UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CONTADURÍA Y
ADMINISTRACIÓN

MEJORAS AL RECONOCIMIENTO ÓPTICO DE CARACTERES Y A LA CORRECCIÓN DE TEXTOS ELECTRÓNICOS EN LOS SISTEMAS DE LECTURA AUTOMÁTICA DE TEXTO

TESIS PROFESIONAL QUE PARA
OBTENER EL TÍTULO DE:
LICENCIADO EN INFORMÁTICA
PRESENTAN:

WULFRANO ARTURO LUNA RAMÍREZ
JORGE BARRÓN MACHADO

ASESOR:
M.C. ESMERALDA URAGA SERRATOS

MÉXICO, D.F. 2004

A mis hermanos y a mis padres, en el entendido de que nunca será suficiente para agradecerles.
A Ek Chua por sus innumerables esfuerzos.
Wulfrano Arturo Luna Ramírez.

A mis padres y hermanos.
Jorge Barrón Machado.

Agradecimientos

El presente trabajo fue realizado con la supervisión académica de la M.C. Esmeralda Uraga Serratos.

Resumen

Los sistemas de lectura automática de texto permiten que las personas invidentes y débiles visuales puedan acceder a un mayor cúmulo de información impresa. Sin embargo, el desempeño de estos sistemas no es completamente satisfactorio. No obstante, dicho desempeño se puede mejorar a través del perfeccionamiento de las diversas partes que constituyen a estos sistemas. Por ello, en esta tesis se presenta tanto un modelo para mejorar el proceso de reconocimiento óptico de caracteres, como un método de corrección de textos electrónicos que minimice los errores de salida del sistema de reconocimiento óptico de caracteres. Las pruebas efectuadas muestran que el modelo de red neuronal convolucional modular, propuesto para realizar el reconocimiento óptico de caracteres, disminuye la tasa de error de reconocimiento hasta en un 68.75%. Por su parte, la evaluación del método de corrección automática de textos evidencia que el uso de categorías gramaticales para realizar la detección de errores y la generación de correcciones potenciales aumenta la tasa de corrección únicamente si el etiquetamiento morfosintáctico del texto es muy preciso.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Desde su surgimiento, la inteligencia artificial ha constituido una fuente de avances notables para las ciencias computacionales, tanto a nivel científico como a nivel tecnológico. Dentro de las diversas investigaciones desarrolladas en este campo se encuentran el reconocimiento óptico de caracteres y el procesamiento del lenguaje natural. Dichas investigaciones han dado lugar a diferentes aplicaciones tecnológicas tales como los sistemas de lectura automática de texto. Estos sistemas resultan de gran ayuda para la población de invidentes y débiles visuales ya que ponen a su alcance información impresa que de otro modo les sería muy difícil o imposible obtener.

Sin embargo, hay que señalar que estas tecnologías todavía distan de ser perfectas, puesto que aún existen errores que hacen que su desempeño se vea afectado negativamente. De esta forma, en un intento por superar estas dificultades, en este trabajo se presentan dos maneras de mejorar los resultados alcanzados hasta el momento por los lectores automáticos de texto.

Para empezar se explican los antecedentes de las disciplinas mencionadas, así como una semblanza de la forma en que han confluido diferentes tecnologías para desarrollar los lectores automáticos, destacando la importancia que éstos tienen para los invidentes y débiles visuales.

A partir de dicha discusión se pasa al planteamiento del problema tratado en este trabajo. También se formulan las hipótesis y se exponen los objetivos de la tesis. Se describe la metodología propuesta para llevar a cabo la investigación. Se menciona la relevancia que tiene este estudio analizándolo desde diferentes perspectivas, y las limitaciones de este trabajo. Finalmente, se da una breve presentación del resto de la obra.

1.1 Antecedentes

Desde la antigüedad el ser humano se ha esforzado en comprender la mente, y por ello, a lo largo de la historia muchos filósofos han tratado de responder diversas interrogantes tales como: ¿qué es el pensamiento?, ¿qué es la mente?, ¿cuáles son las características que nos distinguen de otros entes? [32]. De esta forma, el nombre científico *homo sapiens* (hombre sabio), empleado para designar al ser humano, muestra el gran valor que el hombre le atribuye a su habilidad mental, misma que considera como una de sus características más trascendentales y distintivas [81], y gracias a la cual puede ver, aprender, recordar, razonar y comunicarse a través de un lenguaje.

Así, para muchos filósofos la capacidad de razonamiento constituye una cualidad distintiva del ser humano. Por ejemplo, Dreyfus afirma que el hombre se crea a sí mismo y al mundo de hechos que lo rodean [18]. Según él, este mundo es organizado por el ser humano mediante sus capacidades propias para satisfacer sus necesidades inherentes y por lo tanto es ilógico suponer que un mundo organizado conforme a las capacidades humanas fundamentales (creación, razonamiento, comprensión de un lenguaje natural, etc.) pueda ser accesible por cualquier otro medio.

No obstante, a través de los años han existido personas cuyo interés se ha enfocado en la posibilidad de construir un ser inteligente con capacidades similares a las nuestras. Wilhem Leibnitz (1646–1716) por ejemplo, consideraba factible la creación de un diccionario en el que cada concepto se definiera mediante conceptos más simples, con lo cual creía que la producción de conocimiento científico podría automatizarse. De hecho, construyó un dispositivo mecánico llamado calculadora escalonada, para manipular diversos símbolos no restringidos a la aritmética. Desgraciadamente, su máquina nunca funcionó correctamente [25].

Con el advenimiento de la computadora digital —un dispositivo electromecánico de propósito general para el procesamiento de información—, muchos investigadores comenzaron a vislumbrar como algo posible la creación de un ente capaz de tener ciertas capacidades humanas. Fue así como surgió la inteligencia artificial, disciplina que ha sido abordada desde diversos enfoques a lo largo de su historia. Según lo expuesto en [81] las diferentes corrientes de la inteligencia artificial se pueden agrupar en cuatro grandes categorías en función de los objetivos que se pretende alcanzar.

La primera de estas cuatro categorías está formada por aquellos investigadores que consideran que la inteligencia artificial se debe enfocar en el

estudio y desarrollo de **sistemas que piensen como humanos**. Por ejemplo, para Haugeland el objetivo de la inteligencia artificial es hacer que las computadoras piensen, es decir, que posean mente como los humanos, lo cual es posible ya que, de acuerdo con él, nosotros mismos somos computadoras [32].

Una segunda categoría agrupa a quienes opinan que el objetivo de la inteligencia artificial debe ser el desarrollo de **sistemas que piensen racionalmente**. Así, Winston opina que la inteligencia artificial es el estudio de los cálculos que permiten percibir, razonar y actuar [94].

En la tercera categoría se encuentran los investigadores que consideran que lo importante es el desarrollo de **sistemas que actúen como humanos**. De este modo, Rich considera que la inteligencia artificial estudia la manera de hacer que las computadoras realicen lo que hasta ahora los humanos hacen mejor [80].

Por último, la cuarta categoría incluye a quienes plantean que la inteligencia artificial se enfoca en el estudio y desarrollo de **sistemas que actúen racionalmente**. En este sentido, Luger y Stubblefield afirman que la inteligencia artificial estudia los mecanismos subyacentes del comportamiento inteligente por medio de la construcción y evaluación de dispositivos que implementen dichos mecanismos; la meta es el entendimiento de los principios generales de la conducta racional [53].

Los seguidores de cada uno de estos enfoques han realizado aportaciones valiosas a la inteligencia artificial. No obstante, resulta indispensable adoptar un punto de vista determinado al realizar una investigación con el fin de poder contar con un marco de referencia útil durante el desarrollo de la misma. Para efectos del presente trabajo, se adopta la cuarta categoría cuyo enfoque es la construcción de sistemas que actúan racionalmente, esto es, que hacen lo correcto sin importar si el resultado se obtiene como producto de un proceso de razonamiento: el interés está en el resultado no en la forma de llegar a él.

A partir de todos estos enfoques se han creado diversos campos de estudio dentro de la inteligencia artificial, de los cuales se describirán sólo los dos que se encuentran vinculados con la presente investigación:

1. Visión computacional.

2. Procesamiento del lenguaje natural.

1.1.1 Visión Computacional

La investigación en el área de **visión computacional** o **visión artificial** tiene como objetivo la reproducción artificial del sentido de la vista, objetivo bastante complejo y cuyo logro aún se encuentra distante en la actualidad [56].

En esta área se aprecia la importancia que tiene la capacidad visual tanto para el hombre como para los animales. Por un lado, muchos seres vivos dependen en alto grado de la percepción visual para su supervivencia. Por otra parte, cerca del 75% de la información que maneja el ser humano proviene de la visión. A pesar de que un hombre invidente puede desarrollar plenamente su capacidad mental, y gozar de un buen nivel de vida, la falta de visión le obligará a depender en cierta forma de sus semejantes y de algunos medios de interacción especiales con el entorno. Por consiguiente, se puede concluir que la visión es vital para muchos seres vivos [56].

Una de las disciplinas ubicadas dentro de la visión por computadora es el **reconocimiento óptico de caracteres**, la cual puede abordarse desde dos perspectivas: la científica y la ingenieril. Por un lado, desde el **punto de vista científico** se plantea como objetivo principal la comprensión del concepto abstracto de la forma de un carácter y la manera en que se puede identificar cualquier instancia de tal concepto. Como se puede apreciar, este planteamiento representa un problema altamente complejo, cuya solución aún se encuentra lejana. Por otro lado, el **enfoque ingenieril** busca la construcción de sistemas prácticos, a pesar de no contar con una teoría completa [61].

La importancia de contar con una representación abstracta de un carácter radica en que tal representación contiene las características esenciales que diferencian a cada carácter de los demás, y que permite agrupar a todas las instancias de dicha representación a pesar de las variaciones debidas al tamaño, posición, rotación, etcétera. Si bien dicha representación se halla fuera del alcance actual, el avance en esta disciplina ha sido significativo, pues se han conseguido buenos resultados en problemas cada vez menos restringidos. De esta forma, uno de los temas de investigación actuales es el reconocimiento de caracteres escritos a mano, que representa un problema poco restringido, a diferencia de los problemas abordados inicialmente como el reconocimiento de caracteres impresos de tamaño constante y de una sola fuente [61].

Existen diferentes métodos para llevar a cabo el reconocimiento óptico de caracteres, tales como la comparación de plantillas, los métodos estruc-

turales de clasificación, los clasificadores con funciones discriminativas, los clasificadores bayesianos y las redes neuronales artificiales, entre otros [86]. Dentro de este conjunto de métodos se puede señalar a las redes neuronales artificiales como una de las técnicas más convenientes para llevar a cabo tareas de reconocimiento. Esto se debe a que una red neuronal artificial aprende a clasificar un conjunto de entradas a partir de una serie de datos de entrenamiento, sin requerir el diseño de algoritmos específicos. Incluso, se puede emplear la misma arquitectura de una red para realizar diversas tareas de clasificación [72]. Más aún, las redes neuronales artificiales han demostrado su superioridad sobre otros métodos de clasificación en términos de precisión de reconocimiento, sin requerir demasiados recursos de memoria y con la realización de un número comparativamente bajo de operaciones [51]. Además, gracias a las redes neuronales artificiales se ha incrementado la precisión de los sistemas de reconocimiento, por lo cual muchos sistemas comerciales modernos de reconocimiento óptico de caracteres emplean alguna forma de redes neuronales artificiales [51], [86].

Es importante mencionar que las redes neuronales artificiales constituyen un foco de investigación muy importante e independiente del reconocimiento óptico de caracteres. Esta área surgió a partir de las neurociencias y de los métodos de la física estadística al vislumbrar la posibilidad de construir dispositivos de procesamiento de información capaces de simular la estructura y principios operativos del cerebro humano [35]. Aparte de los aspectos fisiológicos (las relaciones estímulo-respuesta de las neuronas biológicas) y anatómicos (la estructura interna de las neuronas reales), las propiedades computacionales de las neuronas biológicas constituyen uno de los principales motivos de interés en esta área [91]. En efecto, se puede asegurar que el cerebro humano es más rápido y mejor que una computadora digital en la realización de tareas como la visión y el entendimiento del lenguaje natural [35], [91]. Además, el cerebro humano posee otras características importantes [35]:

- Robusteza y tolerancia a fallas. Su desempeño no se afecta de forma significativa a pesar de que diariamente mueren varias de sus neuronas.
- Flexibilidad. Se puede ajustar al ambiente mediante el aprendizaje.
- Puede tratar con información difusa, probabilística, ruidosa o inconsistente.

- Funcionamiento en paralelo.
- Es pequeño y disipa muy poca energía.

Todo esto ha motivado el estudio y la construcción de redes neuronales artificiales compuestas de pequeñas unidades conocidas como neuronas artificiales (modelos simplificados de una neurona real) para realizar diversas tareas y tratar de resolver problemas que resultan intratables si se emplean herramientas convencionales de cómputo [35], [91]. Es así como las redes neuronales artificiales se han llegado a convertir en una alternativa al paradigma tradicional de cómputo [35].

1.1.2 Procesamiento del Lenguaje Natural

En la sección anterior se mencionó la importancia que tiene la visión para poder interactuar con el mundo; sin embargo, aunque los ojos reciben más estímulos del exterior que los oídos, el proceso de comunicación efectivo entre los seres humanos se da a través del habla. Así, la comunicación hablada constituye una capacidad esencial del ser humano que lo distingue de los animales. La estructura acústica y lingüística del habla se halla profundamente relacionada con nuestra habilidad intelectual, y es por ello que resulta un campo de investigación atractivo para la inteligencia artificial. Además, se puede afirmar que el lenguaje ha desempeñado un papel trascendental en nuestro desarrollo cultural y social [24].

El ser humano utiliza el habla para comunicar sus ideas y por lo tanto, espera ser escuchado por un receptor. Dichas ideas las comunica a través de un código, llamado lengua, el cual le permite estructurar sus pensamientos. La información contenida en el habla se divide en tres tipos principales: información lingüística, esto es, lo que el transmisor quiere decir; información sobre los sentimientos y actitudes del hablante; e información sobre la identidad del emisor.

La naturaleza momentánea de la señal de voz indujo al hombre a desarrollar la lengua escrita, la cual es una representación visual de la lengua hablada. De esta forma, el mensaje es capturado en un medio más perdurable y es susceptible de ser comunicado a varias personas sin importar su ubicación geográfica pues no necesita de la presencia emisor-receptor en el mismo lugar y horario. Además, una de sus principales ventajas es la posibilidad de almacenar grandes cantidades de información.

Tomando en cuenta la importancia de la lengua hablada y escrita se han desarrollado sistemas que intentan reproducir el habla e interpretar los mensajes escritos. Esto ha dado lugar al surgimiento del procesamiento de lenguaje natural, disciplina que busca que las máquinas puedan interactuar con el hombre utilizando un lenguaje natural. De este campo de estudio se deriva el proceso de producción artificial de voz y el reconocimiento del habla. El primero, conocido como **síntesis de voz** permite que una máquina transmita información a una persona en forma oral. Para que dicha transmisión ocurra se requiere de unidades que transporten la información. Por este motivo, la investigación en el área de síntesis de voz se encuentra estrechamente vinculada con la investigación sobre las unidades básicas de transporte de información en el habla y sobre los mecanismos de producción del habla [24]. Por su parte, el reconocimiento del habla busca que las máquinas sean capaces de interpretar una señal de voz de forma adecuada, permitiendo así la comunicación hombre-máquina.

Debido a que mucha de la información que el hombre posee se encuentra en medios escritos y al incremento de los textos electrónicos, dentro del área de síntesis de voz se han desarrollado sistemas encargados de reproducir un mensaje escrito mediante una señal de voz, imitando así, la capacidad de lectura del ser humano. Este proceso se denomina **conversión de texto a voz**. Para llevar a cabo su tarea, los sistemas de conversión de texto a voz primero convierten el texto en una representación lingüística y después toman dicha representación para transformarla en una señal acústica [85]. De esta manera, se puede acceder a información escrita sin tener que leer el texto, lo cual trae consigo las siguientes ventajas [24]:

- Los mensajes hablados se pueden escuchar mientras se realizan otras actividades como caminar, manejar, etc.
- Se puede acceder a información remota a través del teléfono.
- Esta forma de comunicación no requiere de papel.

1.2 Discapacidad Visual y Lectores Automáticos de Texto

En nuestra sociedad existen varias diferencias sociales que propician que un grupo extenso de la población del país viva en condiciones de marginación,

esto es, que esté excluido del proceso de desarrollo y por ende no disfrute de sus beneficios, sufra carencias y falta de oportunidades y se encuentre expuesto a riesgos y vulnerabilidades sociales [3]. El CONAPO considera que la educación es un factor importante para estimar el grado de marginación, por esa razón dicho factor es utilizado para estimar el índice de marginación, junto con otras tres dimensiones estructurales: vivienda, ingresos monetarios y distribución de la población. Es por ello que minimizar el número de personas excluidas de la educación es un aspecto clave para disminuir la marginación, además de que pueden alcanzarse otros beneficios, tanto económicos como sociales, puesto que la educación fomenta la incorporación de la población a la planta productiva, la creación de bienes y servicios mediante la innovación tecnológica, el desarrollo científico y cultural y sienta las bases para consolidar una sociedad mejor organizada y con menos desigualdades.

De esta forma, resulta trascendente señalar la existencia de grupos más vulnerables que aún no reciben la ayuda suficiente, por lo que se encuentran en una situación todavía más complicada. Además de sufrir marginación, estos grupos se enfrentan a otros tipos de exclusión debido a su condición física o mental. Tal es el caso de los discapacitados, mismos que de acuerdo con las cifras del XII Censo General de Población y Vivienda [36] conforman el 1.84 por ciento de la población de nuestro país, es decir, aproximadamente 1,795,300 personas, de las cuales el 26.01 por ciento, esto es, 467,040 personas, constituye la población de discapacitados visuales. Dicha población incluye tanto a los invidentes (personas que tienen nula o mínima percepción de luz), como a los débiles visuales (individuos cuya visión es insuficiente para realizar una tarea deseada —a pesar de contar con los mejores lentes correctivos—, pero que aún poseen una percepción de luz suficiente para orientarse con ella y emplearla con propósitos funcionales) [71].

De forma particular, a los discapacitados visuales se les dificulta el acceso a la educación por varias razones, de las cuales, la más obvia es que necesitan materiales de trabajo y espacios especiales que les permitan desarrollar sus capacidades sin restringirlos a obtener la información o a comunicarse a través de medios visuales. Desgraciadamente, no todas las instituciones de asistencia pueden proporcionar estos espacios especiales, según datos del Sistema Nacional de Información sobre Población con Discapacidad [37], en el caso del Distrito Federal, existen 172 asociaciones que atienden a personas con discapacidad visual, mismas que enfrentan diversos problemas como falta de instalaciones, transporte, material y equipo adecuados, escasez de personal profesional, barreras arquitectónicas, financiamiento insuficiente, carencia de

apoyo público y privado, y poca difusión.

Ya que la educación constituye uno de los factores para disminuir la marginación, a lo largo de la historia se han hecho varios esfuerzos por integrar a los invidentes a ella. Uno de los más relevantes es quizá el trabajo llevado a cabo por Luis Braille quien consideraba que los invidentes no tenían porque ser tratados como personas incapaces, y que merecían un trato igual [13].

Motivado por esta idea, en 1895 Luis Braille creó un sistema de lectura y escritura para ciegos formado por seis puntos, los cuales permitían la formación de 64 signos diferentes incluyendo el espacio en blanco. Gracias a este sistema, ahora los ciegos podían leer y escribir, con lo cual se terminaba la época de analfabetismo para los invidentes y débiles visuales [13].

Sin embargo, en la actualidad la literatura en braille es muy escasa, y de ésta, la mayoría está en inglés [17]. A esto se debe agregar el siguiente hecho: de las personas consideradas como invidentes o débiles visuales sólo el 10% emplea el braille como sistema de lectura, y un porcentaje todavía menor lo utiliza para escribir [13].

Esta información resulta preocupante e indica el valor que tienen los esfuerzos encaminados hacia la búsqueda de los medios que hagan posible un mayor y mejor acceso a la información por parte de los ciegos. El contar con recursos que hagan más accesible la información para los invidentes y débiles visuales les permitirá disfrutar de un mejor nivel de vida intelectual, social, cultural y de esparcimiento.

Al hablar de la necesidad de acceso a la información por parte de los ciegos, se hace referencia a un problema que atañe a la sociedad. Para solucionar éste y otros problemas que afectan al ser humano se puede recurrir a la ciencia y a la tecnología, pues ellas pueden y deben ayudarnos en la búsqueda de soluciones satisfactorias a dichas dificultades. En efecto, según Bernal la ciencia tiene tres objetivos fundamentales: el entretenimiento del científico y la satisfacción de su curiosidad natural (objetivo psicológico), el descubrimiento y el entendimiento del mundo (objetivo racional), y la aplicación de dicho entendimiento en la solución de los problemas del bienestar humano (objetivo social) [9]. Por su parte, la tecnología proporciona los medios para que el ser humano controle su entorno físico y social con el fin de obtener resultados prácticos para satisfacer sus necesidades [43].

Por lo tanto, la inteligencia artificial como ciencia, y de forma más particular las tecnologías que se han desarrollado dentro de este campo de estudio, pueden aportar soluciones viables para apoyar a los invidentes y débiles visuales. Específicamente, los avances realizados dentro del campo del reco-

nocimiento óptico de caracteres y dentro del campo del procesamiento del lenguaje natural aportan tecnologías útiles para los discapacitados visuales.

Originalmente, los sistemas de reconocimiento óptico de caracteres no fueron desarrollados con el fin de asistir a las personas invidentes. Mas bien, la aplicación tradicional de dichos sistemas ha sido facilitar la introducción de texto a la computadora sin la necesidad de capturarlo nuevamente. No obstante, tales sistemas pueden resultar muy útiles para una persona invidente, ya que ofrecen una forma de transformar los documentos impresos en un archivo de texto, el cual posteriormente puede ser leído por un programa especial que lo despliegue en una pantalla braille [79]. De la misma forma, los sistemas de conversión de texto a voz tampoco fueron diseñados para ser usados por los invidentes. A pesar de ello, tales sistemas constituyen una de las principales formas en que una persona invidente puede acceder a la información que proporciona una computadora [79].

Como se puede apreciar, la conjunción de ambas tecnologías puede proporcionar un instrumento útil para el acceso a la información por parte de los invidentes y débiles visuales. Empero, se debe precisar lo siguiente: cuando se habla de personas invidentes y débiles visuales, es indispensable distinguir entre dos grupos, aquellos que perdieron la vista en edades en las que ya sabían leer y escribir, y los niños que nacen con ceguera o que la adquieren en edades muy tempranas [33]. Esta distinción se efectúa porque la pertenencia a uno de estos grupos incide directamente en el tipo de asistencia que se les brinda en las escuelas. Además, el empleo de la tecnología sin tomar en cuenta estas previsiones puede conducir a un retraso en la alfabetización de los ciegos debido a la comodidad que brindan los sistemas de conversión de texto impreso en voz comparada con el esfuerzo que implica la lectura táctil. Incluso se puede llegar a la desalfabetización general de los ciegos, y a la larga, a la inutilización del braille si los responsables videntes de la educación, la información y la cultura se dejan llevar por la inmediatez de los datos audibles [13]. No obstante, la tecnología asistencial constituye una extraordinaria herramienta para las personas discapacitadas, ya que les permite realizar tareas que anteriormente hubieran sido muy difíciles e incluso imposibles de lograr. Además, el empleo de la tecnología de asistencia por parte de los invidentes, tanto en sus actividades académicas y laborales como en sus actividades recreativas, ha mejorado notablemente su calidad de vida [2].

Uno de los productos de la tecnología de asistencia está constituido por los lectores automáticos de texto impreso, los cuales son el resultado de la unión

entre los sistemas de reconocimiento óptico de caracteres y los convertidores de texto a voz [33]. Dichos sistemas —formados por un escáner y sintetizadores de voz— resultan de gran ayuda para muchas personas invidentes al permitirles el acceso a una gran cantidad de información impresa. No obstante, los resultados obtenidos no son perfectos y para ciertos documentos la cantidad de errores es tanta que el mensaje hablado se hace ininteligible [15].

1.3 Planteamiento del Problema

Como se puede apreciar, los sistemas de lectura automática involucran el uso de diferentes tecnologías como el reconocimiento óptico de caracteres y el procesamiento del lenguaje natural. Dichas tecnologías aún se encuentran en desarrollo y por ende presentan problemas diversos. Así pues, como lo demuestran estudios recientes [23], [67], los resultados obtenidos por los sistemas de reconocimiento óptico de caracteres no son perfectos, aún cuando dichos sistemas son capaces de entregar resultados con una tasa de precisión de más del 90%. No obstante, estos resultados dependen de la calidad de la imagen sobre la que se trabaje y de su complejidad. Por ejemplo, según un estudio comparativo de programas de reconocimiento óptico de caracteres del 2002 llevado a cabo por la FCMC ningún sistema fue capaz de reconocer un texto con un fondo de texturas y colores.

Algunas de las dificultades enfrentadas por los programas de reconocimiento óptico de caracteres son provocadas por la baja calidad del papel o de la impresión de los documentos originales; las fallas en la segmentación de caracteres de documentos originales que contienen caracteres rotos, superpuestos y fuentes no estandarizadas; la presencia de ruido en la imagen y; la dificultad para trabajar con imágenes de baja resolución [66], [88], [75]. Como se puede constatar, las causas de los errores de reconocimiento son múltiples y es necesario buscar formas efectivas de prevenir y/o corregir dichos errores.

Por una parte, los errores se pueden reducir al mejorar las técnicas de reconocimiento. Así, lo ideal sería contar con un modelo de reconocimiento completamente libre de errores. Aunque esto resulta imposible por ahora, lo realmente factible es disminuir las tasas de error actuales.

Por otro lado, los errores presentes en el texto reconocido pueden reducirse mediante un proceso de corrección ortográfica. Por ello, es importante saber

que en un texto electrónico comúnmente se encuentran diferentes tipos de errores ortográficos que consisten fundamentalmente en [44]:

- Inserción. Añadir un carácter a una palabra.
- Sustitución. Cambiar un carácter de la palabra original por otro.
- Transposición. Intercambiar la posición de dos caracteres adyacentes.
- Omisión/Borrado. Eliminar un carácter de la palabra original.
- Segmentación. Crear dos o más palabras a partir de una.
- Unión. Crear una palabra a partir de dos o más.

Estos errores se propagan también en la estructura de los sintagmas como errores gramaticales (cambios de función y forma de las palabras) y errores semánticos (cambios de significado). Atendiendo a lo anterior, se han hecho otros esfuerzos con el afán de incorporar información lingüística de varios tipos, además de utilizar técnicas de aprendizaje de máquina (o aprendizaje automático). Estos métodos están basados principalmente en técnicas estadísticas, simbólicas y sus combinaciones: árboles y listas de decisión, redes neuronales artificiales, algoritmos genéticos, modelos ocultos de Markov, algoritmos de agrupamiento, n-gramas (conformados por palabras), etc. [46], [44], [57], [89].

Es necesario señalar que estos métodos se han aplicado con resultados todavía no muy satisfactorios en la mayoría de los casos[1] debido a varios factores como la dependencia del método con respecto al corpus y/o lexicones que utiliza (lo que provoca que palabras que no están registradas en él, pero que no son equivocadas se detecten como errores) y a la dificultad para obtener el sentido de las palabras en textos sin restricciones, lo que trae como consecuencia que se creen estructuras erróneas o combinaciones absurdas dentro de un idioma. Lo anterior redunda en que se tengan casos en que no se detectan los errores, se corrigen mal, o se insertan errores donde no los había.

Tomando en cuenta toda esta discusión, los problemas que se tratarán en este estudio son:

[1] Aún cuando se combinan varios métodos, los porcentajes de corrección apenas sobrepasan el 60% [46], [89]

P_1 Determinar si la aplicación de la modularidad en las redes neuronales convolucionales ayuda a reducir la tasa de error de reconocimiento.

P_2 Determinar si la aplicación de un proceso de etiquetamiento morfosintáctico aumenta la tasa de corrección de errores ortográficos en textos electrónicos.

1.4 Hipótesis

La hipótesis correspondiente al problema P_1 es la siguiente:

H_1 Un sistema que emplee un conjunto de n redes neuronales convolucionales, en donde cada una de las redes está entrenada para distinguir a una sola clase de caracteres y rechazar a los $n-1$ caracteres no pertenecientes a tal clase, tendrá una menor tasa de error que la de un sistema que utilice solamente una red neuronal convolucional entrenada para distinguir a un carácter de entre los n caracteres identificables por el sistema.

La hipótesis planteada para el problema P_2 es la siguiente:

H_2 El proceso de corrección ortográfica contextual, basado en la asignación de categorías gramaticales y análisis morfológico de las palabras de un texto electrónico, mejora la tasa de corrección en comparación con el sólo análisis contextual y morfológico.

1.5 Objetivos

Los objetivos de la presente tesis son:

1. Desarrollar un modelo de red neuronal artificial que emplee el concepto de modularidad para llevar a cabo el reconocimiento óptico de caracteres.

2. Disminuir la tasa de error del proceso de reconocimiento óptico de caracteres.

3. Aplicar una técnica de marcación gramatical a un texto electrónico para identificar la categoría de las palabras erróneas contenidas en él.

4. Basándose en la categoría gramatical de las palabras erróneas, aplicar un análisis morfológico y contextual para generar una lista de correcciones potenciales y seleccionar a la mejor candidata para sustituir a la palabra incorrecta en el texto a corregir.

1.6 Metodología

El propósito de la metodología que se describe a continuación es cumplir con los objetivos fijados en la sección anterior y comprobar las hipótesis planteadas. Para ello, se ha dividido esta labor en dos partes: reconocimiento óptico de caracteres y corrección del texto electrónico.

1.6.1 Reconocimiento Óptico de Caracteres

1. Obtener un conjunto de imágenes de entrenamiento, validación y evaluación.

2. Diseñar y entrenar a cada una de las n redes neuronales convolucionales con los conjuntos de entrenamiento recopilados.

3. Integrar a las n redes neuronales convolucionales.

4. Entrenar a una sola red neuronal convolucional para distinguir los n caracteres, la cual servirá de patrón de comparación.

5. Evaluar la arquitectura de n redes neuronales convolucionales especializadas mediante el conjunto de prueba y determinar su tasa de error.

6. Evaluar la red neuronal convolucional general mediante el conjunto de prueba y determinar su tasa de error.

7. Comparar los resultados obtenidos por ambas arquitecturas.

8. Determinar cuál es la mejor arquitectura.

1.6.2 Corrección de Textos Electrónicos

1. Reunir un corpus de textos en español para obtener información estadística.

2. Crear un corpus de prueba integrado por textos en español.

3. Crear un conjunto de diccionarios de categorías gramaticales del español.

4. Crear una matriz de confusión que contenga caracteres parecidos y errores comúnmente introducidos por los sistemas de reconocimiento óptico de caracteres.

5. Implantar una técnica de marcación morfosintáctica para asignar categorías gramaticales a las palabras de un texto.

6. Diseñar e implantar un método que utilice técnicas de análisis morfológico y contextual para detectar y corregir errores en el ámbito de las palabras.

7. Diseñar e implantar un método de corrección ortográfica basado en la categoría gramatical de las palabras.

8. Evaluar los dos métodos antes mencionados con el corpus de prueba y comprar los resultados obtenidos.

1.7 Importancia del Estudio

1.7.1 Relevancia Tecnológica

La arquitectura de red neuronal convolucional modular presentada en este trabajo permite reducir la tasa de error del reconocimiento óptico de caracteres y por ende constituye un avance tecnológico que permitirá el desarrollo de sistemas de reconocimiento más confiables para llevar a cabo diversas tareas, tales como la lectura automática de textos para invidentes.

Por su parte, el método de corrección ortográfica de textos electrónicos basado en la asignación de categorías gramaticales propone una mejor forma de detectar errores y generar correcciones candidatas para cada palabra incorrecta, lo cual implica una contribución tecnológica para los correctores ortográficos automáticos.

1.7.2 Relevancia Humana

Una de las principales razones por las que se busca mejorar el proceso de reconocimiento es aumentar la eficacia de los lectores automáticos de texto. La reducción de errores en estos sistemas resulta trascendental para un invidente, ya que no hay forma de que él se percate de ciertos errores cometidos por el sistema de reconocimiento. Este uso contrasta con las aplicaciones tradicionales de captura de información en que un ser humano puede corregir los errores cometidos por el reconocedor óptico de caracteres al comparar el texto de entrada con el resultado del reconocedor.

De esta forma, los modelos presentados en esta tesis permitirán ofrecer mejores sistemas lectores a los invidentes, con lo cual ellos podrán acceder a un cúmulo mayor de información escrita.

1.8 Limitaciones del Estudio

Los recursos para realizar investigaciones en las áreas de reconocimiento óptico de caracteres y de procesamiento de lenguaje natural son escasos. Dicha escasez se incrementa cuando se trabaja con idiomas distintos al inglés. Por esta razón, el alcance de estas investigaciones siempre es limitado. Para disminuir los efectos negativos que la falta de recursos impone, es conveniente aprovechar los recursos que la comunidad de investigadores hace disponibles en el ámbito académico.

En lo que corresponde al reconocimiento óptico de caracteres se utilizará la base de datos MNIST [51] compuesta de dígitos escritos a mano. Dicha base de datos presenta las siguientes ventajas:

- Los dígitos escritos a mano constituyen un buen punto de referencia para comparar modelos de reconocimiento de figuras.
- Contiene datos del mundo real recolectados de manera rigurosa.
- Ha servido para evaluar diversos modelos de reconocimiento de figuras.
- Se encuentra disponible en Internet de forma gratuita.

Por su puesto, esto limita las pruebas a un conjunto pequeño de clases reduciendo el alcance de los resultados. No obstante, el trabajar con conjuntos más grandes incrementaría notablemente el tiempo requerido para el entrenamiento y prueba de los dos modelos tratados en esta tesis.

Por otro lado, realizar trabajos de procesamiento estadístico de lenguaje natural requiere la existencia de grandes cantidades de información, en este caso, textos electrónicos en español. Esto plantea una seria dificultad si se toma en cuenta que los recursos en nuestro idioma son muy limitados y que la mayoría de ellos no está disponible en los círculos académicos. Tal es el caso de los corpus de texto en español, los cuales son escasos, pequeños y no tienen información lingüística o estadística asociada a ellos. Como es de esperarse, esta situación se acrecenta para el caso de corpus anotados morfosintácticamente. Lo mismo sucede con el caso de los lexicones o diccionarios, los cuales son muy reducidos en número y tamaño y no se basan en estudios lexicográficos de la lengua.

Esto provoca que se tengan que recopilar los datos requeridos cada vez que se realiza una investigación. Por lo tanto, no hay una estandarización en cuanto a la forma en que se guardan o clasifican los recursos. De esta manera, se dificulta crear técnicas que aprovechen la información lingüística y la apliquen al procesamiento de lenguaje natural.

Como consecuencia de lo anterior en este trabajo se reunen los corpus textuales y diccionarios de aquellos sitios en internet que ofrecen una vasta cantidad de textos en español no siempre libres de errores [48]. Además, se creará un corpus etiquetado morfosintácticamente para poderlo emplear en el modelo de corrección ortográfica propuesto.

Finalmente, se debe considerar que el número de pruebas llevadas a cabo es reducido. Esto se debe a que la cantidad de pruebas que puede efectuarse es prácticamente ilimitada. Por ende, la generalidad de los resultados obtenidos refleja dicha restricción.

1.9 Organización de la Tesis

En los siguientes capítulos se analizará de forma detallada cada uno de los modelos que se proponen en este trabajo. Para empezar, en el capítulo 2 se explica el proceso de adquisición y procesamiento de imágenes. Dicho proceso constituye una etapa relevante que debe realizarse antes de poder iniciar el reconocimiento óptico de caracteres.

En los capítulos 3 al 5 se aborda el modelo de red neuronal convolucional modular propuesto. En primer lugar, en el capítulo 3 se expone el marco teórico necesario para entender dicho modelo. Posteriormente, el capítulo 4 presenta el modelo propuesto. Por último, los resultados de la evaluación del

modelo de red neuronal convolucional modular y del modelo de red neuronal convolucional general se discuten en el capítulo 5.

Por su parte, en los capítulos 6 al 8 se trata lo referente al método de corrección ortográfica de textos electrónicos. Así, los fundamentos de la corrección ortográfica se revisan en el capítulo 6. A continuación, en el capítulo 7 se muestra tanto el método de corrección propuesto, como el método de corrección tradicional. Los resultados de las pruebas de los dos métodos se incluyen en el capítulo 8.

Por último, en el capítulo 9 se da una conclusión final con base en los resultados logrados en esta tesis tanto con el modelo de red neuronal convolucional modular como con el método de corrección ortográfica basado en la asignación de categorías gramaticales.

Al final de la tesis se han agregado 3 apéndices. El primer apéndice presenta el código de los programas desarrollados para implementar y probar los modelos. El segundo apéndice contiene el etiquetario de categorías gramaticales utilizado tanto para crear el corpus de entrenamiento, como para realizar la corrección ortográfica. El tercer apéndice está integrado por la matriz de confusión empleada para crear el corpus de prueba.

Capítulo 2

Adquisición y Procesamiento de la Imagen

El reconocimiento óptico de caracteres actúa sobre las imágenes de los caracteres obtenidas a partir de un documento impreso. Dicho documento es un medio de naturaleza analógica que debe ser convertido en una forma digital antes de que una computadora pueda procesarlo [87]. Así, el primer paso consiste en la **captura** del documento mediante algún dispositivo, generalmente un escáner o una cámara de vídeo. Con ello, el contenido de la hoja impresa queda almacenado en la computadora en forma de una imagen digitalizada [69].

Esta imagen digital tendrá que atravesar por un proceso de **segmentación**. Este proceso permitirá, en primera instancia, separar el texto del fondo. Posteriormente, se podrán identificar los renglones, las palabras y los caracteres individuales [69].

Finalmente, una vez separados los caracteres, es necesario aplicar un proceso de **normalización**. Dicho proceso es sumamente relevante para el modelo de reconocimiento propuesto en esta investigación. En este proceso se modifica el tamaño de la imagen, de tal forma que todas las imágenes adopten un tamaño único. Asimismo, el rango de la escala de grises debe reducirse para que la imagen pueda ser procesada por el sistema de reconocimiento óptico de caracteres.

Teniendo como base este esquema, en las siguientes secciones se analizan de manera más detallada los procesos de adquisición y digitalización de la imagen, segmentación y normalización. También se muestran las tareas involucradas en cada proceso y los distintos métodos existentes para llevarlos

a cabo.

2.1 Adquisición y Digitalización de la Imagen

Mientras que en el mundo real los parámetros de las imágenes que percibimos tienen valores continuos, los dispositivos digitales compuestos de un sistema electrónico de imágenes sólo pueden trabajar con imágenes cuyos parámetros sean discretos [1]. Para comprender el proceso que una computadora debe llevar a cabo para adquirir una imagen, resulta ilustrativo analizar el modelo general de una imagen continua.

Una **imagen** es una función continua de intensidad luminosa $f_c(i, j)$ que tiene asignada un valor de brillo para cada punto (i, j) en el plano bidimensional, en donde los valores más altos corresponden a una mayor intensidad. Es importante mencionar que la disposición de los ejes coordenados en una imagen es ligeramente distinta a la disposición común del plano cartesiano. Como se puede ver en la figura 2.1, el origen se halla en la esquina superior izquierda de la imagen. A partir de este punto, los valores de i crecen hacia abajo mientras que los valores de j crecen hacia la derecha [27].

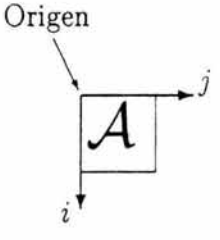


Figura 2.1: Arreglo de los ejes coordenados en una imagen.

Como la luz es una forma de energía, los valores que puede tomar la función de una imagen continua deben ser finitos y mayores a cero [27], es decir:

$$0 < f_c(i, j) < \infty \tag{2.1}$$

En el mundo real, la luz que reflejan los objetos es lo que forma las imágenes que se perciben, por ello se dice que la imagen tiene dos componentes: la iluminación y la reflectancia. La **iluminación** es la cantidad de luz que

incide en la escena, y se denota por la función $l(i,j)$. Por su parte, la **reflectancia** se refiere a la cantidad de luz reflejada por los objetos presentes en la escena, y su función es $r(i,j)$. Considerando estos dos componentes, la imagen está dada por [27]:

$$f_c(i,j) = l(i,j)r(i,j) \tag{2.2}$$

Donde $0 < l(i,j) < \infty$, y $0 < r(i,j) < 1$. Por lo tanto, los fenómenos de **absorción total**, cuando $r(i,j) = 0$, y de **reflectancia total**, cuando $r(i,j) = 1$, quedan excluidos.

Como se puede apreciar, la imagen capturada por un sensor es una función continua $f_c(i,j)$ que debe ser digitalizada para que pueda ser procesada por una computadora [27], [84]. Esta digitalización se debe llevar a cabo tanto sobre el dominio como sobre el rango de la función [84]. Esto significa que se tendrá que efectuar un **muestreo** para obtener valores discretos en el espacio y una **cuantización** para obtener valores discretos de brillo [1], [27].

Una vez que los valores de brillo y las coordenadas han sido discretizados, se obtiene una **imagen digital** que puede representarse mediante una matriz de tamaño $M \times N$ como en la figura 2.2. En dicha matriz los valores de renglón y columna identifican a un punto de la imagen y el elemento correspondiente de la matriz representa el valor de brillo de dicho punto [27], [84]. Los elementos que integran a la matriz son conocidos como **pixeles** o **pels** (abreviaturas de *picture elements*).

$$f_d(i,j) = \begin{bmatrix} f_d(0,0) & f_d(0,1) & \dots & f_d(0,N-1) \\ f_d(1,0) & f_d(1,1) & \dots & f_d(1,N-1) \\ \vdots & & & \\ f_d(M-1,0) & f_d(M-1,1) & \dots & f_d(M-1,N-1) \end{bmatrix}$$

Figura 2.2: La representación de una imagen digital en forma de matriz.

Los valores de M, N y el número de niveles de gris que puede tomar un pixel, generalmente toman valores en potencias de dos; esto es: $M = 2^m$, $N = 2^n$, $G = 2^k$, donde G es el número de niveles de gris en el rango $[0, L]$, conocido como **escala de grises**. En dicha escala de grises el 0 equivale a negro, L (el máximo nivel de gris) representa el blanco, y los valores intermedios representan diferentes niveles de gris, tal y como se muestra en la figura 2.3.

Figura 2.3: El rango de niveles de gris va de 0 a L. El nivel 0 corresponde al color negro mientras que el nivel L corresponde al blanco.

Como consecuencia, el número de bits requerido para almacenar una imagen digital se obtiene mediante la expresión [27]:

$$b = NMk \tag{2.3}$$

Así, la **resolución de la imagen**, es decir el grado de detalle distinguible, depende del número de muestras tomadas (**resolución espacial**), del rango de la escala de grises (**resolución radiométrica**) y del ancho de banda de frecuencias de luz que capta el sensor (**resolución espectral**). Entre más grandes sean estos valores, la imagen digital estará más próxima a la imagen original y por ello será de mayor calidad [84]. Sin embargo, de la ecuación 2.3 se tiene que el crecimiento de estos parámetros también implica un mayor espacio de almacenamiento y por lo tanto el tiempo de procesamiento de la imagen se incrementa también [27].

De estas observaciones se puede deducir la importancia que tiene el equilibrio en la selección de los parámetros de muestreo y cuantización. Las siguientes secciones abordan de manera detallada los aspectos relativos al muestreo y a la cuantización. Además, se incluye una sección donde se explica de manera general el funcionamiento de los dispositivos de captura de imágenes.

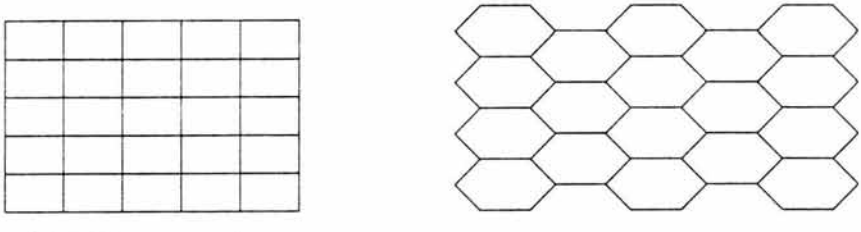
2.1.1 Muestreo

Los dispositivos de captura de imágenes llevan a cabo un proceso de **muestreo** para obtener una imagen discreta en el espacio a partir de una imagen continua en el espacio [1]. De manera general, el proceso de muestreo puede ser visto como la colocación de una rejilla en el espacio bidimensional de forma que las coordenadas de la imagen (i, j) que pertenecen al conjunto de los números reales R^2 son aproximadas a sus valores enteros más cercanos con respecto a la rejilla; así, todo par ordenado (i, j) pertenecerá al conjunto de enteros Z^2 [27], [84].

Para llevar a cabo el muestreo es necesario establecer cuál será la separación entre las muestras y qué forma geométrica tendrá la malla de muestreo [84].

En primer lugar, resulta importante determinar el número mínimo de muestras que se debe tomar de forma que no se pierda información de la imagen. Es decir, se requiere establecer las condiciones que garanticen que el conjunto de muestras obtenido permita la reconstrucción completa de la imagen original [27]. Aunque en la práctica esta reconstrucción completa es imposible, los digitalizadores generalmente emplean intervalos de muestreo bastante pequeños con el fin de minimizar la pérdida de información [84].

El otro aspecto importante para llevar a cabo el muestreo es la disposición de los puntos de muestreo. Estos puntos de muestreo se encuentran dispuestos en forma de rejilla, la cual puede ser rectangular o hexagonal, como se muestra en la figura 2.4. Por lo regular, se emplea una malla rectangular como la de la figura 2.4(a) para llevar a cabo el muestreo. Así, la estructura de datos que generalmente se emplea para almacenar la imagen es una matriz que contiene los valores de brillo de la imagen en cada casilla.



(a) Malla rectangular.

(b) Malla hexagonal.

Figura 2.4: Mallas de muestreo.

Finalmente, resulta interesante observar de que forma se afecta la calidad de la imagen cuando se varía la resolución espacial. Para ilustrar esto, a continuación se muestran unos ejemplos obtenidos mediante la manipulación de una imagen cuadrada ($M = N$).

En la figura 2.5 se muestran cuatro imágenes cuya diferencia estriba en su resolución espacial. La figura 2.5(a) muestra la imagen original de 256×256 pixeles por pulgada con una escala de grises de 256 niveles. Por su parte, las figuras 2.5(b)-(d) muestran el efecto adverso de reducir el número de

muestras en el espacio. La calidad de la imagen se reduce debido a que se deben replicar los pixeles para llenar el espacio sobrante [27].

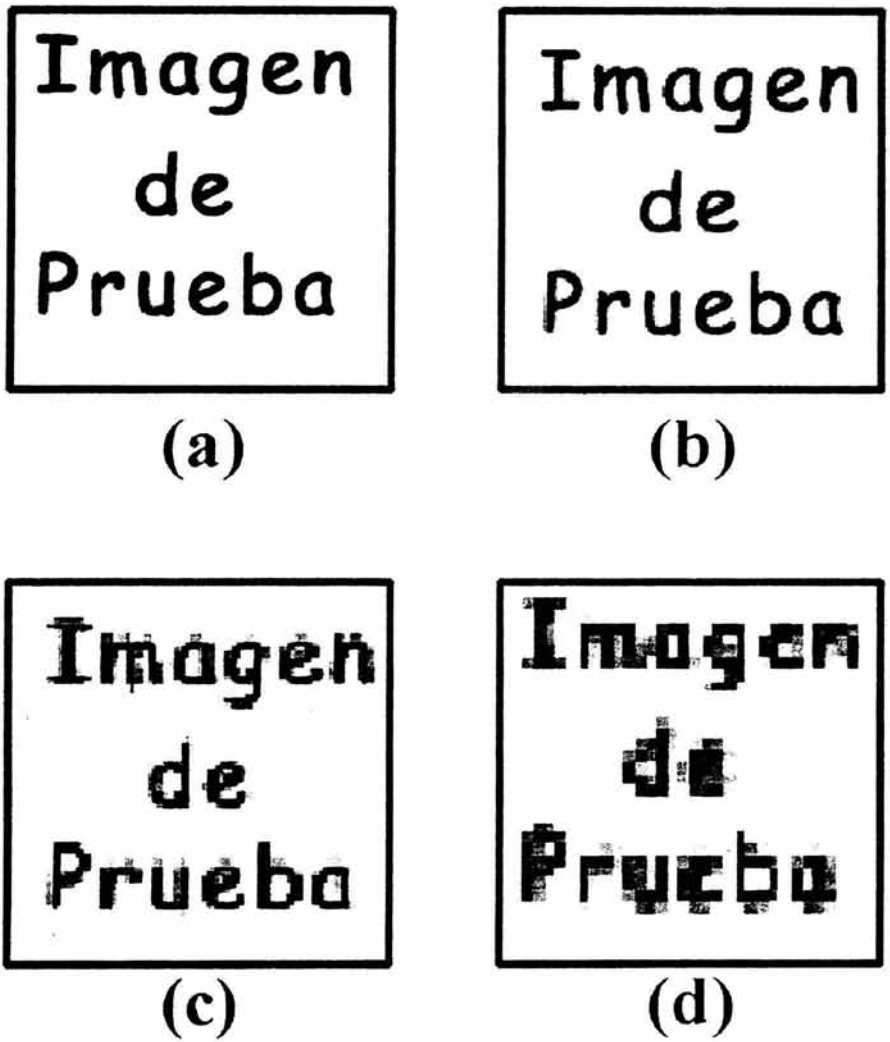


Figura 2.5: Efectos producidos en una imagen por la reducción de la resolución espacial. (a) Imagen original de 256 × 256 pixeles por pulgada con una escala de grises de 256 niveles. (b) Imagen de 128 × 128 pixeles por pulgada. (c) Imagen de 64 × 64 pixeles por pulgada. (d) Imagen de 32 × 32 pixeles por pulgada.

Una comparación a simple vista entre la imagen original y la imagen de la figura 2.5(b) —cuya resolución espacial es de 128×128 pixeles por pulgada— no permite distinguir su diferencia claramente; no obstante, la segunda imagen muestra un granulado mayor que la primera y ha perdido nitidez. La degradación se vuelve más severa en la imagen de la figura 2.5(c) (de 64×64 pixeles por pulgada), cuyo contorno muestra un patrón parecido al de un tablero de ajedrez, y en la cual el efecto de granulado se ha incrementado. Por último, en la imagen de la figura 2.5(d) se aprecia una degradación profunda causada por el número limitado de muestras de la imagen: 32×32 pixeles por pulgada.

2.1.2 Cuantización

La **cuantización** permite que los dispositivos de adquisición de imágenes aproximen los valores de brillo de una imagen a un conjunto finito de niveles de representación simbolizados mediante bits. De esta forma, los valores de la función $f_c(i, j)$ son discretizados, con lo cual los diferentes valores de los niveles de gris quedan comprendidos en el conjunto de enteros Z [27], [84]. Desafortunadamente, este proceso introduce ruido y es irreversible [78].

La cuantización puede ser escalar o vectorial, dependiendo de si se cuantizan las muestras una por una o de forma agrupada. De estas, la **cuantización escalar** es la más sencilla de realizar. Dicha cuantización, dada por la función $Q[x]$, constituye un mapeo de los valores reales de x a un valor discreto de un conjunto finito, con base en la siguiente regla [78]:

$$\text{Si } x \in R_i \Longrightarrow Q[x] = y_i \tag{2.4}$$

Donde $R_i = (x_i, x_{i+1})$ para $i = 1, \ldots, L$ representa intervalos contiguos a lo largo de la línea real, pudiendo ser un intervalo abierto, cerrado o semiabierto. y_i para $i = 1, \ldots, L$ son los **niveles de representación** o **valores de reconstrucción** a los cuales se mapea x. Los valores x_i, que definen las particiones, se conocen como **puntos de decisión**. Esto se muestra gráficamente en la figura 2.6.

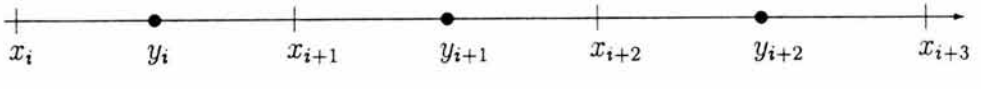


Figura 2.6: Cuantización escalar.

En un **cuantizador uniforme** todos los intervalos son del mismo tamaño y los niveles de reconstrucción se encuentran en los puntos medios. El número de niveles seleccionado depende de las características de la imagen.

No obstante, es importante optimizar la función de cuantización mediante la reducción de la distorsión entre los valores originales y los niveles de representación dado cierto número de niveles. Así, una forma de optimización que se consigue mediante la minimización de la distorsión dado un número L de niveles de representación se conoce como el **cuantizador Lloyd-Max**. En este, cada valor se cuantiza empleando el mismo número de bits.

Si en lugar de utilizar un número fijo de bits para codificar cada valor, se emplea un número variable, se reduce la tasa de distorsión. Esta optimización lleva a un **cuantizador de entropía limitada**, en el cual los códigos de longitud variable para los niveles de representación se pueden conseguir mediante el algoritmo de Huffman o con codificadores aritméticos.

Los resultados obtenidos con la cuantización escalar se pueden mejorar realizando la cuantización de varias muestras simultáneamente, lo cual se conoce como **cuantización vectorial**. Se trata de una generalización de la cuantización escalar en la que un vector N-dimensional continuo x se mapea a un vector N-dimensional discreto si la siguiente regla se cumple:

$$\text{Si } x \in C_i \Longrightarrow Q[x] = y_i \tag{2.5}$$

Donde C_i representa la región o celda i de L celdas N-dimensionales conocidas como **regiones de Voronoi**, que corresponden a las regiones de decisión, y que pueden verse como polígonos sólidos en el espacio N-dimensional. Dichas celdas son contiguas y no se superponen. Los valores y_i son los niveles de representación conocidos como el **libro de códigos**.

Para saber a que región pertenece una muestra se utiliza una **medida de distorsión**:

$$Q[x] = y_i \iff d(x, y_i) \leq d(x, y_j), \qquad j = 0, \ldots, L-1 \tag{2.6}$$

El índice i identifica al vector y_i que dio la mejor comparación y por lo tanto es codificado como la mejor representación del vector. Para reconstruir el vector y_i se busca el contenido de la celda i en el libro de códigos.

El libro de códigos se diseña con base en la función de densidad de probabilidad N-dimensional, pero si se desconoce, se puede obtener mediante un conjunto de entrenamiento compuesto por varios vectores representativos

de la imagen, los cuales se pasan a través de un algoritmo iterativo como K-means.

2.1.3 Dispositivos de Adquisición de Imágenes

En los dispositivos de captura de imágenes existen dos aspectos básicos que se deben considerar [1]. El primero es el **raster de puntos** en el espacio donde las muestras se toman. El segundo es el **efecto de apertura** del sistema que hace que los datos muestreados estén constituidos por un promedio de la imagen dentro de la vecindad del punto de muestreo nominal.

De esta forma, los dispositivos de adquisición de imágenes se agrupan en tres categorías dependiendo del mecanismo de adquisición de muestras que utilicen [1].

Dentro de la primera categoría se hallan los dispositivos que utilizan un mecanismo de punto móvil para la adquisición de los datos. En estos dispositivos la trayectoria del punto y los tiempos de lectura determinan el raster de muestreo, mientras que los efectos de apertura dependen de los puntos de iluminación y de lectura, y del intervalo de tiempo durante el cual la señal de salida del punto lector se promedia para formar un valor de muestra. Algunos ejemplos de este mecanismo incluyen: el haz de electrones de las cámaras de vídeo analógicas; el escaneo mecánico a partir de la rotación de un tambor y del movimiento de un tornillo como en los escáneres de tambor para artes gráficas; la formación del rayo óptico difractivo de los escáneres de punto de venta; y la formación de un arreglo de rayos sincronizado de un radar. Ciertos sistemas como los sistemas aéreos y espaciales para la detección remota de la superficie de la tierra funcionan de modo pasivo, es decir, no hay un punto de escritura y el punto de lectura detecta la radiación que emana naturalmente de la escena.

La segunda categoría utiliza un mosaico de plano focal, compuesto por un arreglo de zonas detectoras. En estos dispositivos, la escena es tomada en la superficie del arreglo y cada detector integra la radiación recolectada del área activa de su superficie, con lo cual se incrementa el efecto de apertura. El arreglo espacial de los detectores es el que determina el raster de muestreo. Algunas de las tecnologías que emplean este mecanismo incluyen a los dispositivos de carga asociada (CCD), los dispositivos de inyección de carga (CID) y los dispositivos CMOS. Estas tecnologías se utilizan en cámaras y vídeo cámaras digitales.

La tercera clase se compone de dispositivos híbridos que emplean ambos

mecanismos. Dentro de esta clase se encuentra el **escáner de cama plana**, el cual se compone de tres partes básicas [86]:

1. Un **detector** y sus partes electrónicas asociadas.
2. Una **fuente de iluminación**.
3. Un **lente de escaneo**.

El proceso de adquisición se inicia cuando la fuente ilumina el objeto mientras que la lente forma la imagen del objeto en el detector. El detector se compone de un arreglo unidimensional de elementos, cada uno de los cuales convierte la luz incidente en una carga o señal analógica. Estas señales analógicas se convierten en una imagen [86]. El proceso de escaneo es llevado a cabo por el detector que se mueve mecánicamente a lo largo de la superficie del documento a escanear [1].

2.2 Segmentación

Para poder llevar a cabo el reconocimiento del texto contenido en una imagen se deben identificar diferentes elementos en la misma. En primer lugar, se habrá de separar el texto del fondo de la imagen, dividir dicho texto en renglones y, finalmente, aislar cada una de las imágenes de los caracteres para que puedan ser identificadas por el clasificador.

Ahora bien, el sistema de reconocimiento recibe cada una de las imágenes y devuelve el símbolo correspondiente a dicha imagen. Este resultado, habrá de ser entregado en el orden apropiado y con los espacios adecuados entre palabras. Por esta razón, la identificación de vocablos resulta importante también [72].

2.2.1 Separación del Texto del Fondo

La **segmentación del texto** permite separar la información textual del fondo, obteniendo un conjunto de regiones disjuntas que corresponden a cada uno de los caracteres del texto, siempre y cuando exista contraste entre el texto y el fondo. Bajo tales condiciones, se pueden emplear métodos globales de segmentación que utilizan histogramas de características, en este caso el brillo, para realizar dicha separación [84].

Uno de los más simples procesos de segmentación es la umbralización. Asumiendo que los objetos no se tocan, y que los niveles de gris del fondo y de los objetos son claramente distintos, se puede determinar un valor de brillo constante, esto es un **umbral**, que permita la segmentación completa de una imagen f_d en un conjunto finito de regiones $R_1 \ldots R_n$ tal que la unión de las n regiones forme la imagen original y ninguna región se traslape con otra:

$$f_d = \bigcup_{i=1}^{n} R_i \qquad R_i \bigcap R_j = \varnothing \quad i \neq j \tag{2.7}$$

Así, la **umbralización** transforma una imagen digital en escala de grises f_d en una imagen binaria f_b, mediante las siguientes ecuaciones:

$$\begin{aligned} f_b(i,j) &= C1 \quad \text{si } f_d(i,j) \geq U \\ f_b(i,j) &= C2 \quad \text{si } f_d(i,j) < U \end{aligned} \tag{2.8}$$

Donde $C1$ y $C2$ son los dos valores de brillo de la nueva imagen binaria; y U es el umbral de segmentación. Por lo general, la imagen f_b emplea el valor 0 (negro) para los pixeles del texto y 255 (blanco) para los pixeles del fondo. Por ejemplo, la figura 2.7 muestra una imagen en escala de grises y su correspondiente representación binaria [84].

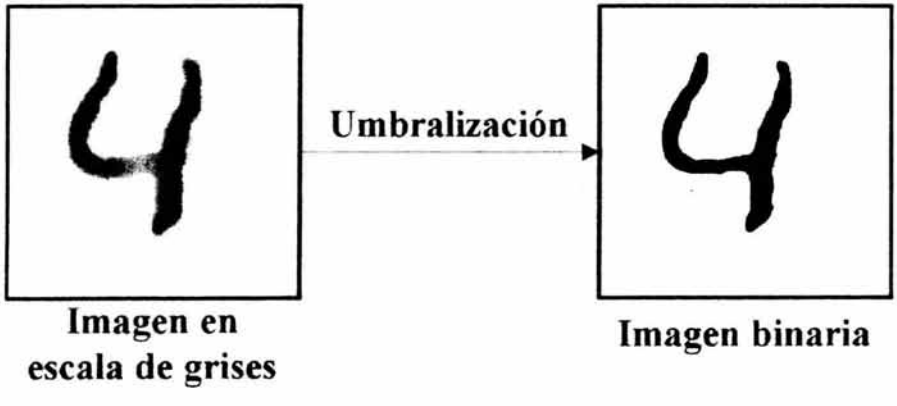


Figura 2.7: El proceso de umbralización convierte una imagen en escala de grises en una imagen binaria.

Existen diferentes métodos para seleccionar el umbral U: métodos de entropía, métodos de relajación, métodos de multiumbralización, métodos de

umbralización óptima, entre otros. Los métodos de **umbralización óptima** aproximan el histograma de una imagen mediante sumas ponderadas de dos o más densidades de probabilidad con una distribución normal. Así, el umbral se elige como el nivel de gris más cercano correspondiente a la probabilidad mínima entre el máximo de dos o más distribuciones normales, con lo cual se obtiene una tasa de error mínima. Este umbral óptimo debe maximizar la varianza de gris entre los objetos y el fondo. Este método se puede aplicar aún si se requiere más de un umbral [84].

En el siguiente algoritmo se muestra la implementación de dicho método asumiendo que la imagen contiene regiones en dos niveles de gris principales, como en el caso del texto [84].

Algoritmo 2.1. Umbral Óptimo
Variables de Entrada:

f_d Una imagen en escala de grises de tamaño $M \times N$

Variables de Salida:

U El valor óptimo del umbral

Variables Locales:

px_total Total de pixeles de f_d

px_fondo Número de pixeles de f_d que pertenecen al fondo

px_texto Número de pixeles de f_d que pertenecen al texto

sum_fondo Suma de los valores de f_d que pertenecen al fondo

sum_texto Suma de los valores de f_d que pertenecen al texto

μ_fondo Media del fondo

μ_texto Media del texto

tmp Variable temporal que almacena el umbral

1. $sum\_fondo = f_d(0,0) + f_d(0,N-1) + f_d(M-1,0) + f_d(M-1,N-1)$
2. $\mu\_fondo = \frac{sum\_fondo}{4}$

3. $sum\_texto = 0$

4. Para $i = 0$ hasta $M - 1$ hacer

   Para $j = 0$ hasta $N - 1$ hacer

   $sum\_texto = sum\_texto + f_d(i, j)$

5. $\mu\_texto = \frac{sum\_texto - sum\_fondo}{px\_total - 4}$

6. $tmp = \frac{\mu\_fondo + \mu\_texto}{2}$

7. Hacer

   (a) $U = tmp$

   (b) $px\_fondo = 0$, $px\_texto = 0$

   (c) $sum\_fondo = 0$, $sum\_texto = 0$

   (d) Para $i = 0$ hasta $M - 1$ hacer

      i. Para $j = 0$ hasta $N - 1$ hacer

         A. Si $f_d(i, j) \geq U$ entonces

            $px\_texto = px\_texto + 1$

            $sum\_texto = sum\_texto + f_d(i, j)$

         B. Si $f_d(i, j) < U$ entonces

            $px\_fondo = px\_fondo + 1$

            $sum\_fondo = sum\_fondo + f_d(i, j)$

   (e) $\mu\_fondo = \frac{sum\_fondo}{px\_fondo}$

   (f) $\mu\_texto = \frac{sum\_texto}{px\_texto}$

   (g) $tmp = \frac{\mu\_fondo + \mu\_texto}{2}$

   Mientras $U \neq tmp$

8. Regresar U

El algoritmo por lo general requiere de 4 a 10 iteraciones y además tiene un buen desempeño en imágenes con una gran variación de contraste [84].

2.2.2 Búsqueda de los Renglones

Los caracteres en un documento están organizados en líneas siguiendo algunas convenciones tipográficas. Estas convenciones pueden ser útiles para localizar los caracteres y las palabras [7]. En el caso más sencillo, un documento está organizado en líneas horizontales, sin columnas, y con un flujo de lectura de izquierda a derecha y de arriba hacia abajo.

Asumiendo esta disposición y partiendo de la imagen binaria, se pueden extraer los renglones de texto de forma fácil. Para ello, se debe obtener el **perfil horizontal** de la imagen, también conocido como **proyección horizontal**. Como se puede ver en el algoritmo 2.2, este perfil o proyección se obtiene contando el número de pixeles negros por cada renglón [62], [72].

Algoritmo 2.2. Perfil Horizontal
Variables de Entrada:

f_b Una imagen binaria de tamaño $M \times N$ con un grado de inclinación de 0

Variables de Salida:

$Perfil$ Un arreglo que representa el perfil horizontal de f_b

1. Para $i = 0$ hasta $M - 1$ hacer
   (a) $Perfil_i = 0$
2. Para $i = 0$ hasta $M - 1$ hacer
   (a) Para $j = 0$ hasta $N - 1$ hacer
   Si $f_b(i, j) = 0$ entonces
   $Perfil_i = Perfil_i + 1$
3. Regresar $Perfil$

De esta forma, si en el perfil la suma correspondiente a un renglón dado es igual a cero, entonces ese renglón está en blanco; de lo contrario, el renglón contiene caracteres [62], [72]. El algoritmo 2.3 muestra la forma en que este perfil horizontal se utiliza en la búsqueda de renglones.

Algoritmo 2.3. Búsqueda de Renglones
Variables de Entrada:

f_b Una imagen binaria de tamaño $M \times N$ con un grado de inclinación de 0

$Perfil$ Un arreglo que representa el perfil horizontal de f_b

Variables de Salida:

$rens\_ini$ Una lista con las coordenadas donde se inicia cada renglón

$rens\_fin$ Una lista con las coordenadas donde se termina cada renglón

Variables Locales:

b_ini Si su valor es verdadero indica que se está buscando el inicio de un renglón

Funciones Locales:

$insertar(l, e)$ Inserta e en la lista l y devuelve la lista resultante

1. b_ini = verdadero
2. Para $i = 0$ hasta $M - 1$ hacer
   (a) Si $Perfil_i > 0$ entonces
   Si b_ini = verdadero entonces
   $rens\_ini = insertar(rens\_ini, i)$
   b_ini = falso
   (b) Si no
   Si b_ini = falso entonces
   $rens\_fin = insertar(rens\_fin, i)$
   b_ini = verdadero
3. Regresar $rens\_ini$ y $rens\_fin$

2.2.3 Separación de las Imágenes de los Caracteres

Gracias a la umbralización, los caracteres presentes en la imagen han quedado separados del fondo. Ahora, la imagen de cada carácter deberá ser extraída de la imagen binaria. El proceso para llevar a cabo esta tarea se puede basar en un análisis de la conectividad del texto.

Así pues, se considera a cada carácter como una **región 8-conectada**. Esto quiere decir que todos los pixeles de la región comparten el mismo valor de brillo del pixel y están conectados unos con otros en forma horizontal, vertical o diagonal. Estos componentes conectados se pueden etiquetar con el propósito de separar cada carácter del texto [21]. Para lograr esto, se revisa cada pixel de la imagen de arriba abajo y de izquierda a derecha buscando los pixeles del texto (negros); y cada que se encuentra un pixel negro un proceso recursivo asigna un nivel de gris mayor a cero a dicho pixel y a todos sus vecinos. Al regresar del proceso, el nivel de gris a usar se incrementa en uno y se continua el barrido de la imagen hasta terminar.

Al final del proceso, los caracteres se hallan identificados con un nivel de gris distinto, y por lo tanto, el total de niveles de gris empleados indica el número de caracteres presentes en el texto. Una vez localizado cada uno de los caracteres se procede a buscar los puntos extremos de cada carácter para encerrar a cada región en un cuadro. Estos cuadros serán bastante útiles para los siguientes pasos.

En primer lugar, se procede a unir los acentos, puntos, diéresis y tildes con sus respectivas letras, buscando por cada recuadro aquellos recuadros que quedan por encima de él y dentro del renglón donde se ubica. La figura 2.8 ilustra este proceso.

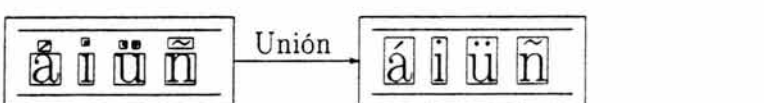


Imagen con recuadros para cada carácter

Imagen con los acentos, puntos, diéresis y tildes unidos

Figura 2.8: Unión de caracteres con sus respectivos puntos diéresis y tildes.

En segundo lugar, las distancias horizontales que separan a cada recuadro de sus vecinos indican el tamaño de los espacios entre caracteres y entre palabras. Con estas distancias se pueden obtener los lugares en que existen espacios entre palabras. Así, debido a que el espacio entre caracteres y entre

palabras es más o menos uniforme en un texto impreso, la distribución del tamaño de estos espacios sigue una distribución normal como se ilustra en la gráfica de la figura 2.9.

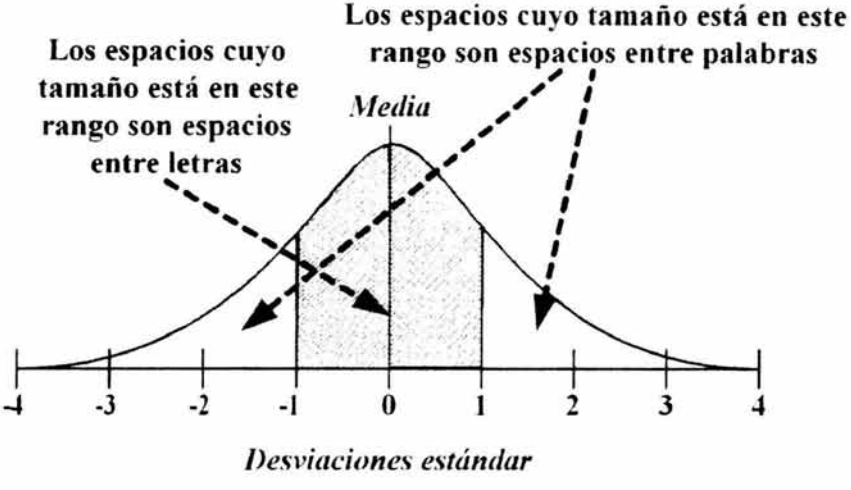


Figura 2.9: La distribución del tamaño de los espacios entre caracteres sigue una distribución normal. Las distancias entre caracteres al interior de una palabra se hallan dentro de una desviación estándar de la media. Las distancias entre palabras vecinas se hallan fuera de esta área.

Los espacios entre palabras son de mayor tamaño que los espacios entre los caracteres de una misma palabra. Sin embargo, la cantidad de espacios entre palabras es menor a la cantidad de espacios entre caracteres. Así, tomando en cuenta la media y la desviación estándar de las distancias entre caracteres se puede afirmar que los espacios entre caracteres se hallarán distribuidos dentro de una desviación estándar de la media. Por lo tanto, 2/3 de los espacios entre caracteres no son espacios entre palabras. El tercio restante de los espacios corresponde a espacios entre palabras.

Estos cálculos se efectúan por cada renglón para garantizar la existencia de la distribución normal. De otra forma, distintos espaciados podrían afectar estos cálculos y provocar errores en la detección de palabras.

2.3 Normalización

Como se verá en el siguiente capítulo, una red neuronal convolucional trabaja sobre imágenes de caracteres con un tamaño normalizado y centradas en el campo de entrada. Además de la normalización del tamaño, el rango de valores de brillo de la imagen también se normaliza [51].

La forma en que estas operaciones deben llevarse a cabo y los métodos empleados para ello se describen en las secciones que se presentan a continuación. Con estas operaciones se concluye el procesamiento de las imágenes. Así, las imágenes quedan listas para ser recibidas por el sistema de reconocimiento óptico de caracteres.

2.3.1 Normalización del Tamaño

El proceso de **normalización del tamaño**, conocido como **escalación**, es equivalente a la modificación de la tasa de muestreo de una imagen digital [16]. Este proceso constituye uno de los tipos de transformaciones geométricas existentes, y para llevarlo a cabo se requieren dos pasos:

1. Realizar la **transformación de las coordenadas** de los pixeles, es decir, obtener las coordenadas de los puntos de la imagen de salida a partir de las coordenadas de los pixeles de la imagen de entrada.

2. Como los puntos obtenidos en el paso anterior no corresponden necesariamente a valores enteros, se deben encontrar los puntos correspondientes en el raster digital y determinar su brillo. Para ello, se realiza una **interpolación de brillo** tomando en cuenta varios puntos en una vecindad [84].

Estos pasos se detallan en las dos secciones siguientes.

Transformación de las Coordenadas

El cambio de escala de una imagen por un factor a en el eje i y un factor b en el eje j se obtiene mediante las ecuaciones:

$$\begin{aligned} i' &= ai \\ j' &= bj \end{aligned} \tag{2.9}$$

Donde los factores a y b están dados por:

$$\begin{aligned} a &= N_{rn}/N_{ro} \\ b &= N_{cn}/N_{co} \end{aligned} \tag{2.10}$$

Donde N_{ro} es el número de renglones de la imagen original; N_{co} se refiere al número de columnas de la imagen original; N_{rn} representa la cantidad de renglones de la imagen normalizada; y N_{cn} corresponde al número columnas de la imagen normalizada [16].

Interpolación de Brillo

Como se dijo anteriormente, las nuevas coordenadas de los pixeles (i', j') no necesariamente tienen valores enteros y, por lo tanto, no tienen una correspondencia con el raster digital de la nueva imagen. Por esta razón, se requiere obtener valores enteros en el raster de salida cuyo valor de brillo podrá obtenerse mediante una interpolación entre los vecinos no enteros [84].

Para comprender mejor como llevar a cabo la interpolación, el problema se puede plantear de otra forma: determinar el brillo del punto original en la imagen de entrada que corresponde al punto de la imagen de salida en el raster discreto. De esta forma, lo que se busca es el valor de brillo del pixel (i', j') de la imagen de salida donde las coordenadas (i', j') se hallan en el raster discreto en la imagen de salida. Tomando en cuenta este nuevo planteamiento, las coordenadas del punto (i, j) de la imagen original se pueden obtener invirtiendo las ecuaciones de escalación [84]:

$$\begin{aligned} i &= i'/a \\ j &= j'/b \end{aligned} \tag{2.11}$$

Dichas coordenadas, en general, no quedan en el raster discreto de la imagen de entrada y por ello su brillo es desconocido. Para obtener el brillo del punto (i, j) será necesario muestrear nuevamente la imagen de entrada.

Existen diversos métodos de interpolación, entre los cuales destacan la interpolación del vecino más cercano, la interpolación bilineal y la interpolación bicúbica. El método que se elija para la interpolación afectará la calidad de la imagen, pues entre más simple sea, mayor será la pérdida en precisión geométrica y fotométrica. Sin embargo, el vecindario empleado para la interpolación no debe ser muy grande, pues incrementa considerablemente la carga computacional [27], [84].

De los tres métodos mencionados, el de la interpolación del vecino más cercano resulta simple de implementar pero los resultados obtenidos son po-

bres. Por su parte, la interpolación bicúbica ofrece resultados más finos, pero su costo computacional es alto. Así, resulta razonable emplear la interpolación bilineal para llevar a cabo la normalización, puesto que los resultados entregados por este método son satisfactorios [27].

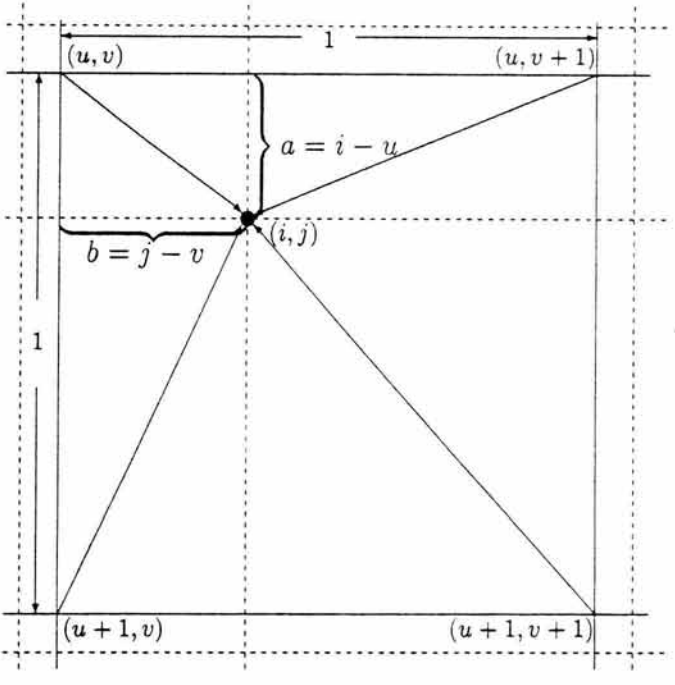


Figura 2.10: Interpolación bilineal. El raster discreto de la imagen original aparece en líneas sólidas. En dicho raster las coordenadas (i, j) del nuevo píxel no son discretas, pero en el raster de la nueva imagen (mostrado en líneas punteadas) las coordenadas de ese píxel son discretas. Lo que se desconoce es el valor de brillo del píxel en la nueva imagen. Para determinar ese valor de brillo se utilizan los valores de brillo de los cuatro pixeles vecinos de la imagen original (u, v), $(u, v+1)$, $(u+1, v)$ y $(u+1, v+1)$ que rodean al nuevo píxel.

La **interpolación bilineal** toma en cuenta cuatro vecinos al punto (i, j) asumiendo que la función de brillo es lineal en este vecindario, tal como se

muestra en la figura 2.10. La ecuación para la interpolación bilineal está dada por [84]:

$$\begin{aligned} f_i(i,j) = \; & (1-a)(1-b)f_d(u,v) \\ & +b(1-a)f_d(u,v+1) \\ & +a(1-b)f_d(u+1,v) \\ & +abf_d(u+1,v+1) \end{aligned} \tag{2.12}$$

Donde u es igual al valor redondeado de i; v es igual al valor redondeado de j; $a = i - u$; y $b = j - v$.

Los valores de brillo de cada uno de los vecinos del punto (i, j) afectan el valor del nuevo píxel en proporción a la distancia a la cual se hallan del mismo. De esta forma, entre más cercano se encuentra el vecino, más contribuye al nuevo valor. Así, en el ejemplo de la figura 2.11 el vecino (u, v) tiene una mayor influencia, pues su peso se multiplica por el valor del área IV dado por $(1 - a)(1 - b)$. Por el contrario, el vecino $(u + 1, v + 1)$ afecta en menor medida el valor ya que su valor de brillo se multiplica por el área I dada por el producto ab.

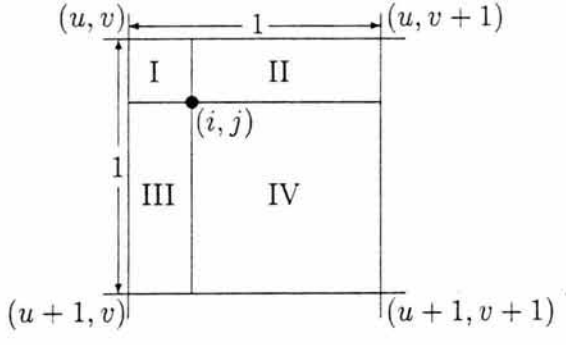


Figura 2.11: Los valores de brillo de cada uno de los vecinos afectan el valor de brillo del punto (i, j) de acuerdo a la distancia que los separa. La ponderación se basa en las áreas de las cuatro regiones. La región I que tiene un área de ab afecta al vecino $(u + 1, v + 1)$. El área de la región II dada por $a(1 - b)$ influye sobre el vecino $(u + 1, v)$. Para el vecino $(u, v + 1)$ se usa la región III cuya área corresponde al producto $b(1 - a)$. La región IV que ocupa un área de $(1 - a)(1 - b)$ afecta al vecino (u, v).

Este tipo de interpolación puede disminuir un poco la resolución y provo-

car que la imagen se haga borrosa, pero el problema de los contornos rectos escalonados es menor que en la interpolación del vecino más cercano.

2.3.2 Centrado de la Imagen

Como se mencionó anteriormente, las imágenes se deben centrar en un campo de entrada de acuerdo con su centro de masa [51]. Para ello, el primer paso es invertir la escala de grises de la imagen, de forma que el 0 corresponda al blanco y el 255 al negro. A partir de esta imagen, el **centro de masa**, también conocido como **centro de gravedad** o **centroide**, se obtiene utilizando las siguientes relaciones [84]:

$$\begin{aligned} i_c &= m_{10}/m_{00} \\ j_c &= m_{01}/m_{00} \end{aligned} \tag{2.13}$$

Donde m_{pq} representa el momento de orden $(p+q)$, dado por la ecuación:

$$m_{pq} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} i^p j^q f_d(i,j) \tag{2.14}$$

Donde i, j son las coordenadas de los pixeles pertenecientes a la región del carácter, es decir, aquellos pixeles cuyo valor de brillo es distinto del fondo (0); M es el número de renglones de la imagen; y N corresponde al número de columnas de la imagen.

Una vez obtenidos los centros de masa, la imagen habrá de trasladarse para que dicho centro coincida con el centro del campo de entrada [51]. De esta forma, cada punto de la imagen de coordenadas (i,j) se traslada a una nueva posición, mediante el uso del desplazamiento (a,b) mediante las ecuaciones:

$$\begin{aligned} i' &= i + a \\ j' &= j + b \end{aligned} \tag{2.15}$$

Donde (i',j') constituyen las coordenadas del nuevo punto [27].

2.3.3 Normalización de la Escala de Grises

Como un último paso, la escala de grises de la imagen de entrada deberá normalizarse, pues el rango con el cual trabaja la red neuronal es menor al

rango utilizado hasta el momento $[0, 255]$ [51]. Las razones por las cuales resulta conveniente normalizar la escala de grises de la entrada son [5]:

1. Evitar que los números más grandes cancelen a los más pequeños.
2. Evitar una saturación prematura de los nodos ocultos de la red neuronal, lo cual dificultaría el proceso de aprendizaje.

Para llevar a cabo la **normalización de la escala de grises**, se deben escalar los valores de intensidad de acuerdo con la siguiente fórmula:

$$f(b_i) = \frac{b_i - b_{min}}{b_{max} - b_{min}}(b'_{max} - b'_{min}) + b'_{min} \tag{2.16}$$

Donde b_i es el valor de gris comprendido en el rango $[b_{min}, b_{max}]$ y $f(b_i)$ es el nuevo valor de gris correspondiente a b_i y que queda comprendido en el nuevo rango $[b'_{min}, b'_{max}]$ [63]. Esta fórmula de carácter lineal resulta suficiente, pues otras técnicas más complicadas no necesariamente producen una mejor solución [5].

2.4 Resumen

Para poder procesar un documento impreso mediante una computadora, es necesario adquirir la imagen de dicho documento. Para ello se emplea un escáner digital, el cual muestrea y cuantiza la imagen original transformándola en una imagen digital que puede ser manejada por una computadora. Esta transformación afecta la calidad de la imagen dependiendo de los parámetros de muestreo y cuantización que se empleen y de la calidad del documento original.

Una vez obtenida la imagen, se procede a segmentarla. Este proceso incluye varias tareas. En primer término, se debe segmentar el texto y separarlo del fondo. Para ello, se emplea el método de umbralización óptima. Posteriormente, se buscan los renglones con la ayuda del perfil horizontal. A continuación, se extraen las imágenes de los caracteres con base en un análisis de la conectividad del texto. Finalmente, se obtienen las posiciones de los espacios entre palabras a partir de un análisis estadístico.

Para concluir, se normaliza el tamaño de la imagen de cada carácter efectuando una transformación de coordenadas lineal y una interpolación de brillo bilineal. Esta imagen se centra de acuerdo con su centro de masa y se

traslada para que dicho centro de masa coincida con el centro del campo de entrada a la red neuronal. Por último, la escala de grises se normaliza para reducir el rango de la misma con el uso de una fórmula lineal.

De esta forma, al final se obtiene una imagen normalizada y centrada por cada carácter. Estas imágenes se encuentran listas para ser procesadas por parte del sistema de reconocimiento óptico de caracteres. El tema del reconocimiento óptico de caracteres será tratado en el próximo capítulo.

Capítulo 3

Reconocimiento Óptico de Caracteres

El ser humano cuenta con una aptitud altamente desarrollada para el procesamiento de patrones, lo cual le permite reconocer figuras fácilmente. Como resultado, la complejidad del reconocimiento óptico de caracteres suele pasarse por alto al considerarse trivial [72]. No obstante, se puede asegurar que dicha tarea pone en juego una de las habilidades fundamentales del hombre: la capacidad de abstracción. En efecto, cada carácter es abstraído por el ser humano, gracias a lo cual puede identificar diferentes instancias del mismo símbolo independientemente de la posición, el tamaño, la rotación, etc. [61].

Ahora bien, implementar el reconocimiento óptico de caracteres en una máquina resulta sumamente complicado. En primer lugar, porque en el alfabeto existen conjuntos de letras cuyas figuras son muy parecidas entre sí. A pesar de ello, los seres humanos son capaces de diferenciarlas, pues aunque tienen similitudes generales, también muestran ciertas diferencias. En segundo lugar, porque todos los caracteres presentan una gran riqueza de figuras. Incluso un mismo carácter está sujeto a muchas variaciones, sobre todo si está escrito a mano. Pese a estas variaciones los humanos frecuentemente reconocen con facilidad un carácter [61]. Estas dificultades permiten apreciar la complejidad de los problemas encontrados en el área del reconocimiento óptico de caracteres.

El tratamiento de tales problemas se puede considerar desde dos puntos de vista distintos: el científico y el ingenieril. Desde la **perspectiva científica**, el objetivo esencial del reconocimiento óptico de caracteres consiste en hallar el concepto general de la forma de un carácter y el mecanismo que permite

identificar cualquier instancia de dicho concepto. Como se puede observar, el problema es sumamente complejo y la búsqueda de su solución aún tomará muchos años de investigación [61]. Sin embargo, el reconocimiento óptico de caracteres se puede abordar desde un **punto de vista ingenieril**, es decir, con la meta de construir sistemas prácticos, a pesar de carecer de una teoría completa. Considerándolo de esta forma, el reconocimiento óptico de caracteres constituye la parte esencial de los sistemas de procesamiento de documentos [61].

El enfoque ingenieril es el que se sigue en el presente trabajo. De esta forma, el **reconocimiento óptico de caracteres** se define como el proceso que transforma la imagen de un carácter en su correspondiente representación simbólica (ver figura 3.1).

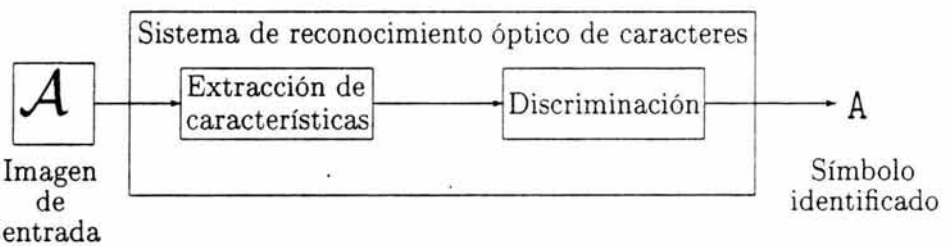


Figura 3.1: El proceso de reconocimiento óptico de caracteres se compone de dos partes principales: extracción de características y discriminación.

Como se ilustra en la figura 3.1, este proceso se puede dividir en dos partes: extracción de características y discriminación [61]. Con la **extracción de características** se determina el conjunto pertinente de características o descriptores que se utilizará para describir a todos los caracteres. Para ello, el extractor de características recibe la imagen del carácter como entrada, a partir de la cual deriva las características que posee el carácter. Por su parte, en el proceso de **discriminación** se emplea el conjunto de características encontrado por el extractor para clasificar al carácter en la clase que le corresponde [86].

Dentro de los múltiples modelos empleados para realizar el reconocimiento óptico de caracteres se encuentran las redes neuronales artificiales. Estas redes han sido utilizadas efectivamente en el reconocimiento óptico de caracteres, tanto impresos [92] como escritos a mano [51], [70]. Los conceptos básicos detrás de estos modelos computacionales se analizan en la primera sección del presente capítulo.

Tomando como base el análisis de las redes neuronales artificiales, en la segunda sección se describe la arquitectura y funcionamiento de las redes neuronales convolucionales. Estas redes constituyen un tipo especial de redes neuronales artificiales diseñado especialmente para tratar con la variabilidad de las imágenes bidimensionales [51].

Para concluir el capítulo se presenta una sección dedicada al análisis de la modularidad en los sistemas de reconocimiento óptico de caracteres y en las redes neuronales artificiales. El objetivo es introducir el concepto de modularidad y mostrar las ventajas que se pueden obtener de su aplicación tanto en los sistemas de reconocimiento como en las redes neuronales artificiales. Esta sección sirve de preámbulo para el capítulo siguiente, en el cual se presenta el modelo de red neuronal convolucional modular propuesto en esta investigación para llevar a cabo el reconocimiento óptico de caracteres.

3.1 Redes Neuronales Artificiales

En la actualidad, las computadoras son capaces de almacenar grandes cantidades de información fácilmente, sus circuitos operan en nanosegundos y pueden ejecutar cálculos aritméticos extensos sin error. Sin embargo, tareas como el entendimiento del lenguaje natural, el reconocimiento de un rostro o el razonamiento de sentido común, que los humanos realizan rutinariamente, se hallan más allá de sus capacidades [45], [91].

Ahora bien, una neurona biológica es extremadamente lenta en comparación con los circuitos de una computadora pues opera en un rango de milisegundos. Sin embargo, los humanos pueden llevar a cabo tareas extremadamente complejas como la interpretación de una escena visual o el entendimiento de una oración en una sola décima de segundo. Esto se debe a que, a diferencia de una computadora convencional, los elementos de procesamiento del cerebro son muchos y actúan en paralelo permitiéndole al cerebro llevar a cabo tareas complejas de reconocimiento de patrones en unos cuantos cientos de milisegundos [22], [45], [82].

Como se puede apreciar, el contar con un sistema artificial capaz de emular las capacidades del cerebro humano permitiría abordar una gran cantidad de problemas que por el momento resultan intratables con las herramientas de cómputo convencionales [91]. Así, con el deseo de construir este tipo de sistemas y con base en los conocimientos biológicos y psicológicos sobre el funcionamiento del cerebro, surge la investigación en el área de redes neuro-

nales artificiales [22], [45], [82].

En la actualidad, esta investigación se nutre de las investigaciones en las áreas de biología, computación, electrónica, matemáticas, medicina, física y psicología [82]. El motivo del interés en las redes neuronales artificiales yace en consideraciones anatómicas y fisiológicas. El interés no es sólo la relación estímulo-respuesta de las neuronas biológicas, sino también la representación de la estructura interna de la neurona [91].

Gracias a las investigaciones, las redes neuronales artificiales han alcanzado notables capacidades de procesamiento de información. Aunado a esto, las redes neuronales artificiales tienen la habilidad de aprender a partir de ejemplos. Como resultado, estos modelos se han convertido en paradigmas eficientes de solución de problemas que suelen usarse en aplicaciones donde es difícil extraer reglas explícitas, pero donde hay suficientes datos disponibles para entrenar a la red con los cuales se puede describir el problema y su solución [82].

Para poder comprender el funcionamiento de estas redes, es necesario comprender primero de qué manera funcionan sus componentes. Para ello, se hace necesario entender el funcionamiento de las neuronas biológicas puesto que constituyen el modelo base de las neuronas artificiales. Así, en las siguientes secciones se analizarán los conceptos relacionados con las neuronas biológicas, las neuronas artificiales y las redes neuronales artificiales.

3.1.1 La Neurona Biológica

La neurona biológica es el bloque básico de construcción del sistema nervioso, el cual está constituido por billones de neuronas de varios tipos y tamaños dependiendo de su localización en el cuerpo [5], [82]. Se trata de células nerviosas que constituyen la unidad básica de procesamiento de información del cerebro y que se agrupan en redes neuronales [91].

La anatomía de la neurona biológica, mostrada en la figura 3.2, comprende tres unidades funcionales principales [5], [82], [91]:

1. **Cuerpo celular o soma.** Contiene los organelos de la neurona, un núcleo que porta información sobre los rasgos hereditarios y plasma que incluye el equipo molecular usado para producir el material que la neurona necesita.

2. **Dendritas.** Son fibras delgadas y ampliamente ramificadas en diferentes direcciones a través de las cuales la neurona recibe señales de

otras neuronas y las pasa al cuerpo celular. El área receptora de las dendritas de una neurona típica es de 0.25 mm^2 aproximadamente.

3. **Axón**. Es una fibra larga que recibe la señal de salida del cuerpo celular y la transporta como impulsos eléctricos (potencial de acción) a lo largo de su camino hasta las ramificaciones que se hallan al final llamadas **botones sinápticos**. De aquí, la señal pasa a través de la sinapsis a las dendritas de las neuronas vecinas.

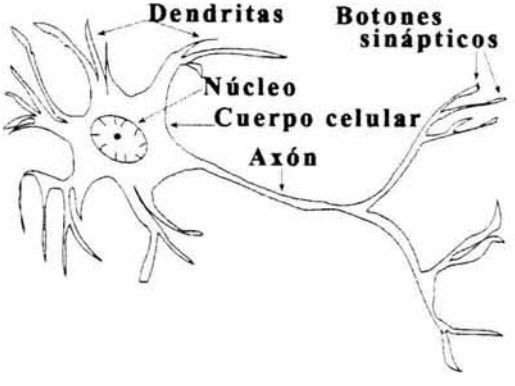


Figura 3.2: Esquema de una neurona biológica.

Las neuronas están conectadas en puntos llamados **sinapsis**, lo cual permite la influencia electroquímica entre ellas [82], [91]. El proceso de transferencia de señales entre dos neuronas se ilustra en la figura 3.3. Un impulso en forma de señal eléctrica viaja dentro de las dendritas hasta llegar al cuerpo celular. De ahí, pasa por el axón hacia la membrana presináptica de la sinapsis. Al llegar a la membrana, un neurotransmisor (un químico) es liberado de las vesículas en una cantidad proporcional a la fuerza de la señal de llegada. El neurotransmisor se difunde dentro del espacio sináptico hacia la membrana postsináptica, y llega así a las dendritas de las neuronas vecinas, forzándolas a generar una nueva señal eléctrica dependiendo del umbral de las neuronas receptoras. La señal generada pasa a través de la segunda neurona de una forma similar [5], [91].

La cantidad de señal que pasa a través de una neurona receptora depende de la intensidad de la señal proveniente de cada una de las neuronas de entrada, de las fuerzas sinápticas y del umbral de la neurona receptora. Debido a que una neurona tiene un gran número de dendritas y sinapsis puede

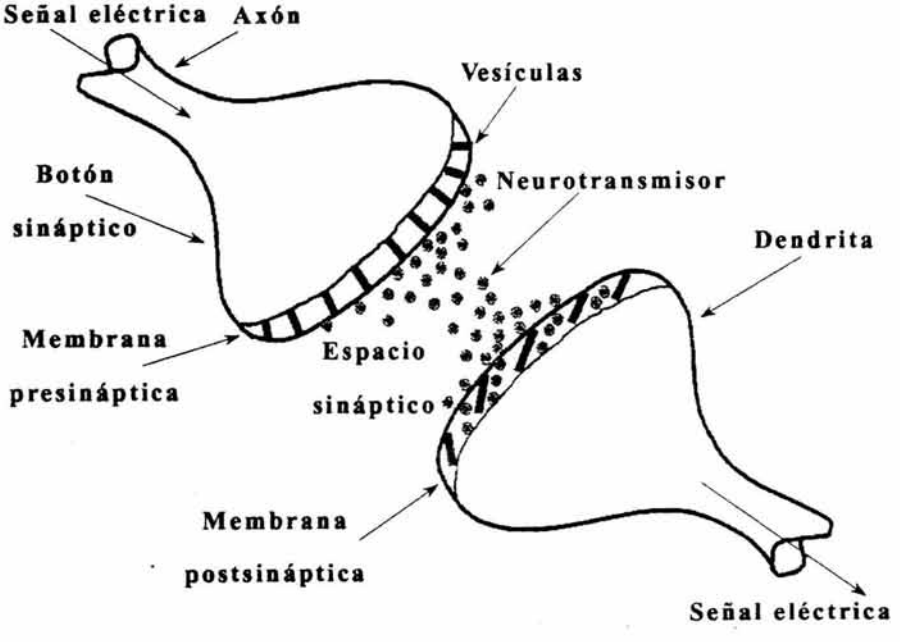


Figura 3.3: Proceso de transferencia de señales entre dos neuronas biológicas.

recibir y transmitir muchas señales simultáneamente. Estas señales excitan o inhiben el disparo de la neurona [5].

Se puede imaginar que las neuronas biológicas realizan la siguiente función: las entradas a la neurona, que pueden variar por la fuerza de las conexiones sinápticas o la frecuencia de la señal de entrada, son sumadas; la suma de entrada es procesada por una función de umbral y cuando el valor de la función excede un umbral, la neurona dispara, es decir, produce una señal de salida [82], [91]. El tiempo de procesamiento de la neurona es de aproximadamente 1 milisegundo por ciclo y su velocidad de transmisión oscila entre 0.6 y 120 m/s, lo cual resulta comparativamente lento para una computadora moderna [82].

3.1.2 La Neurona Artificial

La **neurona artificial** es una unidad de procesamiento basada en la estructura y funcionamiento general de la neurona biológica. Dicha neurona artificial recibe entradas del ambiente o de otras neuronas como estímulo y las combina para obtener un sólo valor de entrada. A dicho valor le agrega un sesgo obteniendo como resultado el **estado de activación** (a) de la neurona. Este estado de activación pasa a través de una **función de activación** (también conocida como **función de transferencia**, **función de ganancia** o **función de aplastamiento**) y la salida (y) se transmite a otras neuronas o al ambiente [5]. Para ilustrar más claramente el funcionamiento de la neurona artificial obsérvese el esquema de la figura 3.4.

En un tiempo t, una neurona j está en un estado de activación a_j. Este estado de activación es calculado como el producto interno (producto punto) de las señales de entrada x que llegan a la neurona y sus fuerzas de unión w, más la adición de un sesgo w_{0j} [5], [82], [91] para lo cual se utiliza la siguiente expresión:

$$a_j = f(x_i, w_{ij}) = \sum_{i=1}^{n} x_i w_{ij} + w_{0j} \tag{3.1}$$

Donde x_i es la i-ésima entrada a la neurona j; w_{ij} es el peso asociado a la conexión que va de la neurona i hacia la neurona j; w_{0j} es el sesgo asociado a la neurona j; y n es el número de entradas que llegan a la neurona j. Los pesos de conexión positivos ($w_{ij} > 0$) incrementan el estado de activación a_j excitando a la neurona, y por ello a los enlaces con pesos positivos se les

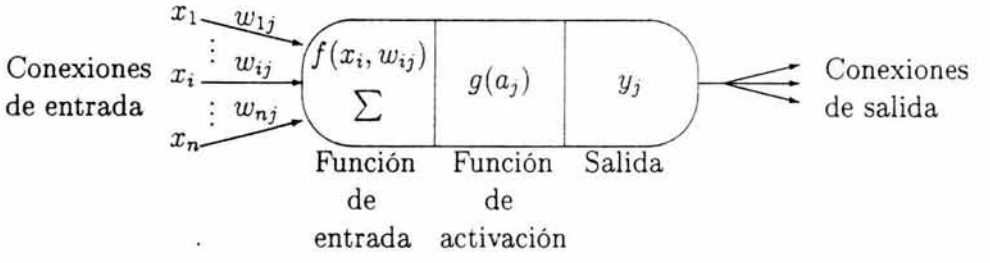


Figura 3.4: Esquema de una neurona artificial. Para cada conexión de entrada x_i hay un peso asociado w_{ij}. Las entradas y sus pesos respectivos se combinan mediante una función de entrada para determinar el estado de activación de la neurona (a_j). Una función de activación g actúa sobre este estado de activación para determinar la salida de la neurona (y_j). Dicha salida se envía hacia el ambiente o hacia otras neuronas, en cuyo caso las neuronas receptoras actúan de forma similar.

conoce como **excitadores**. Los pesos negativos ($w_{ij} < 0$) reducen el estado de activación a_j inhibiendo la actividad de la neurona, por esa razón a las uniones con pesos negativos se les llama **inhibidoras**.

Una función de activación g actuando sobre a_j construye la salida y_j de la neurona j como sigue:

$$y_j = g(a_j) \tag{3.2}$$

Esta salida y_j es enviada hacia las entradas de otras neuronas o hacia el ambiente.

Es así como una neurona artificial establece un mapeo entre la actividad de entrada (estímulo) y la señal de salida (respuesta). De esta forma, la analogía entre una neurona artificial y una neurona biológica es la siguiente [5]:

1. Las conexiones entre neuronas artificiales representan los axones y dendritas.

2. Los pesos de conexión representan las sinapsis.

3. La función de activación simula la actividad en el soma.

Ahora bien, utilizando una sola neurona artificial únicamente se pueden resolver problemas de clasificación en los que los datos son linealmente separables. Lo mismo ocurre con una red de una sola capa de neuronas. Por lo tanto, para resolver problemas no linealmente separables, como el de clasificación de caracteres, se deben utilizar estructuras más complejas con varias capas de neuronas artificiales, con lo cual se obtiene una arquitectura multicapa [45], [82].

3.1.3 Las Redes Neuronales Artificiales

Como se mencionó en la sección anterior, para enfrentarse a los problemas no separables linealmente se necesitan capas de neuronas adicionales entre la **capa de entrada** y la **capa de salida**. Estas capas intermedias no interactúan con el ambiente y por ello se conocen como **capas ocultas**, y a sus neuronas se les conoce como neuronas ocultas. Estas capas intermedias permiten resolver problemas de clasificación no lineales [5], [82]. De esta forma, las neuronas se agrupan en capas empezando con un primer nivel de entrada que recibe las señales del exterior. Estas entradas se propagan a través de varios niveles (niveles ocultos) hasta llegar al último nivel, el nivel de salida [91].

A este arreglo de neuronas en capas se le da el nombre de **red neuronal artificial**. Con el propósito de clarificar lo que se entiende por este término a continuación se presentan algunas definiciones extraídas de diversos autores.

Según Basheer y Hajmeer una red neuronal artificial es una estructura compuesta de elementos de procesamiento simples y adaptables (neuronas artificiales o nodos) densamente interconectados, capaz de llevar a cabo cálculos paralelos masivos para el procesamiento de datos y la representación del conocimiento [5].

Por su parte, Hecht-Nielsen define una red neuronal artificial como una estructura distribuida y paralela de procesamiento de información compuesta de unidades de procesamiento, las cuales cuentan con memoria local, realizan operaciones locales de procesamiento de información y se encuentran interconectadas mediante enlaces unidireccionales [34]. Cada uno de estos elementos de procesamiento genera una señal de salida que envía hacia otros elementos. Dicha señal puede ser de cualquier tipo matemático y es calculada con base en las entradas que recibe la neurona a través de sus conexiones.

Para Hassoun una red neuronal artificial es un modelo de computación paralelo compuesto de unidades básicas de procesamiento adaptables llama-

das neuronas artificiales, las cuales se encuentran densamente interconectadas [31]. Se trata de implementaciones paralelas de sistemas estáticos o dinámicos no lineales. Estas redes son adaptables y aprenden mediante ejemplos, en vez de ser programadas; por esta razón, se pueden aplicar cuando se carece de una comprensión total del problema, pero existen datos de entrenamiento suficientes. Además, su arquitectura paralela permite obtener soluciones rápidamente cuando se implementan en computadoras paralelas.

Desde el punto de vista de Vemuri una red neuronal artificial es un sistema computacional distribuido con un gran número de elementos de procesamiento conectados entre sí [91]. Se trata de un sistema dinámico adaptable, masivamente paralelo y con capacidades de autoaprendizaje capaz de llevar a cabo tareas útiles de procesamiento de información.

Freeman y Skapura opinan que una red neuronal artificial es una colección de procesadores paralelos conectados en forma de una gráfica dirigida, organizada de manera que la estructura de la red se preste al problema en consideración [22].

En su tesis, Schmidt presenta varias definiciones [82]. En la primera de ellas, afirma que las redes neuronales artificiales son redes de nodos adaptables que a través de un proceso de aprendizaje de ejemplos de la tarea almacenan conocimiento experimental y lo hacen disponible para su uso. En otra definición, dice que una red neuronal artificial es un sistema dinámico con interconexiones en un sentido que lleva a cabo el procesamiento mediante su respuesta a entradas. Los elementos de procesamiento son nodos; las interconexiones son enlaces dirigidos. Cada unidad de procesamiento tiene una sola señal de salida que se envía hacia otras unidades. Finalmente, propone que una red neuronal artificial puede considerarse como un conjunto de procesadores simples (neuronas artificiales) interconectados formando un modelo simplificado de las estructuras en el sistema nervioso biológico.

En conclusión, una **red neuronal artificial** es un sistema de procesamiento de información que posee ciertas características de desempeño en común con las redes neuronales biológicas. Se trata de una generalización de los modelos matemáticos de la cognición humana o de la neurona biológica basada en el supuesto de que:

1. El procesamiento de información ocurre en muchos elementos simples llamados neuronas artificiales.

2. Las señales se pasan entre las neuronas artificiales a través de conexiones.

3. Cada conexión tiene un peso asociado que multiplica la señal transmitida. Estos pesos codifican el conocimiento de la red.

4. Cada neurona artificial aplica una función de activación no lineal a su entrada (suma de las señales de entrada ponderadas) para determinar su señal de salida.

5. Estas neuronas son adaptables y por lo tanto permiten que la red aprenda a través de ejemplos.

6. La operación de las neuronas artificiales ocurre de forma distribuida y paralela.

La arquitectura general de una red neuronal artificial se muestra en la figura 3.5. Como se puede apreciar, la idea es tener sistemas neuronales paralelos interconectados basados en unidades simples [82] y compuestos por tres partes: una capa de entrada, una o más capas ocultas y una capa de salida. Así, el flujo de activación va de la capa de entrada a la capa de salida. El conocimiento de la red se codifica en los pesos de las conexiones entre las neuronas artificiales. Los niveles de activación en la capa de salida determinan la salida de la red [45].

La existencia de unidades ocultas permite que la red desarrolle detectores de características complejas [45]. Geométricamente, las capas ocultas se interpretan como hiperplanos adicionales que mejoran la capacidad de separación [82].

Por su puesto, para que la red neuronal sea capaz de realizar su tarea es necesario entrenarla con un conjunto de ejemplos. Para ello, la red se inicia con un conjunto aleatorio de pesos el cual se ajusta cada vez que se presenta un par de entrada-salida. Primero se presenta el ejemplo de entrada a la red para que calcule su salida. Después, en un flujo hacia atrás el valor de salida de la red se compara con la salida objetivo y se estima el error para las neuronas de salida. Los pesos conectados a las neuronas de salida se ajustan para reducir los errores. Entonces se pueden usar las estimaciones del error de las neuronas de salida para derivar los errores estimados en las neuronas de las capas ocultas. Los errores se propagan hacia atrás hasta las neuronas de entrada [45].

Los pesos se actualizan incrementalmente después de la presentación de cada par de entrada-salida. Una vez que se han pasado todos los pares de entrada-salida y que se han ajustado los pesos en cada ocasión, se dice que

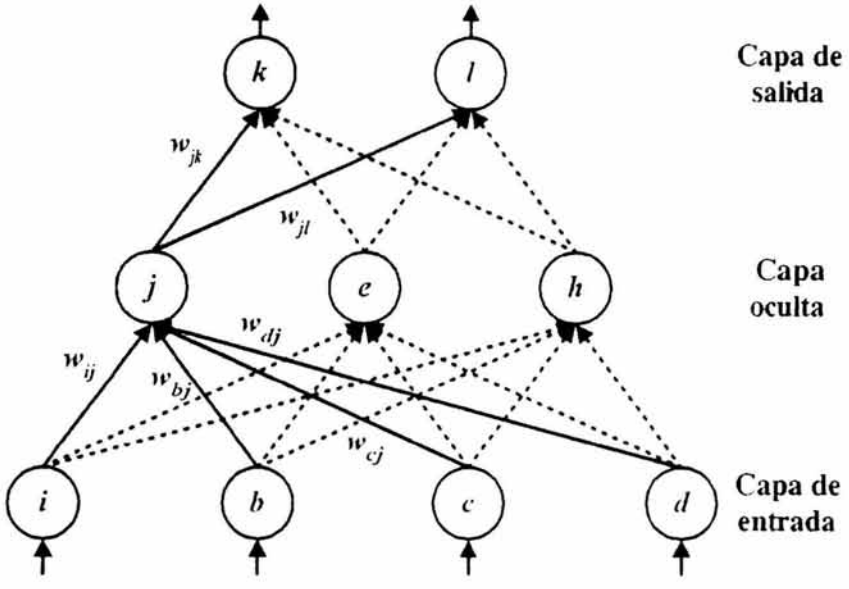


Figura 3.5: Una red neuronal artificial se compone de una capa de entrada, una o más capas ocultas y una capa de salida. Las neuronas de la capa de entrada reciben información del ambiente. Una neurona de una capa oculta recibe entradas de varias neuronas de la capa anterior. Por ejemplo, la neurona j recibe entradas de las neuronas i, b, c y d, teniendo un peso asociado para cada una de las conexiones: w_{ij}, w_{bj}, w_{cj}, y w_{dj}. La neurona j combina estas entradas y sus pesos respectivos, y pasa el resultado por una función de transferencia para obtener su salida. Esta salida es enviada a otras neuronas (k y l en este ejemplo). Finalmente, la capa de salida entrega el resultado del procesamiento de la red al ambiente.

se ha completado una **época** o **ciclo de entrenamiento**. El entrenamiento usualmente requiere de varias épocas antes de que la red pueda llevar a cabo su tarea de manera satisfactoria [45].

La meta es entrenar a un grupo de neuronas para que ejecute una tarea de forma rápida, con tolerancia a fallas, y que pueda generalizar a partir de las entradas vistas. Así pues, la red neuronal no sólo aprende a clasificar las entradas para las que fue entrenada, sino que también puede ser capaz de generalizar entradas que no ha visto, pero que son similares a las entradas vistas durante el entrenamiento. Esta **capacidad de generalización** es una de las principales propiedades de las redes neuronales artificiales [45], [82].

Otras características importantes incluyen su mayor robusteza y superioridad en la clasificación de patrones sobre otras herramientas computacionales gracias a su estructura distribuida [5]. La **robusteza** es lo que permite a la red responder aún cuando el sistema falla, cuando se le presentan entradas fuera de su dominio o ante patrones con ruido. Precisamente, la habilidad de tratar con patrones obscuros o ruidosos es una ventaja significativa de las redes neuronales artificiales sobre las soluciones algorítmicas tradicionales [22].

Además, las redes neuronales artificiales son capaces de resolver problemas para los que se cuenta con datos abundantes, pero en los cuales se carece de una teoría clara sobre la forma en que se debe convertir la entrada en la salida deseada. Por esta razón, usualmente el problema dado a una red neuronal artificial se describe a través de un conjunto de ejemplos [82]. Las redes neuronales artificiales se adaptan así para reproducir las salidas deseadas cuando se les presentan las entradas [22].

3.2 Redes Neuronales Convolucionales

En el modelo tradicional de reconocimiento de patrones, ilustrado en la figura 3.6, un **extractor de características** diseñado a mano recoge la información relevante de la entrada y elimina las variabilidades irrelevantes. Entonces, un **clasificador** entrenable ordena los vectores de características resultantes en clases [51]. En estos métodos se utilizan clasificadores basados en redes neuronales, cuyos datos de entrada se proyectan primero en un espacio de características a través de la extracción de rasgos, y después la red neuronal, que actúa como clasificador, se aplica al espacio de características [4].

El utilizar características reduce la cantidad de memoria necesaria para el

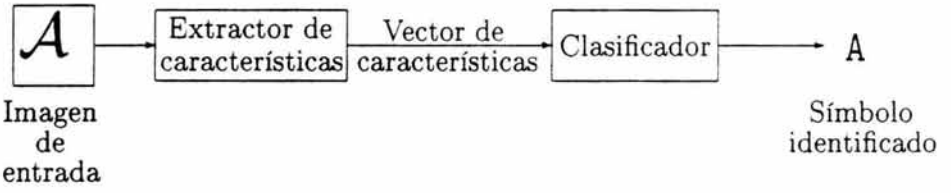


Figura 3.6: El método tradicional de reconocimiento de patrones requiere la extracción de vectores de características antes de poder clasificar a un carácter.

almacenamiento. Además, debido a que la cantidad de datos es menor que en la imagen, también es más rápido el procesamiento. No obstante, se requieren algoritmos que extraigan las características de los datos en bruto, lo cual es un problema difícil pues en muchas situaciones es complicado encontrar las características adecuadas para la clasificación. Además, la extracción de estas características no siempre es confiable [4].

Lo anterior se debe a que la extracción de características es una transformación fija derivada generalmente de conocimiento previo acerca de la tarea. Esto confía en la suposición, probablemente equivocada, de que el diseñador humano puede capturar toda la información relevante de la entrada [51].

Un esquema que puede producir mejores resultados que el usar las características consiste en trabajar con la imagen. En este caso, la red neuronal tiene que alimentarse directamente con imágenes normalizadas en vez de vectores de características. Dicho esquema confía en que el aprendizaje transforme las primeras capas de la red neuronal en un extractor de características apropiado [4], [50].

Así, para efectuar el reconocimiento óptico de caracteres una red puede recibir una imagen normalizada en tamaño y centrada en el campo de entrada. Bajo este esquema, la utilización de una red neuronal multicapa completamente conectada para reconocer la imagen normalizada, tal y como se hace en el modelo tradicional, puede traer serios problemas [51], [49]. El principal problema es la falta de robusteza ante las múltiples variaciones que sufren las imágenes.

Para empezar, en muchas aplicaciones reales los caracteres deben ser segmentados antes del reconocimiento. Los algoritmos de segmentación raramente son perfectos y frecuentemente dejan marcas extrañas en las imágenes de los caracteres (ruido, subrayados, caracteres vecinos), o algunas veces

producen caracteres incompletos al cortarlos demasiado. Estas imágenes no pueden ser normalizadas en tamaño y centradas de forma confiable [51].

La normalización crea además amplias variaciones de tamaño, inclinación y posición vertical de los caracteres individuales [51]. Esto, combinado con la variabilidad en el estilo de escritura causa diferencias en las posiciones de las características distintivas de los objetos de entrada [51], [49], [68].

Por ello, se hace necesario utilizar un reconocedor robusto a tales variaciones que pueda recibir una imagen como entrada. Este reconocedor robusto se logra implementar mediante el empleo de **redes neuronales convolucionales**. Estas redes son una arquitectura de red neuronal especializada que incorpora conocimiento acerca de las invarianzas de las figuras bidimensionales mediante el uso de patrones de conexión locales y la imposición de restricciones sobre los pesos [51].

Este tipo de redes presenta varias ventajas cuando se desea realizar el reconocimiento de imágenes. En primer lugar, las redes neuronales convolucionales están diseñadas específicamente para tratar con la variabilidad de las figuras bidimensionales. Las imágenes tienen una fuerte estructura local bidimensional: los pixeles espacialmente cercanos están altamente correlacionados. De ahí que resulte importante la extracción de características locales, la cual se logra restringiendo las conexiones de los campos receptivos de las unidades ocultas para que sean locales [51].

En segundo lugar, las redes neuronales convolucionales gozan de invarianza a traslaciones de forma automática. Esto se logra forzando la replicación de pesos a través del espacio [51], [49], [68].

Otra ventaja es que las redes neuronales convolucionales están diseñadas para aprender a extraer características relevantes directamente de imágenes de pixeles [51]. Además, las redes neuronales convolucionales generalizan mejor [68].

La idea de las redes neuronales convolucionales se inspiró en la biología [50]. Por esa razón, la topología local de replicación de pesos empleada por las redes neuronales convolucionales es similar a la topología basada en campos receptivos de las redes biológicas. Esta topología mejora la tolerancia a distorsiones locales. Además, gracias a que los pesos se comparten y a la restricción a conexiones locales, se reduce de manera eficiente la complejidad y el número de pesos. Esto es una ventaja cuando se presentan imágenes directamente, en vez de extraer imágenes de forma explícita y reducir los datos como usualmente se hace antes de la clasificación. Además, las redes con una topología local pueden migrarse más efectivamente a una computadora

paralela conectada localmente [68].

La primera realización de las redes neuronales convolucionales fue el neocognitron de Fukushima. En el neocognitron fueron usados por vez primera los campos receptivos. Esta arquitectura fue aplicada en el reconocimiento de dígitos escritos a mano [68].

Desde entonces, la arquitectura de redes neuronales convolucionales ha sido aplicada en la clasificación de imágenes para evitar el preprocesamiento y clasificar las imágenes en bruto directamente [68]. De manera particular, estas redes se pueden aplicar al reconocimiento de caracteres tanto impresos como escritos a mano [51].

Además, las redes neuronales convolucionales son particularmente buenas en el reconocimiento o rechazo de figuras con amplias variaciones de tamaño, posición y orientación [51], [68]. Según unas pruebas realizadas, el reconocimiento preciso ocurre para variaciones de escala hasta en un factor de dos, variaciones de traslación vertical de más o menos cerca de la mitad del alto del carácter y rotaciones de ± 30 grados [51]. De todas las transformaciones, las transformaciones de posición tienen un impacto más alto sobre el desempeño, mientras que la rotación y la escalación no afectan mucho [68]. Así, las redes neuronales convolucionales ofrecen una respuesta parcial al problema de invarianza o robusteza con respecto a distorsiones geométricas [51].

3.2.1 Arquitectura

La arquitectura de una red neuronal influye en su habilidad de generalización. Una buena generalización se puede lograr diseñando una arquitectura que contenga cierta cantidad de conocimiento a priori sobre el problema [50]. De esta forma, las redes neuronales convolucionales incorporan la extracción de características a través de su arquitectura y la reducción de pesos [4].

Las redes neuronales convolucionales combinan tres ideas arquitectónicas para asegurar cierto grado de invarianza a traslaciones, escalaciones y distorsiones [4], [51]:

- Campos receptivos locales.
- Pesos compartidos o replicación de pesos.
- Submuestreo espacial.

En el plano de entrada la red neuronal recibe imágenes de caracteres normalizadas en tamaño y centradas [51]. En las capas siguientes, la entrada a una neurona en una capa dada proviene de un vecindario local en la capa previa, ponderada por un conjunto de pesos. O lo que es lo mismo, la capa actual es la salida de la capa anterior convolucionada con un kernel; de ahí el nombre redes neuronales convolucionales [4].

De esta forma, cada neurona de una capa recibe entradas de un conjunto de neuronas localizadas en un pequeño vecindario de la capa previa tal y como se aprecia en la figura 3.7. En dicha figura cada cuadro representa una neurona y un campo receptivo se representa mediante un conjunto de cuadros en color gris. Estos **campos receptivos locales** permiten que las neuronas extraigan características visuales elementales tales como orillas orientadas, puntos extremos y esquinas. Estas características se combinan mediante capas subsiguientes para detectar características de alto orden [51].

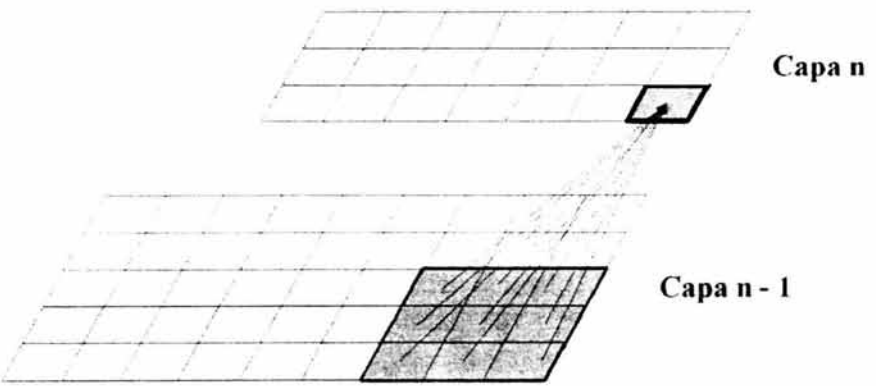


Figura 3.7: Las entradas a una neurona provienen de un campo receptivo local ubicado en la capa previa.

Ahora bien, las distorsiones o traslaciones de la entrada pueden causar que la posición de las características sobresalientes varíe. Por esta razón, se fuerza a que un conjunto de neuronas cuyos campos receptivos están localizados en diferentes lugares de la imagen tengan vectores idénticos de pesos, lo cual se puede ver gráficamente en la figura 3.8. A esto se le conoce como **replicación de pesos**. La replicación de pesos permite que los detectores de características elementales que son útiles en una parte de la imagen se apliquen a lo largo de toda la imagen [49], [51].

Las neuronas en una capa se organizan en planos dentro de los cuales

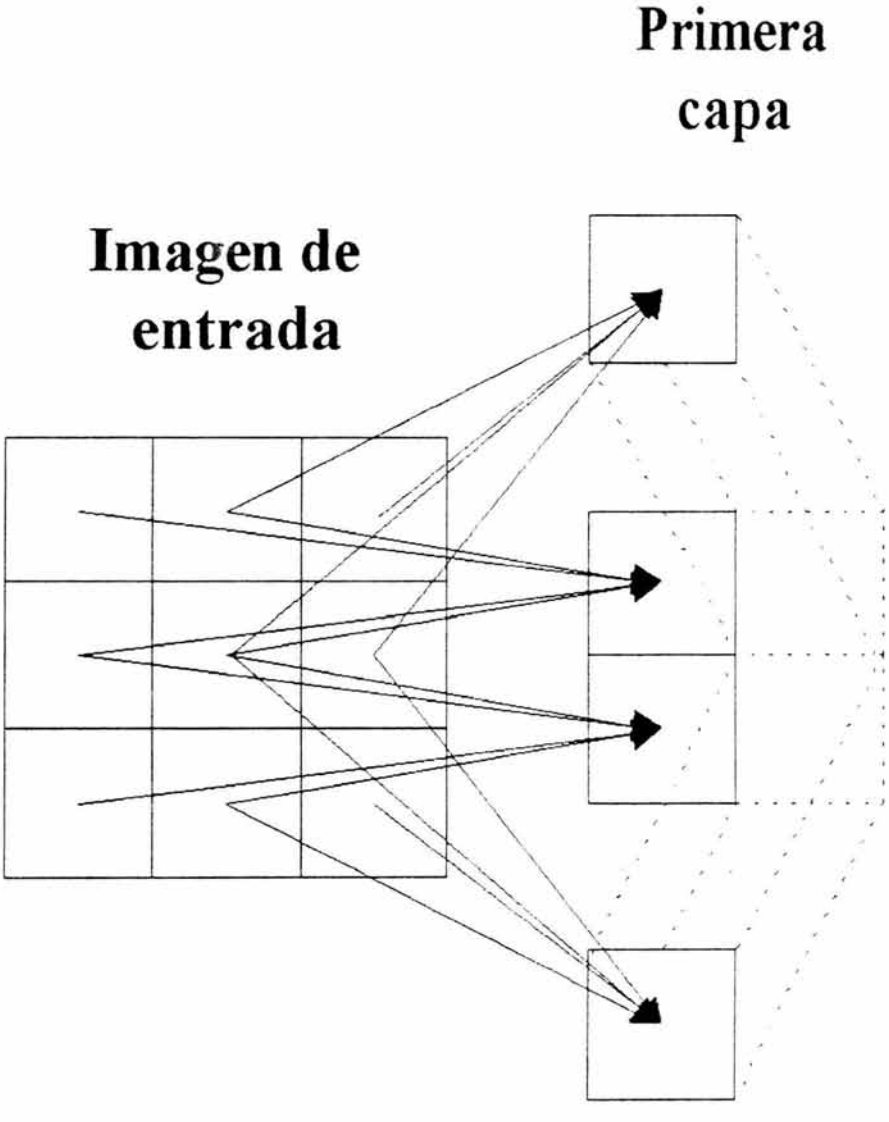


Figura 3.8: Cada neurona de la primera capa aplica el mismo vector de pesos, pero cada una sobre una parte distinta de la imagen.

todas las neuronas comparten el mismo conjunto de pesos (el kernel). El conjunto de las neuronas en tal plano es llamado **mapa de características** [4], [51]. Las neuronas en un mapa de características están restringidas a desempeñar todas la misma operación sobre diferentes partes de la imagen [49], [51]. Una capa convolucional completa como la ilustrada en la figura 3.9 se compone de varios mapas de características (con diferentes vectores de pesos) de forma que se puedan extraer múltiples características en cada ubicación [51].

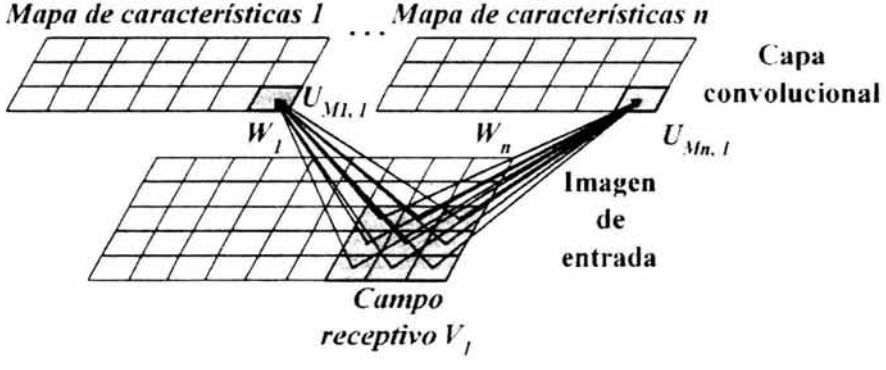


Figura 3.9: Una capa convolucional completa consta de n mapas de características. Todas las neuronas de un mismo mapa utilizan el mismo conjunto de pesos W, y cada una actúa sobre un vecindario distinto de la capa previa (campo receptivo). El mismo vecindario V, es tomado por neuronas homólogas de distintos mapas, sobre el cual aplican un conjunto de pesos distinto. En este ejemplo, el vecindario V_1 es procesado por la neurona 1 del mapa 1 ($u_{M_1,1}$) empleando el conjunto de pesos W_1; y por la neurona 1 del mapa n ($u_{M_n,1}$) empleando el conjunto de pesos W_n.

Siguiendo esta organización, una neurona en un mapa de características se conecta a una área de la capa anterior llamada campo receptivo de la neurona, teniendo así un conjunto de pesos entrenables más el sesgo entrenable. Los campos receptivos de neuronas contiguas de un mapa de características están centrados en neuronas contiguas de la capa previa, por lo tanto los campos receptivos de neuronas vecinas se traslapan [51].

Como todas las neuronas en un mapa de características comparten el mismo conjunto de pesos y el mismo sesgo, pueden detectar la misma característica en todas las posiciones posibles. Los otros mapas de características

de la capa usan otros conjuntos de pesos y sesgo con lo cual extraen diferentes tipos de características locales. Así, en cada ubicación de la entrada se extraen diferentes tipos de características mediante neuronas ubicadas en la misma posición, pero en diferente mapa [51].

Visto de forma secuencial, es como si un mapa de características recorriera la entrada con un sólo campo receptivo local y almacenara los estados de la neurona en una ubicación correspondiente en el mapa de características. Como esta operación es equivalente a una convolución seguida por la adición de un sesgo y la aplicación de una función de activación, a estas redes se les conoce como convolucionales [49], [51]. Aquí, el kernel de convolución es el conjunto de pesos de las conexiones usadas por la neurona del mapa de características. Dicho kernel actúa como detector de características [4].

Estas capas convolucionales tienen la propiedad de que si la imagen de entrada está trasladada, el mapa de características de salida también estará trasladado por la misma cantidad, pero tendrá los mismos valores de salida. Esta propiedad es la base de la robusteza de la red neuronal convolucional a las traslaciones y distorsiones de la entrada [49] [51].

Una vez que se detecta una característica, su posición exacta no es tan importante; sólo es relevante su posición relativa a otras características. No sólo es irrelevante la posición precisa de cada una de las características para identificar el patrón, sino que además puede ser dañina porque las posiciones probablemente varíen para diferentes instancias del carácter. Así, para reducir la precisión con la cual se codifican las posiciones de las características distintivas en el mapa de características, se reduce la resolución espacial del mapa mediante **capas de submuestreo**. Estas capas llevan a cabo un promedio local y un submuestreo, reduciendo la resolución del mapa de características y la sensibilidad de la salida a traslaciones y distorsiones.

De esta forma, las capas de submuestreo tienen un mapa de características para cada mapa de la capa previa. Las neuronas de esta capa operan de forma ligeramente diferente a las neuronas de las capas convolucionales. Así, una neurona de este tipo calcula el promedio de las entradas de su campo receptivo y lo multiplica por un coeficiente entrenable. A este resultado le agrega un sesgo entrenable de la misma forma que las neuronas de las capas convolucionales. Por lo tanto, el estado de activación a_j de una neurona de este tipo queda determinado por la siguiente ecuación:

$$sum = \sum_{i=1}^{n} x_i \qquad a_j = \frac{sum}{n} w_{1j} + w_{0j} \tag{3.3}$$

Donde x_i es la i-ésima entrada a la neurona j; n es el número de entradas a la neurona j; w_{1j} es el coeficiente entrenable y w_{0j} es el sesgo entrenable.

El estado de activación de la neurona a_j, obtenido como resultado, es pasado a través de la misma función de activación g para obtener la salida y_j. Las neuronas contiguas no tienen campos contiguos superpuestos como se puede observar en la figura 3.10 [51].

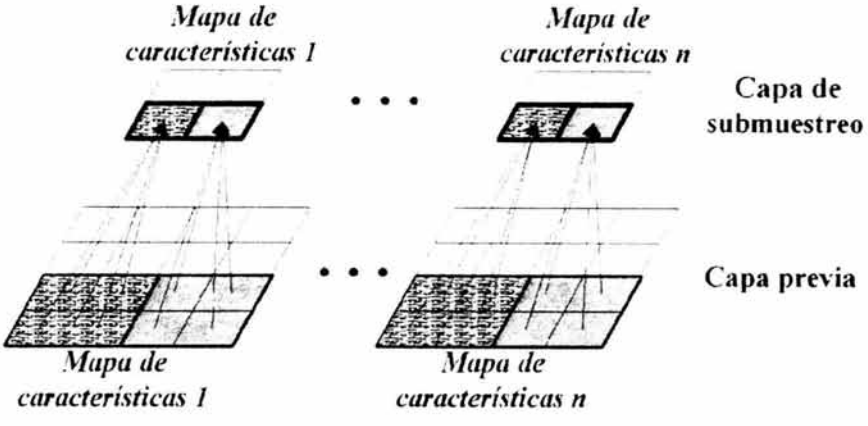


Figura 3.10: Una capa de submuestreo tiene un mapa de características por cada mapa de características de la capa previa.

El coeficiente entrenable y el sesgo controlan el efecto de la no linealidad de la función sigmoide. Si el coeficiente es pequeño las neuronas operan de forma casi lineal, y la capa de submuestreo desdibuja la entrada. Si el coeficiente es grande, las neuronas de la capa de submuestreo llevan a cabo un función OR con ruido o AND con ruido dependiendo del valor del sesgo [51].

En toda la red se alternan capas convolucionales y de submuestreo, de forma que en cada capa, el número de mapas de características se incrementa conforme la resolución espacial se reduce. El grado de invarianza a transformaciones geométricas es grande gracias a la reducción progresiva de la resolución espacial, compensada por el incremento progresivo de la riqueza de la representación (número de mapas de características) [51].

Finalmente, en la última capa, los mapas de características se clasifican mediante una capa totalmente conectada [4]. Así, se obtiene la arquitectura general mostrada en la figura 3.11. La descripción detallada de la red neuronal utilizada en esta tesis se presenta en el siguiente capítulo.

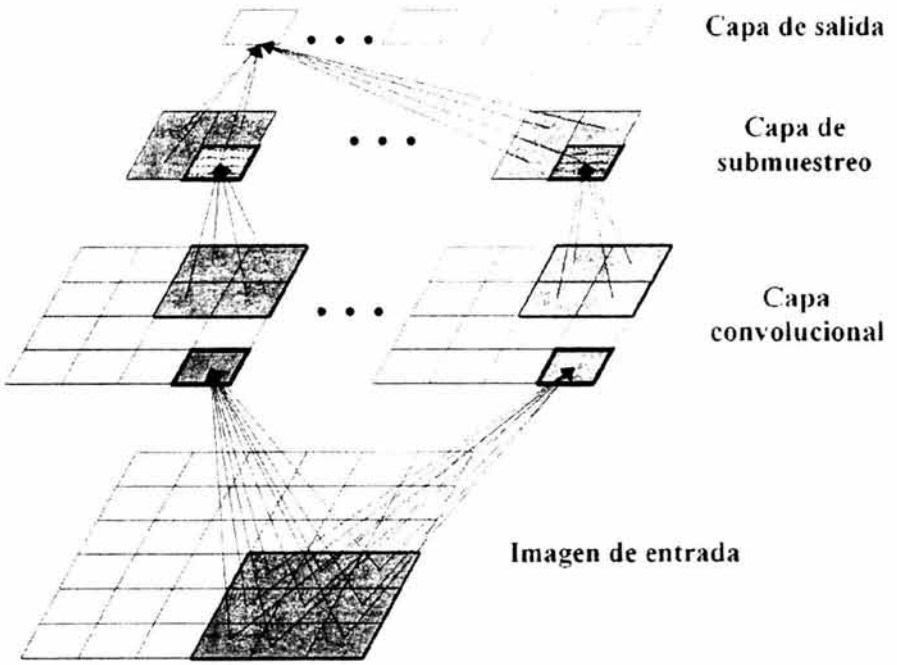


Figura 3.11: Arquitectura general de una red neuronal convolucional.

A través del aprendizaje de los datos de entrenamiento, la red puede hallar el conjunto adecuado de pesos para la extracción de características [4]. Como todos los pesos se aprenden con retropropagación se puede decir que las redes neuronales convolucionales sintetizan su propio extractor de características [51].

Como una ventaja extra del compartir pesos se reduce el número de parámetros libres. Con ello, se disminuye la capacidad de la red neuronal y se reduce el hueco entre el error de prueba y el error de entrenamiento, mejorándose así, la habilidad de generalización de la red [49], [51], [92].

3.2.2 Función de Transferencia

La **función de activación** o **función de transferencia** transforma la suma ponderada de todas las señales de entrada a la neurona para determinar su intensidad de disparo [5]. De entre los diversos tipos de funciones que se pueden emplear para implementar dicha función, las **funciones de activación no lineales** resultan convenientes pues le dan a las redes neuronales su capacidad de no linealidad [52].

De manera particular, en las redes neuronales convolucionales se utilizan las **funciones sigmoides**. Estas funciones poseen la propiedad de continuidad y diferenciabilidad en $(-\infty, \infty)$ lo cual resulta esencial en el aprendizaje [5], [91].

La función de activación sigmoide es una función que se incrementa monotónicamente con asíntotas en algún valor finito conforme se acerca a $\pm\infty$. De éstas forma parte la **tangente hiperbólica** $f(x) = \tanh(x)$, la cual posee la característica de ser simétrica con respecto al origen, lo cual se puede observar en la figura 3.12. Las sigmoides que son simétricas con respecto al origen son preferidas porque es más probable que produzcan salidas que en promedio tienden a cero y, por lo tanto, convergen más rápido que la función logística estándar [52].

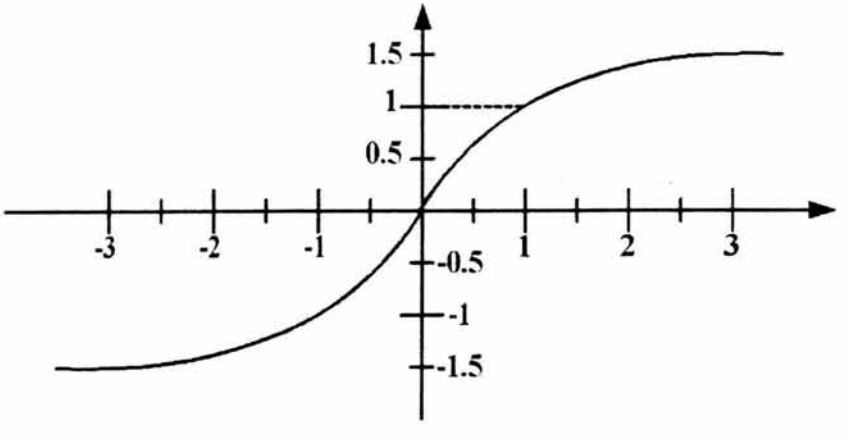


Figura 3.12: La función de tangente hiperbólica $f(x) = 1.7159 \tanh(\frac{2}{3}x)$.

Siguiendo la recomendación dada en [52], las redes neuronales convolucio-

nales implementadas en esta tesis emplean la función $f(x) = 1.7159 \tanh(\frac{2}{3}x)$. Las constantes recomendadas para la sigmoide hacen que la varianza de las salidas se encuentre cercana a 1 cuando se usan con entradas transformadas porque la ganancia efectiva de las sigmoides es aproximadamente 1 sobre su rango útil. Las propiedades particulares de esta sigmoide son [52]:

1. $f(\pm 1) = \pm 1$.
2. La segunda derivada es un máximo en $x = 1$.
3. La ganancia efectiva es cercana a 1.

3.2.3 Base de Datos de Entrenamiento

Las redes neuronales artificiales son capaces de tratar con casos no vistos gracias a la generalización. Para lograr que una red neuronal artificial sea capaz de dicha generalización, se requiere que la cantidad de datos de entrenamiento sea suficientemente grande con el objeto de cubrir todas las posibles variaciones. De otra forma, si el tamaño de la base de datos es insuficiente, la generalización de la red para datos fuera del dominio de entrenamiento se verá afectada de forma adversa [5].

Así, con el propósito de entrenar y evaluar una red neuronal artificial se requiere recolectar un conjunto de muestras distintas que deberá ser dividido en tres partes [5]:

1. **Conjunto de entrenamiento**. Incluye todos los datos pertenecientes al dominio del problema y es usado en la fase de entrenamiento para actualizar los pesos de la red.
2. **Conjunto de validación**. Se usa durante el proceso de entrenamiento para verificar la respuesta de la red ante datos no usados en la actualización de los pesos. Los datos de este conjunto deben ser distintos del conjunto de entrenamiento, pero dentro de los límites de los datos de entrenamiento. Con base en el desempeño medido sobre este conjunto se decide si es necesario aplicar más ciclos de entrenamiento.
3. **Conjunto de prueba**. Una vez que se ha seleccionado la mejor red y antes de implementarla, se usa este conjunto para confirmar su desempeño ante ejemplos no usados en el entrenamiento.

El tamaño que cada uno de estos conjuntos debe tener se selecciona con base en guías generales obtenidas de la experiencia, pues no existen reglas formales que indiquen el tamaño adecuado de cada uno de estos conjuntos. Así, en [51] se sugiere que el tamaño mínimo del conjunto de entrenamiento sea igual al número de parámetros entrenables de la red. Otras tasas sugeridas para el conjunto de entrenamiento son: $\frac{Numero\ de\ ejemplos}{Numero\ de\ pesos} > 4$; $\frac{Numero\ de\ ejemplos}{Numero\ de\ pesos} > 10$ [5].

Por lo que respecta al **particionamiento**, algunos investigadores recomiendan que el 65% de la base de datos sea para el entrenamiento, el 25% para la validación y el 10% para las pruebas [5]. Otros recomiendan que el conjunto de entrenamiento contenga el 50% de las muestras y los conjuntos de validación y pruebas ocupen un 25% cada uno [77]. En general, el conjunto de entrenamiento constituye como mínimo el 50% de la base de datos, y el resto se divide equitativamente entre los conjuntos de validación y pruebas o se divide de forma que el conjunto de validación sea mayor al conjunto de pruebas.

Otro aspecto importante a considerar es el **balanceo de los datos**. Los datos de entrenamiento deben estar distribuidos equitativamente entre las clases para evitar que la red favorezca a las clases sobrerepresentadas [5].

Por último, los datos deben procesarse antes de su uso en el entrenamiento. En primer lugar, como el plano de entrada para una red neuronal convolucional es fijo, se requiere un proceso de normalización de tamaño antes de pasarle imágenes más grandes o más pequeñas [4]. Este proceso se lleva a cabo de la forma descrita en la sección 2.3.1 del capítulo previo. Es importante que esta normalización deforme lo menos posible las imágenes originales y por ello se debe procurar que se preserven las proporciones de la imagen.

Las imágenes normalizadas se centran en un campo receptivo de acuerdo con su centro de masa. El cálculo de este centro de masa y la forma en que se lleva a cabo el centrado de la imagen fueron revisados en el apartado 2.3.2 del capítulo anterior.

Finalmente, los datos deben escalarse dentro de un rango pequeño y uniforme, especialmente cuando los datos de entrada toman grandes valores [5]. Para ello, se emplea la fórmula 2.16 dada en la sección 2.3.3 del capítulo previo. Se recomienda normalizar entre valores ligeramente diferentes de 0 y 1 para evitar la saturación de la función sigmoide, lo cual podría evitar el aprendizaje o hacerlo lento.

3.2.4 Algoritmo de Aprendizaje

El algoritmo empleado para entrenar a las redes neuronales convolucionales está basado en el algoritmo de propagación hacia atrás. Este algoritmo aplica la **regla delta generalizada**, en la cual el error se expresa como una función de los pesos de la red. Lo que se busca es minimizar la media del error cuadrático mediante el uso del método de descenso del gradiente [91].

Estos métodos de gradiente sufren de convergencia lenta conforme el sistema se aproxima a un mínimo local [91]. No obstante, el aprendizaje basado en gradiente aplicado a las redes neuronales convolucionales les permite aprender las características apropiadas de los ejemplos [51].

En el algoritmo empleado se utiliza un **entrenamiento supervisado**. La idea básica es presentar el patrón de entrada (la imagen de un carácter) a la red y propagarlo para calcular la salida de cada capa y la salida final de la red. Para la capa de salida se conocen los valores deseados, es decir, se sabe de antemano qué carácter le corresponde a la imagen presentada y, por lo tanto, los pesos pueden ajustarse de acuerdo con la regla de descenso de gradiente. Para calcular los cambios de los pesos en las capas ocultas, el error en la capa de salida es propagado hacia atrás a estas capas de acuerdo con los pesos de las conexiones. Este proceso se repite para cada muestra del conjunto de entrenamiento hasta completar un ciclo. Un ciclo a través de todo el conjunto de entrenamiento se conoce como **época**. El número de épocas requeridas para realizar el entrenamiento depende en gran medida del error calculado en la capa de salida [82].

Así, teniendo un conjunto de entrenamiento denotado por $T = \{(Z_1, D_1), \ldots, (Z_P, D_P)\}$, donde Z_p es el p-ésimo patrón de entrada y D_p es la salida deseada para Z_p, el algoritmo de entrenamiento es el siguiente [51], [82]:

Algoritmo 3.1. Entrenamiento
Variables de Entrada:

Red Una red neuronal convolucional compuesta por un conjunto de parámetros entrenables W

T Un conjunto de entrenamiento $T = \{(Z_1, D_1), \ldots, (Z_P, D_P)\}$ compuesto por P pares de patrones de entrada Z_p y su correspondiente salida deseada D_p

Variables de Salida:

Red Una red neuronal convolucional cuyos parámetros entrenables W han sido ajustados

Variables Locales:

μ Parámetro de regulación

η Tasa de aprendizaje global

ε_k Tamaño del paso de actualización para el parámetro w_k

w_k k-ésimo parámetro entrenable del conjunto de parámetros entrenables W

Y_p Salida final de la red neuronal convolucional para el patrón p

1. Inicializar W de forma aleatoria
2. Seleccionar μ
3. Mientras la condición de paro sea falsa hacer
   (a) Asignar η de acuerdo al número de iteración actual
   (b) Estimar ε_k sobre m muestras de T
   (c) Mientras haya pares (Z_p, D_p) no usados en T hacer
      i. Tomar uno de los pares no usados (Z_p, D_p) de T de forma aleatoria
      ii. Propagar Z_p a través de *Red* para calcular la salida de cada capa y Y_p
      iii. Calcular el valor de la función de error para el patrón p:

$$E_p = F(D_p, Y_p) \tag{3.4}$$

      iv. Calcular los cambios de los pesos y actualizar los pesos:

$$w_k = w_k - \varepsilon_k \frac{\partial E_p}{\partial w_k} \tag{3.5}$$

4. Regresar *Red*

En las siguientes secciones se ofrece una explicación más profunda de los principales pasos de este algoritmo de aprendizaje. Dicho algoritmo se basa en el **método de diagonal estocástica de Levenberg-Marquardt** [51].

Este algoritmo converge de forma confiable sin requerir ajustes considerables de los parámetros de aprendizaje. Además, corrige los principales problemas de mal condicionamiento de la función de error debidos a peculiaridades de la arquitectura de la red y de los datos de entrenamiento. Así, el algoritmo resulta particularmente útil para las redes de pesos compartidos, porque el compartir pesos crea mal condicionamiento en la superficie de error. Este mal condicionamiento se debe a que un sólo parámetro en las primeras capas puede tener una enorme influencia en la salida. Consecuentemente, la segunda derivada del error con respecto a este parámetro puede ser muy grande, mientras que puede ser muy pequeña para otros parámetros en otra parte de la red [51].

Inicialización de Pesos (Paso 1)

El primer paso en el entrenamiento de la red neuronal convolucional consiste en asignar valores iniciales para los pesos y umbrales de todas las conexiones [5].

La selección inicial de pesos determina el punto de inicio en la superficie de error y, por ello, afecta el resultado del proceso de aprendizaje, controlando si el proceso se termina en el mínimo global o en un mínimo local [82]. Según Hassoun si el vector de pesos inicial se encuentra dentro de la cuenca de atracción de un mínimo local con gran fuerza de atracción, entonces la convergencia del entrenamiento será rápida y la calidad de la solución estará determinada por la profundidad del valle del mínimo local con relación al mínimo global. Pero si la búsqueda se inicia con un vector de pesos ubicado en una región relativamente plana de la superficie de error, la convergencia será muy lenta [31]. Sin embargo, según otros investigadores, la inicialización afecta muy poco la convergencia y la arquitectura final de la red neuronal [5].

Comúnmente, los pesos se inician con valores aleatorios pequeños en un rango con media cero. Lo que se busca es que la función sigmoide sea activada primeramente en su región lineal. Con esto se obtienen dos ventajas. En primer lugar, los gradientes son lo suficientemente grandes para proceder con el aprendizaje. En segundo lugar, la red aprende primero la parte más fácil del mapeo, es decir la parte lineal, y posteriormente la parte no lineal

[52].

La selección de números pequeños es muy importante para evitar la saturación de las neuronas pues de otra forma los cambios de los pesos serían casi nulos. Sin embargo, números muy pequeños pueden llevar a valores gradientes demasiado pequeños, con lo cual el aprendizaje también sería muy lento, pudiendo incluso no convergir [5], [51], [82].

Con el propósito de evitar estos problemas, la inicialización se lleva a cabo neurona por neurona asignando valores aleatorios de una distribución uniforme entre $-r/F_i$ y r/F_i; donde r es un número real que depende de la función de activación de la neurona y F_i es el número de entradas a la unidad i, a la cual pertenecen las conexiones que se desean inicializar [5], [51]. La división entre la cantidad de entradas provoca que la desviación estándar inicial de las sumas ponderadas se encuentre en el mismo rango para cada unidad y que caiga dentro de la región de operación normal de la función sigmoide [51].

Parámetro de Regulación μ (Paso 2)

En el algoritmo de entrenamiento descrito, el tamaño del paso de actualización depende de varios parámetros. Uno ellos es el parámetro de regulación μ. Dicho parámetro evita que el paso de actualización crezca demasiado cuando la estimación de la segunda derivada de la función de error h_{kk} resulta muy pequeña (ver ecuación 3.7) [51].

El parámetro μ es un valor constante que se establece al principio del algoritmo. El valor asignado a dicho parámetro se encuentran en un rango entre 0 y 1:

$$0 < \mu < 1 \tag{3.6}$$

Criterio de Convergencia (Paso 3)

El criterio de convergencia empleado para detener el proceso de aprendizaje es el de **validación cruzada**, puesto que es uno de los criterios de convergencia más confiables aunque requiere de abundantes datos y es más demandante en términos computacionales. Para usar este criterio se emplea el conjunto de validación recolectado. Mediante dicho conjunto se puede vigilar el proceso de entrenamiento, deteniendo el algoritmo en cuanto el error de validación deja de disminuir y empieza a crecer. En este punto el entrenamiento de

la red es óptimo, ya que cuando se prosigue con el entrenamiento, la red memoriza los detalles del conjunto y pierde capacidad de generalización [5].

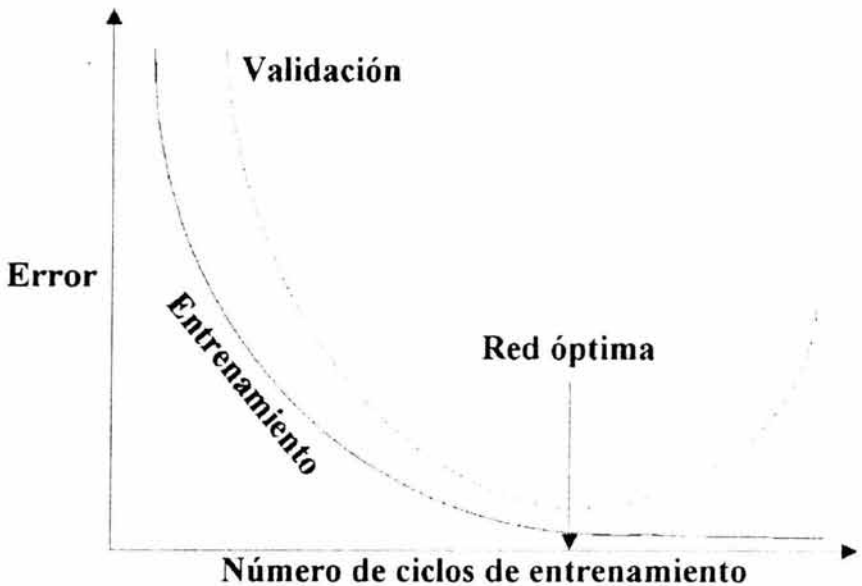


Figura 3.13: Criterio de validación cruzada. El entrenamiento se detiene en cuanto el error de validación deja de disminuir e inicia su crecimiento.

Cuando la red pierde capacidad de generalización, el error de entrenamiento continúa disminuyendo, pero el error en el conjunto de validación comienza a crecer, tal y como se muestra en la figura 3.13. Entrenar demasiado da por resultado una red que sólo sirve como tabla de búsquedas, fenómeno conocido como **sobreentrenamiento** o **memorización**. El entrenamiento excesivo puede llevar a un error de casi 0 en la predicción de los datos de entrenamiento (**recuerdo**); sin embargo, la generalización en los datos de validación se degrada significativamente [5].

Tasa de Aprendizaje (Paso 3a)

La **tasa de aprendizaje** o **coeficiente de aprendizaje** η determina el tamaño del paso de actualización, es decir, el tamaño de los cambios en los

pesos. Este parámetro permite acelerar el entrenamiento haciendo que los cambios en las actualizaciones de los pesos en cada ciclo sean más significativos que sin su uso. Cuando esta tasa es muy pequeña o no es empleada, el entrenamiento converge lentamente hacia la solución. Sin embargo, el uso de una tasa muy grande puede provocar la oscilación de la búsqueda en la superficie de error y evitar la convergencia, tal y como se ilustra en la figura 3.14 [5], [82].

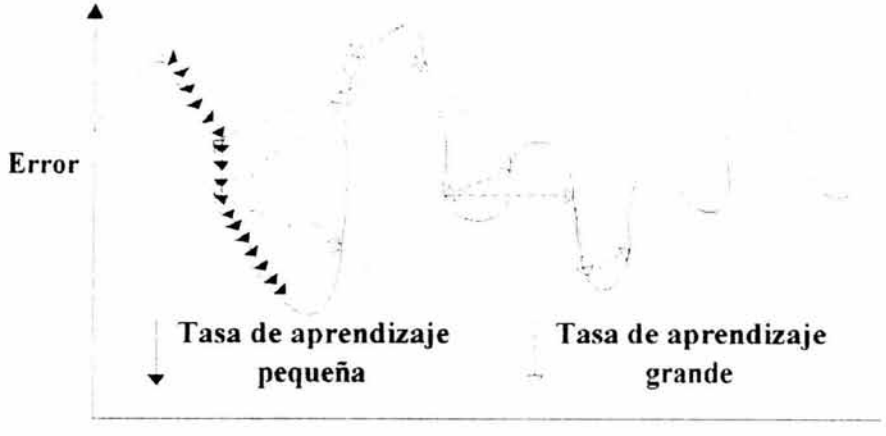


Figura 3.14: La tasa global de aprendizaje influye sobre los cambios de los pesos. Una tasa global demasiado pequeña provoca que el aprendizaje sea lento. Una tasa global demasiado grande produce oscilaciones y evita la convergencia.

Por lo general, se emplea una **tasa de aprendizaje adaptable** que varía a lo largo del entrenamiento, lo cual permite obtener buenas soluciones. Bajo este esquema, al principio se emplea un coeficiente relativamente grande para escapar de los mínimos locales, el cual se reduce conforme el entrenamiento avanza, pues cerca del mínimo el paso debe ser pequeño para no perderlo [5], [82].

Siguiendo estas recomendaciones, en el algoritmo de entrenamiento se emplea una tasa de aprendizaje global adaptable. Los valores de esta tasa se asignan dependiendo del ciclo de aprendizaje actual.

Tamaño del Paso de Actualización (Paso 3b)

En el algoritmo de entrenamiento se calcula una **tasa individual de aprendizaje** ε_k (tamaño de paso) para cada parámetro antes de cada iteración a través del conjunto de entrenamiento. Esta tasa de aprendizaje se calcula usando los términos diagonales de una estimación de la aproximación Gauss-Newton a la matriz de Hessian (segundas derivadas).

Así, los tamaños de paso ε_k son una función de la segunda derivada de la función de error a lo largo del eje w_k:

$$\varepsilon_k = \frac{\eta}{\mu + h_{kk}} \tag{3.7}$$

Donde η corresponde a la tasa global de aprendizaje, μ es el parámetro de regulación y h_{kk} es una estimación de la segunda derivada de la función de error E con respecto a w_k. Entre más grande es h_{kk}, más pequeña es la actualización del peso. La fórmula exacta para calcular h_{kk} a partir de las segundas derivadas con respecto a los pesos de la conexión es:

$$h_{kk} = \sum_{(i,j)\in V_k} \sum_{(k,l)\in V_k} \frac{\partial^2 E}{\partial u_{ij} \partial u_{kl}} \tag{3.8}$$

Donde u_{ij} es el peso de la conexión que va de la unidad j a la unidad i; y V_k es el conjunto de pares de índices tales que la conexión entre i y j comparte el parámetro w_k, esto es:

$$u_{ij} = w_k \forall (i,j) \in V_k \tag{3.9}$$

Sin embargo, h_{kk} no se calcula a partir de dicha fórmula, sino que se hacen tres aproximaciones:

1. Se eliminan los términos que se hallan fuera de la diagonal de la matriz de Hessian con respecto a los pesos de la conexión:

$$h_{kk} = \sum_{(i,j)\in V_k} \frac{\partial^2 E}{\partial u_{ij}^2} \tag{3.10}$$

   Donde los términos $\frac{\partial^2 E}{\partial u_{ij}^2}$ son el promedio sobre el conjunto de entrenamiento de las segundas derivadas locales:

$$\frac{\partial^2 E}{\partial u_{ij}^2} = \frac{1}{P} \sum_{p=1}^{P} \frac{\partial^2 E_p}{\partial u_{ij}^2} \tag{3.11}$$

Estas segundas derivadas locales con respecto a los pesos de conexión se calculan a partir de las segundas derivadas locales con respecto a la entrada total de la unidad:

$$\frac{\partial^2 E_p}{\partial u_{ij}^2} = \frac{\partial^2 E_p}{\partial a_i^2} x_j^2 \tag{3.12}$$

Donde x_j es el estado de la unidad j y $\frac{\partial^2 E_p}{\partial a_i^2}$ es la segunda derivada de la función de error con respecto a la entrada total a la unidad i (denotada a_i). Para calcular estas segundas derivadas se puede emplear la siguiente fórmula:

$$\frac{\partial^2 E_p}{\partial a_i^2} = f'(a_i)^2 \sum_k u_{ki}^2 \frac{\partial^2 E_p}{\partial a_k^2} + f''(a_i) \frac{\partial E_p}{\partial x_i} \tag{3.13}$$

2. Debido a que los términos de la fórmula anterior pueden ser negativos y pueden causar que el algoritmo de gradiente se mueva hacia arriba en vez de hacia abajo, se utiliza una aproximación Gauss-Newton que garantiza que las estimaciones de las segundas derivadas son no negativas. Esta aproximación ignora la no linealidad de la red neuronal, pero no la de la función de error. La ecuación de propagación hacia atrás para las aproximaciones Gauss-Newton de las segundas derivadas es:

$$\frac{\partial^2 E_p}{\partial a_i^2} = f'(a_i)^2 \sum_k u_{ki}^2 \frac{\partial^2 E_p}{\partial a_k^2} \tag{3.14}$$

Como se trata de una suma de productos no negativos, el lado izquierdo es no negativo.

3. El promedio de la ecuación 3.11 no es calculado sobre el conjunto de entrenamiento completo, sino sobre un pequeño subconjunto de este. Además, la estimación no se ejecuta frecuentemente ya que las propiedades de segundo orden de la superficie de error cambian con bastante

lentitud. Por lo tanto, el valor de h_{kk} se estima sobre m patrones antes de cada corrida de entrenamiento. Estas estimaciones no son sensibles al subconjunto particular empleado en el promedio. Esto se debe a que las propiedades de segundo orden de la superficie de error están determinadas principalmente por la estructura de la red, y no por los detalles estadísticos de las muestras.

Función de Error (Paso 3(c)iii)

Para poder trabajar con el algoritmo de aprendizaje presentado es necesario definir la función de error que habrá de ser minimizada durante el proceso de entrenamiento. Así, en el presente estudio se emplea un criterio de entrenamiento discriminativo llamado **criterio del máximo a posteriori**, gracias al cual se introduce competencia entre las clases. El criterio seleccionado maximiza la probabilidad posterior de la clase correcta D_p (o minimiza el logaritmo de la probabilidad de la clase correcta), tomando en cuenta que la imagen puede pertenecer a una de las clases válidas o a la clase de rechazo. En términos de penalización, lo anterior significa que, además de reducir la penalización para la clase correcta, se incrementa la penalización de las clases incorrectas. De esta forma, la función de error está dada por la siguiente ecuación [51]:

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}(Y_{D_p}(Z_p, W) + \log(e^{-j} + \sum_i e^{-Y_i(Z_p,W)})) \tag{3.15}$$

Donde Y_{D_p} es la salida que corresponde a la clase correcta para el patrón de entrada Z_p. En esta ecuación el segundo término es negativo, gracias a lo cual juega un papel competitivo; no obstante, la función de error siempre es positiva pues dicho término es más pequeño o igual al primer término. La constante j es positiva y evita que las penalizaciones de las clases que ya son muy grandes se incrementen aún más. Finalmente, la probabilidad posterior de la clase de rechazo está dada por la proporción entre e^{-j} y $e^{-j} + \sum_i e^{-Y_i(Z_p,W)}$.

Actualización de los Pesos (Paso 3(c)iv)

Los pesos en el algoritmo de entrenamiento se actualizan a través de un **método estocástico**. Así, los pesos se actualizan después de la presentación

de un sólo patrón. Con dicho método las estimaciones del gradiente son aproximadas, pero los parámetros se actualizan con mayor frecuencia que con la **actualización por lotes**, en la cual se suelen presentar todos los ejemplos y calcular el gradiente de forma exacta sobre todo el conjunto de entrenamiento [5], [51].

El motivo principal para seleccionar este método de actualización radica en que la actualización estocástica es más rápida que la actualización por lotes en tareas que involucran conjuntos de datos grandes y con cierto grado de redundancia. Para entender por qué, asumamos que se tiene un conjunto de entrenamiento compuesto de dos copias del mismo subconjunto. Bajo tales condiciones, la acumulación del gradiente sobre todo el conjunto causaría cálculos redundantes. Sin embargo, una corrida sobre todo el conjunto de entrenamiento empleando el método estocástico sería igual a dos iteraciones de aprendizaje completas sobre el pequeño subconjunto. Esta idea se puede generalizar para conjuntos de entrenamiento en donde no existe repetición exacta, pero donde hay cierta redundancia [51], [52].

Otras ventajas de la actualización estocástica sobre la actualización por lotes incluyen, un menor requerimiento de almacenamiento para los pesos y la posibilidad de evitar el atascamiento en un mínimo local gracias a la búsqueda estocástica [5].

La consecución de mejores soluciones se debe al ruido de las actualizaciones. En las redes no lineales usualmente se tienen múltiples mínimos locales de diferentes profundidades. En el aprendizaje por lotes se descubre el mínimo de la cuenca en la cual se colocan los pesos inicialmente. Por su parte, en el aprendizaje estocástico el ruido en las actualizaciones puede provocar que los pesos salten de una cuenca a otra con un mínimo local posiblemente más profundo [52].

En cada iteración de aprendizaje se actualiza un parámetro particular w_k de acuerdo con la regla de actualización dada por la ecuación 3.5 del paso 3(c)iv del algoritmo de entrenamiento. En dicha fórmula, debido a que los pesos se comparten en la red neuronal convolucional, la derivada parcial $\frac{\partial E_p}{\partial w_k}$ está dada por la suma de las derivadas parciales con respecto a las conexiones que comparten el parámetro w_k:

$$\frac{\partial E_p}{\partial w_k} = \sum_{(i,j)\in V_k} \frac{\partial E_p}{\partial u_{ij}} \tag{3.16}$$

Donde u_{ij} es el peso de la conexión que va de la unidad j a la unidad

i; y V_k es el conjunto de pares de índices tales que la conexión entre i y j comparte el parámetro w_k (ver ecuación 3.9).

3.3 Redes Neuronales Artificiales y Modularidad

La estructura convencional de los sistemas de reconocimiento óptico de caracteres, ilustrada en la figura 3.15, consta de tres módulos principales que llevan a cabo las siguientes tareas [51], [70]:

1. **Extracción de características**. Transforma la imagen del patrón de entrada para que pueda ser representado mediante un vector de características X de baja dimensión d o mediante cadenas cortas de símbolos. Un vector de características se compone de un conjunto de x características, de forma que $X = (x_0, x_1, \ldots, x_{d-1})$. Estos vectores de características pueden ser comparados fácilmente y son relativamente invariables con respecto a las transformaciones y distorsiones que no cambian la naturaleza de los patrones de entrada. Este módulo contiene la mayoría del conocimiento previo y es específico para la tarea. También es el foco de la mayoría del esfuerzo de diseño porque está totalmente hecho a mano.

2. **Clasificación**. Teniendo n clases denotadas cada una por ω_i para $0 \leq i < n$, y usando la información del vector X, el módulo de clasificación produce un vector de decisión D que representa las probabilidades de membresía de las n clases. Así, se obtiene un clasificador n de propósito general que contiene un conjunto de parámetros libres o pesos entrenables.

3. **Decisión de clases**. Determina una etiqueta única de clase para el patrón de entrada usando el vector de decisión D.

En la arquitectura convencional, una red neuronal artificial implementa el módulo de clasificación n, es decir, todas las n clases comparten una sola red neuronal artificial grande. La tarea de diseño importante aquí es escoger características con buen poder discriminativo y dejar que la red neuronal divida las regiones de las n clases en el espacio de características elegido. Por supuesto, determinar los límites óptimos de decisión para un módulo

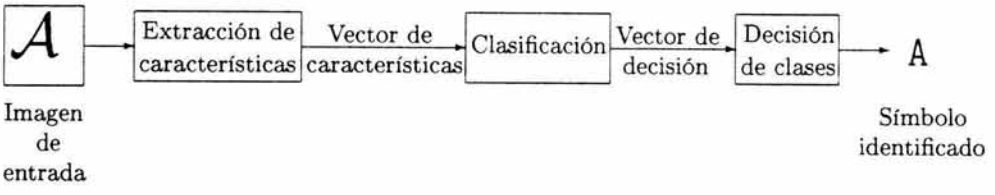


Figura 3.15: La estructura convencional de los sistemas de reconocimiento óptico de caracteres consta de tres partes: extracción de características, clasificación y decisión de clases.

de clasificación n es muy complejo y puede limitar el reconocimiento y los procesos de entrenamiento [70].

La mayoría de las redes neuronales artificiales tiene una estructura monolítica; es decir, se trata de redes totalmente conectadas que trabajan bien en un espacio de entrada muy pequeño. Sin embargo, su complejidad se incrementa y su desempeño decrece a medida que crece la dimensión de la entrada, pues se reduce su capacidad de generalización [82].

Además, las redes neuronales artificiales tradicionales sufren problemas de convergencia y gastan más tiempo en el aprendizaje de espacios de entrada de grandes dimensiones [82]. Los problemas de convergencia ocurren cuando el conjunto de entrenamiento no es lo suficientemente grande comparado con el tamaño del clasificador es decir, el número de parámetros libres [70].

Un factor que influye en el tamaño del clasificador es precisamente el número de caracteres. En los grandes conjuntos de caracteres el tamaño del clasificador es grande y por ello se requiere un conjunto de entrenamiento grande para convergir (Oh y Suen, 2002).

Más aún, la arquitectura convencional es una arquitectura rígida compuesta de una caja negra no estructurada con todas las clases mezcladas. Así, los módulos no pueden ser modificados u optimizados para cada clase [70]. Uno de los principales problemas de este método es que la precisión del reconocimiento depende en gran medida de la habilidad del diseñador para idear un conjunto de características apropiado. Esta es una tarea de enormes proporciones que debe hacerse para cada nuevo problema [51].

3.3.1 Modularidad

La idea de modularidad tiene su inspiración en los organismos biológicos [70]. Por ejemplo, el cerebro tiene una estructura altamente modular y paralela. Una tarea cognitiva puede involucrar diferentes procesos. Los humanos pueden hacer diferentes procesos en paralelo, ya que la mayoría de las tareas involucra una combinación de procesamiento paralelo y serial [82].

El cerebro humano tiene una arquitectura modular y su procesamiento de información es modular. Las funciones individuales se descomponen en subprocesos que se pueden ejecutar en módulos separados sin interferencia mutua. De esta forma, se dividen módulos en submódulos y tareas en subtareas [82]. Por su parte, las redes neuronales convencionales son modulares sólo a nivel de neuronas (granularidad fina) y capas (granularidad burda) [70].

A grandes rasgos, la **modularidad** se define como la división de un objeto complejo en partes más simples. La replicación y la descomposición son dos conceptos clave para la modularidad. Estos conceptos se encuentran tanto en los objetos concretos como en el pensamiento. Además, es difícil discriminarlos con precisión pues frecuentemente ocurren en combinación [82].

La **replicación** es una forma de reutilizar el código. Una vez que un módulo se ha desarrollado y ha probado su utilidad, éste se replica. Lo que se busca es replicar unidades simples para construir una estructura compleja [82].

La **descomposición** frecuentemente se encuentra al enfrentar una tarea compleja. En este caso se descompone un problema complejo en tareas más simples y más fáciles de manejar. Así, la solución se construye a partir de los resultados de las subredes [82].

Por lo que respecta a las redes neuronales artificiales, la modularidad se puede presentar en dos formas, como se ilustra en la figura 3.16. En la primera, se construye una red neuronal grande mediante el uso de módulos, los cuales se implementan como redes neuronales. La arquitectura de un módulo es más simple y las subredes son más pequeñas que una estructura monolítica. En general, la tarea que un módulo tiene que aprender es más simple que la tarea completa, lo cual facilita el entrenamiento. Los módulos se conectan a una red de módulos y son independientes hasta cierto nivel, lo cual permite que el sistema trabaje en paralelo. En este método de modularidad se necesita un sistema de control que permita que los módulos trabajen juntos

de manera útil [82].

En la segunda forma de modularidad se tiene una red neuronal parcialmente conectada. Aquí, la división de los módulos resulta más difusa, ya que un módulo sólo es identificable como una parte de la red localmente conectada. Este tipo de modularidad se halla más próximo a la modularidad biológica [82].

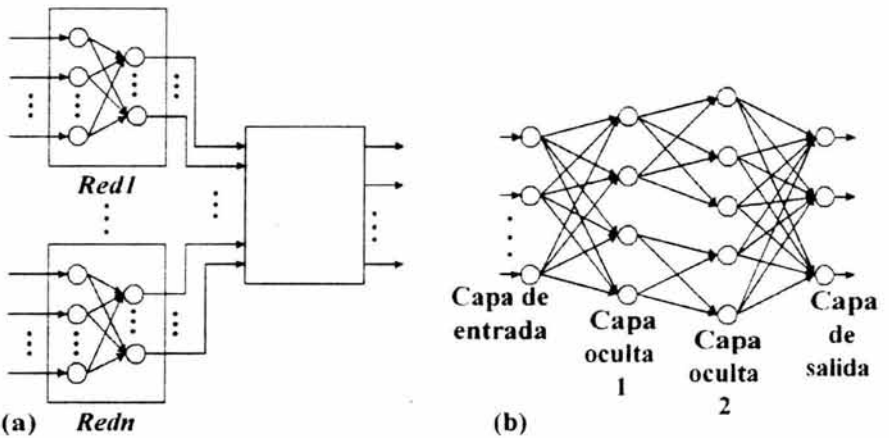


Figura 3.16: La modularidad en las redes neuronales artificiales puede presentarse en dos formas. (a) Una red neuronal modular compuesta de varias redes neuronales simples. (b) Una red neuronal modular basada en conexiones locales.

El concepto de arquitectura neuronal modular fue desarrollado por Jacobs. En su arquitectura emplea varias redes expertas y una red de salida. La tarea se descompone en múltiples subtareas y cada subtarea es asignada a una red experta. La red de salida controla el resultado final mediante el uso de las salidas de las redes expertas. Esta arquitectura fue empleada en una tarea de visión dividiéndola en dos subtareas: identificación del tipo de objeto y localización del objeto [70].

3.3.2 Ejemplos de Arquitectura Modular

Son varios los autores que han presentado propuestas de modularidad en los sistemas de reconocimiento y en las redes neuronales artificiales. Esta modularidad puede adoptar diversas formas como: la división en una parte

supervisada y una parte no supervisada, la división entre lineal y no lineal, las redes jerárquicas, etc. [70].

Por ejemplo, Bao-Qing Li y Baoxin Li integran dos redes neuronales convolucionales con diferente tamaño de entrada [4]. De esta forma, el factor de escalación es menor para aquella red neuronal convolucional cuyo plano de entrada se encuentra más cercano a la escala del texto, con lo cual se reduce la posibilidad de introducir nuevas características. Las salidas de estas dos redes se introducen en una red de dos capas para integrar las salidas de las redes neuronales convolucionales de forma no lineal.

Este modelo es aplicado a la detección de texto y a la segmentación de texto y figuras en chino, obteniendo resultados ligeramente superiores a los de la arquitectura no modular. A partir de estos resultados, Bao-Qing Li y Baoxin Li concluyen que se requieren más experimentos antes de poder atribuir alguna ventaja al uso de múltiples redes neuronales convolucionales [4].

Por su parte, Wang y Jean utilizan tres redes neuronales para construir un sistema de reconocimiento óptico de caracteres [92]. La idea es emplear una resolución baja para procesar la mayoría de los caracteres limpios y no confusos, y un esquema más complicado de reconocimiento cuando se necesita resolución alta. De esta forma, se tienen las siguientes redes:

1. Red_1. Es una red neuronal totalmente conectada con una capa oculta. Esta red es rápida, simple y usa baja resolución para procesar los caracteres limpios y no ambiguos que constituyen la mayoría de los caracteres. Red_1 comete pocos errores y tiene una tasa de rechazo baja, rechazando solamente aquellos caracteres con ruido o distorsión severos o cuya figura es similar a otras a baja resolución.

2. Red_2. Es una variante de las redes neuronales convolucionales de LeCun. Esta red poderosa y sofisticada sólo es invocada cuando se necesita examinar mayores detalles de discriminación con alta resolución o cuando los caracteres con ruido o distorsión no pueden ser reconocidos confiablemente por Red_1.

La diferencia en la arquitectura y la resolución entre Red_1 y Red_2 permite que las características extraídas por ambas sean diferentes y por lo tanto complementarias. Además, es probable que cometan diferentes tipos de errores complementarios entre sí. Por ello, cuando ninguna de las dos redes

puede reconocer el patrón de entrada debido a ruido o distorsión excesivos, la decisión final se toma ponderando la evidencia de ambas.

A parte de estas dos redes, Wang y Jean introducen una red simple y rápida de alta resolución (Red_{12}) que sirve como filtro antes de Red_2 [92]. La arquitectura de esta red es similar a la arquitectura de Red_1, pero trabaja con la resolución de Red_2. Con esto se reduce el número de llamadas a Red_2 y, por lo tanto, se mejora la velocidad de reconocimiento.

En su investigación, Oh y Suen presentan una breve revisión de diversos modelos de arquitectura modular. Estos modelos se discuten a continuación [70].

En la arquitectura OCON (*one-class-one-network*, una clase una red) se emplea una red para reconocer a cada dígito. Se define una función de costo nueva adaptada a la arquitectura y la fórmula de aprendizaje de retropropagación del error se modifica para incorporar esta función de costo. Además, se utiliza un esquema de podado selectivo de actualizaciones para excluir las muestras que rara vez actualizan los pesos.

Otros investigadores proponen un modelo de red modular adaptable para el reconocimiento de números en un sistema de lectura de cheques de banco. La red se compone de tres partes: una capa localmente conectada, una capa de subred y una capa de decisión. En la capa de subred hay diez subredes independientes, cada una responsable de una clase en particular. La primera capa pasa el vector de características a la red y la tercera capa calcula la decisión final de reconocimiento. Los investigadores reportan una mejor precisión.

Otro diseño para el reconocimiento de números escritos a mano se basa en reglas. En este diseño cada módulo está especializado en una clase específica y se implementa mediante la extracción de un conjunto de reglas sintácticas a partir de los esqueletos de los patrones.

En otro experimento se calculan muchas variables relacionadas con la figura del patrón. Estas variables se utilizan para diseñar un conjunto de reglas veto para cada clase. La tarea de una regla veto es aceptar las muestras de la clase correspondiente, y rechazar las muestras de las otras clases. El problema de este método basado en reglas es la dificultad para desarrollar un módulo de aprendizaje automático.

Inspirado en la modularidad del sistema nervioso, Schmidt propone un modelo de red neuronal modular cuyos componentes son redes multicapa [82]. Este modelo se basa en una estructura piramidal homogénea en la cual se usan redes más pequeñas para formar una red grande no totalmente

conectada.

Su modelo tiene una capa de módulos de entrada y un módulo de decisión. Todos los módulos son redes neuronales multicapa totalmente conectadas. Cada variable de entrada se conecta sólo a un módulo de entrada de forma aleatoria. Las salidas de todos los módulos de entrada se conectan a la red de decisión. Así, se logra un entrenamiento más rápido y se mejora la generalización, pero se introducen ciertas limitaciones.

Al tratarse de una red no totalmente conectada, el número de pesos se reduce. Además, la arquitectura da por resultado una red neuronal modular paralela y tolerante a fallas. Esta arquitectura combina dos métodos de generalización:

1. Cada red tiene la habilidad de generalizar sobre su espacio de entrada.

2. Se puede generalizar de acuerdo a la similaridad de los patrones de entrada.

Por su parte, el entrenamiento de esta arquitectura es más rápido que el de una red neuronal monolítica en el mismo problema debido a que:

1. El número de conexiones de la red neuronal modular y por ende el número de pesos es mucho menor. Menos pesos conducen a menos operaciones durante el entrenamiento, resultando directamente en un aumento de la velocidad de aprendizaje.

2. Los módulos de la capa de entrada son mutuamente independientes, así que el entrenamiento puede llevarse a cabo en paralelo. Con esto, el tiempo puede reducirse al máximo requerido por un módulo de entrada más el entrenamiento del módulo de decisión.

3. La división del vector de entrenamiento en partes ayuda a enfocarse en los atributos comunes.

Sin embargo, dividir el conjunto de entrenamiento puede traer problemas. El principal problema es que se puede incrementar el número de vectores de entrada iguales con valores de salida diferentes, especialmente en módulos con un número pequeño de variables de entrada.

Finalmente, Schmidt también lleva a cabo la revisión de otras formas de modularidad. Estos modelos se describen en los siguientes párrafos [82].

El primer modelo propone la descomposición de la tarea de reconocimiento: el objeto complejo (carácter) se detecta mediante el reconocimiento de sus subpatrones. Por lo tanto, la red neuronal está basada en un esquema de detección de dos niveles. En el primer nivel se reconocen los subpatrones. Aquí se usan múltiples detectores para cada segmento con el objetivo de reconocer patrones trasladados. El segundo nivel detecta la clase de acuerdo con la información del primer nivel. Esta red neuronal es capaz de reconocer patrones distorsionados, con ruido, escalados y trasladados.

Otra arquitectura se compone de dos partes generales: un subsistema de división hecho de subredes capaces de manejar un subconjunto de datos; y un subsistema de integración que determina cual de las salidas de la subred será usada como salida final. El conjunto de entrenamiento se divide de acuerdo al error de entrenamiento, y cada subred aprende su parte del conjunto de entrenamiento. El proceso se repite hasta que los módulos del subsistema de división pueden aprender los subconjuntos con un error total lo suficientemente pequeño. El subsistema de integración se entrena para determinar a que partición pertenece un vector de entrada.

En otro modelo se segmentan los datos de entrenamiento automáticamente y se emplean módulos diferentes para manejar diferentes subconjuntos. La idea básica está basada en un algoritmo de aprendizaje de multi-filtrado. De esta forma, los patrones se clasifican en diferentes niveles. En el primer nivel algunos patrones se reconocen correctamente, mientras que otros no. Las muestras clasificadas correctamente se sacan del conjunto de entrenamiento. El siguiente nivel sólo se entrena con las muestras restantes. Este proceso se repite hasta que todos los patrones se han clasificado correctamente. Cada nivel genera una red neuronal con la habilidad de reconocer un subconjunto del conjunto de entrenamiento original. Así, las redes de este modelo enfrentan una tarea de reconocimiento más simple que el problema total.

Los módulos se consideran en orden jerárquico: la salida del primer módulo que clasifique el patrón es tomada como respuesta del sistema. Sólo si el módulo no puede clasificar el patrón, se llama a la red del siguiente nivel. Todos los módulos trabajan en paralelo y por ello la velocidad del sistema es casi independiente del número de módulos.

Entre las ventajas de este modelo están que la descomposición del problema y la generación de la red apropiada es hecha por el algoritmo. Además, la idea de filtrado parece muy intuitiva para el entendimiento y el pensamiento humano. Este concepto de sistemas paralelamente masivos que crecen solos es muy prometedor. No obstante, el problema es que no es fácil decidir si la

clasificación de un patrón desconocido en un cierto nivel es correcta.

Otra idea es reducir la dimensión de entrada en módulos separados dentro de una red. Así, el sistema para un vector de entrada n-dimensional consiste en n módulos competitivos. Cada módulo está conectado sólo a $n-1$ entradas, y cada entrada está conectada a $n-1$ módulos. El aprendizaje competitivo en cada módulo se basa en vectores de entrada ligeramente diferentes de dimensión $n-1$. En una segunda etapa, una capa de reconocimiento decide de acuerdo a las salidas de todos los submódulos. Esta arquitectura paralela es superior a una única red competitiva n-dimensional.

En otra propuesta se tiene una arquitectura heterogénea compuesta por dos módulos. El primer módulo es un mapa autoorganizado que sirve para reducir la dimensión de la entrada. El segundo módulo es una red multicapa que trabaja sobre el espacio de entrada reducido.

Un último modelo a discutir es otra estructura heterogénea. En este modelo se entrena a una red sobre la salida de nodos lógicos adaptables. La idea es que la distribución de los valores de salida puede tener información importante acerca del patrón de entrada. Así, se usan todas las salidas como vector de entrada a una red neuronal.

3.3.3 La Modularidad de Clases

Con el propósito de reducir la tasa de error en el reconocimiento óptico de caracteres, se puede aplicar el concepto de modularidad de clases a las redes neuronales artificiales. Esto da lugar a una arquitectura modular de clases. En dicha arquitectura, una red neuronal modular se compone de n subredes neuronales, cada una responsable de discriminar una clase de las otras $n-1$ clases [70].

En la modularidad de clases cada clase es manejada de forma independiente de las otras clases. Así, el problema de clasificación n se descompone en n subproblemas de clasificación 2, en cada uno de los cuales se debe discriminar a una clase de las otras $n-1$ clases. Por lo tanto, la arquitectura modular de clases está compuesta de n subclasificadores, cada uno responsable de un problema de clasificación 2. La tarea de un subclasificador para una clase dada es discriminar las muestras provenientes de la clase de las otras $n-1$ clases [70].

Como se puede apreciar, el problema de clasificación 2 se resuelve mediante un clasificador 2 diseñado específicamente para la clase correspondiente. De esta forma, la arquitectura modular utiliza n subredes cada una entrenada

de forma separada. Esta arquitectura difiere de la arquitectura convencional en la cual se entrena a una gran red para clasificar a todas las clases [70].

En su estudio, Oh y Suen diseñan una red neuronal modular compuesta por n redes neuronales convencionales específicas para cada carácter [70]. Así, cada red neuronal específica tiene una capa de entrada, una capa oculta y una capa de salida totalmente conectadas como se muestra en la figura 3.17. Por lo tanto, deben utilizar un módulo de extracción de características para obtener el vector de características que será usado por las n clases.

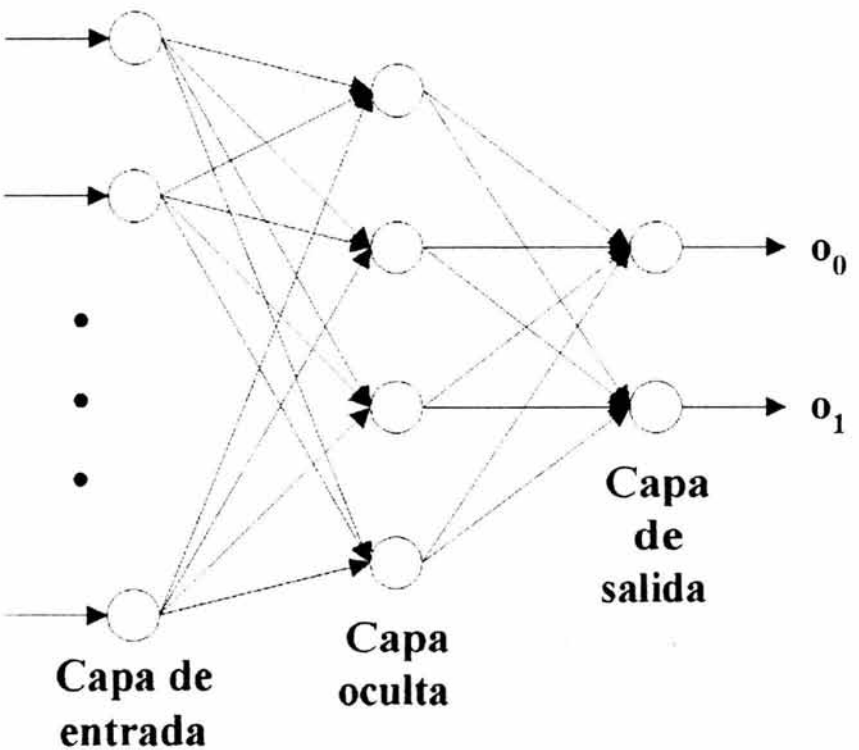


Figura 3.17: Arquitectura de las redes neuronales específicas empleadas por Oh y Suen como módulos de la red neuronal modular.

Oh y Suen demuestran la efectividad de su red neuronal modular en términos de convergencia y capacidad de reconocimiento sobre números escritos a mano (10 clases), letras mayúsculas (26 clases), pares de números unidos (100 clases) y caracteres coreanos (352 clases) [70]. Los resultados de las

pruebas muestran que el espacio entre las tasas de reconocimiento de prueba de la red neuronal modular y de la red neuronal no modular crece entre más clases están involucradas. Así, se demuestra la efectividad de la red neuronal modular en problemas de clasificación de grandes conjuntos.

3.4 Resumen

La habilidad del hombre para reconocer fácilmente los caracteres presentes en un texto se debe en gran medida a su capacidad de generalización. No obstante, la implementación de dicha habilidad en una computadora es una labor compleja.

De ahí que varios investigadores hayan decidido abordar el problema del reconocimiento de caracteres desde una perspectiva ingenieril. Con ello buscan la construcción de sistemas que puedan ser útiles para el ser humano. Dichos sistemas convierten la imagen de un carácter en su respectiva representación simbólica.

Para llevar a cabo el proceso de reconocimiento óptico de caracteres se han empleado las redes neuronales artificiales con muy buenos resultados. Estas redes neuronales artificiales se componen de unidades simples llamas neuronas artificiales. Dichas neuronas constituyen un modelo basado en la anatomía y funcionamiento de las neuronas biológicas.

De manera general, las neuronas artificiales combinan sus entradas ponderándolas con un conjunto de pesos para obtener un valor al cual agregan un sesgo. Como resultado se obtiene un valor llamado estado de activación. Este valor es procesado por una función de transferencia para obtener la salida de la neurona artificial.

Al agruparse múltiples neuronas en varias capas se obtiene una arquitectura de red neuronal artificial multicapa. En esta arquitectura se identifican tres tipos de capa: la capa de entrada, las capas ocultas y la capa de salida.

Estas redes aprenden a realizar su tarea mediante el entrenamiento con un conjunto de ejemplos y un algoritmo de aprendizaje. Así es como aprenden a reconocer caracteres.

Para realizar el reconocimiento de caracteres se pueden emplear las redes neuronales convolucionales, las cuales están diseñadas de manera especial para tratar con imágenes bidimensionales como la imagen de un carácter. Estas redes trabajan directamente con la imagen normalizada de un carácter. Con ello, se evita la construcción manual de un extractor de características

y, por ende, se obtienen mejores resultados en el reconocimiento.

Ahora bien, con el propósito de mejorar los resultados del reconocimiento, varios autores han propuesto la aplicación de la modularidad sobre las redes neuronales artificiales. De entre varios de los modelos de modularidad, la modularidad de clases propuesta por Oh y Suen parece ser uno de los modelos más prometedores. No obstante, Oh y Suen utilizan un modelo convencional de reconocimiento, y emplean la extracción manual de características en su modelo [70].

Por supuesto, resulta interesante investigar cuál podría ser el efecto de aplicar el modelo de modularidad de clases a las redes neuronales convolucionales. Esta combinación da lugar a un modelo de red neuronal convolucional modular. Este modelo es el objeto de estudio del siguiente capítulo.

Capítulo 4

El Modelo de Red Neuronal Convolucional Modular

Como se ha mencionado en otros capítulos, los resultados entregados por el reconocimiento óptico de caracteres no son perfectos debido a diversos problemas causados por múltiples factores. Una de las formas en que se puede mejorar el resultado entregado por los sistemas de reconocimiento óptico de caracteres es mediante la búsqueda, diseño y uso de mejores técnicas de reconocimiento.

De esta forma, la investigación en redes neuronales artificiales ha conducido al diseño de redes neuronales convolucionales. Estas redes neuronales constituyen una de las mejores técnicas empleadas para llevar a cabo el reconocimiento óptico de caracteres debido a que fueron diseñadas de manera específica para tratar con la variabilidad de las imágenes bidimensionales mediante la replicación de pesos de las neuronas, gracias a lo cual también logran cierta resistencia al ruido. Para emplear este tipo de redes neuronales en un sistema de reconocimiento, los desarrolladores deben diseñar la red neuronal y entrenarla mediante un conjunto de ejemplos que incluya a todas las clases de caracteres que se desea que reconozca el sistema [51].

En relación a este punto, algunos investigadores han demostrado las ventajas de un enfoque modular en el diseño de redes neuronales artificiales, en lo que se refiere a precisión del reconocimiento y tiempo de entrenamiento [70] [82].

Partiendo de dichas investigaciones, en esta tesis se propone un modelo para el reconocimiento óptico de caracteres que aplique los conceptos de modularidad a las redes neuronales convolucionales. En las siguientes sec-

ciones se analiza el modelo propuesto, junto con la descripción detallada del procedimiento de selección de los conjuntos de entrenamiento, validación y prueba; el proceso de entrenamiento y los parámetros de aprendizaje.

4.1 La Red Neuronal Convolucional Modular

La **red neuronal convolucional modular** presentada en este trabajo emplea el concepto de modularidad de clases propuesto por Oh y Suen [70]. Así, el problema de clasificar n caracteres se divide en n subproblemas de clasificación, para los cuales se diseña una **red neuronal específica** entrenada para reconocer un sólo carácter y rechazar los $n-1$ caracteres restantes. Además, se incluye un módulo de decisión de clases que integra las salidas de las n redes neuronales convolucionales.

Teniendo n clases de caracteres, cada una denotada por ω_i, $0 \leq i < n$, entonces la red neuronal convolucional modular consiste en n subredes M_{ω_i}, $0 \leq i < n$, cada una responsable de una sola clase. La tarea específica de cada subred M_{ω_i} es clasificar a los caracteres en uno de dos grupos: Ω_0 y Ω_1, donde:

$$\Omega_0 = \{\omega_i\} \qquad \Omega_1 = \{\omega_j \mid 0 \leq j < n, j \neq i\} \tag{4.1}$$

La arquitectura de cada una de las subredes será la de una red neuronal convolucional, en vez de la red completamente conectada propuesta por Oh y Suen [70].

De esta forma, siguiendo el modelo propuesto en [51], una red neuronal convolucional estará compuesta de seis capas. La red neuronal recibe a la entrada una imagen de 32×32 pixeles, donde cada carácter ha sido normalizado en tamaño a 20×20 pixeles y centrado en una imagen de 28×28 mediante el cálculo de su centro de masa, el cual se hace coincidir con el centro del campo de 28×28 como se ilustra en la figura 4.1. En caso de ser necesario, se agregan pixeles de fondo para cubrir los 32×32 pixeles. De esta forma, las características distintivas como orillas o esquinas pueden aparecer en el centro del campo receptivo de los detectores de características de más alto nivel de la red. Las imágenes originales se encuentran en escala de grises $[0, 255]$, sin embargo, estos valores se normalizan para quedar en un rango de -0.1 para los pixeles del fondo (blancos) y 1.175 para los pixeles del frente (negros). Con esto la media de la entrada se hace casi 0, y la varianza aproximadamente 1, con lo cual se acelera el aprendizaje.

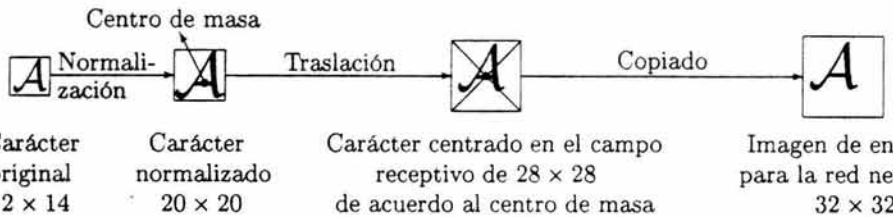


Figura 4.1: La imagen del carácter debe ser procesada antes de pasar a través de la red neuronal convolucional. Primero el carácter se normaliza en tamaño a 20×20 pixeles conservando las proporciones de la imagen. A continuación, se calcula el centro de masa de esta imagen normalizada. Posteriormente se traslada la imagen para que su centro de masa coincida con el centro de un campo de 28×28 pixeles y se copia en un campo de 32×32 pixeles.

La primera capa es una capa convolucional (C1) compuesta de seis mapas de características. Cada mapa de características está compuesto de 28×28 neuronas, cada una de las cuales se conecta a un vecindario de 5×5 en la entrada, el cual se conoce como **campo receptivo**, tal y como se aprecia en la figura 4.2. Así, cada neurona consta de 25 pesos entrenables más un sesgo entrenable. Los campos receptivos de neuronas contiguas del mapa de características están centrados en puntos contiguos correspondientes en la imagen, por eso los campos receptivos de neuronas contiguas se traslapan. Por ejemplo, los campos receptivos de neuronas contiguas horizontalmente se superponen por cuatro columnas y cinco renglones. Todas las neuronas de un mismo mapa de características comparten el mismo conjunto de 25 pesos y el sesgo para detectar las mismas características en todos los lugares posibles de la entrada. Por consiguiente, en la primera capa se extraen seis tipos diferentes de características mediante seis neuronas en la misma posición, con lo cual se forman seis diferentes mapas de características. El número de parámetros entrenables para esta capa es de 156: (25 entradas + 1 sesgo) $\times$ 6 mapas. El número de conexiones es de 122,304: (25 entradas + 1 sesgo) $\times$ (28×28 neuronas) $\times$ 6 mapas.

La segunda capa es una capa de submuestreo (S2) con 6 mapas de características de 14×14 neuronas, uno para cada mapa de la capa anterior. El campo receptivo de cada neurona es un área de 2×2 en el correspondiente mapa de características de la capa C1, lo cual se puede observar en la figura 4.3. Cada neurona toma sus cuatro entradas, las suma, multiplica el resultado por un coeficiente entrenable y agrega un sesgo entrenable, pasando el

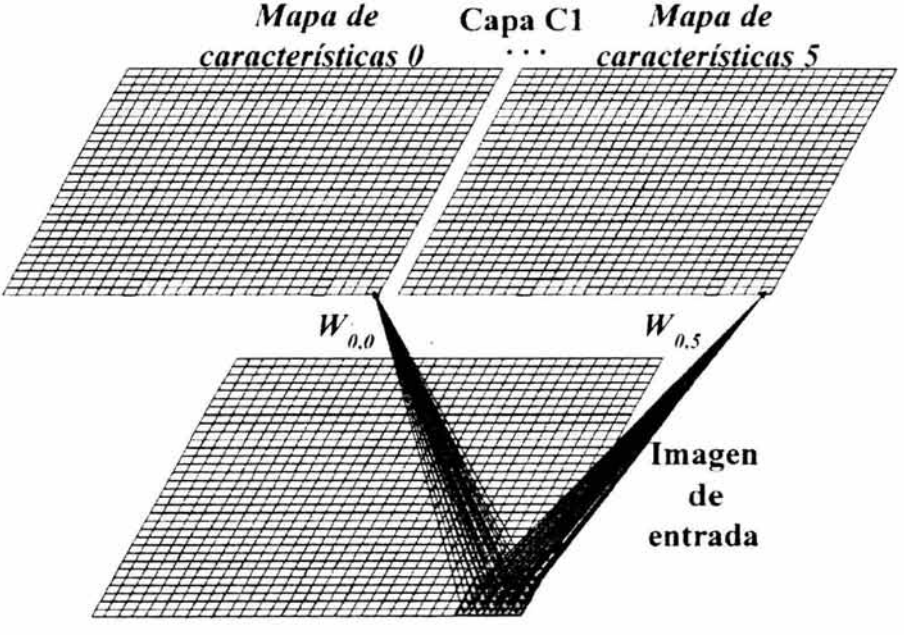


Figura 4.2: La capa C1 se compone de 6 mapas de características de 28×28 neuronas. Cada mapa de características emplea un campo receptivo de 5×5 neuronas para extraer las características de la imagen.

resultado por la misma función sigmoide empleada por las neuronas de las capas convolucionales. Como los campos receptivos no se superponen, se obtienen mapas de características de la mitad del tamaño de los de la capa anterior. Esta capa tiene 12 parámetros entrenables: (1 coeficiente + 1 sesgo) × 6 mapas. El número de conexiones de la capa es de 5,880: (4 entradas + 1 sesgo) × (14 × 14 neuronas) × 6 mapas.

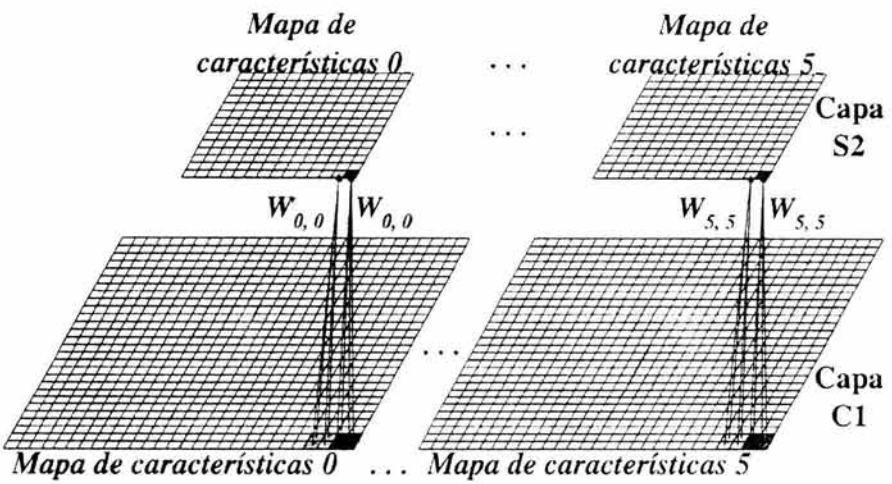


Figura 4.3: La capa S2 se compone de 6 mapas de características de 14 × 14 neuronas, uno por cada mapa de la capa C1. Cada mapa de características de la capa S2 emplea un campo receptivo de 2 × 2 neuronas para muestrear los datos del correspondiente mapa de la capa C1.

La tercera capa (C3) es otra capa convolucional con 16 mapas de características de 10 × 10 neuronas. Cada neurona en cada mapa de características tiene varios campos receptivos de 5 × 5 en la misma posición, pero en diferentes mapas de características de la capa S2 de acuerdo con la tabla 4.1. Por ejemplo, el primer mapa de características de la capa C3 tiene un conjunto de 75 pesos por neurona, siendo los mismos para todas las neuronas del mapa (figura 4.4). La cantidad de sesgos es de 16, uno para cada mapa de la capa C3, y las operaciones son similares a C1. La razón para no conectar cada mapa de características de S2 a cada mapa de C3 es forzar la asimetría de la red, con lo cual diferentes mapas de características extraen diferentes características, de preferencia complementarias, porque obtienen diferentes conjuntos de entradas. Además, el número de conexiones es menor. Con el

esquema de la tabla 4.1, los primeros 6 mapas toman entradas de cada uno de los subconjuntos contiguos de 3 mapas de S2, los siguientes 6 toman entradas de cada uno de los subconjuntos contiguos de 4 mapas de S2. Los siguientes 3 toman entradas de conjuntos discontinuos de 4, y el último toma entradas de todos los mapas de S2. De esta forma, la capa C3 tiene 1,516 parámetros entrenables: 25 pesos por cada campo receptivo $\times$ 60 campos receptivos + 16 sesgos. El número de conexiones de C3 es de 151,600: (25 entradas $\times$ 60 campos receptivos + 16 sesgos) $\times$ (10 $\times$ 10 neuronas).

		Capa C3														
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Capa S2	0	X				X	X	X			X	X	X	X		X	X
	1	X	X				X	X	X			X	X	X	X		X
	2	X	X	X				X	X	X			X		X	X	X
	3		X	X	X			X	X	X	X			X		X	X
	4			X	X	X			X	X	X	X		X	X		X
	5				X	X	X			X	X	X	X		X	X	X

Tabla 4.1: Cada columna indica que mapas de características de S2 se combinan mediante las neuronas en un mapa de características de C3.

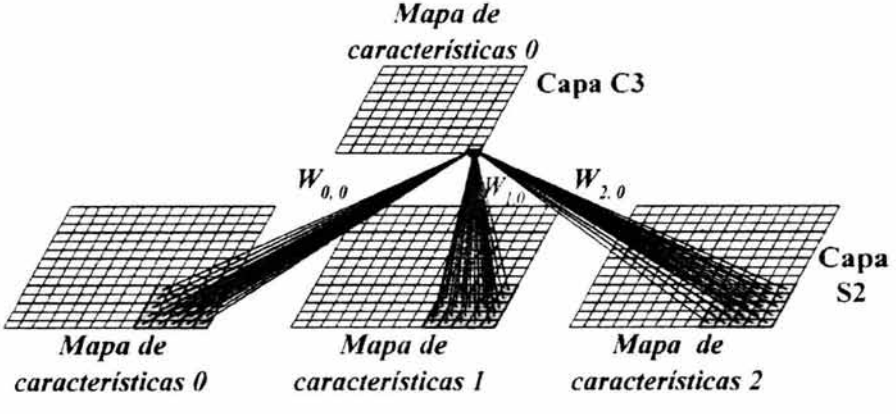


Figura 4.4: Las neuronas del mapa 0 de la capa C3 reciben entradas de los mapas 0, 1 y 2 de la capa S2.

La cuarta capa (S4) es otra capa de submuestreo con 16 mapas de tamaño 5 $\times$ 5 y que funciona de forma similar a S2, pero trabajando sobre C3, como se muestra en la figura 4.5. Tiene 32 parámetros entrenables: (1 coeficiente

+ 1 sesgo) × 16 mapas. Esta capa tiene 2,000 conexiones: (4 entradas + 1 sesgo) × (5 × 5 neuronas) × 16 mapas.

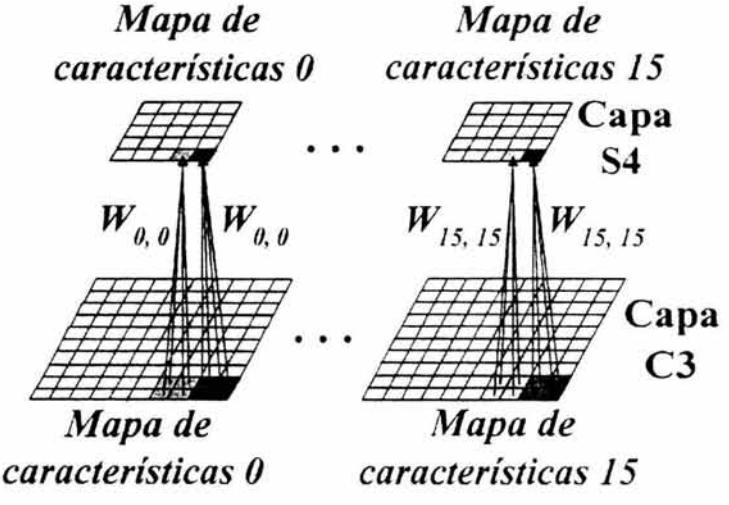


Figura 4.5: La capa S4 tiene un mapa de características por cada mapa de características de la capa C3. Cada uno de los 16 mapas de características de la capa S4 utiliza un campo receptivo de 2 × 2 neuronas para efectuar el muestreo del mapa correspondiente de la capa C3.

La quinta capa (C5) es la última capa convolucional y consta de 120 mapas de características de tamaño 1 × 1, ya que el tamaño de los mapas de S4 es el mismo que el del campo receptivo: 5 × 5 (ver figura 4.6). Cada uno de los 120 mapas se conecta con cada uno de los 16 mapas de S4 dando un total de 48,120 conexiones entrenables: (25 entradas × 16 mapas) × 120 mapas + 120 sesgos.

Finalmente, la capa de salida contiene 2 neuronas totalmente conectadas con C5 y por ello tiene 242 parámetros entrenables: 120 mapas × 2 neuronas + 2 sesgos. La figura 4.7 muestra el esquema de conexión entre la capa C5 y la capa de salida.

De esta forma, el total de parámetros entrenables de la red es de 50,078. Las dos neuronas de la capa de salida se denotan o_0 y o_1 para Ω_0 y Ω_1 respectivamente (ver ecuación 4.1). La arquitectura completa de la red neuronal convolucional se presenta en la figura 4.8.

La figura 4.9 muestra la arquitectura completa de la red neuronal convolucional modular formada por las n redes neuronales convolucionales M_{ω_i}.

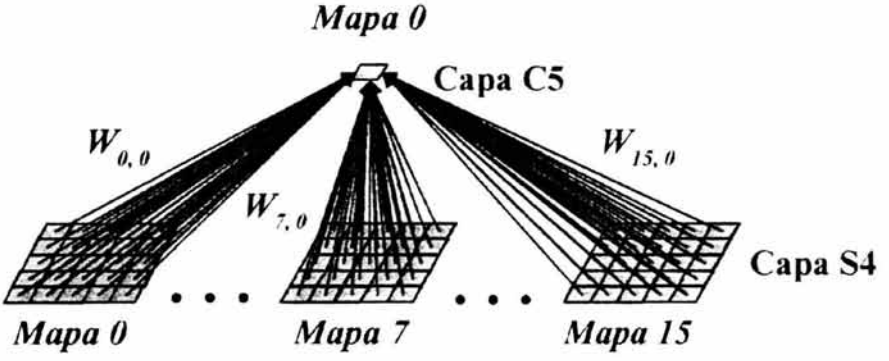


Figura 4.6: La capa C5 se compone de 120 mapas de características de tamaño 1×1. A través de un campo receptivo de 5×5 neuronas, cada mapa de características de la capa C5 obtiene los datos provenientes de los mapas de la capa S4.

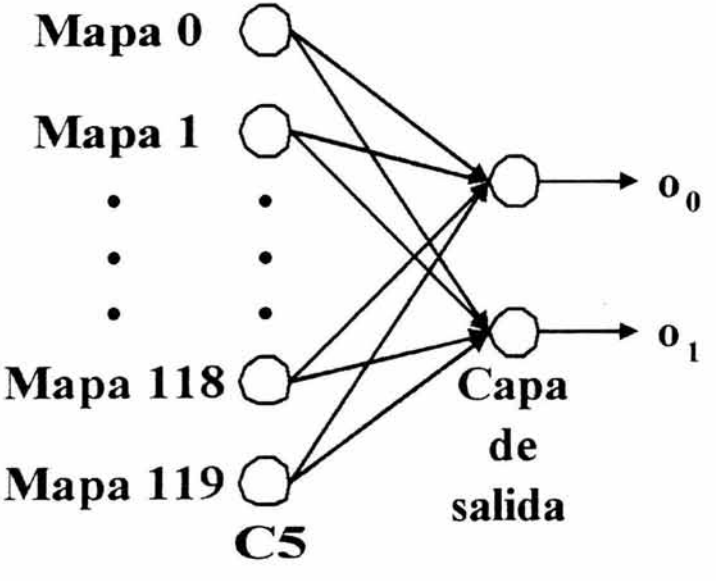


Figura 4.7: Los 120 mapas de características de la capa C5 se conectan completamente con las dos neuronas de la capa de salida.

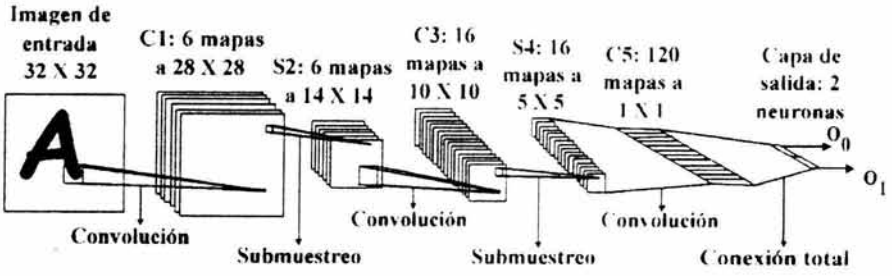


Figura 4.8: Arquitectura de la red neuronal convolucional para la identificación de un carácter.

Las n redes reciben la imagen de entrada y generan su vector de salida $D_{\omega_i} = (o_0, o_1)$.

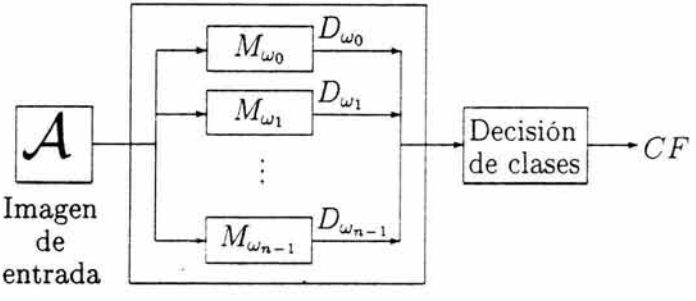


Figura 4.9: Arquitectura de la red neuronal convolucional modular.

La decisión final CF sobre la clase a la que pertenece la imagen de entrada se efectúa tomando en cuenta los vectores de salida de las n redes neuronales especializadas de acuerdo con el algoritmo 4.1. De manera general, el funcionamiento del algoritmo es el siguiente: se toma la salida con menor nivel de penalización de cada una de las redes y, si una y sólo una red no rechazó el carácter, entonces el resultado es la clase correspondiente a esa red, de lo contrario, el carácter se rechaza.

Algoritmo 4.1. Decisión de clases
Variables de Entrada:

D El conjunto de los n vectores de salida de las n redes neuronales convolucionales especializadas

Variables de Salida:

CF La clase asignada a la imagen de entrada por parte de la red neuronal convolucional modular

Variables Locales:

n El número de clases a reconocer

ω_i La clase $i, 0 \leq i < n$

M_{ω_i} La red neuronal convolucional especializada en el reconocimiento de la clase ω_i

Ω_0 Conjunto donde la red M_{ω_i} almacena los caracteres pertenecientes a la clase $\omega_i : \Omega_0 = \{\omega_i\}$

Ω_1 Conjunto donde la red M_{ω_i} almacena los caracteres que no pertenecen a la clase $\omega_i : \Omega_1 = \{\omega_j \mid 0 \leq j < n, j \neq i\}$

o_0 Salida para Ω_0

o_1 Salida para Ω_1

$Clase_{\Omega_0}$ Lista que contiene los caracteres que cada red M_{ω_i} identifica como pertenecientes a la clase ω_i

Funciones Locales:

$insertar(l, e)$ Inserta el elemento e en la lista l y devuelve la lista resultante

$cabeza(l)$ Devuelve el elemento que se halla en la cabeza de la lista l

$tam(l)$ Devuelve el número de elementos de la lista l

1. Para $i = 0$ hasta $n - 1$ hacer
   (a) si $D_{\omega_i}[o_0] < D_{\omega_i}[o_1]$ entonces
      i. $Clase_{\Omega_0} = insertar(Clase_{\Omega_0}, \Omega_0)$
2. Si $tam(Clase_{\Omega_0}) = 1$ entonces

(a) $CF = cabeza(Clase_{\Omega_0})$

3. Si no

(a) $CF = Rechazado$

4. Regresar CF

4.2 La Red Neuronal Convolucional General

La arquitectura de una red neuronal convolucional específica descrita en la sección anterior, generalmente se emplea para el reconocimiento de los n caracteres. Así, se diseña y entrena una sola **red neuronal convolucional general** para el reconocimiento de todos los caracteres.

Debido a que una sola red se encarga de clasificar las imágenes de entrada, en la arquitectura general de red no se requiere de un módulo de decisión de clases, y únicamente se debe entrenar una red. No obstante, esta red contiene un mayor número de conexiones y pesos entrenables.

Así, la principal diferencia entre la red neuronal convolucional general y una red neuronal convolucional específica yace en la capa de salida. En la red neuronal general, esta capa consta de n neuronas de salida totalmente conectadas con la capa convolucional C5. El resto de las capas permanece igual.

4.3 Selección de las Muestras de Entrenamiento, Validación y Prueba

Con el propósito de poder probar el modelo presentado en esta tesis es necesario entrenar y evaluar a la red neuronal convolucional modular. Se requiere también entrenar y evaluar a una sola red neuronal convolucional para el reconocimiento de todo el conjunto de caracteres. Esta última red servirá de comparación con el modelo propuesto para verificar si el modelo modular presentado es superior en términos de reconocimiento al modelo convencional.

Por lo tanto, se requiere reunir un conjunto de muestras que sirva para llevar a cabo el entrenamiento, validación y prueba de los modelos. De esta forma, la base de datos utilizada para entrenar y probar los dos modelos

descritos en esta tesis es la misma base de datos empleada en los experimentos descritos en [51].

Esta base de datos fue construida a partir de dos bases de datos del NIST: Special Database 3 y Special Database 1. Estas bases de datos contienen imágenes binarias de dígitos escritos a mano. Las imágenes contenidas en SD-3 son más limpias y fáciles de reconocer que las imágenes de SD-1. La razón de esto es que SD-3 fue recolectada entre los empleados de la oficina del censo, mientras que SD-1 fue tomada de estudiantes de secundaria.

A partir de estas bases de datos del NIST, se formó la base de datos MNIST (Modified NIST) [51]. En primer lugar, se separó SD-1 en dos partes: 250 escritores para el conjunto de entrenamiento y los siguientes 250 para el conjunto de prueba. De esta forma, se obtuvieron dos conjuntos con casi 30,000 ejemplos cada uno. El nuevo conjunto de entrenamiento se completó con suficientes ejemplos de SD-3 hasta reunir 60,000 ejemplos de entrenamiento. Por su parte, el conjunto de prueba se redujo a 10,000 muestras: 5,000 ejemplos de SD-1 y 5,000 muestras de SD-3.

En la presente investigación la división de esta base de datos se hará tomando en cuenta los lineamientos sugeridos en [5], quedando de la siguiente forma:

1. Conjunto de entrenamiento: 60,000 muestras (85.72% de la base de datos).

2. Conjunto de validación: 5,000 muestras (7.14% de la base de datos).

3. Conjunto de prueba: 5,000 muestras (7.14% de la base de datos).

El tamaño del conjunto de entrenamiento es ligeramente superior al tamaño mínimo recomendado en [51] con una tasa de ejemplos por pesos de:

$$\frac{60,000 \text{ ejemplos}}{50,078 \text{ parámetros entrenables}} = 1.198130915771396621270817524661 53$$

Dado este conjunto de entrenamiento, la red neuronal convolucional general puede ser entrenada de forma directa. Sin embargo, para entrenar a la red neuronal convolucional modular, es necesario entrenar a cada una de las redes neuronales específicas que la componen.

Por esta razón, para poder entrenar a cada red neuronal convolucional en el reconocimiento de una clase ω_i se organizarán las muestras del conjunto de

entrenamiento en dos grupos: Z_{Ω_0} y Z_{Ω_1} de forma que Z_{Ω_0} tenga muestras de las clases en Ω_0 y Z_{Ω_1} de las clases en Ω_1 (ver ecuación 4.1)[70]:

$$Z_{\Omega_0} = \bigcup_{\omega_k \in \Omega_0} Z_{\omega_k} \qquad Z_{\Omega_1} = \bigcap_{\omega_k \in \Omega_1} Z_{\omega_k} \tag{4.2}$$

Estos conjuntos de entrenamiento no están balanceados entre las dos clases Ω_0 y Ω_1. Sin embargo, el conjunto de entrenamiento total si está balanceado, es decir, hay un número similar de muestras para cada carácter y por ello Z_{Ω_1} contiene aproximadamente $n-1$ veces más muestras que Z_{Ω_0}.

Son varios los motivos por los cuales se seleccionó la base de datos MNIST para evaluar el modelo presentado en esta tesis. En primer lugar, el reconocimiento de dígitos individuales escritos a mano es una excelente prueba de referencia para la comparación de los métodos de reconocimiento de figuras [51]. Esto se debe a que la gente tiende a personalizar la forma en que escribe los dígitos. Estas sutiles variaciones en el estilo son difíciles de tratar debido a que se incrementa el tamaño del espacio de entrada a examinar [22]. En segundo lugar, esta base de datos contiene datos reales recolectados y tratados adecuadamente. Finalmente, la base de datos ya ha sido utilizada para probar otros métodos de reconocimiento.

4.4 Entrenamiento de la Red Neuronal Convolucional Modular

Para entrenar a la red neuronal convolucional modular propuesta se debe entrenar a cada una de las redes que la componen mediante los conjuntos mencionados en la sección anterior. El algoritmo empleado para dicho propósito fue presentado en la sección 3.2.4 del capítulo anterior. El propósito de esta sección es presentar los valores que se asignarán a los parámetros del algoritmo para efectos de esta investigación.

Como se mencionó en el capítulo previo, los pesos se suelen inicializar con valores aleatorios pequeños en un rango con media cero. Así, en esta investigación los pesos se inicializan con valores aleatorios empleando una distribución uniforme entre $-2.4/F_i$ y $2.4/F_i$, donde F_i es el número de entradas a la neurona i, a la cual pertenecen las conexiones que se desea inicializar [51].

Para prevenir que el paso de actualización crezca más de lo debido, al parámetro de regulación μ se le asigna el valor de 0.02 [51].

Por su parte, en el capítulo previo se describió el criterio de paro que se emplea: la validación cruzada. Para usar este criterio se utiliza el conjunto de validación recolectado para vigilar el proceso de entrenamiento, deteniendo el algoritmo en cuanto el error de validación deja de disminuir y empieza a crecer [5]. En caso de que el proceso de aprendizaje llegue al ciclo número 21, el entrenamiento se detiene y el conjunto de pesos de la red así entrenada se toma como el conjunto de parámetros óptimo [51].

En cuanto a la tasa de aprendizaje global, se emplea un coeficiente de aprendizaje adaptable η que varía a lo largo del entrenamiento. Bajo este esquema, al principio se emplea un coeficiente relativamente grande para escapar de los mínimos locales, el cual se reduce conforme el entrenamiento avanza, con el fin de no perder el mínimo [5] [82]. Los valores de la tasa de aprendizaje global se asignan dependiendo del ciclo de aprendizaje actual de acuerdo con la tabla 4.2 [51].

Época	Valor de η
1, 2	0.0005
3, 4, 5	0.0002
6, 7, 8	0.0001
9, 10, 11, 12	0.00005
13 en adelante	0.00001

Tabla 4.2: Valor asignado a la tasa global de aprendizaje η correspondiente al ciclo de entrenamiento en que se emplea.

Finalmente, el tamaño del paso ε_k se estima sobre 500 muestras del conjunto de entrenamiento.

4.5 Resumen

En este capítulo se presentó el modelo de red neuronal convolucional modular propuesto en esta tesis. Dicho modelo aplica el concepto de modularidad de clases a las redes neuronales convolucionales con el propósito de obtener una red que mejore el desempeño de los sistemas de reconocimiento óptico de caracteres.

En este modelo se entrena a un conjunto de redes neuronales convolucionales cada una especializada en el reconocimiento de un sólo carácter. Por lo tanto, se obtienen tantas redes como caracteres haya que reconocer. La imagen de un carácter se pasa por cada una de las redes y la decisión final sobre la clase a la que pertenece el carácter se toma considerando el resultado de todas las redes neuronales especializadas.

Para entrenar a esta red neuronal convolucional modular se entrena a cada una de las redes neuronales convolucionales especializadas empleando el conjunto de entrenamiento usado en [51]. Dicho entrenamiento se lleva a cabo con el algoritmo de aprendizaje presentado en el capítulo anterior, utilizando los parámetros presentados en este capítulo.

Con el propósito de evaluar este modelo se entrena también a una red neuronal convolucional general para el reconocimiento de todos los caracteres. Para ello, se emplea el mismo conjunto de entrenamiento, algoritmo de aprendizaje y parámetros de entrenamiento.

La evaluación tanto del modelo general como del modelo modular propuesto se realiza con el conjunto de prueba mencionado en la sección 4.3. Estas pruebas servirán de comparación para determinar si el modelo propuesto sobrepasa al modelo convencional en términos de reconocimiento. De esta forma, el siguiente capítulo presenta los resultados de la evaluación, así como las conclusiones que se derivan de tales resultados.

Capítulo 5

Evaluación del Modelo de Red Neuronal Convolucional Modular

El desempeño de la arquitectura de red neuronal convolucional modular debe ser comparado con el desempeño del modelo de red neuronal convolucional general. De esta forma, se puede comprobar si la hipótesis H_1, planteada en el capítulo 1, es confirmada o rechazada.

Por esta razón, es necesario diseñar y entrenar una red neuronal convolucional general que reconozca los n caracteres que de forma individual reconoce cada una de las n redes neuronales convolucionales que integran a la red neuronal convolucional modular. Dicha red neuronal convolucional general fungirá como patrón de comparación.

Por lo tanto, el primer paso consiste en la integración de las n redes neuronales convolucionales en una sola red neuronal convolucional modular tal y como se especifica en el capítulo anterior. Esta es la red que será sometida a las pruebas. El segundo paso es el diseño y entrenamiento de la red neuronal convolucional general. La arquitectura de esta red general será la misma que la de las redes neuronales convolucionales especializadas, y será entrenada con los mismos datos, empleando para ello el algoritmo y parámetros de entrenamiento ya descritos.

Los resultados de las pruebas de estos modelos deben ser organizados con el fin de poder analizarlos adecuadamente. Además, se debe decidir qué métricas serán recolectadas durante las pruebas para describir dichos resultados. Estos temas son el objeto de la primera sección de este capítulo.

En la siguiente sección se presentan los resultados obtenidos de las pruebas en forma de matrices, tablas y gráficas. Asimismo, se comparan los resultados

obtenidos por ambos modelos.

En la última sección se discuten las razones que llevan a la aceptación o rechazo de la hipótesis con base en los resultados. De igual forma, se proporcionan las conclusiones que se derivan del presente estudio. Finalmente, se mencionan algunas líneas de investigación futura obtenidas a partir de esta tesis.

5.1 Criterios de Evaluación

Como se mencionó en el capítulo previo, las dos redes neuronales convolucionales serán probadas con dígitos escritos a mano. El desempeño de las redes neuronales convolucionales se evaluará utilizando las métricas que se describen en las siguientes líneas.

En primer lugar, se debe obtener la cantidad de caracteres que fueron reconocidos correctamente, incorrectamente, y los no reconocidos (rechazados). Estos resultados se despliegan en una **matriz de confusión** [42] como la que se muestra en la figura 5.1, en donde las columnas representan la clasificación del símbolo por parte de la red neuronal y los renglones representan el valor verdadero del símbolo. Dicha matriz de confusión permite examinar el comportamiento de la red neuronal, mostrando cuál y qué tan severa fue la confusión [72].

		Reconocidos como					
		a	b	c	d	Rechazados	Incorrectos
Carácter verdadero	a	9	0	0	1	0	1
	b	0	8	0	0	2	0
	c	2	0	6	1	1	3
	d	1	0	0	9	0	1
	Errores	3	0	0	2	3	5

Figura 5.1: Matriz de confusión.

A partir de estos datos se calculan tres tasas: la tasa de reconocimiento de caracteres, la tasa de error y la tasa de rechazo. La **tasa de reconocimiento de caracteres** mide el porcentaje de exactitud obtenido en el reconocimiento de los caracteres, y está dada por [42] [67]:

$$Tasa\ de\ reconocimiento\ de\ caracteres = \frac{Caracteres\ reconocidos}{Total\ de\ caracteres} \quad (5.1)$$

Los **errores** son la suma de eliminaciones, inserciones y sustituciones necesarias para convertir la salida de la red neuronal en la salida deseada [67]. Esta información se reporta como porcentaje de ejemplos clasificados incorrectamente y, generalmente resulta más ilustrativa que la tasa de reconocimiento de caracteres. Por ejemplo, al reducir la tasa de error del 4% al 2% se puede afirmar que el porcentaje de error se redujo en un 50% [77]. La **tasa de error** se obtiene mediante la siguiente fórmula:

$$Tasa\ de\ error = \frac{Caracteres\ reconocidos\ incorrectamente}{Total\ de\ caracteres} \quad (5.2)$$

Por su parte, los **rechazos** incluyen aquellos caracteres que la red neuronal no pudo clasificar, los cuales no cuentan como errores [72]. La **tasa de rechazo** se obtiene de forma similar a las anteriores:

$$Tasa\ de\ rechazo = \frac{Caracteres\ no\ reconocidos}{Total\ de\ caracteres} \quad (5.3)$$

Por supuesto, una red neuronal que indica que no puede clasificar un caracter es superior a una que lo clasifica erróneamente. Por ello es importante minimizar la tasa de error, incluso a costa de incrementar un poco la tasa de rechazo. Sin embargo, la tasa de reconocimiento de caracteres por si sola no toma en cuenta que un rechazo es mejor que una mala clasificación, pues ambas actúan en contra de la tasa de reconocimiento. Para considerar este hecho se emplea la **tasa de fiabilidad**, la cual se calcula de la siguiente forma [72]:

$$Tasa\ de\ fiabilidad = \frac{Tasa\ de\ reconocimiento\ de\ caracteres}{1 - Tasa\ de\ rechazo} \quad (5.4)$$

Así, la fiabilidad es alta cuando ocurren pocos errores de clasificación. Sin embargo, los rechazos también deben contarse de alguna forma, pues también afectan la calidad de la red neuronal. Por ejemplo, considérese una red neuronal que clasifica correctamente el 50% de los caracteres y rechaza el otro 50%. Evidentemente, el funcionamiento de esta red no puede considerarse

correcto, aún cuando su fiabilidad sea del 100%. En consecuencia, una buena red neuronal combina una tasa alta de fiabilidad con una tasa de reconocimiento alta, lo cual se puede medir a través de la **tasa de aceptabilidad** [72]:

$$\begin{aligned} \textit{Tasa de aceptabilidad} = \; & \textit{Tasa de fiabilidad} \times \\ & \textit{Tasa de reconocimiento de caracteres} \end{aligned} \tag{5.5}$$

5.2 Resultados

Con el fin de poder evaluar los modelos de redes neuronales convolucionales es necesario entrenar dichos modelos. Para ello, se emplean los conjuntos de entrenamiento y validación descritos en el capítulo anterior. Así, cada una de las redes neuronales convolucionales especializadas se entrena para reconocer un dígito en particular. Además, se entrena a la red neuronal convolucional general que reconocerá todos los dígitos.

Debido a la inicialización de pesos aleatoria, los resultados de varias corridas de entrenamiento difieren aún cuando se emplee la misma arquitectura, algoritmo de aprendizaje, parámetros de entrenamiento y base de datos. Por lo tanto, es necesario llevar a cabo varias corridas de entrenamiento para cada red. Esto hará que los resultados reportados sean más confiables [77].

Siguiendo esta recomendación, en el presente estudio se realizaron cinco corridas de entrenamiento para cada red neuronal convolucional. En las figuras 5.2 a 5.12 se muestran las gráficas de error de las diez redes neuronales convolucionales especializadas y de la red neuronal convolucional general. Cada gráfica muestra la tasa de error sobre el conjunto de validación para cada una de las redes obtenida en las cinco corridas de entrenamiento.

Por lo que se refiere a las 10 redes neuronales convolucionales especializadas, las gráficas muestran que el entrenamiento en cada una de las cinco corridas de entrenamiento produce resultados similares. Las mayores divergencias se encuentran en las redes para los dígitos 8 y 9, para las cuales la diferencia en la tasa de error entre la red con mayor tasa y la red con menor tasa es de 1.14% y 1.16% respectivamente. En consecuencia, se puede afirmar que el entrenamiento del modelo de red neuronal convolucional modular producirá resultados similares.

En cuanto a la red neuronal convolucional general, se observa una mayor diferencia entre las diversas corridas. De esta forma, la diferencia en la tasa

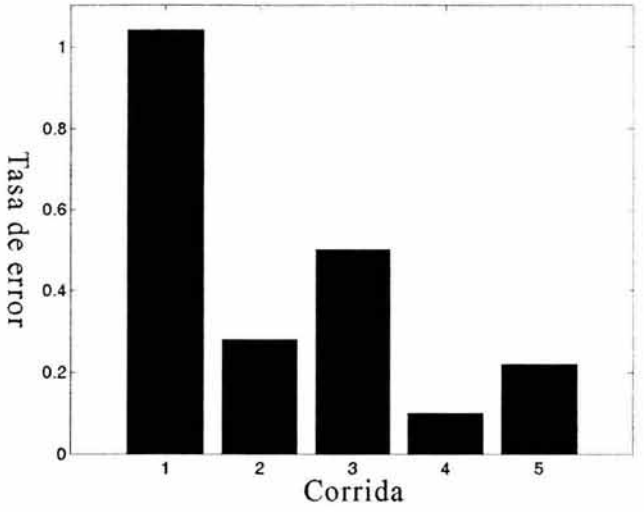


Figura 5.2: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 0.

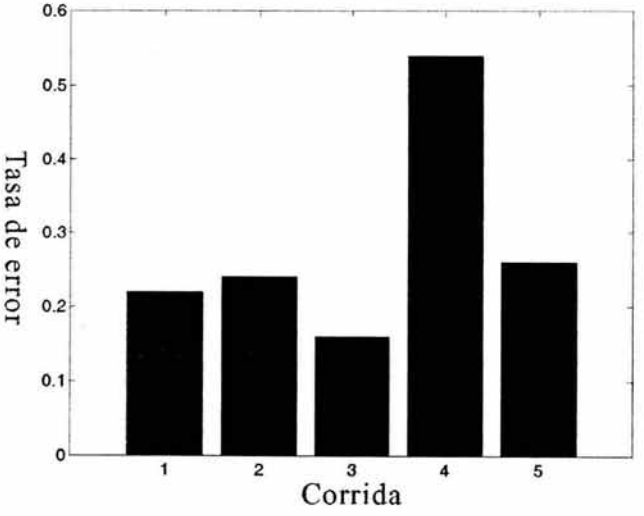


Figura 5.3: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 1.

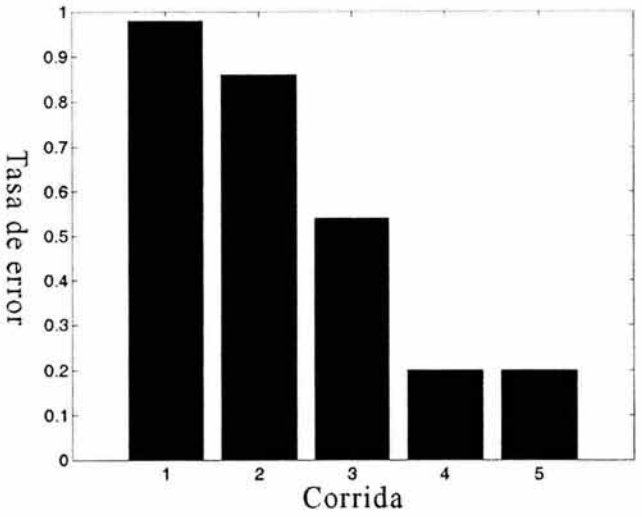


Figura 5.4: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 2.

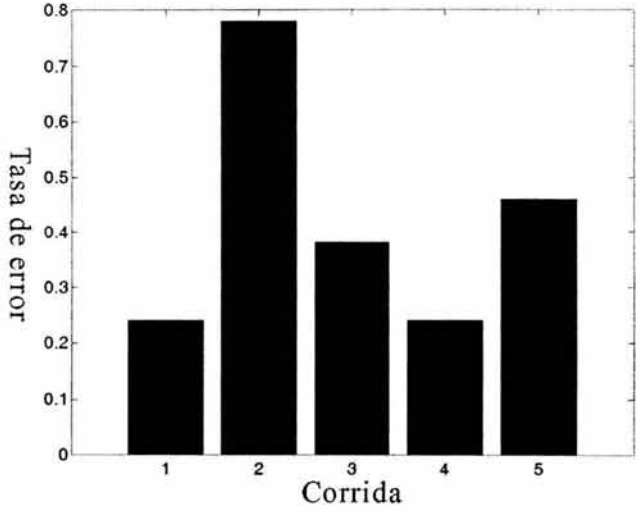


Figura 5.5: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 3.

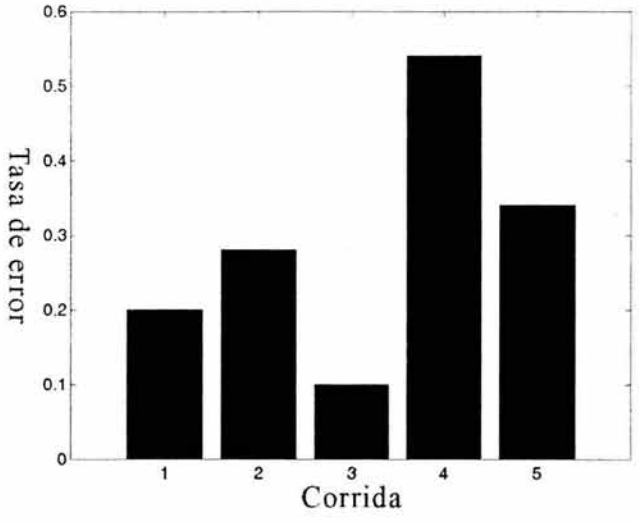


Figura 5.6: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 4.

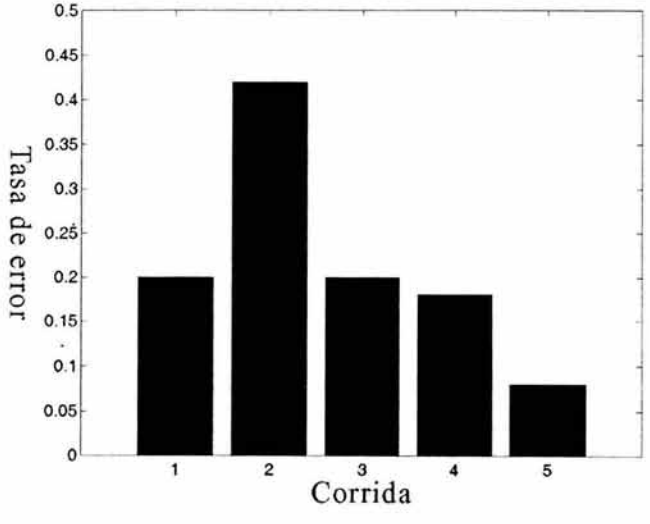


Figura 5.7: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 5.

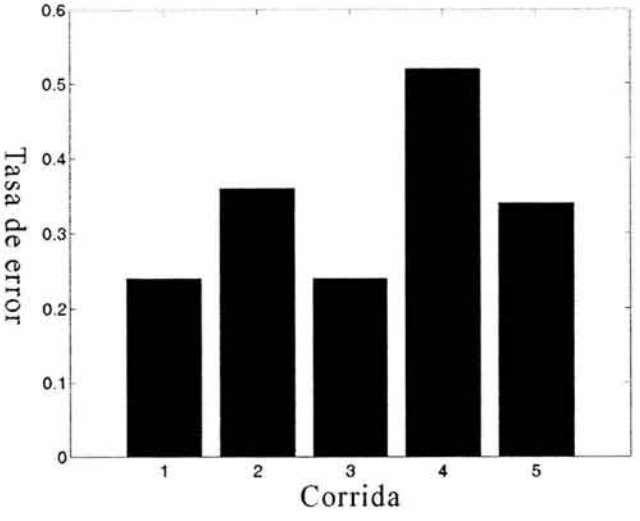


Figura 5.8: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 6.

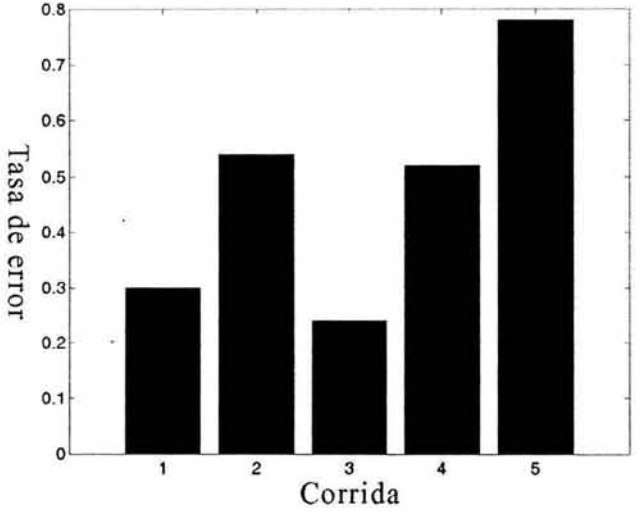


Figura 5.9: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 7.

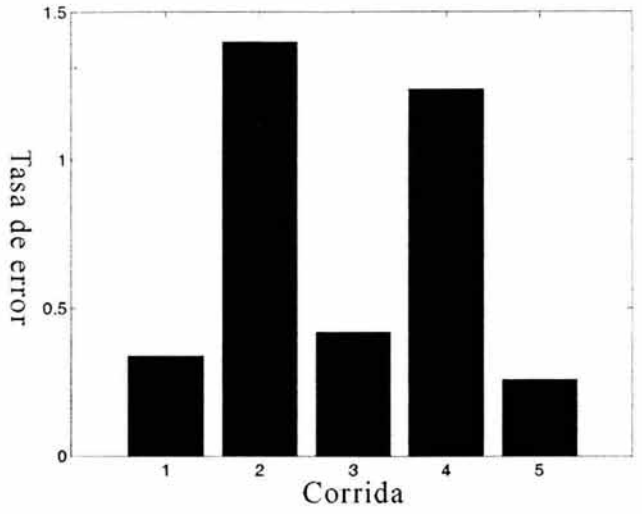


Figura 5.10: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 8.

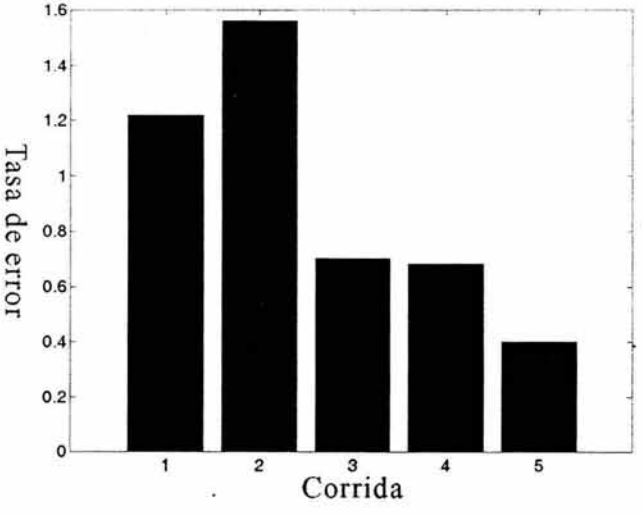


Figura 5.11: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal especializada en el reconocimiento del dígito 9.

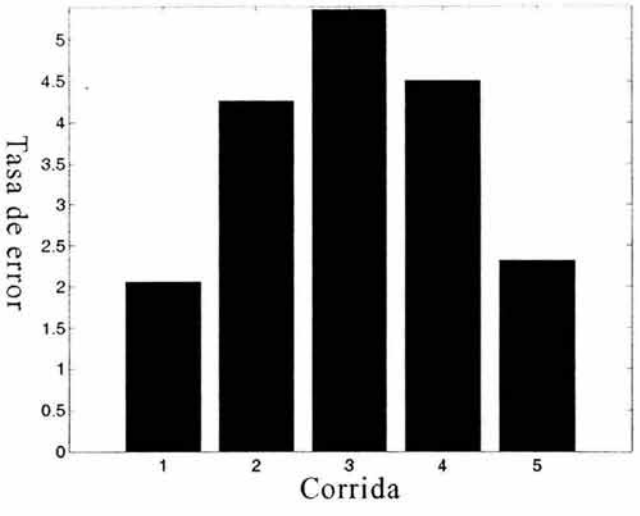


Figura 5.12: Tasa de error sobre el conjunto de validación para las cinco corridas de entrenamiento de la red neuronal general.

de error entre la red con la tasa más alta y la red con la tasa más baja es de 3.3%. No obstante, el entrenamiento de este modelo también produce resultados similares entre sí.

A partir de estas gráficas se pueden obtener datos estadísticos relevantes para analizar el comportamiento de los modelos. En las tablas 5.1 a 5.11 se muestran los resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por cada una de las redes neuronales convolucionales. Estos resultados incluyen las tasas de reconocimiento, error y rechazo, así como el número de ciclos de entrenamiento requeridos para hallar el conjunto de pesos óptimo de la red. Además, se resalta en color gris la corrida de entrenamiento que dio la red con mayor desempeño y en letras negritas la corrida de entrenamiento que produjo la red con menor desempeño.

Como se puede ver en la tabla 5.1, en cada una de las corridas de entrenamiento se producen redes neuronales convolucionales para el dígito 0 cuyo desempeño es muy similar. Por lo tanto, en caso de que se entrenara una sola red es de esperarse que dicha red tenga un desempeño muy cercano al desempeño promedio mostrado. Por otra parte, el desempeño de la red es excelente puesto que la tasa de error es bastante baja y la tasa de reconocimiento es muy alta.

Para el dígito 0, la red neuronal convolucional seleccionada como la mejor

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	**98.960**	**1.040**	**0.000**	**1**
2	99.720	0.280	0.000	3
3	99.460	0.500	0.040	1
4	**99.360**	**0.100**	**0.540**	**4**
5	99.740	0.220	0.040	6
Media	99.448	0.428	0.124	3

Tabla 5.1: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 0.

es la correspondiente a la corrida de entrenamiento 4. Como se puede ver en la figura 5.13 para entrenar a esta red se requirieron 4 ciclos de entrenamiento.

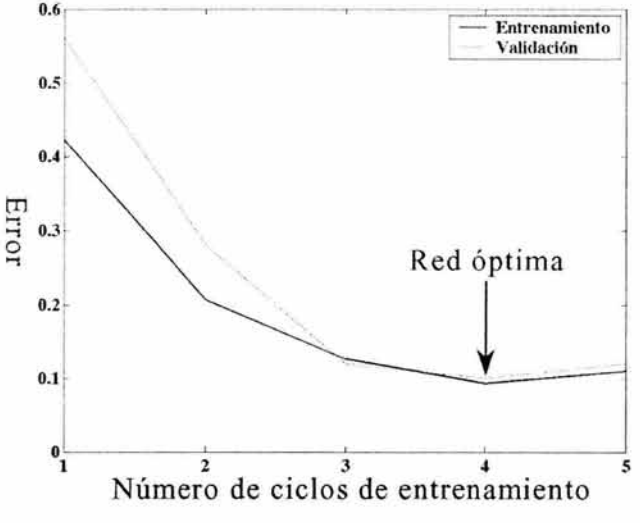


Figura 5.13: Para entrenar a la red neuronal convolucional especializada en el dígito 0 se requirieron 4 ciclos de entrenamiento.

Los resultados obtenidos por la red neuronal especializada en el dígito 1 se muestran en la tabla 5.2. Estos resultados son de los mejores en varios aspectos. En primer lugar, el entrenamiento produce redes con un desempeño similar. Gracias a esto, se garantiza que el entrenamiento producirá

una red con tasas bastante cercanas a las tasas promedio, incluso cuando el entrenamiento se detiene al finalizar una época. En segundo lugar, la tasa de reconocimiento para la red en todas las corridas de entrenamiento se mantiene por arriba del 99%. En tercer lugar, la tasa de error de la red obtenida en la mejor corrida es muy baja. Finalmente, la tasa de reconocimiento de la red obtenida en la mejor corrida es la más alta de todas.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	99.780	0.220	0.000	6
2	99.760	0.240	0.000	5
3	**99.560**	**0.160**	**0.280**	**5**
4	**99.460**	**0.540**	**0.000**	**1**
5	99.740	0.260	0.000	4
Media	99.660	0.284	0.056	4.2

Tabla 5.2: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 1.

La gráfica de la figura 5.14 corresponde a la tercera corrida de entrenamiento de la red neuronal convolucional especializada en el dígito 1. En ella se muestra la forma en que se reduce la tasa de error hasta obtener la red óptima en un total de 5 ciclos.

La tabla 5.3 refleja los resultados obtenidos por la red neuronal convolucional especializada en el dígito 2. Debido a que los resultados son similares, la probabilidad de que una red alcance las tasas medias es alta. Para esta red, la tasa de error de la red obtenida en la mejor corrida de entrenamiento se mantiene baja aunque la tasa de reconocimiento se halla por debajo del 99%. Esto se debe a que la tasa de rechazo es mayor en comparación con las dos redes anteriores.

La red obtenida en la cuarta corrida de entrenamiento fue seleccionada como la mejor en el reconocimiento del dígito 2. Su gráfica de entrenamiento aparece en la figura 5.15. Dicha gráfica muestra que la cantidad de ciclos de entrenamiento requeridos fue de 4.

Los resultados de la red neuronal convolucional especializada en el dígito 3 son mostrados en la tabla 5.4. El proceso de entrenamiento entrega resultados similares en cada una de las corridas de entrenamiento. Por ende, las tasas

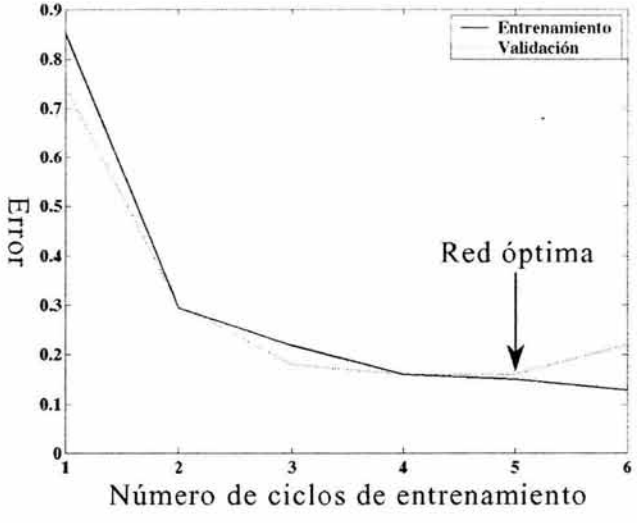


Figura 5.14: Para entrenar a la red neuronal convolucional especializada en el dígito 1 se requirieron 5 ciclos de entrenamiento.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	**99.020**	**0.980**	**0.000**	**1**
2	99.060	0.860	0.080	1
3	99.460	0.540	0.000	3
4	**98.520**	**0.200**	**1.280**	**4**
5	98.480	0.200	1.320	3
Media	98.908	0.556	0.536	2.4

Tabla 5.3: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 2.

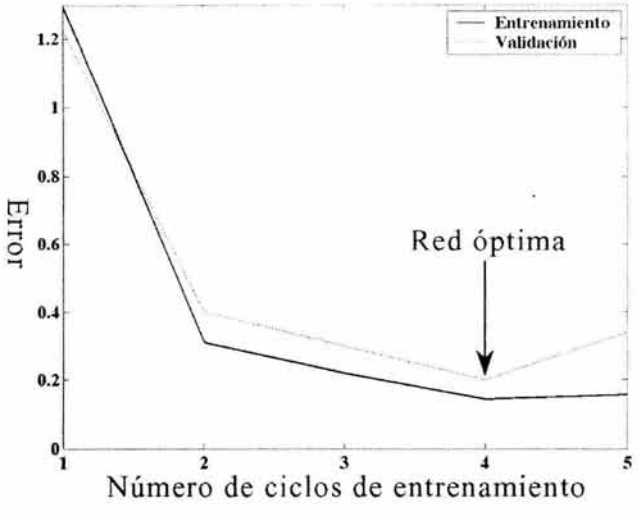


Figura 5.15: Para entrenar a la red neuronal convolucional especializada en el dígito 2 se requirieron 4 ciclos de entrenamiento.

obtenidas por la red se hallan bastante cercanas a las tasas promedio. Por otra parte, el desempeño de la red es bastante aceptable tomando en cuenta que la tasa de error es muy baja y que la tasa de reconocimiento es grande.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	98.260	0.240	1.500	2
2	**99.200**	**0.780**	**0.020**	**1**
3	99.620	0.380	0.000	5
4	**99.160**	**0.240**	**0.600**	**5**
5	99.100	0.460	0.440	3
Media	99.068	0.420	0.512	3.2

Tabla 5.4: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 3.

Como se ilustra en la figura 5.16, para entrenar a la red encargada de reconocer el dígito 3 se necesitaron 5 ciclos de entrenamiento. La gráfica mostrada corresponde a la red obtenida en la cuarta corrida de entrenamien-

to, la cual fue seleccionada como la mejor.

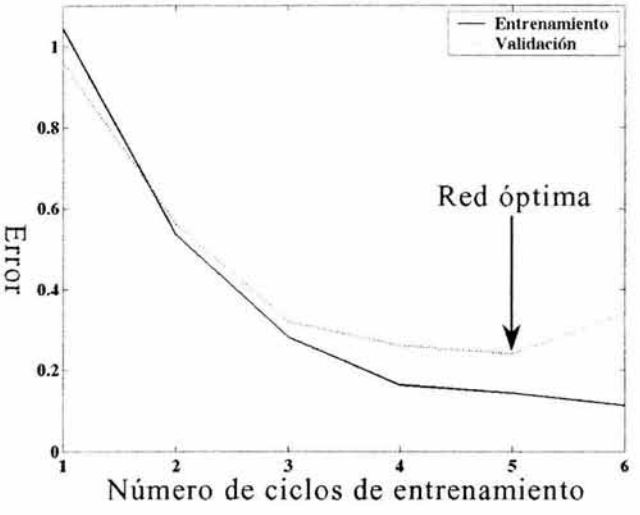


Figura 5.16: Para entrenar a la red neuronal convolucional especializada en el dígito 3 se requirieron 5 ciclos de entrenamiento.

En la tabla 5.5 aparecen los resultados de validación para las cinco corridas de entrenamiento de la red neuronal convolucional especializada en el dígito 4. A diferencia de las redes anteriores, el entrenamiento de esta red produce redes con una tasa de rechazo mayor, con lo cual se afecta también la tasa de reconocimiento. Así, aunque la red generada en la corrida 5 tiene una tasa de error baja, la tasa de rechazo es alta. Como resultado, la tasa de reconocimiento se reduce. De esta forma, la mejor corrida tiene una tasa de reconocimiento ligeramente baja aun cuando la tasa de error es muy baja.

Para el dígito 4 se seleccionó la red generada en la corrida de entrenamiento 3. Como se puede ver en la figura 5.17, esta red se entrenó en 2 ciclos de entrenamiento.

Las tasas desplegadas en la tabla 5.6 muestran que el entrenamiento de la red neuronal convolucional especializada en el dígito 5 genera redes con un desempeño muy cercano al promedio. La tasa de error de la red obtenida en la mejor corrida de entrenamiento es la menor de todas. Sin embargo, la tasa de reconocimiento se reduce por el ligero incremento en la tasa de rechazo.

En la figura 5.18 aparece la gráfica de entrenamiento de la red neuronal convolucional para el reconocimiento del dígito 5. Esta gráfica corresponde

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	99.180	0.200	0.620	5
2	99.720	0.280	0.000	4
3	**98.300**	**0.100**	**1.600**	**2**
4	**98.680**	**0.540**	**0.780**	**1**
5	96.860	0.340	2.800	1
Media	98.548	0.292	1.160	2.6

Tabla 5.5: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 4.

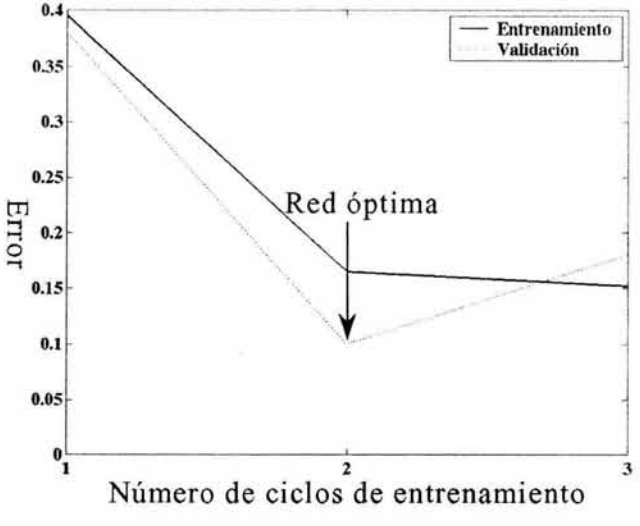


Figura 5.17: Para entrenar a la red neuronal convolucional especializada en el dígito 4 se requirieron 2 ciclos de entrenamiento.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	99.040	0.200	0.760	3
2	**99.580**	**0.420**	**0.000**	**4**
3	99.000	0.200	0.800	3
4	99.260	0.180	0.560	4
5	**98.340**	**0.080**	**1.580**	**4**
Media	99.044	0.216	0.740	3.6

Tabla 5.6: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 5.

a la red obtenida en la quinta corrida de entrenamiento. Dicha red requirió de 4 ciclos de entrenamiento.

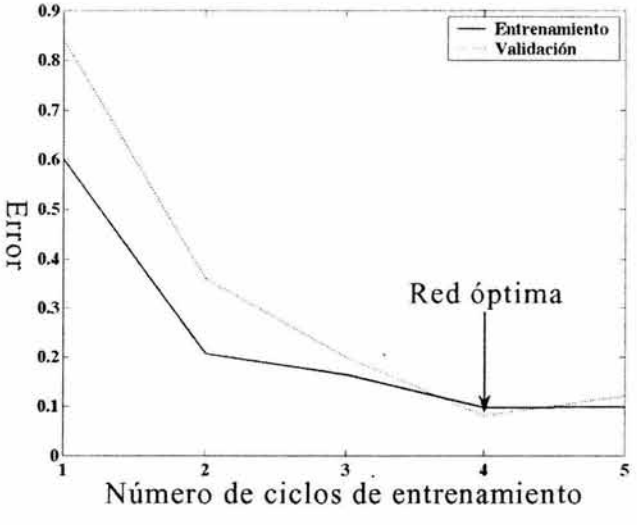


Figura 5.18: Para entrenar a la red neuronal convolucional especializada en el dígito 5 se requirieron 4 ciclos de entrenamiento.

Para el dígito 6 se obtuvieron redes neuronales convolucionales con resultados similares en las diferentes corridas de entrenamiento. Esto se puede apreciar en la tabla 5.7. La misma tabla muestra que el desempeño de la red obtenida en la mejor corrida de entrenamiento es bastante satisfactorio ya

que se tiene una tasa de error baja y una tasa de reconocimiento alta.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	98.840	0.240	0.920	3
2	99.640	0.360	0.000	6
3	**99.220**	**0.240**	**0.540**	**5**
4	**99.480**	**0.520**	**0.000**	**2**
5	99.640	0.340	0.020	5
Media	99.364	0.340	0.296	4.2

Tabla 5.7: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 6.

La gráfica de entrenamiento de la figura 5.19 corresponde a la red obtenida en la tercera corrida de entrenamiento. Dicha red está especializada en el reconocimiento del dígito 6 y su entrenamiento requirió de 5 ciclos.

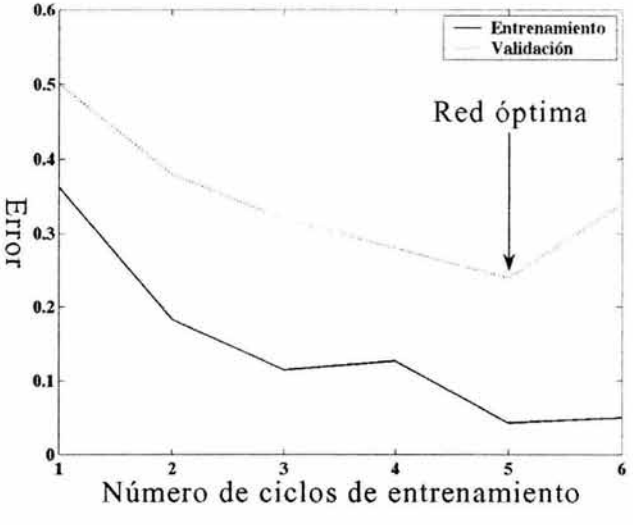


Figura 5.19: Para entrenar a la red neuronal convolucional especializada en el dígito 6 se requirieron 5 ciclos de entrenamiento.

Para la red neuronal convolucional del dígito 7 la tabla 5.8 muestra que los resultados obtenidos después del entrenamiento son similares. Además,

los resultados de la red producida en la mejor corrida de entrenamiento son bastante aceptables tanto en términos de tasa de error como en términos de tasa de reconocimiento.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	98.080	0.300	1.620	3
2	99.460	0.540	0.000	5
3	**99.280**	**0.240**	**0.480**	**7**
4	99.480	0.520	0.000	5
5	**98.940**	**0.780**	**0.280**	**2**
Media	99.048	0.476	0.476	4.4

Tabla 5.8: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 7.

Como se puede apreciar en la figura 5.20 el entrenamiento de la mejor red neuronal convolucional especializada en el dígito 7 tomó 7 ciclos. Esta gráfica corresponde a la red de la tercera corrida de entrenamiento.

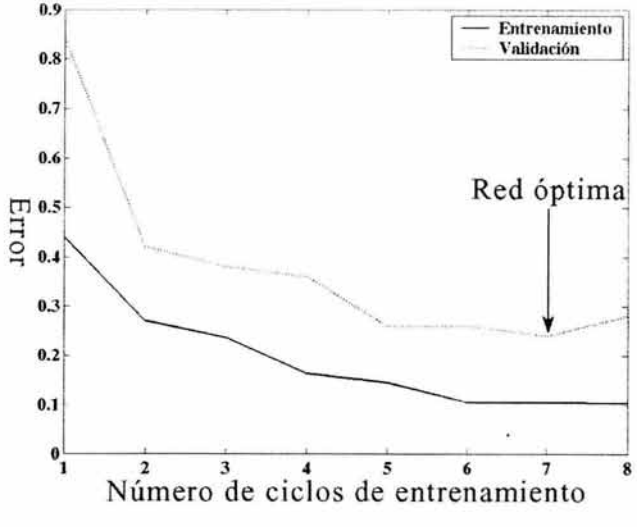


Figura 5.20: Para entrenar a la red neuronal convolucional especializada en el dígito 7 se requirieron 7 ciclos de entrenamiento.

Los resultados obtenidos por la red neuronal convolucional especializada en el dígito 8 se muestran en la tabla 5.9. Los resultados de la red obtenida en la mejor corrida son muy satisfactorios en términos de tasa de error. A pesar de ello, la tasa de rechazo conduce a un resultado ligeramente bajo en términos de tasa de reconocimiento.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	99.660	0.340	0.000	7
2	**98.600**	**1.400**	**0.000**	**1**
3	99.580	0.420	0.000	4
4	98.760	1.240	0.000	1
5	**97.960**	**0.260**	**1.780**	**4**
Media	98.912	0.732	0.356	3.4

Tabla 5.9: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 8.

En la figura 5.21 se muestra la gráfica de entrenamiento para la red obtenida en la quinta corrida de entrenamiento. Esta red especializada en el dígito 8 requirió de 4 ciclos de entrenamiento para convergir.

La red neuronal convolucional para el dígito 9 tiene el menor desempeño de entre todas las redes especializadas como se puede ver en la tabla 5.10. Para empezar, los valores de la tasa de reconocimiento y la tasa de rechazo se hallan más dispersos. Así, la tasa de rechazo es de 0% en ciertas corridas y de 3.44% en otra. Esto provoca que la tasa de reconocimiento fluctúe de forma similar. Como resultado, no es tan probable que el entrenamiento de una red alcance el desempeño promedio. Por otro lado, la tasa de rechazo de la red generada por la mejor corrida de entrenamiento es la mayor de las tasas de rechazo de todas las redes con lo cual se reduce ligeramente la tasa de reconocimiento. No obstante, la red producida en esta corrida de entrenamiento fue elegida como la mejor porque la tasa de error es baja aunque mayor a la tasa de error de las otras redes especializadas.

La quinta corrida de entrenamiento generó la red seleccionada para el reconocimiento del dígito 9. Este entrenamiento requirió de un único ciclo, lo cual se puede observar en la figura 5.22.

Finalmente, la tabla 5.11 muestra los resultados obtenidos en las cinco

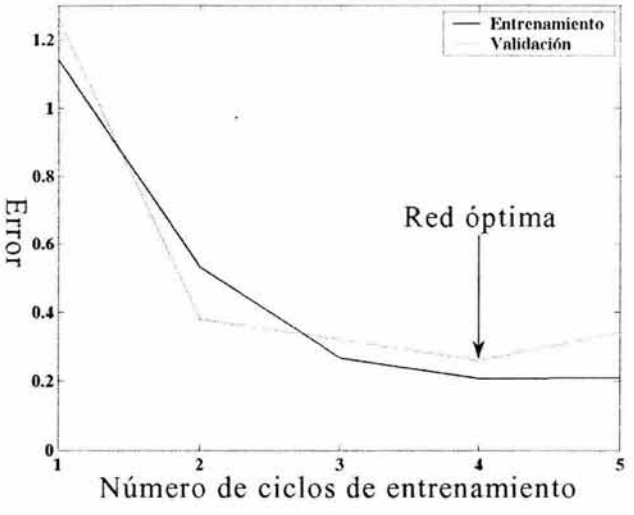


Figura 5.21: Para entrenar a la red neuronal convolucional especializada en el dígito 8 se requirieron 4 ciclos de entrenamiento.

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	98.780	1.220	0.000	1
2	**98.440**	**1.560**	**0.000**	**1**
3	99.300	0.700	0.000	5
4	99.300	0.680	0.020	9
5	**96.160**	**0.400**	**3.440**	**1**
Media	98.396	0.912	0.692	3.4

Tabla 5.10: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional entrenada para reconocer el dígito 9.

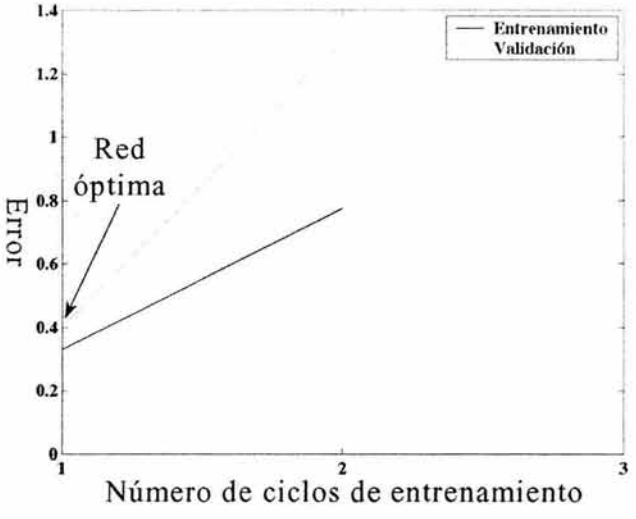


Figura 5.22: Para entrenar a la red neuronal convolucional especializada en el dígito 9 se requirió 1 sólo ciclo de entrenamiento.

corridas de entrenamiento por la red neuronal convolucional general. Como se puede apreciar, la fluctuación de la tasa de reconocimiento es mayor que en las demás redes. No obstante, ahora esta variación es provocada por la tasa de error. Así, las tasas de error resultan mayores en esta red, mientras que las tasas de rechazo son menores. Esta es la principal diferencia con respecto a las redes neuronales especializadas, las cuales tienen tasas de error bajas. Esto se refleja en tasas de reconocimiento menores para la red neuronal convolucional general en relación a las redes neuronales especializadas.

El entrenamiento de la mejor red neuronal convolucional general se llevó a cabo en 6 ciclos. Esta red, cuya gráfica de entrenamiento se presenta en la figura 5.23, se generó en la primera corrida de entrenamiento.

Los datos estadísticos mostrados en las tablas anteriores, sirven para tener una idea de qué desempeño se puede esperar si únicamente se entrena una red neuronal [77]. Así, la media indica cual es el resultado promedio que garantiza el modelo.

El criterio empleado para seleccionar a la mejor red de entre las diferentes corridas de entrenamiento es el siguiente:

1. La red con menor tasa de error se selecciona como la mejor.

2. En caso de que haya más de una red con la misma tasa de error, se

Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
1	**97.280**	**2.060**	**0.660**	**6**
2	93.980	4.260	1.760	1
3	**93.400**	**5.360**	**1.240**	**1**
4	93.980	4.500	1.520	1
5	97.040	2.320	0.640	7
Media	95.136	3.700	1.164	3.2

Tabla 5.11: Resultados sobre el conjunto de validación obtenidos en cada corrida de entrenamiento por la red neuronal convolucional general.

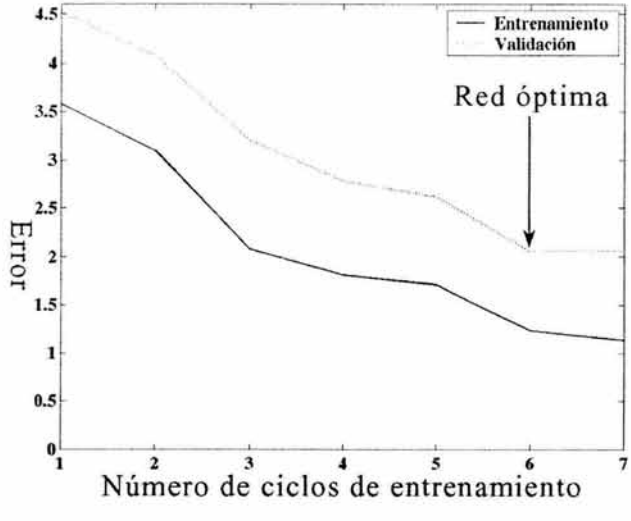


Figura 5.23: Para entrenar a la red neuronal convolucional general se requirieron 6 ciclos de entrenamiento.

selecciona la red con menor tasa de rechazo.

3. En caso de que dos o más redes compartan la misma tasa de rechazo, se selecciona la red que haya empleado el mayor número de ciclos en su entrenamiento.

4. Si todavía existen dos o más redes iguales, se escoge cualquiera al azar.

La tabla 5.12 reúne la información correspondiente a las mejores redes incluyendo tanto a las redes neuronales convolucionales específicas, como a la red neuronal convolucional general. Como se puede ver, la mayor tasa de error corresponde a la red neuronal convolucional general. Además, esta red tiene una tasa de reconocimiento baja que sólo es superior a la tasa de reconocimiento de la red neuronal convolucional especializada en el dígito 9. Por su puesto, las tasas de reconocimiento bajas en las redes neuronales convolucionales especializadas se deben al incremento en la tasa de rechazo y no a errores de clasificación.

Red	Corrida	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Épocas requeridas
0	4	99.36	0.10	0.54	4
1	3	99.56	0.16	0.28	5
2	4	98.52	0.20	1.28	4
3	4	99.16	0.24	0.60	5
4	3	98.30	0.10	1.60	2
5	5	98.34	0.08	1.58	4
6	3	99.22	0.24	0.54	5
7	3	99.28	0.24	0.48	7
8	5	97.96	0.26	1.78	4
9	5	96.16	0.40	3.44	1
General	1	97.28	2.06	0.66	6

Tabla 5.12: Resultados sobre el conjunto de validación de las mejores corridas de entrenamiento de cada red neuronal convolucional.

Con fines comparativos, la figura 5.24 muestra el número de ciclos de entrenamiento requeridos por cada una de las redes neuronales convolucionales específicas y por la red neuronal convolucional general. Como se puede ver, las diferencias no son grandes. Lo más notable es que la red neuronal convolucional del dígito 9 se entrenó en sólo un ciclo.

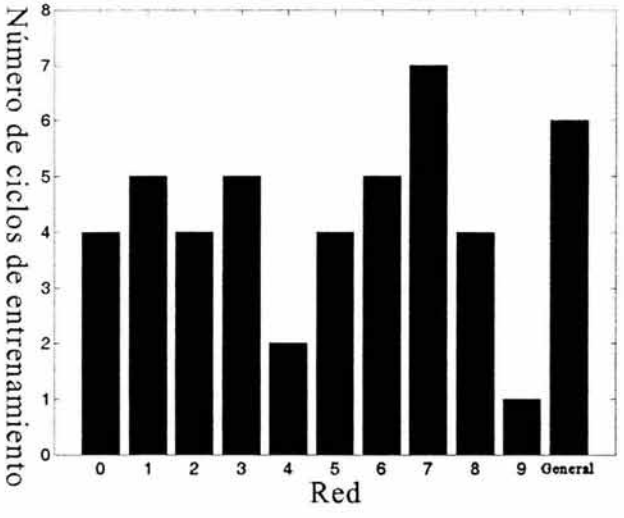


Figura 5.24: Ciclos de entrenamiento requeridos para obtener la mejor red de cada una de las redes neuronales convolucionales.

Una vez determinada la mejor red para cada una de las redes neuronales convolucionales específicas, dichas redes se evaluaron con el conjunto de prueba. La tabla 5.13 muestra un concentrado de los resultados obtenidos por cada una de estas redes. Como se puede apreciar, las tasas de error son bastante bajas, siendo incluso de 0% para el dígito 5. Además, las tasas de reconocimiento son bastante altas aunque el desempeño de las redes para el dígito 8 y 9 fue un poco menor. Aún así, los resultados de estas dos redes son bastante buenos considerando que tuvieron una tasa de error mayor con el conjunto de validación.

Por su parte, debido a que la tasa de error es baja, la tasa de fiabilidad se mantiene por arriba del 99.5% para todas las redes especializadas. Por supuesto, la tasa de fiabilidad es de 100% para la red del dígito 5. Ahora bien, la tasa de aceptabilidad también es alta aunque un poco menos para las redes de los dígitos 8 y 9 debido a que tienen tasas de rechazo mayores.

En la parte inferior de la tabla 5.13 aparece el promedio de las tasas. A partir de este promedio se puede ver que las redes neuronales convolucionales especializadas tienen un alto desempeño. Por un lado, la tasa media de reconocimiento es excelente, y en combinación con una tasa media de error sumamente baja, conduce a una tasa de confiabilidad muy cercana al 100%. Por otro lado, la tasa de aceptabilidad es muy buena también gracias a que

Red	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Tasa de fiabilidad	Tasa de aceptabilidad
0	99.56	0.14	0.30	99.860	99.420
1	99.84	0.06	0.10	99.940	99.780
2	99.10	0.10	0.80	99.899	99.000
3	99.52	0.22	0.26	99.779	99.300
4	99.08	0.12	0.80	99.879	98.960
5	99.52	0.00	0.48	100	99.520
6	99.76	0.06	0.18	99.940	99.700
7	99.64	0.04	0.32	99.960	99.600
8	98.70	0.12	1.18	99.879	98.580
9	98.26	0.32	1.42	99.675	97.941
Promedio	**99.30**	**0.12**	**0.58**	**99.881**	**99.180**

Tabla 5.13: Resultados sobre el conjunto de prueba para cada una de las redes neuronales convolucionales especializadas.

la tasa media de rechazo no es tan grande.

Finalmente, se llevó a cabo la evaluación de la red neuronal convolucional modular integrada. Los resultados obtenidos se muestran en la matriz de confusión de la figura 5.25. Como se puede ver, la cantidad de errores es de solamente 15. No obstante, el número de rechazos es elevado: 115.

		Reconocidos como											
		0	1	2	3	4	5	6	7	8	9	Rechazados	Incorrectos
Dígito verdadero	0	503	0	0	0	0	0	1	0	0	1	15	2
	1	0	557	0	0	0	0	1	0	1	0	5	2
	2	0	1	492	0	0	0	0	1	0	0	8	2
	3	0	0	0	496	0	0	0	0	1	0	13	1
	4	0	0	0	0	470	0	0	0	1	0	11	1
	5	0	0	0	1	0	427	1	0	0	0	7	2
	6	0	0	0	0	0	1	491	0	0	0	4	1
	7	0	0	3	0	0	0	0	499	0	0	14	3
	8	0	0	1	0	0	0	0	0	471	0	13	1
	9	0	0	0	0	0	0	0	0	0	464	25	0
	Errores	0	1	4	1	0	1	3	1	3	1	115	15

Figura 5.25: Matriz de confusión de la red neuronal convolucional modular.

La mayoría de los errores se debe a confusiones por la similitud de la

forma. Por ejemplo, el 0 se confundió en una ocasión con el 6 y en otra con el 9; un 2 se confundió con un 7 y tres imágenes del 7 con un 2. Resulta interesante notar que la red para el dígito 9 no cometió error alguno, pero fue la que rechazo más dígitos: 25. El menor número de rechazos fue de 4 para el dígito 6. La mayor cantidad de errores fue de sólo 3 para el dígito 7.

La tabla 5.14 muestra además todas las tasas calculadas para la red neuronal convolucional modular. Como se puede ver, los resultados para cada carácter son inferiores a los obtenidos por las redes neuronales de manera aislada. El resultado global muestra que la red es confiable debido a que su tasa de error es baja. No obstante, la tasa de reconocimiento se ve afectada por el incremento en la tasa de rechazo, y por ende, la tasa de aceptabilidad se reduce ligeramente.

Carácter	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Tasa de fiabilidad	Tasa de aceptabilidad
0	96.731	0.385	2.885	99.604	96.348
1	98.759	0.355	0.887	99.642	98.406
2	98.008	0.398	1.594	99.595	97.611
3	97.255	0.196	2.549	99.799	97.059
4	97.510	0.207	2.282	99.788	97.303
5	97.936	0.459	1.606	99.534	97.479
6	98.992	0.202	0.806	99.797	98.791
7	96.705	0.581	2.713	99.402	96.128
8	97.113	0.206	2.680	99.788	96.908
9	94.888	0.000	5.112	100	94.888
Total	**97.400**	**0.300**	**2.300**	**99.693**	**97.101**

Tabla 5.14: Resultados sobre el conjunto de prueba para la red neuronal convolucional modular.

En comparación, los resultados obtenidos por la red neuronal convolucional general se muestran en la matriz de confusión de la figura 5.26. La situación aquí se invierte, pues el número de caracteres incorrectos sube a 48, mientras que la cantidad de caracteres rechazados se reduce a sólo 19.

Nuevamente, muchos de los errores obedecen a confusiones debidas a la figura. El 0 por ejemplo, se confundió en dos ocasiones con el 6 y en otras dos con el 8; el 2 se confundió dos veces con el 7, y el 7 cuatro veces con el 2; el 4 se confundió tres veces con el 9. El dígito 9 figura nuevamente con el mayor número de rechazos (7) y el menor número de errores (2). Para los dígitos 1 y 3 el número de rechazos es de cero, pero el número de errores es el mayor: 7.

		Reconocidos como											
		0	1	2	3	4	5	6	7	8	9	Rechazados	Incorrectos
Dígito verdadero	0	513	0	0	0	1	0	2	1	2	0	1	6
	1	0	557	0	0	1	0	2	0	4	0	0	7
	2	0	1	495	0	0	0	0	2	2	0	3	4
	3	0	0	2	503	0	0	0	1	2	2	0	7
	4	0	0	0	0	477	0	0	1	0	3	1	4
	5	1	0	0	1	0	429	2	0	1	0	2	5
	6	0	0	2	0	0	0	492	0	0	0	2	2
	7	0	1	4	0	0	0	0	510	0	0	1	5
	8	0	0	2	0	0	2	0	2	477	0	2	6
	9	1	0	0	0	0	0	0	1	0	480	7	2
	Errores	2	1	10	1	2	2	6	8	11	5	19	48

Figura 5.26: Matriz de confusión de la red neuronal convolucional general.

La tabla 5.15 incluye las tasas de desempeño obtenidas de estos resultados. Como se puede apreciar, la tasa de error es mayor en el modelo general, y por lo tanto el modelo modular resulta superior en términos de fiabilidad. No obstante, la tasa de reconocimiento es inferior en el modelo modular, puesto que la tasa de rechazo es más grande que en el modelo general. Como resultado, la tasa de aceptabilidad del modelo general es ligeramente superior.

Carácter	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Tasa de fiabilidad	Tasa de aceptabilidad
0	98.654	1.154	0.192	98.844	97.513
1	98.759	1.241	0.000	98.759	97.533
2	98.606	0.797	0.598	99.198	97.815
3	98.627	1.373	0.000	98.627	97.274
4	98.963	0.830	0.207	99.168	98.140
5	98.395	1.147	0.459	98.848	97.262
6	99.194	0.403	0.403	99.595	98.792
7	98.837	0.969	0.194	99.029	97.878
8	98.351	1.237	0.412	98.758	97.129
9	98.160	0.409	1.431	99.585	97.752
Total	**98.660**	**0.960**	**0.380**	**99.036**	**97.709**

Tabla 5.15: Resultados sobre el conjunto de prueba para la red neuronal convolucional general.

Los resultados tanto de la red neuronal convolucional modular como de la red neuronal convolucional general se muestran en la tabla 5.16. Esta tabla permite comparar fácilmente los dos modelos de red neuronal convolucional, mostrando que ambos modelos presentan resultados similares. Como se puede apreciar, el modelo modular reduce la tasa de errores en un 68.75%.

Red	Tasa de reconocimiento	Tasa de error	Tasa de rechazo	Tasa de fiabilidad	Tasa de aceptabilidad
Modular	97.400	0.300	2.300	99.693	97.101
General	98.660	0.960	0.380	99.036	97.709

Tabla 5.16: Resultados sobre el conjunto de prueba para la red neuronal convolucional modular y para la red neuronal convolucional general.

5.3 Conclusiones

El principal objetivo de esta tesis es comprobar si las hipótesis planteadas son respaldadas o no. Para ello se diseñaron los modelos a evaluar, y se prepararon y aplicaron las pruebas pertinentes sobre dichos modelos. Así, en esta sección se discuten los resultados relacionados con la hipótesis H_1.

Los resultados mostrados en la sección anterior apoyan la hipótesis debido a que el modelo de red neuronal convolucional modular obtuvo una tasa de error menor a la tasa de reconocimiento del modelo de red neuronal convolucional general. La tasa de error de la red neuronal convolucional general es 3.2 veces mayor que la tasa de error de la red neuronal convolucional modular.

Por otro lado, el modelo de red neuronal convolucional modular obtuvo una tasa de reconocimiento de caracteres ligeramente menor a la tasa de reconocimiento del modelo de red neuronal convolucional general. No obstante, esto se debe a que la tasa de rechazo de la red neuronal convolucional modular es seis veces mayor a la tasa de rechazo de la red neuronal convolucional general.

Ahora bien, el resultado promedio obtenido por las redes neuronales convolucionales especializadas operando de manera aislada sobre el conjunto de prueba es superior en todos los aspectos a los resultados obtenidos por la red neuronal convolucional general. En primer lugar, la tasa media de reconocimiento es del 99.3% contra el 98.66% obtenido por la red neuronal

convolucional general. En segundo lugar, la tasa de error de la red neuronal convolucional general es 8 veces más grande que la tasa de error promedio de las redes neuronales convolucionales especializadas. En tercer lugar, debido a la baja tasa de error, las redes neuronales convolucionales especializadas alcanzan una tasa de fiabilidad promedio mayor que la tasa de fiabilidad de la red neuronal convolucional general. Finalmente, a pesar de que la tasa de rechazo media es ligeramente superior en las redes neuronales convolucionales especializadas, la tasa de aceptabilidad de las redes neuronales convolucionales especializadas es mayor.

Los resultados obtenidos por las redes neuronales convolucionales especializadas muestran que la tasa de reconocimiento de caracteres se mejora al hacer que una red maneje una cantidad menor de clases. La reducción de esta tasa a su vez provoca un incremento de la tasa de rechazo. Al integrar estas redes neuronales convolucionales especializadas en un sólo modelo modular, la tasa de rechazo se incrementa un poco más. Por lo tanto, se puede afirmar que este modelo implica el incremento de la tasa de rechazo.

De las observaciones anteriores, se puede concluir que la aplicación de la modularidad sobre las redes neuronales convolucionales mejora su desempeño al reducir la tasa de error. No obstante, se pueden efectuar más experimentos en los cuales se consideren diferentes variables a modificar. A continuación, se proporciona una serie de investigaciones recomendadas como trabajo futuro para poder determinar con mayor certeza si el modelo modular aplicado a las redes neuronales convolucionales es realmente superior a las redes neuronales convolucionales convencionales.

Para empezar, se podría investigar cuál es el efecto de emplear distintos criterios de convergencia en el entrenamiento de las redes. Algunos criterios que se podrían investigar incluyen la validación cruzada con base en la tasa de reconocimiento y el entrenamiento con un número fijo de ciclos como en [51]. Además, se podría modificar el criterio de selección de la mejor corrida para basarlo en la tasa de reconocimiento. La modificación del criterio permitiría averiguar si es posible reducir la tasa de rechazo sin incrementar la tasa de error.

El uso de diferentes criterios de decisión de clases también proporciona un gran número de experimentos posibles. Se puede comenzar con un criterio basado en la regla "el ganador lo toma todo". También se podría experimentar con el empleo de umbrales variables. Por ejemplo, tomando las dos clases con menor penalización, el umbral para decidir cuál clase es la ganadora podría basarse en la similitud gráfica. De esta forma, para clases gráficamente

parecidas el umbral de separación debería ser mayor que para clases de figura diferente.

Otro aspecto que se puede someter a experimentación es el cambio de los parámetros de aprendizaje. Asimismo, la arquitectura convolucional podría variarse con el propósito de ajustarla a cada clase, en vez de emplear exactamente la misma arquitectura. Estas variaciones incluyen el cambio del número de capas, el número de mapas de características en cada capa, el tamaño del campo receptivo y la forma de combinar los mapas de características.

El tamaño de la base de datos puede ser otro factor a investigar. De esta manera, se podría comparar cómo se afecta el desempeño de ambos modelos dependiendo del tamaño de la base de datos. Es posible que el modelo modular sea más robusto a reducciones en el tamaño de la base de datos gracias a que la tarea que debe aprender una red especializada es menor que la labor que debe aprender la red general.

Varios autores han mencionado que las ventajas de la arquitectura modular se incrementan conforme el número de clases aumenta. Por esta razón, puede ser muy interesante analizar cuál es el comportamiento de este modelo modular en comparación con el modelo general para conjuntos con distinto número de clases.

Finalmente, sería recomendable llevar a cabo experimentos con imágenes distorsionadas y con ruido para averiguar si el modelo modular es más resistente a las transformaciones y el ruido que el modelo general. Para probar esto sería necesario efectuar evaluaciones controladas de los modelos con diferentes niveles de ruido y con distintas transformaciones, así como combinaciones de estas. Como resultado se podrían obtener los límites de resistencia de los modelos para cada tipo de distorsión y para diferentes niveles de ruido.

Capítulo 6

Corrección Ortográfica

El proceso de reconocimiento óptico de caracteres convierte un texto impreso en un archivo electrónico. Existen diversos factores por los que pueden presentarse errores de reconocimiento, mismos que deben corregirse en un proceso posterior con el fin de evitar que incidan negativamente en su uso posterior, por ejemplo en la reproducción acústica del texto mediante un conversor de texto en habla. Como se mencionó en el capítulo uno de esta tesis, para enmendar los errores provenientes del reconocedor óptico se han propuesto procesos de corrección ortográfica de palabras aisladas principalmente. Éstos atienden sólo a la estructura de las palabras, ya sea mediante una búsqueda en diccionarios o realizando un análisis de las letras que las constituyen [6]. Sin embargo, los resultados logrados con ellos no son muy satisfactorios, lo que hace necesario que se lleve a cabo una revisión que incorpore un mayor conocimiento lingüístico.

El objetivo de hacer una revisión lingüística en textos electrónicos es aumentar la efectividad en el proceso de corrección para que el contenido del archivo electrónico refleje íntegramente el contenido del texto impreso. Por ello, se deben aplicar de manera conjunta técnicas que incorporen conocimiento ortográfico, es decir morfológico, y sintáctico. El conocimiento morfológico se refiere específicamente a la forma de las palabras del texto reconocido; el conocimiento sintáctico tiene que ver con la función de las palabras dentro del sintagma analizado en el proceso de revisión.

Así, combinando los dos conocimientos puede crearse un mecanismo de procesamiento de lenguaje natural (PLN) que identifique una palabra incorrecta, y seleccione la palabra correcta que deba sustituirla, basándose en su forma y en la función que cumple dentro de la estructura lingüística a la que

pertenece.

En el presente capítulo se exponen algunos conceptos básicos que son de utilidad para entender el proceso de corrección de errores ortográficos en textos electrónicos. Se explican las tres etapas del proceso de corrección tradicional: detección, selección y sustitución. Además, se mencionan algunas características de los lexicones y las principales técnicas de corrección que los investigadores han desarrollado, como la distancia mínima de edición entre dos cadenas y los modelos de lenguaje basados en n-gramas. Finalmente se explica el proceso de etiquetamiento morfosintáctico automático el cual es usado en el modelo de corrección ortográfica propuesto en esta tesis y se exponen dos de las aplicaciones de etiquetamiento morfosintáctico automático que han tenido mejores resultados, los etiquetadores estocásticos y el etiquetador basado en el algoritmo de aprendizaje TBL.

6.1 Conceptos Básicos

Debido a que en este capítulo se trata el tema de la corrección ortográfica de errores, es relevante entender los conceptos de palabra y sintagma en primer término, puesto que es en el nivel de las palabras donde principalmente se producen los errores en los textos electrónicos (por ejemplo, los obtenidos mediante un proceso de reconocimiento óptico de caracteres) y es en sintagmas como se agrupan las palabras de un texto.

De acuerdo con la lingüística, se denominan **signos lingüísticos** o **palabras** a aquellos signos tanto auditivos como visuales que pueden **articularse**, es decir, relacionarse entre sí mediante ciertas normas de distribución, orden y dependencia, para formar cadenas o frases. Una palabra es una unidad resultante de asociar dos aspectos inseparables: el significante (los signos articulados que la forman) y el significado (la idea atribuida al significante) [8].

Un texto está formado por diferentes símbolos, entre los cuales se tienen los siguientes:

Las letras del alfabeto español:

A B C D E F G H I J K L M N Ñ O P Q R S T U V W X Y Z Á É Í Ó Ú Ü
a b c d e f g h i j k l m n ñ o p q r s t u v w x y z á é í ó ú ü

Los números arábigos:

0 1 2 3 4 5 6 7 8 9

Los números romanos:

I V X C M L D

Los signos ortográficos:

. , ; : "' ¡ ! ¿ ? ( ) { } [ ]

Además de aquellos otros símbolos que tienen un uso específico, por ejemplo:

$\%\$ + - <=> / \times$

Cabe señalar que dentro de un texto, una palabra está diferenciada por espacios en blanco [8]. Por ello, se consideran palabras todas aquellas secuencias de uno o más símbolos distintos al espacio en blanco.

Ahora bien, un **sintagma** es una cadena lineal horizontal de palabras articuladas. La importancia del sintagma es que dentro de él cada palabra adquiere su valor gramatical (como género, número, tamaño, aspecto, etc.) [8] y dicho valor puede ser utilizado en el proceso de corrección.

Para la presente tesis, es de principal interés la **gramática** porque es la disciplina lingüística que estudia las funciones y formas de las palabras. La forma de las palabras es estudiada por la morfología y la sintaxis estudia la función de éstas dentro del lenguaje [28] [8].

Por otro lado, uno de los conceptos más importantes en este trabajo es el de **paradigma**, el cual se define como una agrupación de elementos lingüísticos (palabras, fonemas, etc.). El criterio de agrupación puede basarse en la semejanza o asociación de ideas que cada elemento posee, como las de similitud o contraste [8],[19]. De esta manera, dos elementos lingüísticos a y a' pertenecen al mismo paradigma *si y sólo si* pueden sustituirse en un mismo sintagma de forma mutua. Esto es, si existen dos sintagmas válidos dentro de la lengua: bac y $ba'c$ [19] (Donde b y c son elementos lingüísticos también).

Una forma de clasificar las palabras de una lengua es dividirlas en **categorías gramaticales** también denominadas **partes del discurso** [55]. Para realizar esta clasificación se han utilizado criterios morfológicos, semánticos y sintácticos [28],[19]. Sin embargo, no puede hablarse de una clasificación única, ya que esos mismos criterios permiten realizar varias divisiones sin que haya un consenso en cuanto a cuál división de las palabras es la mejor [19].

Para cerrar esta sección de términos básicos, mencionaremos a continuación los conceptos de corpus y de lexicón, así como el de probabilidad, ya que son los elementos principales de este trabajo.

El **corpus lingüístico** es una colección de muestras (o realizaciones) de la lengua, tanto de voz como de texto [60],[64].

En el caso del corpus de texto, también puede contener información lingüística relacionada con las muestras, frecuentemente morfosintáctica [60]. Se llama **morfosintáxis** al estudio de las palabras que toma en cuenta las relaciones de éstas en el texto o discurso. Es decir, utiliza información morfológica y sintáctica. De esta forma, un corpus puede anotarse o etiquetarse con información acerca de las categorías gramaticales que tienen las palabras que lo forman [73]. En esta tesis se utilizará un corpus etiquetado de esta manera.

De forma general los corpus de texto presentan las siguientes características [73]:

- Representan datos reales, no inventados.
- Pueden realizarse diferentes estudios sobre un mismo corpus. Por ejemplo el análisis léxico referido a la búsqueda de colocaciones, esto es, contextos específicos y relaciones entre palabras dentro de un discurso, o bien el conteo de las apariciones de una palabra (o conjunto de palabras) dentro del corpus para determinar escalas de frecuencia.

De acuerdo con los objetivos de esta tesis, se pueden obtener los siguientes datos al utilizar corpus textuales:

1. Obtener la frecuencia de una determinada palabra en un texto o grupo de textos.
2. Encontrar los contextos específicos en los que aparece una palabra. Lo que nos da la información sobre el uso que ésta tiene.
3. Determinar a qué categoría gramatical pertenece una palabra cuando aparece en el corpus.

Una cuestión que se tiene que tomar en cuenta es qué tipo de textos se recopilan, debido a que esto incide directamente en el dominio en el que será aplicado el procesamiento lingüístico, por ello, dichos textos deben ser representativos de éste para obtener resultados significativos.

El **lexicón** o **diccionario** es una lista de palabras relevantes (o alguna otra unidad lingüística) dentro de un contexto de aplicación (y en algunos casos se incluye también información lingüística asociada a ellas). Según Kukich [47], es preferible el uso de la palabra lexicón puesto que se refiere a una lista de palabras, sintagmas o estructuras más complejas [74], que son relevantes para un tema, campo o clase en particular. Atendiendo a esto, se utilizará el término lexicón.

Ahora bien, cuando se habla de probabilidades de los elementos lingüísticos, se entiende el término **probabilidad** como el cociente entre la frecuencia de aparición de una unidad lingüística en el corpus utilizado y el número total de esas unidades. Esto se expresa en la ecuación 6.1:

$$P(e) = \frac{f(e)}{N} \tag{6.1}$$

Donde:

$P(e)$ probabilidad del elemento lingüístico e.

$f(e)$ frecuencia de aparición del elemento lingüístico e en el corpus.

N número total de elementos lingüísticos en el corpus.

Por último, se comenta que para obtener resultados aceptables en la corrección de errores debe combinarse la información probabilística con técnicas de búsqueda en lexicones, ya que el cálculo de probabilidades no es suficiente para realizar la detección y corrección de errores [47].

6.1.1 Sistemas de Procesamiento de Lenguaje Natural

El objetivo del procesamiento de lenguaje natural (PLN) es construir sistemas computacionales capaces de llevar a cabo operaciones útiles a partir de una entrada en lenguaje natural [20].

Estas operaciones pueden ser: la transformación de una representación lingüística a otra (como la traducción o la categorización gramatical) o la comunicación entre el hombre y las computadoras mediante el uso del lenguaje natural [64]. De esta forma, cualquier programa de computadora que procese estructuras lingüísticas y no simplemente palabras, es considerado un programa de PLN [60].

Por otro lado, de acuerdo con Moreno Sandoval [60] se denomina **lenguaje** a cualquier mecanismo de transformación y codificación de información. Debido a que las computadoras pueden, al igual que los humanos, manipular símbolos y por tanto información, se dice que también utilizan lenguajes [60]. Por ello, se hace una distinción entre los lenguajes naturales (humanos), como el español o el náhuatl y los lenguajes artificiales (creados para las computadoras), como los lenguajes de programación. Debido al tema de esta tesis, se utiliza el término *lenguaje* para referirse sólo al lenguaje natural.

Los sistemas de PLN hacen uso de conocimiento lingüístico, ya que éste es de gran ayuda para modelar ciertos aspectos del lenguaje. Por eso, en diferente medida, de acuerdo con el tipo de tarea que realicen, estos sistemas hacen uso de diferentes niveles de conocimiento lingüístico, a saber [60],[93],[30],[28],[29]:

- Nivel **fonético** y **fonológico**. Ayuda a entender las realizaciones acústicas de los elementos lingüísticos. Generalmente se utiliza sólo en los sistemas que procesan el habla.

- Nivel **ortográfico** o **morfológico**. Describe la estructura y aspecto externo de las palabras, es decir, la forma en que se articulan las letras que las integran.

- Nivel **sintáctico**. Permite describir la organización de las palabras en los enunciados y clasificarlas en categorías gramaticales de acuerdo a la función que desempeñan dentro de ellos.

- Nivel **semántico**. Permite asignar un significado a los elementos lingüísticos dentro de un contexto determinado.

- Nivel **pragmático** o **contextual**. Establece los referentes extralingüísticos por los cuales debe interpretarse un objeto que está siendo aludido en una oración. También se refiere al uso del lenguaje en una situación o contexto específico. Se divide en dos tipos:

  - **Del discurso**. Toma en cuenta la información emitida anteriormente para realizar una interpretación ya sea anafórica o temporal.
  - **Del mundo**. Es la información conceptual del entorno (el mundo real) de los hablantes, que les permite comprender información no expresada explícitamente.

Los sistemas de PLN, como todos los sistemas computacionales, pueden construirse mediante técnicas de ingeniería de software [30]. Una de esas técnicas es la modularidad, la cual divide un problema complejo en problemas más simples para poder abordarlos de manera individual [93],[60]. Como ya se ha mostrado, el conocimiento lingüístico puede también estratificarse en niveles lo que facilita que un sistema de PLN (construido modularmente), se oriente al tratamiento de un aspecto del lenguaje en particular. Es decir, cada uno de sus componentes se enfoca a su modelo de lenguaje de manera relativamente aislada con respecto a los otros.

Lo anterior ha motivado el desarrollo de herramientas especializadas que posteriormente facilitan la creación de sistemas más grandes [60]. Tal es el caso de los **correctores ortográficos**, que son sistemas que revisan un texto en busca de errores con el objeto de corregirlos de tal manera que el texto revisado puede utilizarse directamente por otros sistemas de PLN.

Entre las aplicaciones que requieren un proceso de revisión ortográfica al texto de entrada se encuentran las siguientes [47]:

- Edición de código y texto asistida por computadora.
- Traducción automática.
- Tutoriales para el aprendizaje de idiomas.
- Interacción con bases de datos.
- Transcripción fonética.
- Reconocimiento óptico de caracteres (tanto impresos como escritos a mano).
- Reconocimiento y síntesis del habla.

En la siguiente sección se describe el proceso de corrección de errores ortográficos en textos electrónicos y las técnicas que serán utilizadas en este trabajo para la construcción de un método automático de corrección ortográfica.

6.2 Proceso de Corrección de Errores en Textos Electrónicos

El **proceso de corrección** de errores presentes en textos electrónicos, de manera tradicional, se realiza en tres fases [47],[44],[39]: la detección de errores, la generación de correcciones candidatas y la corrección.

Los investigadores han desarrollado sistemas de corrección que llevan a cabo las tres fases anteriores de manera simultánea y también de forma separada. Pero, debido a la diferencia en cuanto a complejidad se refiere, sobre todo entre la detección de errores y la selección de la mejor corrección candidata, debe hacerse una distinción entre ellas [47]:

La **detección de errores ortográficos** consiste, básicamente, en revisar que una cadena de caracteres se ajuste a un modelo de lenguaje, el cual puede ser tan solo una determinada lista de palabras [47] o bien una estructura gramatical (morfológica, sintáctica o semántica) [74].

La **generación de correcciones candidatas** consiste en localizar una o varias correcciones potenciales para cada palabra detectada como error.

La **corrección** tiene como objetivo seleccionar la mejor corrección candidata o potencial. Va más allá de la localización y ponderación de palabras candidatas a sustituir a la cadena errónea [47]. Esto se debe tanto a las características morfológicas de las lenguas naturales, como a las restricciones sintácticas y semánticas, las cuales inciden significativamente en la solución del problema [47],[26].

Tomando en cuenta lo anterior, los investigadores han desarrollado dos tipos de detección y corrección cuya implementación depende de las características de la aplicación en la que sea utilizada [47]:

1. **Detección y corrección interactiva.** Presentan ante el usuario una lista de sugerencias para que sea él quien determine en última instancia la más adecuada para sustituir una cadena identificada como error. Un ejemplo del uso de esta forma de corrección se encuentra en los procesadores de texto.

2. **Detección y corrección automática.** Se busca que el sistema sea quien realice todo el trabajo de manera independiente sin solicitar ayuda del usuario. Además, muchas veces se necesita que éste procesamiento se haga en tiempo real, por ejemplo, para el caso de los conversores

de texto en habla, es por ello que en este trabajo nos enfocaremos en el desarrollo de un método de este tipo.

Además, las técnicas de corrección se dividen en dos tipos, según tomen o no en cuenta el contexto de las palabras analizadas para realizar la corrección de errores ortográficos. A continuación se mencionan estos dos tipos de corrección:

1. **Corrección de palabras aisladas**. Únicamente toma en cuenta la pertenencia de la palabra analizada a un modelo de lenguaje (una lista de palabras o restricciones morfotácticas). Sólo corrigen errores de palabras no válidas dentro de la lengua.

2. **Corrección contextual de palabras**. Toma en cuenta el contexto en el cual aparece la palabra, es decir, el sintagma al que pertenece. Comúnmente se utilizan las n palabras anteriores o siguientes a la palabra detectada como incorrecta, con ello se logran corregir errores de palabras no válidas y palabras reales que no corresponden al contexto (están mal articuladas).

En las secciones que siguen se presenta una clasificación de los errores que pueden encontrarse en los textos electrónicos, también se muestran los modelos de lenguaje basados en n-gramas y los lexicones computacionales. Posteriormente se explican las tres fases del proceso de corrección de errores. Debido a que estas fases están estrechamente relacionadas (la detección precede a la generación de correcciones candidatas y la corrección depende de la primera para que sus resultados sean buenos) es difícil separarlas totalmente, incluso algunos métodos las realizan de manera conjunta. A pesar de ello, con fines explicativos, se presentarán de manera separada según lo expuesto en [47],[44],[74],[40],[39].

6.3 Tipos de Errores

Para abordar el tema de la detección y corrección automáticas, es de ayuda conocer qué tipos de errores son los que se presentan en la lengua escrita. A continuación se muestran diferentes clasificaciones de los errores de acuerdo a diferentes criterios, cabe señalar que las divisiones frecuentemente se intersectan entre sí, pero son de utilidad para tratar el tema de una manera ordenada.

6.3.1 Errores Originados según la Forma de Obtener el Texto

En esta primera división, se toma en cuenta la forma en que se obtuvo el texto en formato electrónico, por ello, los errores pueden ser generados por [47],[44]:

- **Sistemas de cómputo** (dispositivos electrónicos):
  - Fallas en el reconocimiento óptico de caracteres.
  - Fallas en el reconocimiento del habla.
- **Humanos**:
  - Imprecisiones al capturar los textos, debidas a la adyacencia de las teclas o a una mala coordinación.
  - Desconocimiento de la representación ortográfica correcta de algunos fonemas.

6.3.2 Tipos de Errores según la Palabra que Generan

Además de la clasificación antes señalada, los errores pueden caer en alguno de los siguientes casos con consecuencias muy distintas para el proceso de detección y corrección [47],[44]:

- Los que provocan que una cadena de caracteres se convierta en una palabra que no pertenece a una lengua, llamados **errores de palabras no válidas**.
- Aquellos que resultan en otra palabra válida, llamados **errores de palabras reales**.

6.3.3 Clasificación General de Errores

Las clases de errores antes señaladas pueden dar lugar también a otras categorías de errores. Los siguientes tipos de errores son generados por máquinas o por humanos y pueden originar tanto palabras válidas como no válidas:

- **Tipográficos**. Contienen caracteres equivocados. Por ejemplo: *Naci***6**, en lugar de *Nació*; *com***in***icar* en lugar de *com***un***icar*; *ba***n***o* en lugar de *ba***ñ***o*.

- **Gramaticales.** Violan las reglas de articulación de la lengua. Por ejemplo: *a* en vez de **h***a*; *cambi***o** en lugar de *cambi***ó**; **rr***enta* en vez de **m***enta*.

- **Cognitivos.** Se deben a carencias en el conocimiento ortográfico. Por ejemplo: *lien***s***o* en vez de *lien***z***o*; **b***erde* en lugar de **v***erde*; *di***c***iplina* en vez de *di***sc***iplina*.

- **Fonéticos.** Son representaciones equivocadas de ciertos fonemas. Son un caso especial de los errores cognitivos, generalmente son muy difíciles de corregir porque deforman en gran medida la escritura de las palabras. Por ejemplo: *Gua***ll***aba* en lugar de *Gua***y***aba*; **Gü***evo* en vez de *Huevo*; *cone***cc***ión* y no *cone***x***ión*; *sub***rr***ayar* en lugar de *sub***r***ayar*.

En la figura 6.1 se muestran las clasificaciones anteriormente mencionadas.

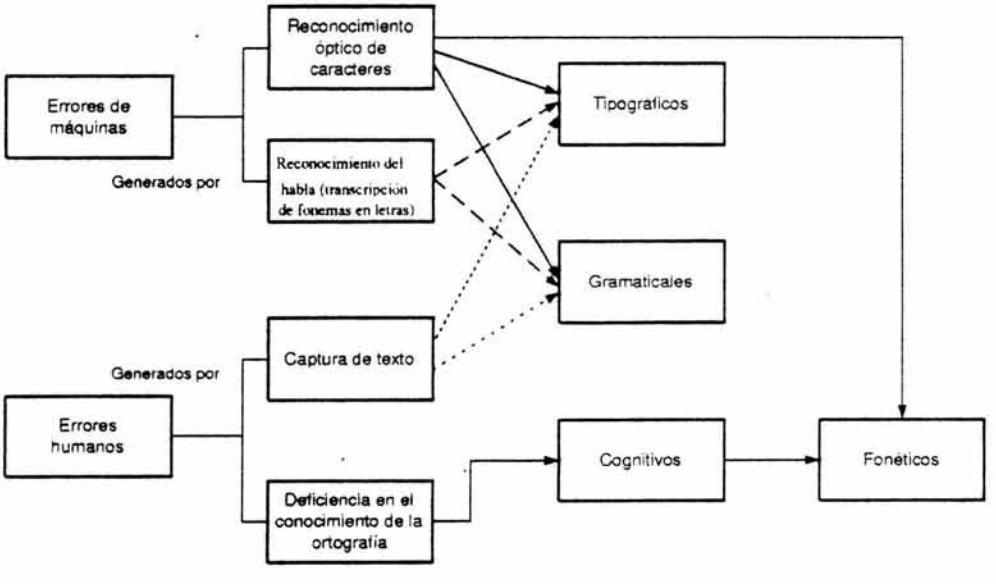


Figura 6.1: En esta figura se muestran los tipos de errores de los textos electrónicos según su origen.

6.3.4 Tipos de Errores según los Niveles de Conocimiento Lingüístico

Además, de acuerdo con lo señalado en el apartado 6.1.1 el conocimiento lingüístico puede estratificarse en niveles. Así, atendiendo a su pertenencia a cada uno de estos niveles podemos clasificar los errores en [47]:

- **Morfológicos**. Violan restricciones en la formación de las palabras. Por ejemplo: *Comunicacion* por *Comunicación*; o *conprobar* por *comprobar*.
- **Sintácticos**. Incumplen restricciones en el orden de articulación de las palabras y obtienen funciones (categorías) incorrectas dentro de un sintagma. Un ejemplo es: *Genaro ha* **venir** frente a *Genaro ha* **venido**.
- **Semánticos**. Producen cambios equivocados en el significado de las palabras en un contexto determinado. Por ejemplo *Pedro* **casó** *al venado*, en lugar de: *Pedro* **cazó** *al venado*.
- De la **estructura del discurso**: provocan incoherencias en las relaciones de los componentes del discurso. Por ejemplo en: *La* **docena** *trágica fueron diez sangrientos días* se hace referencia equivocadamente a doce en lugar de a diez días dentro de la misma oración.
- **Pragmáticos**. Causan que cambie el mensaje que se quiere transmitir en el texto. Por ejemplo un cambio de letra en la oración siguiente: *El* **c***opal es mexicano* hace que el mensaje original: *El* **n***opal es mexicano*, cambie significativamente.

6.3.5 Errores en el Nivel de Palabras

Por último, se muestra una clasificación que se orienta sólo a los casos de error que ocurren en el nivel de las palabras (generados por humanos o por reconocedores ópticos de caracteres) [47],[44]. Cabe señalar, que éstos pueden dar como consecuencia que se produzcan cualesquiera de los tipos de errores antes mencionados. La clasificación es la siguiente:

- **Inserción**. Es añadir un carácter a una palabra. Por ejemplo: *l***a***e* en vez de *le*; *practr**i**car* en lugar de *practicar*.
- **Repetición**. Es repetir un carácter de una palabra (ponerlo dos veces). Por ejemplo: *cab***b***eza* en vez de cabeza; *tin***n***a* y no *tina*.

- **Omisión**. Elimina un carácter de una palabra. Por ejemplo: *toar* en vez de *to**m**ar*; *camnar* en lugar de *cam**i**nar*.

- **Substitución**. Cambia un carácter de la palabra original por otro u otros. Por ejemplo: *M**e**xico* en vez de *M**é**xico*; *co**m**er* en vez de *cor**r**er*; *va**ñ**o* en lugar de *va**n**o*; *e**s**tera* en lugar de *e**n**tera*.

- **Transposición**. Intercambia la posición de dos caracteres adyacentes. Por ejemplo: *est**ri**ar* en vez de *est**ir**ar*; *com**op**ner* por *com**po**ner*; *ajon**oj**lí* en lugar de *ajon**jo**lí*.

- **Segmentación**: Es formar dos o más palabras a partir de una sola. Por ejemplo: *ca**n** **c**ión* a partir de: *can**c**ión*; *e**s** **p**era* a partir de: *e**sp**era*; *simple* **m***ente* y no *simple**m**ente*.

- **Unión**: Forma una palabra a partir de dos o más. Por ejemplo: *despertaren* a partir de: *despertar* **e**n; *me**p**ertenece* en vez de: *me* **p***ertenece*; *de**l**concertista* a partir de: *de**l** **c**oncertista*.

Según lo expuesto por Kukich [47], los errores de segmentación son generados más frecuentemente por reconocedores ópticos de caracteres, en tanto que los de unión se deben a errores humanos.

Ahora bien, la importancia de esta última clasificación radica en que las técnicas de corrección y detección que se han creado aprovechan las características de estos tipos de error. Las más exitosas toman en cuenta conocimiento lingüístico en los niveles morfológico y sintáctico principalmente [47]. Los sistemas que incorporan conocimiento lingüístico de los otros niveles son usados en aplicaciones específicas, y por lo tanto, son dependientes del dominio [60].

En este trabajo se utilizarán técnicas estadísticas para realizar la corrección de errores en textos electrónicos, por ello, antes de explicar la fase de detección, es preciso exponer en qué consisten los modelos de lenguaje denominados n-gramas y los lexicones computacionales, porque ambos pueden ser utilizados en diversas etapas del proceso de corrección tanto para realizar una corrección contextual, como una de palabras aisladas.

6.4 N-gramas

Con el objetivo de identificar si una secuencia de elementos lingüísticos (palabra, morfema, letra, etc.) es válida dentro de una lengua, como el español, se utilizan modelos de lenguaje. Dichos modelos representan las formas y las restricciones para combinar los elementos lingüísticos para construir estructuras válidas.

Los **n-gramas** son un tipo de modelo estadístico del lenguaje [41], el cual permite estimar la probabilidad de una secuencia de letras o palabras, lo que ayuda a "predecir" la siguiente unidad lingüística a partir de la(s) anterior(es), es decir, tomando en cuenta la historia [55]. Los n-gramas representan modelos de Markov en tanto que se basan en la "suposición de Markov", la cual establece que una unidad lingüística sólo es afectada por el contexto local (por ejemplo, las palabras precedentes) [41].

Un n-grama es una secuencia de n letras o palabras obtenida a partir de un corpus. Con ellos se pueden obtener las **probabilidades de transición** de las unidades lingüísticas del corpus. Esto es, la probabilidad de aparición de una determinada secuencia de elementos lingüísticos en el corpus. Los n-gramas se han utilizado tanto en la detección como en la generación y ponderación de correcciones potenciales de errores de palabras, ya sea que se realicen estas tareas de manera conjunta o separada [47]. Los n-gramas se han empleado también en la corrección de palabras aisladas y en contexto.

De acuerdo con el número de unidades lingüísticas que se utilicen para formar el n-grama, se tienen los siguientes tipos: monogramas, bigramas y trigramas para valores de n de 1 a 3, respectivamente.

Por ejemplo: los n-gramas (al nivel de letras) ($n = 1 \ldots 3$) de la palabra *informática* se muestran en la tabla 6.1.

Orden del n-grama	N-gramas
Monogramas	{i,n,f,o,r,m,á,t,i,c,a}
Bigramas	{in,nf,fo,or,rm,má,át,ti,ic,ca}
Trigramas	{inf,nfo,for,orm,rmá,mát,áti,tic,ica}

Tabla 6.1: Ejemplos de n-gramas de letras de la palabra *informática*

Por otro lado, en la tabla 6.2 se muestran los n-gramas de palabras de la siguiente oración:

La informática es interesante

Orden del n-grama	N-gramas
Monogramas	{La} {Informática} {es} {interesante}
Bigramas	{La,informática}{informática,es}{es,interesante}
Trigramas	{La,informática,es}{informática,es,interesante}

Tabla 6.2: Ejemplos de n-gramas de palabras de la frase *La informática es interesante*

A cada n-grama se le asocia información estadística, es decir, su probabilidad de aparición.

En el caso de los n-gramas de letras, en ocasiones se les asocia información posicional, o sea, información de la posición de cada n-grama dentro de las palabras del lexicón o corpus.

De esta forma, los n-gramas de palabras representan la probabilidad de que n palabras aparezcan juntas en el corpus. Es decir, muestran una aproximación de la probabilidad de una determinada palabra, a partir de la palabra previa. Así, la probabilidad de un sintagma puede obtenerse a partir de las probabilidades de todas las palabras que lo conforman [41], obtenidas de acuerdo con el modelo del lenguaje utilizado (los n-gramas).

Los modelos estadísticos de lenguaje, basados en n-gramas pueden extenderse a tantas palabras como se quiera. Sin embargo hay que tomar en consideración que mientras mayor sea el número de palabras a considerar, se necesitará un corpus más grande para que las probabilidades sean representativas de un dominio en particular. En la ecuación 6.2 se presentan las expresiones matemáticas para el cálculo de las estimaciones estadísticas de los n-gramas.

$$P(w_1 \ldots n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1}) \tag{6.2}$$

Donde:

w_k k-ésima palabra.

P probabilidad de que la palabra w_k esté precedida por w_{k-1}.

n total de palabras de la secuencia analizada (n-grama).

Hay que tomar en cuenta las siguientes consideraciones para el cálculo de las estimaciones estadísticas:

- Debido a que probabilidades muy pequeñas pueden ocasionar que se presenten errores en las computadoras porque no se pueden representar adecuadamente (*underflow errors*). Una forma de realizar éstos cálculos es el empleo de los logaritmos de cada probabilidad, los cuales son sumados y al resultado se le calcula el antilogaritmo para obtener la probabilidad buscada [41].

- Una previsión más es evitar que se produzcan probabilidades iguales a cero (si la palabra no existe en el corpus) o demasiado bajas (si su frecuencia en el corpus es muy reducida) ya que éstas pueden afectar negativamente las estimaciones de otras palabras.

El último punto es consecuencia de que las probabilidades obtenidas se multiplican y basta con que una de ellas sea cero para que se produzca una estimación inadecuada. En el caso de la corrección de errores se entiende que las palabras erróneas no estarán en el corpus, por tanto, es imprescindible ocuparse de este tema. Una forma de resolver lo anterior es aplicar un proceso de descuento o suavizado. La idea es reevaluar las probabilidades de las palabras no presentes en el corpus auxiliándose de las probabilidades de las que aparecen en él [41]. El método de descuento que se utilizará en el presente trabajo para calcular la probabilidad de los n-gramas de palabras, es el descuento de Good-Turing, el cual ha dado buenos resultados y es comúnmente usado para trabajos de PLN en la lengua inglesa [41],[55]. Este método utiliza los n-gramas con mayor frecuencia para reestimar la probabilidad de los de frecuencia baja o igual a cero. Esto se expresa en la ecuación 6.3.

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \tag{6.3}$$

Donde:

c número de veces que aparece un n-grama en el corpus.

c^* número de veces que aparece un n-grama en el corpus después de que se ha aplicado el descuento.

N número de n-gramas que aparecen c veces en el corpus.

N_{c+1} número de n-gramas que aparecen $c+1$ veces en el corpus.

Pese a que con el método Good-Turing se evita el cálculo de probabilidades iguales a cero, no se puede extender éste para todos los n-gramas, puesto que para el valor máximo de c dentro del corpus no existe un $c+1$ con el cual calcular el valor de c^* para ese valor [55]. Una forma de solucionar esto es introducir un umbral k para decidir cuáles n-gramas deben ser descontados porque los valores muy altos de c no lo necesitan [55].

Esto se calcula mediante la expresión 6.4.

$$\begin{aligned} &\text{Si } c > k \text{ entonces} \\ &\qquad c^* = c \\ &\text{de lo contrario} \\ &\qquad c^* = (c+1)\frac{N_{c+1}}{N_c} \end{aligned} \tag{6.4}$$

Donde:

k umbral de descuento. Determina si se calcula el descuento para un n-grama que aparece c veces en el corpus.

Como última parte, se comenta una pequeña guía para generar n-gramas, de acuerdo con Manning y Schütze [55] se realizan los pasos siguientes:

1. Preprocesar el corpus. Consiste en la separación de palabras y signos de puntuación. Puede realizarse insertando espacios entre ellos o bien asignando etiquetas especiales para identificar a éstos últimos.

2. Realizar las estimaciones estadísticas antes mencionadas para c y c^*.

6.5 Lexicones Computacionales

Debido a que en este trabajo se utilizará un lexicón para detectar errores de palabras, a continuación se hace una descripción de los lexicones computacionales que se han utilizado en sistemas de PLN [60],[64],[47]. Cabe señalar también, que el uso de lexicones no se restringe a la búsqueda de errores en palabras aisladas, sino que también puede utilizarse para palabras en contexto. Además, son un elemento clave para la generación de correcciones candidatas para el método de corrección propuesto en el presente trabajo.

Como se indicó en la introducción, un lexicón es una lista de elementos lingüísticos que son relevantes para llevar a cabo una tarea. Por ello, al implementar la búsqueda en lexicones como forma de detección se debe enfrentar el siguiente problema: el tamaño de los lexicones. Puesto que a mayor tamaño del lexicón, mayor será la demanda de recursos de cómputo (almacenamiento y tiempo de ejecución), lo que hace que la velocidad total de respuesta del sistema se decremente. Por ello, se debe hacer énfasis en dos aspectos esenciales en la constitución del lexicón: el contenido del lexicón y la forma en que se representa dicho contenido. Del contenido depende el espacio de búsqueda, mientras que de la forma de representarlo depende el método de búsqueda que se utilizará.

La **representación** del contenido de un lexicón computacional se realiza mediante un tipo de estructura de datos (de los lenguajes de programación) [60]. Las estructuras más utilizadas son las siguientes [60],[47]:

- **Listas lineales**. Son estructuras de datos en las cuales la información está representada de manera lineal [11],[60], es decir, cada elemento (llamado nodo) se almacena uno a uno en un orden secuencial.

- **Árboles de búsqueda (trie)**. Consiste en organizar la información en una estructura jerarquizada, de tal forma que cada elemento del árbol puede contener a su vez otro árbol de búsqueda. Los nodos iniciales tienen la tarea de conducir la búsqueda hacia los nodos subordinados [14] y así, finalmente encontrar la meta, en este caso, la palabra buscada. Por ejemplo, cada nodo inicial puede ser la primera letra o sílaba (o cualquier otra unidad lingüística) de las palabras y los nodos intermedios las letras que constituyen a las palabras almacenadas.

- **Tablas hash**. Esta estructura permite asignar a cada elemento que será almacenado, una clave numérica que es utilizada después para buscar ese elemento en el lexicón. Representan la forma más eficiente de utilizar los lexicones [47] ya que reducen considerablemente el número de comparaciones al realizar la búsqueda. Para crear la clave se utiliza una función también llamada hash, la cual debe ser lo suficientemente buena para no generar colisiones, esto es, que se asigne una misma dirección para dos elementos distintos.

En la figura 6.2 se muestran ejemplos de estas formas de representar los lexicones computacionales.

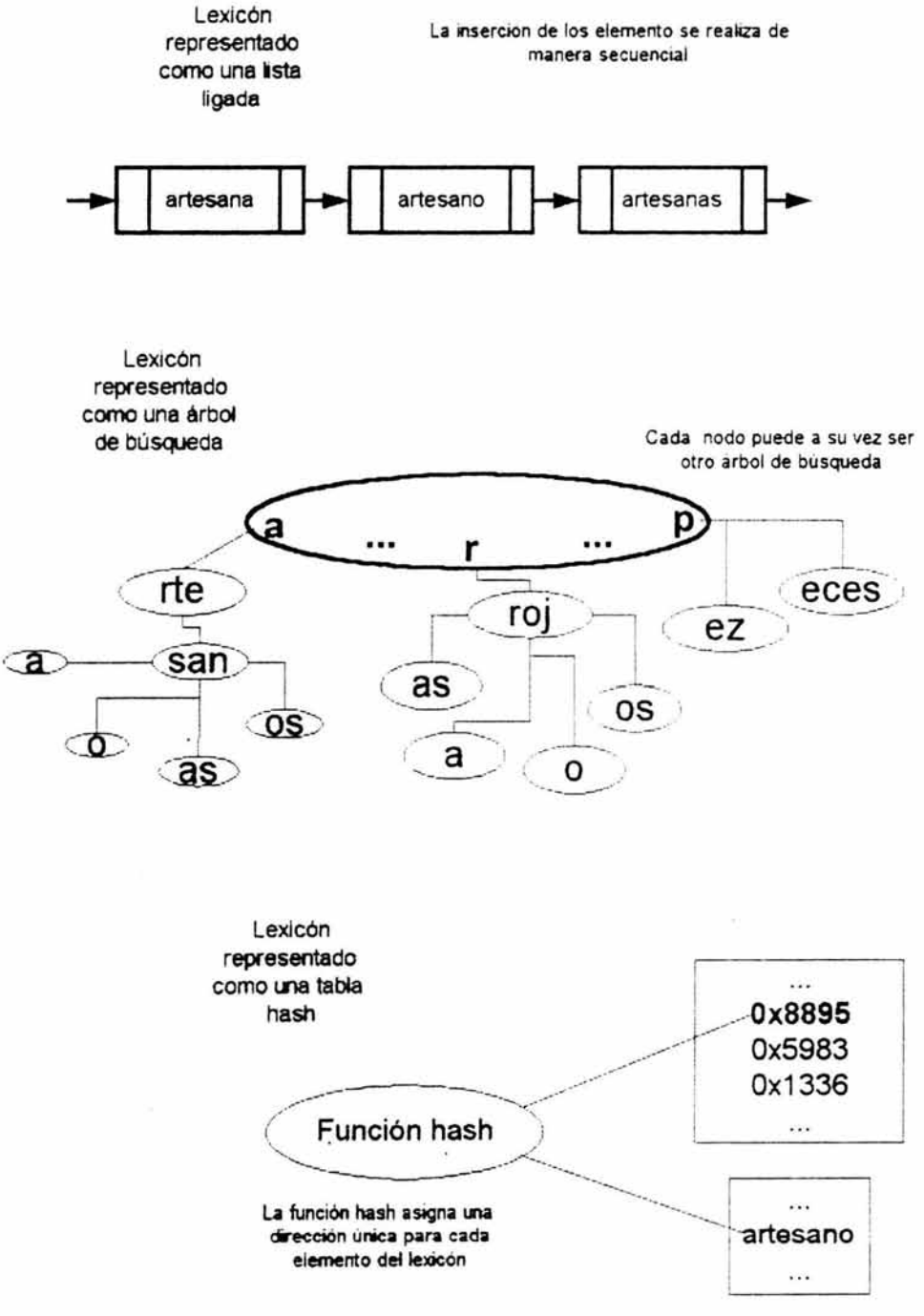


Figura 6.2: Representación de lexicones computacionales.

Por otro lado, el **contenido** del lexicón puede ser cualquiera de las siguientes unidades lingüísticas[1]:

- **Palabras completas**. Representan la forma ortográfica completa de una palabra. De esta forma, se tienen que almacenar todas las posibilidades que una palabra presenta.

- **Variantes de palabras**. Esta forma de almacenar la información se apoya en el hecho de que una palabra puede dividirse en unidades más pequeñas, llamadas **morfemas** que son la mínima forma con significado de la lengua [28],[8]. A su vez, los morfemas se dividen en lexemas y gramemas. Los **lexemas** indican la idea principal de una palabra [28],[8], en tanto que los **gramemas** son las formas variables de las palabras, expresan modificaciones en su idea principal, como los accidentes gramaticales (género, número, tiempo, modo y persona) [8]. Así, en el lexicón se almacenan sólo algunas variantes de las palabras. Por ejemplo, se guardan sólo los lemas o raíces en lugar de todas las variantes. Con ello se obtiene también una reducción en el tiempo de búsqueda, ya que el programa no tiene que comparar la palabra buscada con todas sus posibles variantes.

- **N-gramas**. También pueden almacenarse n-gramas, ya sea de letras o de palabras (en el caso de la corrección contextual). El objetivo es almacenar información preprocesada para que ésta sea utilizada en las siguientes tareas del método de detección empleado.

En la figura 6.3 se muestran ejemplos de lexicones con contenidos diferentes según lo expuesto anteriormente.

Además de tomar en consideración el contenido y la representación de los lexicones, para disminuir el tiempo de búsqueda y el procesamiento, se han utilizado esquemas de particionamiento del lexicón [47]. Esto se refiere a dividir el contenido del lexicón en sublexicones. Hay que tomar en cuenta diferentes criterios para hacer la división del lexicón. Por ejemplo el dominio de la aplicación en que se va a utilizar, la frecuencia de aparición de las palabras que lo constituyen y aún el tamaño de éstas. Con respecto a este último hay que destacar que se tienen pocos estudios acerca de la frecuencia

[1]Hay que señalar que el contenido de un lexicón puede ser únicamente la *entrada*, es decir, el elemento de búsqueda, o bien puede constituirse con la entrada e información sobre ésta

de errores con relación al tamaño de las palabras [47], sin embargo, hay que tomar en consideración los siguientes dos hechos: las palabras más frecuentes tienden a ser asimismo las más cortas [47],[54] y los errores en palabras cortas son mucho más difíciles de corregir [47].

Basándose en lo anterior se pueden utilizar varios esquemas de particionamiento, por ejemplo:

- **Frecuencia de las palabras**. Atendiendo a la frecuencia de aparición de las palabras constitutivas (u otras unidades lingüísticas) se tendrían tres niveles de particionamiento (cuyo tamaño varía ascendentemente) [47]: en el primero se guardarían las palabras más frecuentes. En un segundo nivel se almacenarían las palabras específicas del dominio de aplicación. Finalmente, en el último lexicón se almacenarían las palabras con la menor frecuencia de aparición.

- **Tamaño de las palabras**. También puede dividirse el lexicón por el tamaño de las unidades lingüísticas. De esta forma, se tendrían varios lexicones de acuerdo con el tamaño seleccionado para hacer la división.

- **Categorías gramaticales**. Para el caso de las palabras, otra de las formas de disminuir el espacio de búsqueda consiste en crear lexicones de categorías gramaticales, así, cada lexicón contendrá sólo palabras que corresponden a su misma categoría gramatical. Por ejemplo: un lexicón de verbos, sólo contendrá verbos, mientras que uno de sustantivos sólo almacenará éste tipo de palabras.

En el presente trabajo se utilizarán lexicones con palabras completas, un esquema de particionamiento basado en tamaños de palabras y categorías gramaticales y para representar el lexicón se usará una tabla hash.

6.6 Detección de Errores

Como se indicó en la sección 6.2, en el proceso de corrección se llevan a cabo tres actividades: detección, generación de correcciones potenciales y corrección (la más compleja de ellas). Además, éstas pueden realizarse tomando en cuenta únicamente la palabra analizada o bien, tomando en cuenta también su contexto, es decir, la estructura gramatical a la que pertenece (las palabras que la rodean dentro del sintagma en el que se encuentra) [47],[44],[40],[39].

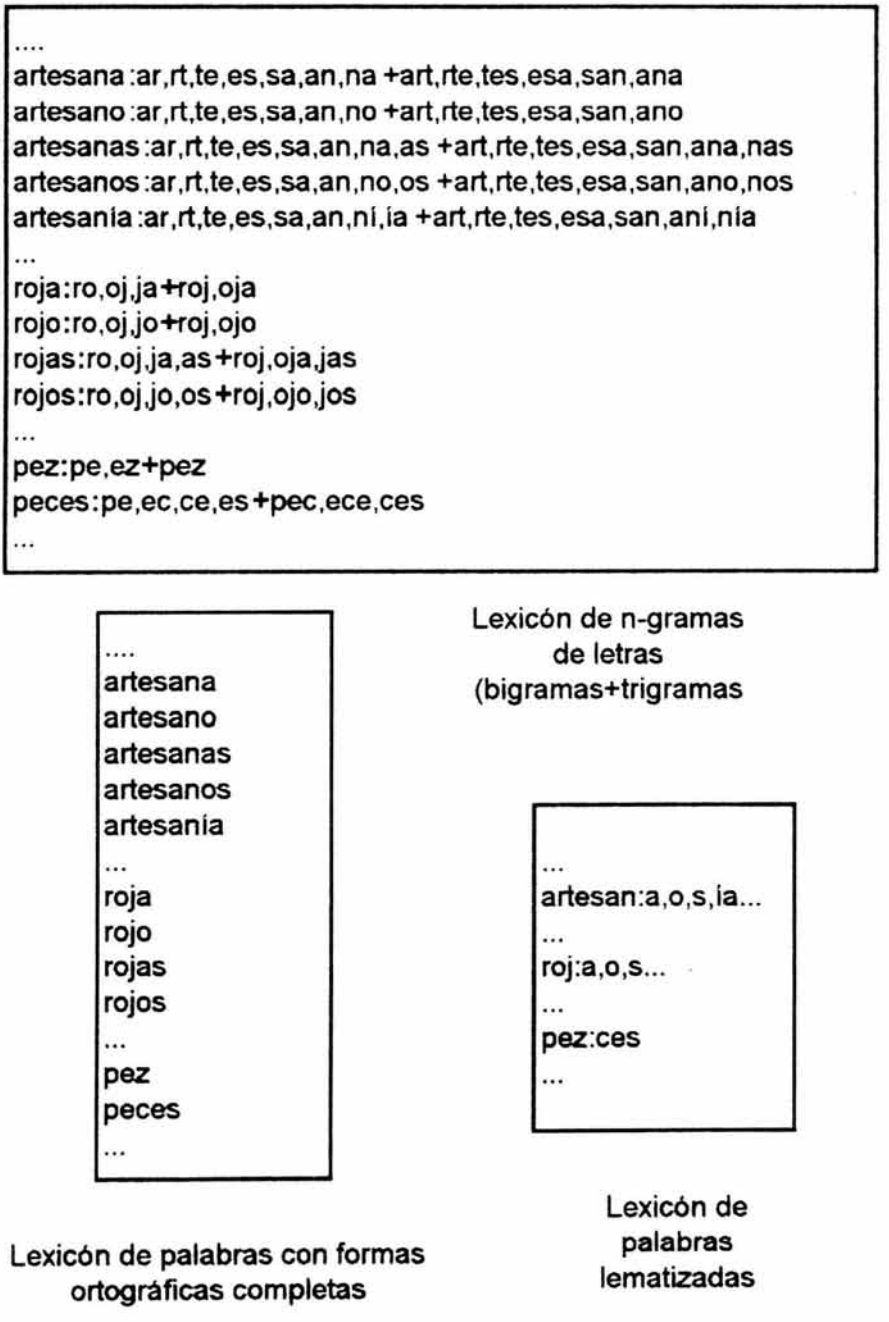


Figura 6.3: Ejemplos de lexicones con diferente contenido.

En esta sección se describe cómo se realiza la detección de errores en textos electrónicos. Se mencionan dos formas de detección de errores: la detección de errores en palabras aisladas y la detección contextual de errores. La primera de ellas analiza las palabras que integran el texto individualmente, mientras que la segunda toma en cuenta las otras palabras del sintagma, es decir, el contexto de las palabras analizadas.

6.6.1 Detección de Errores en Palabras Aisladas y en Contexto

Como ya se indicó, la detección de errores puede realizarse tomando en cuenta sólo palabras aisladas y también el contexto de éstas, aunque hay que aclarar que el contexto al que se hace referencia es un contexto local, es decir, de las palabras contiguas a la palabra analizada cada vez y no al contexto del sintagma completo [47].

La detección de errores consiste en verificar que una palabra (o un conjunto de ellas) pertenezca a un modelo de lenguaje.

En lo que se refiere a la *detección de errores en palabras aisladas*, el modelo de lenguaje puede ser una lista de palabras válidas (lexicón) o bien, un conjunto de restricciones morfotácticas de la lengua en la que se está trabajando. En el primer caso, se considera que una palabra es incorrecta si no se encuentra una entrada en el lexicón que coincida con ella. La principal ventaja del uso de lexicones es que, con un lexicón bien construido, disminuirá el número de palabras aceptadas de manera indebida o de falsas alarmas de error, esto es, palabras que aunque no muy frecuentes, pertenecen a la lengua en cuestión [47]. En el caso de un análisis morfotáctico, se puede dar el caso de que se acepten palabras que no rompan las reglas de articulación de los morfemas de la lengua, pero que no tengan significado en ella. Generalmente, la detección de errores en palabras aisladas sólo identifica errores de palabras no válidas, y pasa por alto aquellos errores que pudieron dar origen a palabras válidas. Por ejemplo, en el siguiente sintagma, se encuentra un error que no puede ser detectado analizando palabras aisladas: "ya *se* he dicho", puesto que la palabra *se* es válida dentro del español y seguramente se encontrará en el lexicón, sin embargo no es una construcción válida.

En contraste, la *detección contextual de errores* identifica también errores de palabras no válidas. El modelo de lenguaje utilizado consiste en un lexicón de n-gramas de palabras. De esta forma, todas las secuencias de n palabras

del texto son comparadas con los n-gramas del lexicón, y para identificarlas como errores, pueden tomarse dos criterios: que la secuencia de palabras sea de muy baja probabilidad (esto es, su probabilidad de transición), o bien, que no exista dentro del modelo de lenguaje [47]. Continuando con el ejemplo anterior, analizando la secuencia de palabras "ya *se* he dicho" con bigramas y trigramas sería identificada como un error, ya que es muy poco probable que se encuentren en el lexicón los n-gramas siguientes:

Trigramas:

$$\{ya,se,he\} \text{ y } \{se,he,dicho\}$$

Bigramas:

$$\{se,he\}$$

Una vez que se han detectado las palabras incorrectas del texto, se procede a generar las correcciones potenciales y finalmente a la selección de la corrección candidata que sustituya al error. En las secciones siguientes se explican estos dos procesos.

6.7 Generación de Correcciones Potenciales

El objetivo de esta segunda fase es crear un conjunto de palabras denominado **correcciones potenciales**, entre las que se encuentre la que sustituirá a cada una de las palabras del texto detectadas como incorrectas. Para llevar a cabo esta tarea se requiere de un lexicón y un criterio de comparación para extraer las correcciones candidatas del lexicón en función de las palabras incorrectas [60],[89].

A continuación se describen tres técnicas para generar las correcciones candidatas a partir de un lexicón: análisis de n-gramas de letras, comparación de claves y mínima distancia de edición.

Hay que señalar que éstas técnicas son utilizadas también para seleccionar la palabra que habrá de sustituir a la palabra incorrecta por lo que en esta sección sólo se indica el uso que puede darse para generar correcciones potenciales, mientras que en la sección siguiente se describe cómo se emplean para realizar la ponderación y selección de la mejor corrección candidata.

Análisis de n-gramas de letras. Se parte de un lexicón de palabras cuyas entradas están formadas con los n-gramas de letras de las palabras que

los integran. Así, se obtienen los n-gramas de la palabra incorrecta y éstos son comparados con las entradas del lexicón y de esta manera se pueden extraer bastantes candidatas revisando los n-gramas que se deseen [89]. Por ejemplo, suponiendo que como resultado de un proceso de reconocimiento óptico de caracteres, se obtiene la palabra *infomática*, podrían recuperarse las siguientes candidatas para esa palabra si se busca con su primer trigrama de letras (*inf*):

$$\{inf\} = [informática,\ información,\ informar,\ informe]$$

Con ello, se obtendría el conjunto de correcciones potenciales entre las cuales está la palabra que realizaría la corrección adecuadamente. Hay que señalar que pueden tomarse cualesquiera de los n-gramas que forman las palabras para crear las entradas del lexicón. Por ejemplo, si se toman los tres primeros n-gramas de las palabras se evita que las palabras que tienen errores en las primeras letras se queden sin correcciones candidatas porque sus n-gramas no coincidan con los n-gramas del lexicón. De esta forma se podrían obtener de varias maneras correcciones potenciales para la palabra incorrecta *nformática*:

$$\{nfo\} = [informática,\ información,\ informar,\ informe]$$
$$\{for\} = [informática,\ información,\ informar,\ informe]$$
$$\{orm\} = [informática,\ información,\ informar,\ informe]$$

Similitud entre claves. En esta técnica se crea una clave para cada palabra analizada. Para que la clave sea efectiva debe mantener las características fundamentales de las palabras y ser insensible a errores en éstas [76], de tal forma que palabras con una ortografía similar tengan claves similares o aún idénticas. De esta manera, una palabra tendrá una o varias correcciones potenciales y el tiempo de búsqueda será breve pues se evita el recorrido de todo el lexicón.

Existen diversas maneras de crear las claves, puede tomarse en cuenta la abreviatura, el orden de las letras, el tamaño de las palabras, la pronunciación silábica, o bien, alguna otra convención.

Una forma de implantar esta técnica es la creación del "**esqueleto de la palabra**" (que también podría utilizarse para construir tablas hash [44]), esta clave consiste en la separación de consonantes y vocales de la palabra analizada en el orden de aparición tomando la primera letra como inicio del esqueleto, eliminando la repetición de signos [76].

Por ejemplo, el esqueleto de *comunicado* sería:

Consonantes *{cmnd}*, vocales *{ouia}*

La búsqueda se realiza tomando como criterio la coincidencia de la primera letra de cada palabra del lexicón, con la primera letra de la palabra analizada.

Aquí vale la pena aclarar que al realizar una búsqueda alfabética, un esquema de particionamiento alfabético del lexicón sería muy funcional. Sin embargo hay que tomar en cuenta la siguiente información: según Kukich [47] , los errores en palabras ocurren con menor frecuencia en la primera letra de éstas. Pero lo anterior aplica sobre todo para textos generados por humanos, puesto que Klein & Kopel [44], mencionan que los reconocedores ópticos de caracteres introducen errores en la primera letra con una mayor probabilidad que en el resto de la palabra.

Por lo tanto realizar una búsqueda alfabética con esta técnica no es apropiado para corregir errores provenientes de un proceso de reconocimiento óptico.

Mínima distancia de edición (*MDE*). Ésta técnica, también conocida como distancia entre dos cadenas calcula el número mínimo de operaciones de edición, es decir inserciones, borrados, sustituciones o transposiciones, necesario para convertir una cadena en otra. A cada operación se le asigna un peso que varía en función de la forma en que se implementa, así, dichos pesos pueden ser enteros, o números reales (por ejemplo, la probabilidad de cada operación). De esta forma, se buscan en el lexicón aquellas palabras que tengan las distancias menores con respecto a las palabras incorrectas detectadas. En términos generales, la MDE requiere m comparaciones entre la palabra errónea detectada y las palabras del lexicón, donde m es el número de palabras de éste.

A cada operación se le asigna un peso que varía en función de la forma en que se implemente. Por ejemplo, Levenshtein propone los siguientes valores [41]:

- inserción = 1
- borrado = 1
- substitución = 2 (puede ser representada como una inserción y un borrado)

A continuación se muestra el algoritmo para calcular la MDE entre dos palabras que utiliza los pesos sugeridos anteriormente para las operaciones de edición.

Algoritmo 6.1. Mínima Distancia de Edición
Variables de Entrada:

$p1$ Cadena de caracteres (palabra) uno

$p2$ Cadena de caracteres (palabra) dos

Variables de Salida:

d la mínima distancia de edición entre las dos palabras $p1$ y $p2$

Variables Locales:

p_i carácter de la columna i

p_j carácter del renglón j

$tam(p)$ tamaño de la palabra p, es decir, su número de caracteres

i i-ésima letra de $p1$, es decir, las columnas de la matriz

j j-ésima letra de $p2$, es decir, los renglones de la matriz

$D(i,j)$ la distancia entre la letra de la columna i y la de la columna j

costoI el peso que se le asigna a la operación de inserción de un carácter, en este caso es 1

costoB el peso asignado a la operación de borrado. También es 1

costoS el peso que se asigna a la operación de sustitución de un carácter por otro. En este caso es 2, porque se toma como un borrado y una inserción. Si los caracteres son iguales, el valor es 0

Funciones Locales:

$min(a,b,c)$ función que devuelve el menor de los tres valores que recibe: a, b y c

$creaM(a,b)$ función que crea una matriz de a por b elementos

1. $n = tam(p1)$
2. $m = tam(p2)$
3. $D = creaM(n+1, m+1)$
4. $D(0,0) = 0$
5. para $i = 0$ hasta n hacer

   para $j = 0$ hasta m hacer

   $D(i,j) = min(D(i-1,j) + costoI(p_i), D(i-1,j-1) + costoS(p_j, p_i), D(i,j-1) + costoB(p_j))$
6. $d = D(i,j)$
7. Regresar d

Para ejemplificar el cálculo de la MDE, mediremos la distancia mínima entre *pemo* y *perro*, que es **3**, debido a que presenta una sustitución y una inserción. La tabla 6.3 ilustra la matriz de distancias que resulta al calcular la MDE anterior.

		p	**e**	**m**	**o**
	0	1	2	3	4
p	1	**0**	1	2	3
e	2	1	**0**	1	2
r	3	2	1	**2**	3
r	4	3	2	3	**4**
o	5	4	3	4	**3**

Tabla 6.3: Matriz de distancias entre perro y pemo. La diagonal principal, en negritas, indica la MDE para cada renglón y columna (es decir, cada carácter de las palabras comparadas).

Analizando el último valor de la celda de la tabla 6.3, se tiene lo siguiente:

$D(4,5) = min(D(3,5)+costoI(o), D(3,4)+costoS(o,o), D(4,4)+costoB(o))$

Donde:

$costoI(o) = 1$

$cotoS(o, o) = 0$

$costoB(o) = 1$

$D(3, 5) = 4 + 1 = 5$

$D(3, 4) = 3 + 0 = \mathbf{3} \leftarrow$ *menor valor*

$D(4, 4) = 4 + 1 = 5$

Por lo tanto, el valor de D(4,5) es **3**.

Finalmente se comenta que han existido varias modificaciones a esta técnica, desde las que permiten tratar errores no adyacentes hasta las que incorporan valores específicos para transformaciones fonéticas o teclas adyacentes.

En este trabajo se empleará un lexicón con n-gramas de letras para generar las correcciones potenciales de las palabras incorrectas que se detecten en el texto. A continuación se describe la fase final del proceso de corrección de errores: la ponderación y la selección de las correcciones potenciales.

6.8 Corrección de Errores

Como se indicó anteriormente, esta fase es la más compleja dentro del proceso de corrección de errores en textos electrónicos. En ella, se realiza una ponderación del conjunto de correcciones potenciales que se obtuvo en la fase anterior y se lleva a cabo una discriminación de las palabras que lo integran con el fin de seleccionar las candidatas más adecuadas para sustituir a las palabras incorrectas.

En la ponderación se les asignan pesos a las palabras del conjunto de correcciones potenciales con el fin de descartar aquellas palabras que tengan una diferencia mayor con la palabra incorrecta. Los criterios para asignar pesos pueden basarse en el parecido morfológico y en la probabilidad de que una palabra dada sea la corrección candidata más adecuada de acuerdo con el modelo de lenguaje [47]. A continuación se exponen algunas de las técnicas de ponderación y selección que se han desarrollado.

Análisis de n-gramas de palabras. Como se indicó en las secciones anteriores, los n-gramas de palabras son secuencias de n palabras que aparecen juntas en un corpus de texto que tienen asociada información estadística,

como su frecuencia o su probabilidad de aparición [60],[55]. Se utilizan para ponderar cada una de las correcciones candidatas de acuerdo con la probabilidad de transición de la secuencia formada con las correcciones potenciales y las palabras precedentes y siguientes a la palabra incorrecta en el texto. Mientras más probable resulte una construcción, mayor será la ponderación que reciba la corrección candidata. De esta forma, las correcciones potenciales de mayor peso son avaladas por el modelo de lenguaje utilizado [60].

Análisis de n-gramas de letras. De la misma forma que en la generación de correcciones candidatas, en esta fase, se obtienen los n-gramas de letras de las correcciones potenciales y de la palabra incorrecta. Una vez que se tienen, se procede a comparar el número de coincidencias que presentan ambos conjuntos de n-gramas. Con ello, se tiene una medida de similitud entre las palabras del conjunto de correcciones candidatas y la palabra incorrecta [89]. El uso de n-gramas de letras ayuda a capturar información léxica de las palabras del lexicón y permite obtener correcciones válidas [60].

Hay que tomar en cuenta los siguientes puntos:

- Según indica Kukich [47], un estudio realizado por Dahl y Cherkassky en 1990 muestra que el uso de trigramas y bigramas de letras proporciona mejores ponderaciones para varios tipos de errores sin importar el tamaño de las palabras.

- La presencia de subcadenas en el conjunto de correcciones potenciales. Esto se refiere a que palabras que están contenidas en otras, por ejemplo *ferro***carril** y *carril* pueden recibir una ponderación equivocada que haga que el peso de la corrección potencial sea distorsionado. Para evitar esto, se utiliza el siguiente *coeficiente de similitud para trigramas* según lo expuesto en el trabajo de Angell et al en 1983 (citado en [47]):

$$\frac{C}{max(i,c)} \tag{6.5}$$

Donde:

C número de trigramas comunes entre la palabra incorrecta y la corrección potencial.

i tamaño de la palabra incorrecta (número de caracteres).

c tamaño de la corrección potencial (número de caracteres).

$max(i, c)$ función que regresa el mayor de sus argumentos: c, i.

De esta forma, la ponderación para bigramas y trigramas de las siguientes correcciones potenciales de la palabra *nformática* (*informática* e *información*) sería como sigue:

Bigramas:
nformática {*nf*, ***fo,or,rm,má,át,ti,ic,ca***}
informática {*in,nf*, ***fo,or,rm,má,át,ti,ic,ca***} = 8 coincidencias
nformática {*nf*, ***fo,or,rm***,*má,át,ti,ic,ca*}
información {*in,nf*, ***fo,or,rm***,*ma,ac,ci,ió,on*} = 3 coincidencias

Trigramas:
nformática {***nfo,for,orm,rmá,mát,áti,tic,ica***}
informática {*inf*, ***nfo,for,orm,rmá,mát,áti,tic,ica***} = 8 coincidencias
nformática {***nfo,for,orm,rmá,mát,áti,tic,ica***}
información {*inf*, ***nfo,for,orm***,*rma,mac,aci,ció,ión*} = 3 coincidencias

Aplicando la fórmula 6.5 se obtiene:

$$\begin{gathered} nformática = i = 9 \\ informática = c = 11 \\ \frac{8}{max(9,11)} = \frac{8}{11} = 0.72727272 \\ información = c = 11 \\ \frac{3}{max(9,11)} = \frac{3}{11} = 0.27272727 \end{gathered}$$

Con esto, la corrección potencial mejor ponderada es *informática*.

Comparación de esqueletos. Como se indicó en la sección 6.7, el esqueleto de una palabra está constituido por sus consonantes y vocales, en su mismo orden de aparición y sin repeticiones, tomando como inicio la primera letra de la palabra, por ejemplo, *informática* = {*infrmtc*|*oáa*}. La ponderación consiste en tomar el número de coincidencias entre los esqueletos de las palabras detectadas como incorrectas y sus respectivas correcciones potenciales. Por ejemplo:

nformática = **nfrmtc** | **oáa**
informática = **infrmtc** | **oáa** = 9 coincidencias
información = **infmc** | **oa***ó* = 6 coincidencias

De esta manera, será la corrección *informática* la que resulte con una mejor ponderación.

Mínima distancia de edición (MDE). Consiste en evaluar el parecido de dos palabras basándose en la mínima distancia de edición, que consiste en calcular el número de inserciones, borrados, sustituciones y transposiciones necesarios para convertir una palabra en otra.

Así, calculando las distancias entre las palabras incorrectas y sus respectivas correcciones potenciales, se ponderará con pesos mayores a las correcciones potenciales que tengan las menores distancias de edición con la palabra incorrecta. Por ejemplo, las distancias entre *nformática*, y sus correcciones potenciales: *informática* e *información*, son las siguientes:

$$d(\textit{informática, nformática}) = 1$$
$$d(\textit{información, nformática}) = 9$$

Por lo que la corrección mejor ponderada será: *informática*.

6.9 Asignación de Categorías Gramaticales

Como se mencionó en la introducción de este capítulo, las palabras de una lengua pueden clasificarse de acuerdo con ciertos criterios. De esta forma, los lingüistas han creado el concepto de partes del discurso o categorías gramaticales.

También se mencionó que los corpus de texto pueden contener información adicional, por ejemplo, información sobre las categorías gramaticales a las que pertenecen las palabras que en él aparecen.

Existen dos formas de anotar un corpus de texto: la **forma manual** y la **forma automática**. La primera consiste en que uno o varios especialistas determinen a qué categorías pertenecen las palabras del corpus y se las asignen directamente en el texto mediante alguna notación o convención que la identifique, generalmente denominada **etiqueta**. La segunda forma realiza el etiquetamiento utilizando un programa de computadora que haga esta tarea.

Las ventajas de la forma automática son principalmente: la facilidad de procesamiento y la eficacia que proporciona [73], ya que puede emplearse para procesar grandes cantidades de texto sin que su tasa de aciertos disminuya por factores como el cansancio o la distracción. En este trabajo se utilizará un etiquetador automático.

6.9.1 Proceso de Etiquetamiento Automático

En el proceso de etiquetamiento automático se realizan básicamente las siguientes tareas [60]:

1. Reconocer las palabras de un texto.
2. Asignarles la categoría gramatical que les corresponda.

Hay que señalar que pueden darse dos casos:

1. Que las palabras no sean ambiguas morfosintácticamente, de esta forma, la asignación de sus categorías no presenta mayores dificultades.
2. Que las palabras sean ambiguas morfosintácticamente, y a partir de ellas decidir cuál categoría les debe ser asignada.

Del último punto se concluye que el proceso de etiquetamiento automático puede entenderse como un proceso de desambiguación [60],[55],[57], esto es porque muchas palabras pueden tener más de una categoría gramatical y el programa debe ser capaz de determinar cuál es la más adecuada en un sintagma dado. Por ejemplo, la palabra *bajo* puede referirse a un verbo o a un adjetivo apreciativo, dependiendo del contexto:

(Yo) *Bajo* enseguida; (verbo)
Juan es *bajo* de estatura. (adjetivo apreciativo)

De los etiquetadores automáticos destacan por su grado de precisión los que emplean técnicas estocásticas o estadísticas y de aprendizaje automático [60],[55],[57]. Hay que mencionar que en el fondo todos los algoritmos de aprendizaje son estadísticos [60], la diferencia se hace de acuerdo con lo siguiente [59]:

Los algoritmos estadísticos tienen el conocimiento de forma explícita, en el modelo de probabilidades que utilizan, por lo que se asume cierta intervención humana. Mientras que los algoritmos de aprendizaje automático tratan de crear procedimientos automáticos que "aprendan" una tarea a partir de una serie de ejemplos.

Antes de describir los dos tipos de etiquetadores automáticos conviene señalar que el diseño del **conjunto de etiquetas** o **etiquetario** depende de las pretensiones de la investigación o aplicación que se lleve a cabo y de las características de la lengua en particular en la que se trabaje [60].

Partiendo de lo anterior, existen dos enfoques para el diseño del etiquetador [55]:

1. El **predictivo**. Busca predecir el comportamiento de las palabras en un cierto contexto, en donde las palabras circundantes influyen significativamente.

2. El **clasificatorio**. Busca dar información acerca de la clase gramatical de las palabras.

Además, para el diseño del etiquetario hay que tomar en cuenta lo siguiente: hacer un conjunto de etiquetas muy refinado, que indique las más sutiles distinciones entre las categorías gramaticales, ayuda a que la predicción de etiquetas para las palabras de un sintagma se facilite, sin embargo, la precisión disminuye y la complejidad del etiquetamiento se incrementa [60],[55].

Antes de presentar un ejemplo, se señala uno de los problemas mayores de los etiquetadores automáticos [55]: el de las palabras desconocidas. Como se mencionó antes, una palabra puede no encontrarse en el corpus o en el lexicón a causa de que es poco usual o de un dominio distinto al de éstos, sin embargo, puede darse el caso de que esta palabra sea errónea y por esa razón no se encontrará jamás entre los datos de entrenamiento. En las secciones siguientes indicaremos la forma en que se ha enfrentado este problema.

En la tabla 6.4 se muestra la siguiente oración con sus etiquetas morfosintácticas:

Así, la Filosofía ha podido tomarse como la totalidad de la ciencia humana.

Así	**,**	**la**	**filosofía**	**ha**
C10##LN	M05####	D00##SF	N00#~Sf	VHaPD2N
podido	**tomarse**	**como**	**la**	**totalidad**
VImT~#N	VImI~#E	B06~~LN	D00##SF	N00#~Sf
de	**la**	**ciencia**	**humana**	**.**
P00###N	D00##SF	N00#~Sf	A05#1SF	M04####

Tabla 6.4: Ejemplo de texto anotado morfosintácticamente

Como se muestra en la tabla 6.4, a cada palabra le corresponde una cierta etiqueta en función de su morfosintáxis. Para ilustrar lo anterior, se muestra la tabla 6.5 donde se toman cuatro palabras del ejemplo antes citado.

Filosofía	N00#~Sf	Sustantivo común singular femenino.
humana	A05#1SF	Adjetivo calificativo positivo singular femenino.
de	P00###N	Preposición simple.
como	B06~~LN	Adverbio relativo invariable

Tabla 6.5: Ejemplo de categorías morfosintácticas y su significado

En las secciones siguientes se describen dos formas de realizar el etiquetamiento automático de textos electrónicos: los etiquetadores estadísticos y los que se basan en aprendizaje automático.

6.9.2 Etiquetadores Estadísticos

Los programas de etiquetamiento automático de categorías gramaticales basados en técnicas estocásticas, en particular, los que utilizan Modelos Ocultos de Markov [60],[57],[55] (que son a los que haremos referencia en esta sección), han alcanzado grandes porcentajes de precisión, más de 95% de acierto [60]. Para decidir la etiqueta adecuada, esto es, para desambiguar la categoría sintáctica de la palabra, los etiquetadores estadísticos generalmente utilizan dos fuentes de información [60],[55]:

1. **Información sintagmática**. Deciden qué categoría asignar a una palabra a partir del contexto en el que se encuentra, que en este caso está formado por las categorías de las otras palabras.

2. **Información léxica**. Consiste en un lexicón que contiene la categoría más probable de cada palabra o bien información morfológica para reconocer formas flexionadas (sufijos, prefijos, etc.).

Los etiquetadores estocásticos utilizan los siguientes componentes:

- Un modelo de Markov para calcular la probabilidad condicionada de que a una palabra le corresponda determinada etiqueta.

- Un conjunto de etiquetas.

Los etiquetadores estocásticos asumen que las secuencias de categorías gramaticales en un texto pueden modelarse como una cadena de Markov [55], esto es:

1. La etiqueta de una palabra depende sólo de la etiqueta de la palabra previa, es decir, posee un **horizonte limitado**.

2. La dependencia anterior no cambia a través del tiempo, es decir, es una **dependencia invariante** o **estacionaria**.

Para el caso de las palabras desconocidas se ha planteado realizar algunas de las siguientes acciones [55]:

- Asignar una distribución de probabilidades sobre todo el lexicón o una parte de él.

- Utilizar lexicones morfológicos para asignar etiquetas a estas palabras.

- Crear un modelo de inferencia estadística.

Algunas de las desventajas de los anotadores estocásticos son [6],[55],[10]:

- Necesitan de un corpus etiquetado previamente para obtener la información estadística contextual y un corpus de texto no etiquetado para ser entrenados. Mientras más grandes sean los corpus de texto, mejores serán los resultados alcanzados por el programa. Lo anterior plantea una paradoja, puesto que anteriormente indicamos que con los etiquetadores automáticos se intenta minimizar el etiquetamiento manual, y si bien es cierto que esto se cumple, se requiere que antes haya un trabajo importante de anotación manual para que funcione de forma óptima.

- Cuando no se tiene información léxica disponible, no se tiene un buen desempeño del etiquetador.

- No modelan relaciones de palabras distantes entre sí dentro del sintagma y menos aún dentro del texto.

- Cuando se usan en un dominio de aplicación distinto, los resultados pueden variar significativamente. Lo mismo sucede cuando se utilizan en un idioma distinto [60],[55].

6.9.3 Etiquetadores Basados en Aprendizaje Automático

Debido a las dificultades de los etiquetadores estocásticos, se han propuesto otras formas de realizar el etiquetamiento automático. Una de ellas, que ha obtenido bastante éxito, es el aprendizaje automático [10],[57].

El aprendizaje automático busca "construir algoritmos que permitan obtener [...] una descripción del concepto que subyace a un conjunto de observaciones o ejemplos de aprendizaje. Esta descripción debe ser coherente con el conjunto de observaciones y debe permitir predecir futuras observaciones del mismo problema." [57] (p. 137).

La aplicación de aprendizaje automático a problemas de lenguaje natural se fundamenta en que los problemas de desambiguación pueden ser abordados como **problemas de clasificación** [57]. De esta forma, la clasificación puede tratarse desde tres perspectivas de aprendizaje [59],[57]:

1. **Aprendizaje no supervisado**. A partir de un conjunto de observaciones se establecen clases o categorías en los datos.

2. **Aprendizaje supervisado**. Se define de antemano la regla mediante la cual puede clasificarse una observación dentro de clases preestablecidas.

3. **Aprendizaje semisupervisado**. Se utilizan datos de ejemplo para que el algoritmo cree las reglas de clasificación.

El algoritmo de aprendizaje que se utilizará en este trabajo es el propuesto por Eric Brill a principios de los noventa, denominado *Transformation-Based Error-Driven Learning* (TBL), que ha conseguido una tasa importante de aciertos [10],[57]. En la sección siguiente se describe dicho algoritmo.

6.9.4 Etiquetamiento Basado en TBL

El etiquetamiento basado en el algoritmo de aprendizaje llamado TBL ha alcanzado porcentajes de acierto mayores al 96% para el idioma inglés [10].

El algoritmo TBL genera una lista de **reglas de transformación**, las cuales codifican las interdependencias entre etiquetas y palabras, seleccionando aquellas reglas que minimizan el error en la asignación de etiquetas

[55],[10]. Una regla de transformación está constituida por una regla de reescritura y un contexto de activación.

Las tres principales ventajas del etiquetado automático basado en el algoritmo TBL o simplemente etiquetado TBL son [55],[10]:

1. Puede aprovechar una amplia gama de regularidades léxicas y sintácticas (sufijos, prefijos, contextos, etc.).

2. El conocimiento aprendido es almacenado en un conjunto de reglas deterministas simples, a diferencia de otras formas de etiquetar automáticamente, como las tablas de estadísticas en el caso de los etiquetadores estocásticos.

3. El entrenamiento del etiquetador puede realizarse de dos maneras:

   - **No supervisado**. Las reglas son aprendidas automáticamente sin utilizar un corpus etiquetado manualmente.
   - **Semisupervisado**. Se utiliza un pequeño corpus etiquetado manualmente para que el algoritmo cree las reglas.

A continuación se describe el etiquetamiento semisupervisado con base en lo expuesto en [57],[55],[10].

Los datos de entrada requeridos para el etiquetamiento son:

1. Un corpus etiquetado manualmente.

2. Un lexicón que consta de palabras y sus etiquetas más frecuentes.

El etiquetador consta de tres componentes:

1. El **anotador inicial**. Es un programa que realiza un etiquetamiento preliminar al texto de entrada.

2. El **espacio de transformaciones** admitido o **plantilla de transformaciones**. Representa los **contextos de activación** de las reglas de transformación, dichos contextos pueden estar condicionados por la palabra actual y por combinaciones de palabras y etiquetas. En la tabla 6.6 se muestra un ejemplo de una regla de transformación y sus contextos de activación.

   Además, existen transformaciones cuyo contexto de activación se basa en la morfología de las palabras.

Regla	*E1* → *E2* (sustituye la etiqueta *E1* por la *E2*)
Contextos de activación	La palabra **precedente** (o **siguiente**) está etiquetada como EZ La palabra precedente (o siguiente) es *W* La palabra anterior a la precedente es *X*

Tabla 6.6: Una regla de transformación está asociada a un contexto de activación.

3. El **algoritmo de aprendizaje**. Consiste en una función que realiza las ponderaciones entre los resultados de las transformaciones y el corpus de entrenamiento. El algoritmo de aprendizaje es el siguiente:

Algoritmo 6.2. TBL
Variables de Entrada:

C El corpus de texto etiquetado manualmente

L El lexicón de palabras y sus etiquetas más frecuentes

Variables de Salida:

T Un conjunto de reglas de transformación

Variables Locales:

C_k Es la k-ésima palabra del corpus

E Es la tasa de error. El número de palabras mal etiquetadas

e Es el umbral de error permitido

t Es el número de palabras del corpus T

1. $k = 0$
2. Hacer
   (a) $k = k + 1$
   (b) $v =$ la transformación u_i que minimiza $E(u_i(C_k))$

(c) Si $(E(C_k)) - (v(C_k)) < e$ entonces

finalizar las repeticiones

(d) $C_{k+1} = v(C_k)$

(e) $T_{k+1} = v$

3. Mientras $k < t$

4. Regresar $T_1 \ldots T_k$

El proceso de transformación se realiza como sigue:

1. A cada palabra del texto de entrada se le asigna su etiqueta más frecuente mediante el anotador inicial.

2. El texto así anotado es comparado con el corpus de entrenamiento, de esta forma son aprendidas las reglas de transformación. El algoritmo TBL crea una lista con las transformaciones que minimizan las diferencias, esto es, los errores, con respecto al corpus de entrenamiento.

El proceso se muestra en la Figura 6.4. El algoritmo de aprendizaje TBL selecciona iterativamente las mejores transformaciones, es decir, aquellas que reducen la tasa de error, la cual es medida como el número de palabras mal etiquetadas con respecto al corpus de entrenamiento. El algoritmo se detiene cuando no encuentra alguna transformación que mejore la tasa de error. Es una búsqueda voraz de la secuencia óptima de transformaciones.

Las transformaciones pueden aplicarse tanto de forma inmediata como de forma retardada. En el primer caso la aplicación de las transformaciones puede influenciar a las subsecuentes. Por ejemplo, si se aplica la siguiente regla de forma inmediata:

Sustituye la etiqueta A por la etiqueta B si la etiqueta de la palabra anterior es A.

Produciría el resultado: $AAAA \rightarrow ABBB$.

En tanto que una aplicación retardada produciría: $ABAB$.

Cabe destacar que este algoritmo asigna las categorías de **Nombre común** o **Nombre propio** para las palabras desconocidas según inicien o no con una letra mayúscula. Posteriormente, gracias a las reglas de transformación basadas en la morfología de las palabras, pueden corregirse los errores que resulten de esta asignación inicial.

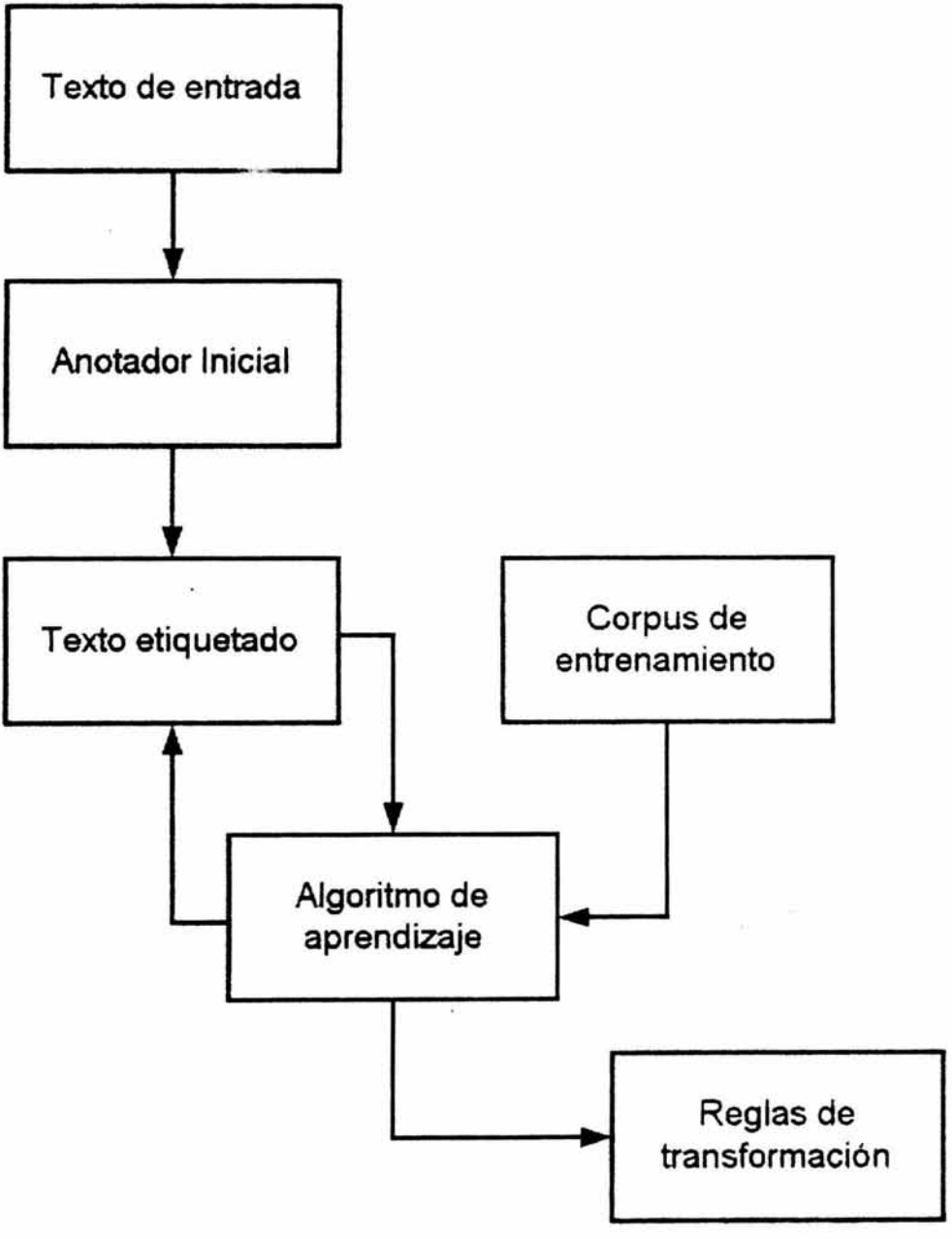


Figura 6.4: El proceso de aprendizaje del etiquetador basado en TBL. Adaptado de [10]

6.10 Resumen

En este capítulo se mostró cómo puede estratificarse el conocimiento lingüístico en niveles y se indicó que es posible incorporarlo a un sistema de cómputo para ayudar a realizar la tarea de corrección de errores en textos electrónicos. Además se mostraron varias clasificaciones de los tipos de errores que se han identificado y se hizo una descripción de los modelos de lenguaje basados en n-gramas. Se señalaron las características de los lexicones computacionales, las formas de representarlos y algunas estrategias para minimizar su tamaño, y por ende, el tiempo de búsqueda.

También se expusieron algunas técnicas de detección, ponderación y selección de errores (de palabras aisladas y en contexto) basadas en estadística y en la morfología de las palabras, como son: búsqueda en diccionarios , análisis de n-gramas de letras y palabras, esqueletos de palabras y la distancia mínima de edición entre dos cadenas. Por último, se expuso el proceso de asignación automática de categorías gramaticales a las palabras de un texto mediante anotadores estocásticos y basados en el algoritmo de aprendizaje llamado TBL.

En este trabajo se utilizará un esquema de particionamiento del lexicón, un modelo de lenguaje basado en bigramas de palabras, monogramas, bigramas y trigramas de letras, el cálculo de la distancia mínima de edición y creación de esqueletos de palabras, además de un anotador basado en TBL.

En el siguiente capítulo se presentarán las aplicaciones y experimentos realizados con los modelos antes mencionados para tratar el problema de la corrección contextual automática de errores en textos electrónicos.

Capítulo 7

El método de Corrección Ortográfica

En el capítulo 1 de esta tesis se estableció la hipótesis H_2, que a la letra dice:

> El proceso de corrección ortográfica contextual, basado en la asignación de categorías gramaticales y análisis morfológico de las palabras de un texto electrónico, mejora la tasa de corrección en comparación con el sólo análisis contextual y morfológico.

Con el fin de poner a prueba la hipótesis anterior, en este capítulo se presenta un método de corrección automática de errores en textos electrónicos en español basado en un análisis morfológico y contextual de las palabras (CATMC), el cual será comparado con otro método de corrección automática que realiza dicho análisis basándose en la asignación de categorías gramaticales, denominado CATCG.

El método CATMC representa la forma de corrección "tradicional" que se ha empleado para revisar ortográficamente textos electrónicos y se utiliza aquí para establecer un marco de comparación entre el método propuesto en esta tesis y los métodos existentes.

El método CATCG es el que se propone en este trabajo. Tiene como base los conceptos de paradigma y categorías gramaticales[1], el primero establece que varios elementos lingüísticos pueden agruparse de acuerdo con un criterio, en este caso ese criterio es la clasificación de las palabras en categorías gramaticales (verbos, sustantivos, pronombres, etc.). Por ello, resulta atractivo

[1] Véase la sección de conceptos básicos del capítulo 6 de esta tesis.

utilizar miembros del mismo paradigma para crear los conjuntos de corrección de cada palabra incorrecta, ya que de esta forma, se está incorporando conocimiento morfosintáctico al corrector ortográfico.

A continuación se describen los datos y los dos métodos de corrección utilizados. En cada método, se indica la forma en que se realiza la detección, la generación de correcciones potenciales y finalmente la corrección de las palabras identificadas como erróneas.

7.1 Descripción de los Datos Utilizados

Los datos que se ocuparon para trabajar con los métodos de corrección CATMC y CATCG son los siguientes:

1. Un corpus de texto (plano) (CT)
2. Un corpus de texto anotado (CA)
3. Un corpus de texto con errores insertados de forma automática y aleatoria (CE).
4. Un conjunto de lexicones de palabras, divididos en uno general y nueve de categorías gramaticales.
5. Un modelo de lenguaje basado en bigramas de palabras.

Enseguida se describen estos datos con mayor detalle.

7.1.1 Corpus Textuales y Etiquetario

Las características del **corpus de texto** utilizado para este trabajo se muestran en la tabla 7.1.

Los textos fueron recopilados de los siguientes sitios virtuales:

1. Antología del Ensayo Ibero e Iberoamericano:
   http://ensayo.rom.uga.edu/antologia/
   Contiene textos sobre filosofía y crítica iberoamericana. La parte que se utilizó fue la correspondiente a los textos en español de los siguientes temas:
   - Ensayo del Siglo XX-Iberoamérica:

Idioma	Español
Formato	Código ASCII
Archivos	497
Líneas de texto	137,103
Palabras	2, 877, 834
Caracteres	18, 075, 061

Tabla 7.1: Características del corpus de texto

  - http://ensayo.rom.uga.edu/antologia/XXA/index.htm
- Crítica:
  - México:
    http://ensayo.rom.uga.edu/critica/mexico/
  - General:
    http://ensayo.rom.uga.edu/critica/generales/
  - Teoría/Retórica del Ensayo Estudios Generales:
    http://ensayo.rom.uga.edu/critica/ensayo/
  - Manifiestos:
    http://ensayo.rom.uga.edu/critica/manifiestos/
  - Pensamiento ecológico:
    http://ensayo.rom.uga.edu/critica/ecologia/
  - Pensamiento de la liberación:
    http://ensayo.rom.uga.edu/critica/liberacion/
  - Retórica:
    http://ensayo.rom.uga.edu/critica/liberacion/
  - Teoría y crítica:
    http://ensayo.rom.uga.edu/critica/teoria/

2. Biblioteca Virtual Miguel de Cervantes:
   http://cervantesvirtual.com/

   Es un proyecto de la Universidad de Alicante, que reúne una gran cantidad de textos en español. Los tres textos utilizados en este trabajo fueron los siguientes:

   - Discurso del método de Descartes. Traducción y prólogo de Manuel G. Morente.

- Dos ideas de la filosofía Pro y contra la filosofía de la filosofía. José Gaos y Francisco Larroyo.
- Elogio de la locura Erasmo de Rotterdam. Traducción, prólogo y notas de Pedro Voltes Bou.

El corpus de texto plano se usó para obtener de la información estadística correspondiente al modelo de lenguaje, esto es, el conjunto de bigramas de palabras y su frecuencia de aparición en los textos.

7.1.2 Corpus Anotado

Hay que señalar que se tomaron muestras aleatorias del corpus de texto plano con el fin de integrar el corpus de texto anotado, el tamaño de estas muestras es del 0.1509% del corpus total, es decir 4,346 cadenas de caracteres. Además, se escogieron diez textos para formar el conjunto de entrenamiento.

El objetivo del **corpus de texto anotado** es entrenar el etiquetador automático basado en TBL que se utilizará en el método de corrección CATCG. Dicho etiquetador se describió en el capítulo 6.

Las características del corpus de texto anotado se muestran en la tabla 7.2.

Idioma	Español
Formato	Código ASCII
Archivos	1
Líneas de texto	243
Palabras	4346

Tabla 7.2: Características generales del corpus de texto anotado.

En la siguiente sección, se describe cómo se entrenó el etiquetador automático y cómo se obtuvo el corpus de texto anotado.

7.1.3 Entrenamiento del Etiquetador Automático

A continuación se describe brevemente la forma en que se entrenó el etiquetador automático. Como resultado de dicho entrenamiento, se obtuvo el *corpus*

de texto anotado[2].

El etiquetador automático usado en esta tesis necesita un corpus de texto anotado. A partir de este corpus aprende las reglas para realizar futuras anotaciones de texto. Como es de esperarse, mientras más grande sea el corpus, mejor será el desempeño del algoritmo de aprendizaje y se obtendrán resultados más satisfactorios. Por ello, el corpus de texto anotado que se utilizó, fue generado de forma híbrida, empleando para ello tanto la anotación manual como el mismo etiquetador automático. De esta forma, se realizaron dos acciones apoyándose en los programas (utilerías) que la misma distribución del anotador automático ofrece: el etiquetamiento manual del texto y el entrenamiento del anotador automático.

Etiquetamiento manual de texto

En primer término, se expanden las abreviaturas, se apartan los signos de puntuación de las cadenas alfanuméricas y se separan los sintagmas dejando sólo uno por renglón. Después se asigna la etiqueta morfosintáctica correspondiente a cada palabra de acuerdo con el formato siguiente: *<palabra>/<etiqueta>*. Por ejemplo, el siguiente sintagma:

"Antología del Ensayo Hispánico"

Aparece anotado como sigue a continuación:

Antología/N00#~Sf **del**/P03###N **Ensayo**/N00#~Sm
Hispánico/A05~~SM

En este trabajo se utilizó un texto de 572 cadenas de caracteres y signos de puntuación, como primer corpus etiquetado manualmente.

Entrenamiento

Para llevar a cabo el entrenamiento se realizan varias acciones, a saber:

1. Dividir el corpus de texto anotado manualmente (en adelante CE) en dos partes. El algoritmo de aprendizaje utiliza la primera parte para

[2]El proceso de obtención del corpus de texto anotado sigue los pasos indicados en la documentación que viene incluida en la distribución del etiquetador automático utilizado. Véase la dirección electrónica de donde pueden obtenerse los programas del etiquetador en el apéndice A.

generar las reglas de transformación que asignan las etiquetas morfosintácticas más probables a palabras desconocidas. La segunda parte del corpus es empleada para generar las reglas contextuales de transformación.

2. Crear un corpus no etiquetado (CNE). Como ya se indicó, para asignar las etiquetas más probables a las palabras desconocidas se utiliza la primera parte del corpus anotado manualmente. Para ello es necesario retirar las anotaciones hechas manualmente al corpus.

3. Crear los archivos de listas, bigramas y lexicones. Para el entrenamiento del anotador, se requieren los archivos siguientes:

   - **Lista de palabras**. Contiene las palabras existentes en el CNE ordenadas por su frecuencia de aparición de forma descendente.
   - **Lista pequeña de palabras etiquetadas**. Es una lista del número de veces que una palabra está anotada con una cierta etiqueta en el CE. El formato de la lista es: <palabra> <etiqueta> <frecuencia>.
   - **Lista grande de bigramas**. Contiene todos los pares de palabras existentes en el corpus CNE.

4. El siguiente paso es ejecutar el programa de aprendizaje con los archivos antes creados. De acuerdo con la documentación del anotador automático, se utiliza el parámetro 30 para indicar que sólo use contextos de bigramas para las palabras que estén dentro de las 30 más frecuentes. Así, se gana eficiencia en el procesamiento.

5. A continuación se crean los lexicones de entrenamiento que ayudarán al programa a aprender reglas contextuales para mejorar la precisión del etiquetamiento. Aquí se utiliza la segunda parte del corpus que se obtuvo al inicio del entrenamiento. Los lexicones creados son los siguientes:

   - **Lexicón de entrenamiento**. Contiene líneas con palabras y sus posibles etiquetas, de la forma:
     <palabra> <etiqueta 1> <etiqueta 2> ... <etiqueta n>.
   - **Lexicón final**. Es el lexicón que será utilizado por el anotador automático cuando sea ejecutado con el fin de anotar nuevo texto.

- **Segundo corpus no etiquetado (CNE2)**. Es un corpus de texto no anotado. Se genera a partir de la segunda parte del corpus etiquetado creada al inicio del entrenamiento.

En este punto, el anotador está listo para etiquetar nuevo texto.

6. Continuar el entrenamiento con el algoritmo para que aprenda reglas contextuales. En esta parte se complementan las reglas de transformación aprendidas hasta ahora por el programa.

7. Como resultado se tiene un archivo de reglas contextuales, el cual es utilizado por el anotador automático.

   Después de esto, el etiquetador automático está listo para ser utilizado en la anotación de texto nuevo.

8. Etiquetamiento de nuevo texto. Una vez que se ha entrenado, el etiquetador se utiliza para seguir etiquetando texto nuevo y hacer crecer el corpus anotado.

9. Corrección de errores y reentrenamiento del anotador.

   Finalmente, se corrigen los errores que se presentan en la anotación de nuevo texto, con ello se tiene un corpus más grande a partir del cual puede volverse a entrenar el anotador y mejorar su precisión. Este proceso puede continuar hasta que se considere suficiente. En nuestro caso, se alcanzó un corpus de 4,345 palabras.

7.1.4 Etiquetario

El **etiquetario** utilizado para hacer las anotaciones se basa en la información obtenida de los etiquetarios y de los trabajos que a continuación se mencionan:

- Una propuesta de codificación morfosintáctica para corpus de referencia en lengua española. Tesis de Aurora Martín de Santa Olalla Sánchez [58].

- Proyecto de Fin de Carrera de Azucena Jiménez Pozo sobre categorización gramatical [38].

- Propuesta de marcaje gramatical del corpus en castellano [12].

- Manual de gramática española. Gramática de la palabra, de la oración y del texto [28].

En el etiquetario cada identificador de categoría gramatical (etiqueta) tiene un tamaño fijo de 7 caracteres, cuyo significado es el que se muestra en la tabla 7.3.

Posición del carácter	Significado
1	Clase
1-3	Subclase
4	Tiempo
5	Modo/Grado
6	Persona/Número
7	Género

Tabla 7.3: Las etiquetas se forman con 7 caracteres. A cada carácter se le asigna un significado según su posición.

En lo que se refiere a las **categorías gramaticales** (esto es, las clases de palabras en que se divide el corpus anotado), se encuentran representadas en el etiquetario las que se mencionan en la tabla 7.4, junto con el número de palabras del corpus que pertenecen a cada categoría gramatical. El etiquetario completo se muestra en el apéndice B.

A continuación se describe el corpus de texto con errores (conjunto de prueba).

7.1.5 Corpus de Texto con Errores

Para crear el **conjunto de prueba** se integró un **corpus de texto con errores**, para ello, se escogieron de forma aleatoria 10 archivos del corpus de texto, de entre aquellos que tienen de 9 a 20 líneas de texto.

Para hacer que los archivos del conjunto de prueba tuvieran errores, se llevaron a cabo dos tareas:

1. Insertar errores automáticamente.

2. Digitalizar los textos mediante un escáner y su reconocedor óptico de caracteres.

Clase	No. de palabras
Verbos	434
Preposiciones	614
Pronombres	172
Interjecciones	0
Conjunciones	279
Artículos	437
Adjetivos	562
Adverbios	140
Sustantivos	1033
Numerales	73
Fechas	21
Miscelánea	580

Tabla 7.4: Las 12 categorías gramaticales representadas en el etiquetario.

Textos	Líneas	Cadenas	Caracteres	Errores1	Errores2
1	15	79	587	7	5
2	17	92	634	7	8
3	16	126	831	13	2
4	11	147	1068	14	9
5	14	150	985	15	4
6	19	260	1770	25	28
7	17	310	2061	30	10
8	20	358	2045	55	52
9	17	591	3800	116	29
10	19	1153	6774	41	15
Total	**165**	**3,266**	**20,555**	**323**	**162**

Tabla 7.5: Características del corpus de texto con errores que se utiliza como conjunto de prueba para evaluar a los dos métodos presentados en este capítulo. La columna Errores1 muestra los errores insertados automáticamente. La columna Errores2 muestra los errores provenientes del reconocimiento óptico de caracteres, sólo se contemplan los errores que afectan a las palabras del texto.

El conjunto de prueba tiene las características indicadas en la tabla 7.5.

Para insertar errores automáticamente a los archivos, cada uno fue tratado bajo el siguiente procedimiento:

1. El texto del archivo es formateado de manera que queden 10 palabras por línea.
2. Se selecciona aleatoriamente una palabra de cada línea del archivo formateado.
3. También aleatoriamente, se define una posición dentro de la palabra. En esta posición se crea el error.
4. De forma aleatoria se selecciona uno de tres tipos de error:
   (a) Inserción: consiste en añadir un carácter a la palabra en la posición seleccionada anteriormente.
   (b) Borrado: consiste en eliminar el carácter de la palabra que está en la posición seleccionada.
   (c) Sustitución: consiste en cambiar el carácter de la posición seleccionada en el paso 3 por otro de la matriz de confusión escogido de forma aleatoria también.
5. La palabra modificada de esta forma, es reinsertada en la línea correspondiente.

Así, se tiene un texto con un error distinto (aleatorio) cada diez palabras.

La matriz de confusión utilizada para generar los errores de sustitución se muestra en el apéndice C. Es una adaptación de las que se presentan en los trabajos de Torres [90], Myka [65], y Sinha [83].

La digitalización de los textos se llevó a cabo utilizando un escáner *Hewlett Packard*, Modelo *ScanJet 3200c*. Cada texto se fotocopió una vez a partir de la impresión original, con el objetivo de reducir la calidad de la imagen y provocar con ello errores de reconocimiento. Las características del escáner se muestran en la tabla 7.6.

7.1.6 Lexicones

Los lexicones son utilizados por los métodos de corrección CATMC y CATCG para detectar posibles palabras erróneas. El primer método utiliza un lexicón,

Escáner	HP ScanJet 3200c
Software de Reconocimiento Óptico de Caracteres	HP Presicionscan LT
Resolución	600 x 1200 dpi
Formato de salida del texto	Texto (.txt) o Texto enriquecido (.rtf)

Tabla 7.6: El escáner utilizado para reconocer los documentos fotocopiados y su sofware de reconocimiento óptico de caracteres.

llamado general, que contiene palabras de todas las categorías gramaticales, de ahí su nombre. En tanto que el segundo método, hace uso de lexicones de categorías gramaticales, los cuales contienen exclusivamente palabras de cierta categoría gramatical.

Los lexicones fueron tomados del Diccionario del Español Usual en México [48], además de palabras tomadas del corpus de texto descrito anteriormente. En la tabla 7.7 se muestran las características de cada uno de ellos. Cabe aclarar que el lexicón correspondiente a la categoría Miscelánea no se presenta en esta tabla porque se insertó directamente en el código del programa y consiste en los símbolos distintos a las letras y números, por ejemplo: *; : , . ¡ ! ¿ ? {}[]*.

Por último, se menciona el formato de los lexicones. El contenido de cada lexicón se compone de: la forma ortográfica completa de cada palabra y sus n-gramas de letras. Las palabras con un tamaño menor que cuatro letras tienen sólo sus monogramas. Las palabras de cuatro letras o más, tienen bigramas y trigramas. Todas las palabras están en minúsculas.

La estructura de los lexicones es la siguiente:

palabra : monogramas Si el tamaño es <= 3 *letras*

palabra : bigramas+trigramas Si el tamaño es >= 4 *letras*

7.1.7 El Modelo de Lenguaje

Para modelar el lenguaje representado en el corpus de texto, se utilizaron bigramas de palabras. Dichos bigramas se calcularon revisando todos los sintagmas que componen el corpus de texto y llevando el conteo de la frecuencia de aparición de cada uno de los distintos pares de palabras. Se toma como palabra aquella secuencia de caracteres que está separada por espacios

Lexicón	A	B	C	D
General	131,648	42	1	17
Verbos	10,206	30	2	11
Preposiciones	32	13	1	7
Pronombres	83	8	2	5
Interjecciones	26	7	2	5
Conjunciones	38	20	1	8
Artículos	9	3	2	3
Adjetivos	5,508	26	2	14
Adverbios	1,516	23	2	12
Sustantivos	14,517	25	1	12

Tabla 7.7: Características de los lexicones de categorías gramaticales. Las columnas representan la siguiente información: A = No. total de palabras, B = Tamaño mayor de palabra, C = Tamaño menor de palabra y D = Tamaño promedio de palabra.

en blanco dentro del sintagma analizado. Al momento de realizar el conteo de parejas, se separan los signos de puntuación que tengan las palabras y se convierten de mayúsculas a minúsculas.

Cada bigrama está estructurado de la forma siguiente:

$$pal_i\ pal_{i+1} \mid frecuencia \qquad \text{para } i = 1 \ldots S-1$$

Donde:

S es el número de palabras de cada sintagma del corpus.

pal_i es la palabra que aparece en la i-ésima posición dentro del sintagma.

pal_{i+1} es la palabra que aparece inmediatamente después de la i-ésima posición dentro del sintagma.

$frecuencia$ es el número de veces que el bigrama aparece dentro del corpus.

Por ejemplo, en el siguiente sintagma se encuentran los bigramas que se enuncian en la tabla 7.8.

El modelo de lenguaje, tiene las características que se exponen en la tabla 7.9.

Sintagma	<I> el rostro femenino de la teología .
Bigramas	<I> el|160 el rostro|39 rostro femenino|5 femenino de|6 de la|25,489 la teología|848 teología .|41

Tabla 7.8: Ejemplo de bigramas de palabras con sus frecuencias.

Total de Bigramas	566,521
Frecuencia mayor de bigrama	25,542
Frecuencia menor de bigrama	1
Frecuencia media de bigrama	4

Tabla 7.9: Frecuencias mayor, menor y media de los bigramas.

Por último, cabe destacar que se aplicó el descuento Good-Turing a los bigramas del modelo de lenguaje, esto con el objetivo de calcular la probabilidad de cada uno de ellos de forma adecuada. Tomando en cuenta el promedio de frecuencias del modelo de lenguaje, se tomó 4 como el umbral de descuento para recalcular las probabilidades de cada bigrama.

En la tabla 7.10 se muestran algunos bigramas con sus frecuencias y con sus probabilidades después de que se les aplicó el descuento Good-Turing.

Bigramas sin descuento	**Bigramas con descuento**
demasiada prisa|1	demasiada prisa|0.358617581894879
y consolidado|2	y consolidado|1.19180721545851
el derroche|3	el derroche|2.16756870738736

Tabla 7.10: Ejemplo de bïgramas de palabras y sus probabilidades recalculadas mediante el descuento Good-Turing.

En las siguientes secciones se describen los métodos de corrección ortográfica implementados. En primer término se menciona el método CATMC y posteriormente se expone el método propuesto en esta tesis, denominado CATCG.

7.2 Corrección Morfológica y Contextual

En términos generales, el método CATMC realiza el proceso de corrección automática como se expone en los siguientes puntos:

1. **Detección contextual**. Para realizar la detección contextual de errores, se revisa cada una de las palabras del texto siguiendo dos pasos:

   - Búsqueda en el lexicón general. Cada palabra se compara con las palabras de un lexicón general, de no encontrarse entre éstas, se considera que esa palabra es incorrecta.
   - Búsqueda en el modelo de lenguaje. Se forman dos bigramas: uno toma la palabra anterior y la palabra en turno y el otro toma la palabra en turno y la palabra posterior. Estos bigramas se buscan dentro del modelo de lenguaje, de no encontrarse, se considera como incorrecta la palabra en turno.

2. **Generación de correcciones potenciales**. Se realiza un proceso de búsqueda de correcciones candidatas (o potenciales) en un lexicón de n-gramas de letras. Dependiendo del tamaño de la palabra incorrecta, los n-gramas pueden ser monogramas, bigramas o trigramas.

3. **Corrección**. Una vez que se tiene el conjunto de correcciones potenciales, se analiza contextual y morfológicamente con el objetivo de determinar cuál es la más parecida a la palabra detectada como errónea. El análisis contextual consiste en verificar cuál corrección potencial es más frecuente en el modelo de lenguaje basado en bigramas de palabras. De esta manera, se crea el conjunto de palabras sustitutas, las cuales se incorporarán al texto de entrada intercambiándolas por las palabras erróneas correspondientes.

En la figura 7.1 se muestra la arquitectura del método CATMC.

Enseguida se detallan los componentes de este método y su funcionamiento.

7.2.1 Componentes del Método CATMC

Este método utiliza los siguientes datos[3]:

[3]Estos datos fueron descritos al inicio de este capítulo.

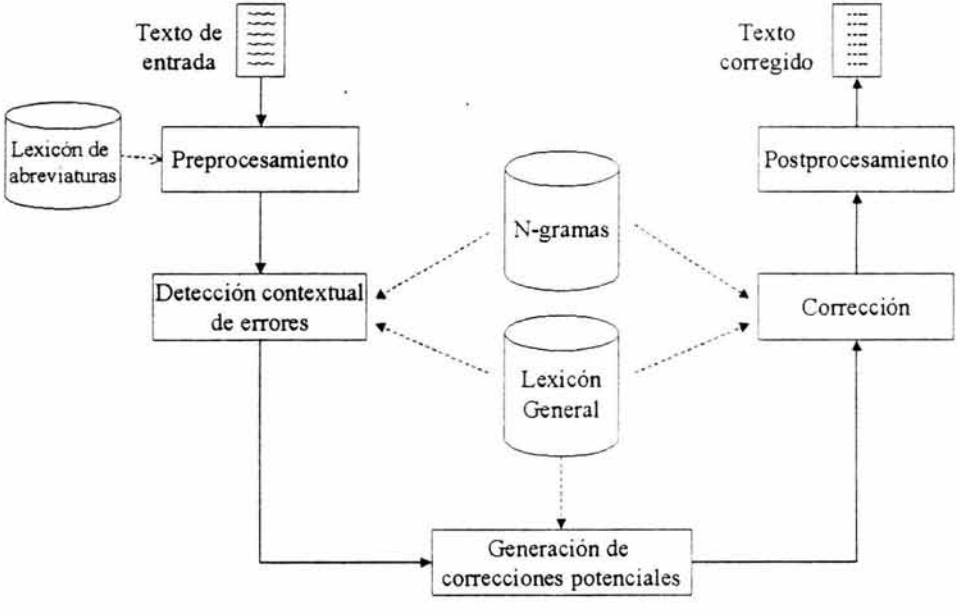


Figura 7.1: Arquitectura del método de corrección automática morfológico-contextual de textos electrónicos, denominado CATMC.

1. Un lexicón general

2. Un modelo de lenguaje basado en bigramas

El método CATMC se compone de los módulos que se mencionan a continuación, los cuales implementan las técnicas descritas en el capítulo 6.

1. **Módulo de preprocesamiento**. Se encarga de darle al texto un formato apropiado para las fases del proceso de corrección. Lleva a cabo las siguientes tareas:

   - Obtención del texto.
   - Expansión de abreviaturas.
   - Separación de sintagmas y signos
   - Separación de mayúsculas y minúsculas.

2. **Módulo de detección de errores**. Determina si una palabra del texto es incorrecta dentro de la lengua española. En primer término, crea los lexicones de búsqueda y posteriormente busca las palabras en éstos. Este módulo realiza las actividades siguientes:

   - Crea el lexicón de búsqueda de palabras (LP).
   - Crea el lexicón de búsqueda de correcciones potenciales de n-gramas de letras (LCP).
   - Crea el lexicón de búsqueda del modelo de lenguaje (LN).
   - Busca en el lexicón LP cada palabra del texto.
   - Busca en el lexicón LN los bigramas que se pueden formar con las palabras del texto.

3. **Módulo de generación de correcciones potenciales**. Obtiene el conjunto de correcciones potenciales para cada palabra del texto que fue detectada como incorrecta. Para ello, realiza una búsqueda en el lexicón LCP.

4. **Módulo de selección de correcciones potenciales**. Decide cuál de las palabras de cada conjunto de correcciones potenciales sustituirá finalmente a las palabras detectadas como incorrectas. Lleva a cabo una ponderación de las correcciones potenciales mediante las actividades siguientes:

- Análisis de n-gramas de palabras y de letras.
- Cálculo de la mínima distancia de edición entre candidatas con igual ponderación morfológica.
- Comparación de esqueletos.
- Comparación de tamaño de palabras y de esqueletos.

5. **Módulo de postprocesamiento.** Se encarga de sustituir las palabras incorrectas por sus respectivas correcciones y regresarle al texto su aspecto original. Por ello, realiza las siguientes tareas:

   - Sustitución de correcciones candidatas.
   - Restitución de mayúsculas.
   - Creación del archivo de texto corregido.

En las siguientes secciones se detalla el funcionamiento del método CATMC, el cual lleva a cabo el proceso de corrección automática en 5 fases: preprocesamiento, detección de errores, generación de correcciones potenciales, selección de la corrección potencial final y postprocesamiento.

7.2.2 Preprocesamiento

En esta fase, el método CATMC carga en memoria el contenido de los siguientes archivos de texto plano (código ASCII):

1. El texto de entrada.
2. El modelo de lenguaje (bigramas).
3. El lexicón general.
4. El lexicón de abreviaturas.

Una vez que ha realizado lo anterior, se crea el archivo donde se almacenará el resultado de la corrección y se realiza un *proceso de expansión de abreviaturas y separación de sintagmas y signos ortográficos.* Este proceso es mostrado en la figura 7.2.

En la parte de expansión de abreviaturas, se identifica si una palabra es encontrada en la lista de abreviaturas, de ser así, se sustituye por la palabra no abreviada. Esto se lleva a cabo con el fin de evitar que esas

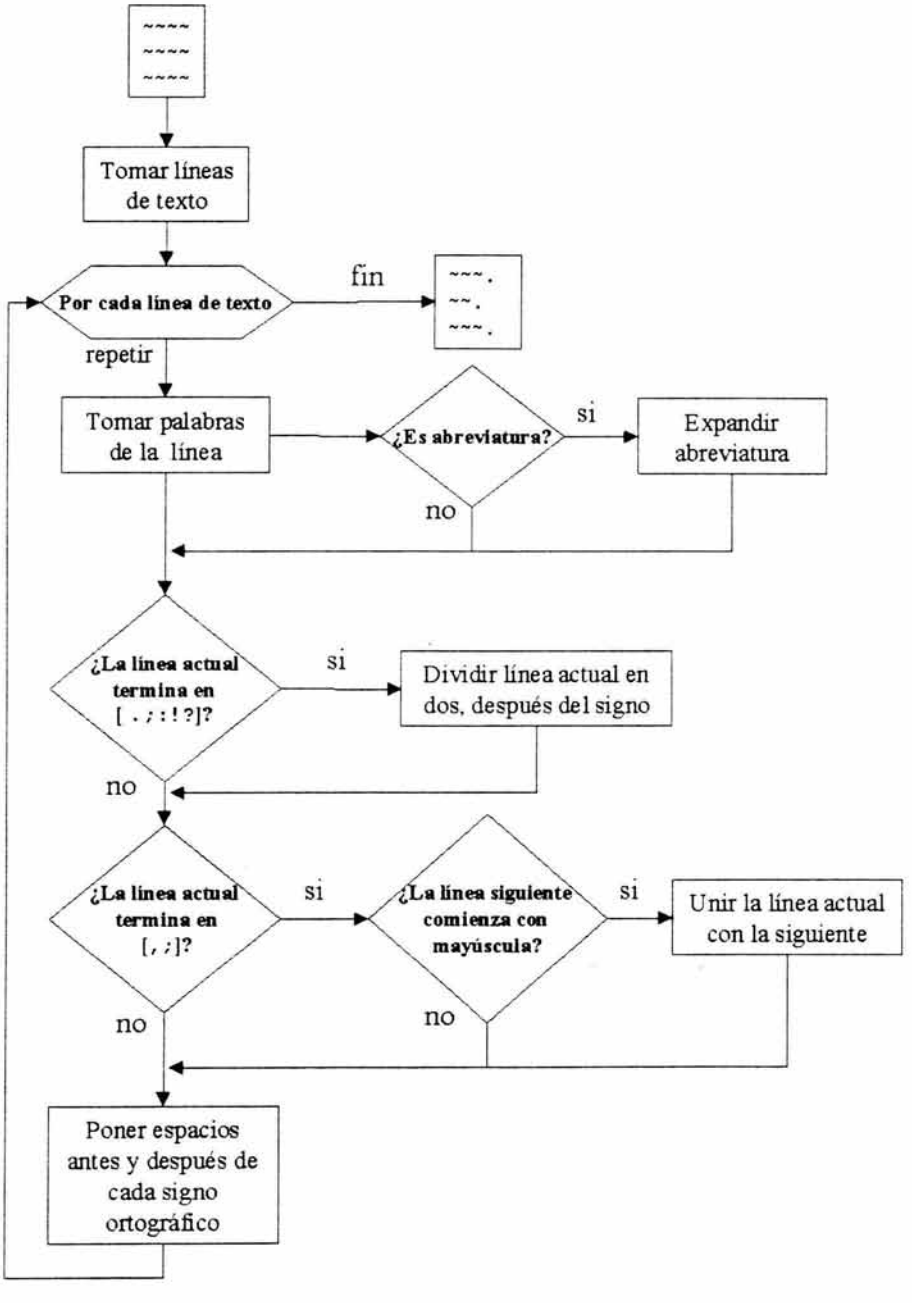


Figura 7.2: Proceso de expansión de abreviaturas y separación de sintagmas y signos ortográficos.

palabras se tomen como errores durante la fase de detección. A continuación, el método CATMC toma las palabras del texto de entrada, y realiza un proceso de separación de los signos de puntuación. Además, separa el texto en sintagmas, tomando como criterio si existe un punto (.), un punto y coma (;), un signo de interrogación o admiración que cierra (? !) o bien, dos puntos (:). Algunos de los caracteres que son separados en esta etapa son los siguientes:

" ' < > { } ( ) - ¿ ? ¡ ! | [ ] . : , ; $ #
+ − % & ^ \ / = @ _ ` ~ ¢ £ ¥
ª ¬ º ® ¯ ° ± ¹ ² ³ µ ¶ ÷ Ø ©

De esta forma, los caracteres antes listados, son tomados como una palabra, pero no afectan a la fase de detección de errores puesto que están aislados de las palabras a las que estaban ligados.

Además, en esta fase se lleva a cabo la separación de las letras mayúsculas de las palabras. El objetivo es que todas las palabras del texto estén en minúsculas, ya que en la fase de detección de errores no se toman en cuenta las mayúsculas. Las letras mayúsculas son cualquiera de las letras del alfabeto español que se muestran a continuación:

A B C D E F G H I J K L M N Ñ O P Q R S T U V W X Y Z Á É Í Ó Ú Ü

Mientras que las minúsculas son las letras siguientes:

a b c d e f g h i j k l m n ñ o p q r s t u v w x y z á é í ó ú ü

En la figura 7.3 se muestra cómo se lleva a cabo esta separación de mayúsculas.

Con el fin de no perder la información de cuáles palabras tienen letras mayúsculas, se realiza lo siguiente:

- En palabras que inician con una mayúscula, se substituye la mayúscula encontrada, por la letra minúscula que le corresponde y se marca la palabra con el identificador *MAY*, que significa que contiene una letra inicial mayúscula (exceptuando la Ü, ya que en español no existen palabras con Ü inicial y posiblemente sea un error).

- Si es una palabra con todas las letras mayúsculas, se sustituyen todas sus letras por minúsculas. Las palabras se marcan con el identificador *TMAY*.

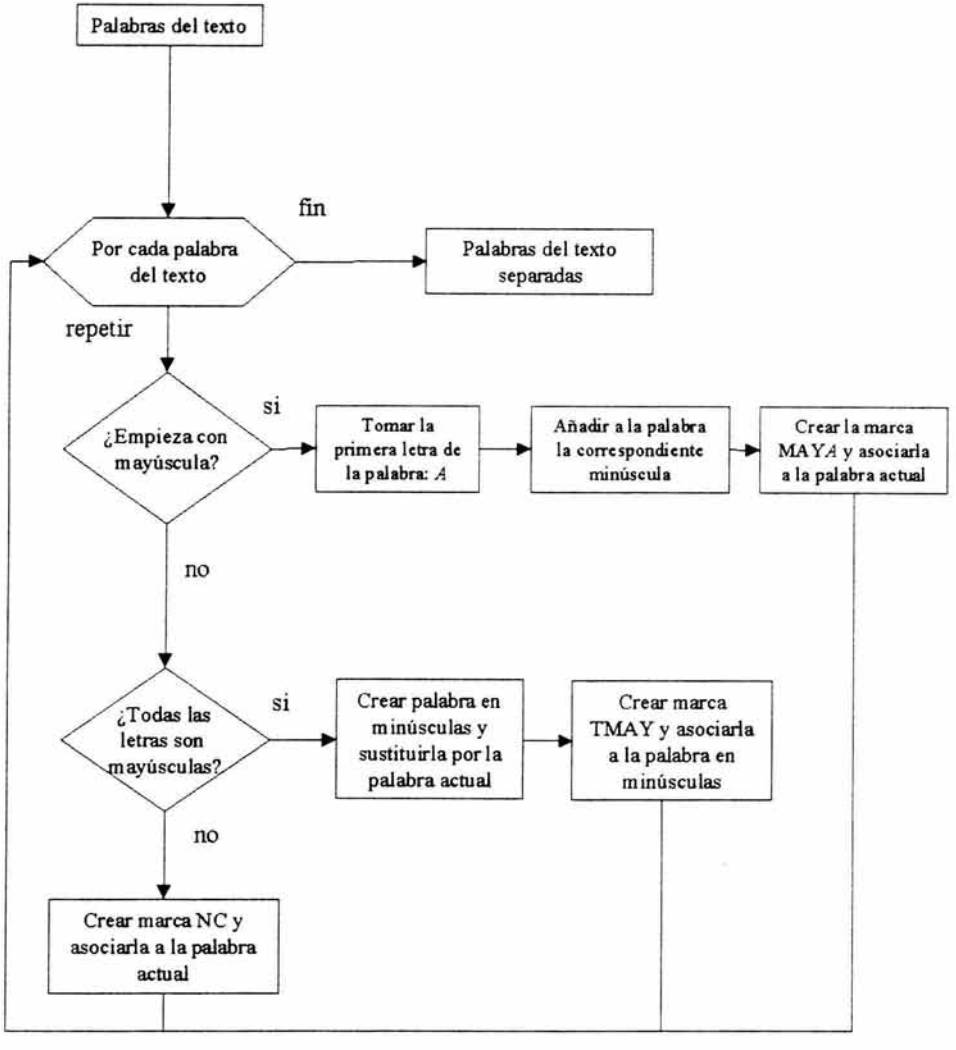


Figura 7.3: Proceso de separación de mayúsculas y minúsculas.

- Las palabras restantes no sufren cambio alguno. Éstas se marcan con el identificador *NC* (No hay carácter). En este caso puede ser que existan palabras con mayúsculas intermedias, las cuales serán revisadas como están porque posiblemente esas mayúsculas se deban a un error.

De esta forma, todas las palabras del texto constituyen un binomio con la siguiente estructura:

[TMAY] [palabra]
[MAY] [palabra]
[NC] [palabra]

En los siguientes ejemplos se muestran palabras que están marcadas con alguno de los identificadores antes mencionados.

yo = [NC][yo]
vencida = [NC][vencida]
*Valle, = [MAY**V**][**v**alle]*
*Nació = [MAY**N**][**n**ació]*
*Pedro = [MAY**P**][**p**edro]*
*TODAS = [**T**MAY][todas]*

De esta manera, quedan a disposición de las otras fases únicamente los caracteres que integran las palabras (con las letras en minúsculas). Por último, se inserta al principio de cada sintagma la pseudopalabra: *<I>* (que indica inicio de sintagma). Esto se realiza porque en la fase siguiente se buscan los bigramas de las palabras que forman el sintagma tomando como pareja la palabra anterior, y obviamente, la primera palabra no tiene una palabra precedente con que sea buscada en el modelo de lenguaje.

7.2.3 Detección de Errores

Esta fase consta de dos partes: creación de los lexicones de búsqueda y detección contextual de errores. La primera es indispensable porque estructura los datos donde se realizará la búsqueda. Por su parte, la búsqueda identifica los errores en el texto de entrada. A continuación se describen estas dos partes de la detección de errores.

Lexicones de Búsqueda

A partir del lexicón general, se crean los lexicones de búsqueda de palabras, de correcciones potenciales y de bigramas. La forma en que se almacenan estos datos es mediante la generación de una clave a partir de cierta información de cada palabra del lexicón general. Lo anterior se realiza con el objetivo de evitar una búsqueda secuencial, lo que sería muy caro computacionalmente.

La generación de la clave antes mencionada se lleva a cabo aprovechando la característica del lenguaje de programación empleado[4], que permite usar las estructuras de datos llamadas "arreglos asociativos". Un arreglo asociativo genera una clave a partir de un índice para identificar qué elemento se requiere utilizar, de la siguiente forma: *{índice} = valor.*

Además, se pueden crear arreglos multiclaves, los cuales permiten que un dato tenga dos índices que lo identifiquen. De este modo, se crea una tabla *hash* con cada lexicón. Es por ello, que lo que aquí se denomina lexicón de búsqueda es solamente un arreglo asociativo.

A continuación se indica la forma en que se integra el índice de cada palabra en los lexicones de búsqueda.

1. El lexicón de búsqueda de palabras (LP) tiene como índices los datos siguientes:

   {Tamaño de la palabra}{palabra}

   Por ejemplo: *{1}{a}, {2}{la}, {4}{acre}.*

   El valor de cada elemento para todos los casos es 1, esto es, el número de veces que aparecen en el lexicón general.

2. En el lexicón de correcciones potenciales (LCP), cada elemento tiene como valor los n-gramas de letras de la palabra buscada, esto es, *{Índice de búsqueda} = n-gramas.*

   El índice de búsqueda se integra con el conjunto siguiente:

   {Tamaño de la palabra,n-grama de búsqueda}

   A su vez, el n-grama de búsqueda puede estar formado de varias maneras, dependiendo del tamaño de las palabras. En los siguientes incisos se muestra cómo se forma el n-grama de búsqueda:

[4] Véase el apéndice A donde se muestran las herramientas de desarrollo utilizadas.

- Si el tamaño de la palabra es *menor a cuatro letras*: toma las x primeras letras, donde $x = 1 \ldots 3$. Con cada letra distinta se creará un índice diferente. En la tabla 7.11 se muestran algunos ejemplos.

A	B	C
a:a	1,a	a:a
tú:t,ú	2,t	tú:t,ú
tú:t,ú	2,ú	tú:t,ú
asa:a,s,a	3,a	asa:a,s,a
asa:a,s,a	3,s	asa:a,s,a
asa:a,s,a	3,a	asa:a,s,a

Tabla 7.11: Ejemplos de llaves del lexicón de búsqueda para palabras de tres letras o menos. A = Entrada del lexicón general, B = Clave del lexicón de búsqueda, C = Valor de clave en el lexicón de búsqueda.

En este último ejemplo sólo se crean dos llaves debido a que la tercera es la misma que la primera, por lo tanto, en el arreglo asociativo se suprime una de éstas dos puesto que crean la misma clave.

- Si el tamaño de la palabra es de cuatro letras: la llave se integra con los dos únicos trigramas que posea. Cada uno creará un índice diferente. En la tabla 7.12 se muestran ejemplos de esto.

A	B	C
acre:acr,cre+ac,cr,re	4,acr	acre:acr,cre+ac,cr,re
acre:acr,cre+ac,cr,re	4,cre	acre:acr,cre+ac,cr,re

Tabla 7.12: Ejemplos de llaves del lexicón de búsqueda para palabras de cuatro letras o más. A = Entrada del lexicón general, B = Clave del lexicón de búsqueda, C = Valor de clave en el lexicón de búsqueda.

- De otra manera: la llave se forma con los tres primeros trigramas que posea la palabra (cada uno creará un índice distinto). Por ejemplo, la entrada del lexicón general:
  abanderar:aba,ban,and,nde,der,era,rar+ab,ba,an,nd,de,er,ra,ar
  produciría los siguientes índices:

$$\{9,aba\} = abanderar{:}aba,ban,and,nde,der,era,rar + ab,ba,an,nd,de,er,ra,ar$$
$$\{9,ban\} = abanderar{:}aba,ban,and,nde,der,era,rar + ab,ba,an,nd,de,er,ra,ar$$
$$\{9,and\} = abanderar{:}aba,ban,and,nde,der,era,rar + ab,ba,an,nd,de,er,ra,ar$$

3. Por último, se crea el lexicón de búsqueda del modelo de lenguaje (LN) para almacenar los bigramas del modelo de lenguaje, con la estructura siguiente: $\{palabra1\} = palabra2 - frecuencia$

   Esto es, cada elemento del lexicón LN tiene como índice la primera palabra del bigrama, y su valor es la segunda palabra unida por un guión con la frecuencia del bigrama. Lo anterior permite identificar rápidamente los bigramas para una palabra dada.

Detección Contextual de Errores

Una vez que se han creado los lexicones de búsqueda, se procede entonces a identificar cuáles palabras del texto son incorrectas, es decir, se hace la detección contextual de errores. En esta parte se crea el conjunto de palabras incorrectas PE. Se toma como palabra errónea aquella que no se encuentra en los lexicones de búsqueda.

En el algoritmo 7.1 se indica cómo se realiza la detección de errores.

Algoritmo 7.1. Detección Contextual de Errores
Variables de Entrada:

P Lista de palabras del texto a corregir

LN Lexicón de búsqueda del modelo de lenguaje (n-gramas)

Variables de Salida:

PE Lista de palabras incorrectas

Variables Locales:

k Representa el índice de la palabra de P que se está revisando en cada iteración

P_k Es la k-ésima palabra de P

$|P|$ Es el número de palabras de la lista P

LP Es el lexicón de palabras

pa Palabra que se está revisando en cada iteración (P_k)

ant Palabra anterior a pa

$post$ Palabra posterior a pa

$bandera$ Indica si no ha sido encontrado un bigrama de pa

$< I >$ Pseudopalabra insertada al inicio de cada sintagma

Funciones Locales:

$agregar(l, e)$ Función que añade el elemento e al conjunto l

1. $k = 0$
2. Mientras $k < |P|$ hacer
   (a) $ant = P_{k-1}$
   (b) $pa = P_k$
   (c) $post = P_{k+1}$
   (d) Si $pa \neq < I >$ entonces
      i. Si $pa \notin LP$ entonces
         $agregar(PE, pa)$
      ii. si no
         Si $[ant, pa] \notin LN$ entonces
         a. Si $ant \notin LP$ entonces
            $agregar(PE, ant)$
         b. si no
            $bandera = 1$
         c. Si $[pa, post] \notin LN$ entonces
            1 Si $post \notin LP$ entonces
               $agregar(PE, post)$

2 si no
Si $post \notin PE$ entonces
$bandera = 2$

d. Si $bandera = 2$ entonces
$agregar(PE, pa)$

e. Si $bandera = 1$ entonces
Si $ant \notin PE$ entonces
$agregar(PE, pa)$

(e) $k = k + 1$

3. Regresar PE

Como el algoritmo lo indica, la detección se realiza palabra por palabra. Se verifica que las palabras del texto estén en el lexicón de búsqueda de palabras LP. De no encontrarse alguna, se asume que la palabra es incorrecta. Cuando si se encuentran en el lexicón LP, se procede a buscar cada palabra en el modelo de lenguaje, para ello, se crean dos bigramas con las palabras actual *pa*, anterior *ant* y siguiente *post*. Si no se encuentran los bigramas así formados, se verifica que las palabras *ant* y *post* estén en el lexicón LP. De esta forma, se entiende que el bigrama no existe porque una de las palabras que lo constituyen es incorrecta, ya sea la palabra anterior o la posterior, en cuyo caso se ingresan al conjunto PE.

Así, para que la palabra actualmente buscada *pa* se considere correcta, deben encontrarse los dos bigramas formados con las palabras anterior y posterior.

En el caso de que se encuentre sólo un bigrama y de que todas las palabras (*ant*, *pa* y *post*) estén en LP se considera la palabra actual como incorrecta sólo en el caso de que las otras dos no estén ya en el conjunto PE.

7.2.4 Generación de Correcciones Potenciales

Esta etapa se encarga de generar las correcciones potenciales de cada palabra detectada como incorrecta, de acuerdo con el parecido que tengan con ésta. La búsqueda de correcciones candidatas se lleva acabo en el lexicón LCP. Lo anterior se realiza en cuatro pasos:

1. Se calcula la longitud de la palabra, es decir, el número de letras que contiene. Esto se realiza para buscar en el lexicón LCP sólo aquellas

correcciones potenciales que tengan un tamaño similar, esto es: un carácter menos, el mismo número de caracteres y un carácter más.

2. Se crean los bigramas y en su caso, los trigramas de la palabra errónea, esto se determina de la siguiente manera:

   (a) Si el tamaño de la palabra es menor o igual a tres caracteres: se toman las letras de la palabra.
   Por ejemplo: *a = a, al = a,l, ala = a,l,a.*

   (b) Si el tamaño es mayor a tres caracteres se realizan dos operaciones:

      i. Crear los trigramas de la palabra. Por ejemplo:

      cóndor = cón,ónd,ndo,dor

      almendras = alm,lme,men,end,ndr,dra,ras

      óseo = óse,seo

      ii. Crea los bigramas de la palabra. Por ejemplo:

      Cóndor = có,ón,nd,do,or

      Almendras = al,lm,me,en,nd,dr,ra,as

      Óseo = ós,se,eo

   De tal forma, se crea una de las estructuras siguientes:

   palabra:monogramas

   palabra:trigramas+bigramas

   Por ejemplo:

   el e,l

   cóndor: cón,ónd,ndo,dor+có,ón,nd,do,or

   óseo: óse,seo+ós,se,eo

   almendras: alm,lme,men,end,ndr,dra,ras+al,lm,me,en,nd,dr,ra,as

   Naci6: Nac,aci,ci6+Na,ac,ci,i6

3. Después, se separan los trigramas y bigramas de cada palabra. Esto se hace para crear una, dos o tres "llaves de búsqueda"(según los n-gramas de la palabra), cada llave se integra con los primeros tres trigramas de la palabra errónea y con los siguientes tamaños de palabras: $|P|$ (El tamaño de la palabra calculado en el paso 1), $|P| - 1$ y $|P| + 1$. En el caso de palabras de menos de cuatro caracteres se toma el bigrama o el trigrama único. Por ejemplo, en la tabla 7.13 se muestra cada llave de búsqueda para la palabra *cóndor*.

Llave de búsqueda	**Valor de la llave**
{|P|}**{n-grama1}**	{6}{cón}
{|P + 1|}{n-grama1}	{7}{cón}
{|P - 1|}{n-grama1}	{5}{cón}
{|P|}**{n-grama2}**	{6}{ónd}
{|P + 1|}{n-grama2}	{7}{ónd}
{|P - 1|}{n-grama2}	{5}{ónd}
{|P|}**{n-grama3}**	{6}{ndo}
{|P + 1|}{n-grama3}	{7}{ndo}
{|P - 1|}{n-grama3}	{5}{ndo}

Tabla 7.13: Llaves de búsqueda

De la misma forma se integran llaves de búsqueda para los n-gramas de las palabras siguientes:

[alm][lme][men] para *almendras*

[a][l][a] para *ala*

[a] para *a*

4. Con cada llave de búsqueda se obtienen del lexicón de búsqueda las palabras que tengan como índice esa llave y se crea el conjunto de correcciones potenciales (CP). Por ejemplo, para la palabra errónea *naci6*, se tendría un conjunto de correcciones potenciales como el siguiente:

[nación, nació, nacido, nacida, hacina, yacio, yacija, vació, vacile, vaciar, tracio, tinaco, tacita, sació, saciña, sacio, saciar, sacia, reacio, ración, racimo, racima, nacían, nací, naco, hacia, guacia, gracia, glacis, facies, facial, dación, dacio, cacica, ación, acirón].

En el caso de que no se encuentre ninguna palabra candidata, la palabra incorrecta se marca con un identificador *NHS* (No Hay Sustituta), para que no haya más procesamiento puesto que se toma la misma palabra detectada como errónea para sustituirla en el texto original.

7.2.5 Selección de Correcciones Potenciales

En esta fase se determina cuál de las palabras del conjunto de correcciones potenciales sustituirá a la palabra errónea en el texto original. Para ello, se realiza una ponderación contextual y morfológica y una selección de las correcciones potenciales para cada palabra incorrecta detectada, con lo cual se crea el conjunto de palabras sustitutas PS.

En el algoritmo 7.2 se muestra cómo se realiza la ponderación de las palabras de CP, mientras que en el algoritmo 7.3 se muestra la forma en que se hace la selección de la mejor corrección potencial de una palabra incorrecta.

Algoritmo 7.2. Ponderación Contextual y Morfológica
Variables de Entrada:

PE Conjunto de palabras incorrectas detectadas

CP Conjunto de correcciones potenciales de cada palabra incorrecta

Variables de Salida:

CPP Conjunto de correcciones potenciales ponderadas

Variables Locales:

w Guarda el índice de la palabra que se está analizando en cada iteración

s Guarda el índice de la corrección potencial que se está analizando en cada iteración

p Es la palabra que se está analizando actualmente

$CP_{p,s}$ Es la corrección potencial de p que se analiza en la iteración s

ant Palabra que precede a la actual

post Palabra posterior a la actual

coinciden Guarda el número de coincidencias entre los n-gramas de la palabra actual y la corrección potencial en turno

$|PE|$ Tamaño de PE (número de palabras)

$|p|$ Tamaño de p (número de caracteres)

$|CP_{p,s}|$ Tamaño de $CP_{p,s}$

$Ngrp$ Guarda los n-gramas de letras de p

$Ngrps$ Guarda los n-gramas de letras de $CP_{p,s}$

$bgrs$ Guarda el número de bigramas de letras comunes entre dos palabras

Funciones Locales:

$hazNgramasL(pal)$ Crea los n-gramas de letras de la palabra *pal* que recibe

$comparaTrigrs(a, b)$ Obtiene el número de trigramas de letras comunes a dos palabras a y b

$comparaBigrs(a, b)$ Obtiene el número de bigramas de letras comunes a dos palabras a y b

$max(a, b)$ Función que devuelve el mayor de sus dos argumentos a y b

$concatena(c, v)$ Asocia al elemento c el valor v

$ingresa(l, e)$ Ingresa el elemento e, al conjunto l

1. $w = 0$
2. $s = 0$
3. Mientras $w < |PE|$ hacer
   (a) $p = PE_w$
   (b) Si $p \neq NHS$ entonces
      i. $Ngrp = hazNgramasL(p)$
      ii. $ant = PE_{w-1}$

iii. $post = PE_{w+1}$

iv. Mientras $s < |CP_{p,s}|$ hacer

a. $Ngrps = hazNgramasL(p)$

b. $coinciden = 0$

c. Si $[ant, p] \in LN$ entonces

$coinciden = coinciden + 1$

d. Si $[p, post] \in LN$ entonces

$coinciden = coinciden + 1$

e. $coinciden = coinciden + comparaTrigrs(Ngrp, Ngrps)$

f. $bgrs = comparaBigrs(Ngrp, Ngrps)$

g. $coinciden = coinciden + bgrs$

h. $coinciden = \frac{coinciden}{max(|p|,|CP_{p,s}|)}$

i. $concatena(CP_{p,s}, coinciden)$

j. $ingresa(CPP, CP_{p,s})$

v. $s = s + 1$

(c) $w = w + 1$

4. Regresar CPP

A continuación, se explica el algoritmo 7.2 y se dan algunos ejemplos.

En primer lugar, una vez que se tiene el conjunto CP, se realiza una primera ponderación contextual de cada una de las palabras sustitutas de CP, se forman bigramas con las palabras del texto anterior y posterior a la palabra incorrecta analizada. Por ejemplo, para la palabra incorrecta inicial *nació*, y tres de sus correcciones potenciales: *nación*, *nació* y *nacido*:

Bigramas anteriores	**Bigramas posteriores**
<I> nación	nación en
<I> nació	nació en
<I> nacido	nacido en

Así, de encontrarse en LN los bigramas (de palabras) formados con las correcciones potenciales y las palabras anterior y posterior a la palabra incorrecta analizada, se aumenta en una unidad la ponderación para las correcciones potenciales por cada bigrama encontrado.

Después de la ponderación morfológica, se identifica el número de bigramas y trigramas idénticos que tenga cada palabra sustituta con la palabra errónea. En la tabla 7.14 se muestra un ejemplo de esto.

Palabra	Trigramas	Bigramas	Coincidencias
naci6	**nac, aci**, ci6	**na, ac, ci**, i6	
Nación	**nac, aci**, ció, ión	**na, ac, ci**, ió, ón	5
Nació	**nac, aci**, ció	**na, ac, ci**, ió	5
Dacio	Dac, **aci**, cio	da, **ac, ci**, io	3
Hacia	Hac, **aci**, cia	ha, **ac, ci**, ia	3

Tabla 7.14: Ejemplos de palabras ponderadas mediante sus bigramas y trigramas de letras

El número de coincidencias de los n-gramas de letras de cada corrección potencial se suma a la ponderación contextual anterior. De esta forma, se tiene la ponderación final para cada una de las palabras del conjunto CP. Así, se obtiene el conjunto ponderado de correcciones potenciales CPP, las cuales son contrastadas en el paso siguiente para seleccionar una única corrección potencial.

Como segunda parte de esta fase, se lleva a cabo la selección de las palabras que sustituirán a las incorrectas. Esta selección se basa en la ponderación anterior e implementa las técnicas expuestas en el capítulo 6. El algoritmo 7.3 muestra la manera en que se realiza esta selección para cada palabra incorrecta.

Algoritmo 7.3. Seleccionar Corrección Final
Variables de Entrada:

PE Conjunto de palabras incorrectas detectadas

CPP El conjunto de correcciones potenciales ponderadas

Variables de Salida:

PS Conjunto de correcciones potenciales seleccionadas, es decir, las palabras que sustituirán a las incorrectas

Variables Locales:

$|PE|$ Número de palabras del conjunto PE

CPP_p Conjunto de correcciones potenciales ponderadas CPP para la palabra p

$|CPP_p|$ Número de palabras del conjunto CPP_p

w Guarda el índice del número de palabra de cada iteración

s Guarda el índice de la corrección potencial analizada en cada iteración

p Es la palabra analizada en cada iteración, es decir la n-ésima palabra de PE (PE_n)

$CP_{p,s}$ Es el elemento s del conjunto CP para la palabra p

ant Es la palabra anterior a p

post Es la palabra posterior a p

candA Es la corrección potencial número s de la palabra p

candF Es la corrección potencial final de la palabra p

pesoF Guarda el valor del peso final de p

pesoA Guarda el valor del peso de la corrección potencial $CP_{p,s}$

$frAA$ Probabilidad del bigrama [ant,candA]

$frPA$ Probabilidad del bigrama [candA,post]

$frPF$ Probabilidad del bigrama [ant,candF]

$frAF$ Probabilidad del bigrama [candF,post]

esqA Almacena el esqueleto de *ant*

esqF Almacena el esqueleto de *candF*

esqp Almacena el esqueleto de p

kA Almacena el número de coincidencias entre *esqA* y *esqp*

kF Almacena el número de coincidencias entre *esqF* y *esqp*

dA Almacena la diferencia de tamaño entre *esqA* y *esqp*

dF Almacena la diferencia de tamaño entre *esqF* y *esqp*

Funciones Locales:

mde(y, x) Calcula la mínima distancia de edición entre dos palabras x y y

esqueleto(x) Obtiene el esqueleto de la palabra x

ordena(C) Ordena de mayor a menor el conjunto C

buscaBigr$(p1, p2)$ Busca el bigrama formado con las palabras $[p1, p2]$ en LN y devuelve su probabilidad

concidencias$(e1, e2)$ Devuelve el número de coincidencias entre los esqueletos $e1$ y $e2$

ingresa(l, e) Ingresa el elemento e, al conjunto l

1. $w = 0$
2. $s = 0$
3. Mientras $w < |PE|$ hacer
   (a) $p = PE_w$
   (b) $pesoF = 0$
   (c) Si $CPP[p] \neq NHS$ entonces
      i. $ordena(CPP_p)$
      ii. Mientras $s < |CPP_p|$ hacer
         A. $pesoA =$ ponderación de la palabra $CPP_{p,s}$

B. $candA = CPP_{p,s}$

C. Si $pesoF > pesoA$ entonces

   a. candF = candA

   b. pesoF = pesoA

D. si no

   Si $pesoA = pesoF$ entonces

   i. $frAA = buscaBigrs(ant, candA)$

   ii. $frPA = buscaBigrs(candA, post)$

   iii. $frAF = buscaBigrs(p, candF)$

   iv. $frPF = buscaBigrs(candF, post)$

   v. Si $frAA > frAF$ entonces
   $pesoF = pesoF - 0.5$

   vi. Si $frAA < frAF$ entonces
   $pesoA = pesoA - 0.5$

   vii. Si $frPA > frPF$ entonces
   $pesoF = pesoF - 0.5$

   viii. Si $frPA < frPF$ entonces
   $pesoA = pesoA - 0.5$

   ix. Si $pesoA > pesoF$ entonces

      a. candidF = candA

      b. pesoF = pesoA

   x. si no

   Si $pesoF = pesoA$ entonces

   I $mdeA = mde(candA, p)$

   II $mdeF = mde(candF, p)$

   III Si $mdeA < mdeF$ entonces
   $candF = candA$

   IV si no

      A. $esqA = esqueleto(candA)$

      B. $esqF = esqueleto(candF)$

      C. $esqp = esqueleto(p)$

      D. $kA = coincidencias(esqA, esqp)$

E. $kF = coincidencias(esqF, esqp)$

F. Si $kA > kF$ entonces

$candF = candA$

G. si no

Si $kA = kF$ entonces

i) $dA = |esqA - esqp|$

ii) $dF = |esqA - esqp|$

iii) Si $dA < dF$ entonces

$candF = candA$

iii. $s = s + 1$

(d) $ingresa(PS, candF)$

(e) $w = w + 1$

4. Regresar PS

A continuación se describe el funcionamiento de este algoritmo.

La selección se realiza tomando, una por una, las palabras incorrectas y sus respectivas correcciones potenciales del conjunto CPP. En el caso de que la palabra errónea no esté marcada como *NHS*, se realizan tres pasos: ordenamiento, selección y si es necesario desambiguación.

- Ordenamiento. Se ordena el conjunto de correcciones potenciales de mayor a menor tomando como criterio el número de coincidencias de bigramas y trigramas calculado en la parte de ponderación de correcciones potenciales. De esta forma, se descartan las palabras con menor parecido morfológico y se toma la de mayor parecido, o en su caso, se desambiguan las que tengan una igual ponderación.

- Selección. La selección de la mejor candidata se realiza analizando el conjunto de correcciones potenciales palabra por palabra según la ponderación anteriormente mencionada. Cuando se encuentran dos candidatas con igual ponderación, se realiza un proceso de desambiguación, esto se repite iterativamente hasta que se encuentra una corrección potencial final o se termina de revisar todo el conjunto de correcciones potenciales. El proceso de desambiguación se describe enseguida.

- Desambiguación. Para decidir cuál es la corrección potencial más adecuada, se toma la corrección candidata anterior con igual peso y se compara con la más reciente. Se tienen las siguientes formas de desambiguar las palabras candidatas a ser la corrección final: análisis de bigramas de palabras, comparación de la mínima distancia de edición, comparación de los esqueletos de las palabras y comparación del tamaño de las palabras y de sus esqueletos. A continuación se explica cada una de ellas.

1. Análisis de bigramas. Se realiza una búsqueda en el modelo de lenguaje basado en bigramas de palabras para determinar cual es la corrección potencial más probable dentro del modelo. Para ello se toma la palabra anterior a la palabra errónea en el texto original. Nuevamente, se utiliza la pseudopalabra *<I>*, que indica que la palabra en cuestión está en el inicio de sintagma. Así, se tiene el bigrama formado por *<palabra anterior del texto> <corrección potencial>*.

   En la tabla 7.15 se tiene un ejemplo de análisis de bigramas para la palabra *Naci6*, con sus candidatas *nación* y *nació*, con igual ponderación de 5.

Palabra	Bigrama
nació	<I> nació
Nación	<I> nación

Tabla 7.15: Análisis de bigramas para *Naci6* y *nación* y *nació*

Se tienen las posibilidades siguientes:

(a) Se encuentran los bigramas dentro del modelo. En este caso, se comparan las probabilidades de los bigramas y se selecciona la mayor. La de menor probabilidad es descartada asignándole una ponderación morfológica menor, así, la corrección potencial seleccionada podrá contrastarse con otras candidatas con igual ponderación si éstas existen. La misma situación se da en el caso de que no se encuentre uno de los bigramas, esto es, alguna de las dos candidatas no aparece en el modelo, por lo que es descartada en favor de la que sí aparece.

(b) En el caso de no encontrarse ninguno de los dos bigramas, o de que sus probabilidades sean iguales, ninguna de las dos candidatas se descarta, sino que se deja el turno a la comparación de la mínima distancia de edición.

2. Comparación de la mínima distancia de edición. Aquí se calcula la mínima distancia de edición entre las correcciones potenciales y la palabra errónea. Se selecciona como la mejor corrección potencial aquella que tenga la menor distancia. Por ejemplo, la mínima distancia de edición entre la palabra incorrecta *naci6* y las palabras correctas *nación* y *nació* son:

   MDE(naci6,nación) = 3

   MDE(naci6,nació) = 2

   Como es evidente, la palabra que quedaría como corrección potencial final es: *nació*.

3. Comparación de esqueletos y tamaños. Si el análisis de bigramas y la mínima distancia de edición no son suficientes para desambiguar una palabra, se procede a calcular el esqueleto tanto de las palabras candidatas, como de la palabra errónea.

   Hay que tomar en cuenta que el esqueleto de una palabra, como se calcula aquí, no comprende las vocales ni los números, sino *sólo las consonantes sin repeticiones.* La razón de hacer esto es que, de acuerdo con lo expuesto en Pollock y Zamora [76] y Klein y Kopel [44], se tienen las consideraciones siguientes: a) las consonantes ofrecen mayor información que las vocales; b) el orden de las consonantes se preserva en la mayoría de las palabras; c) los errores generados por los reconocedores de caracteres ocurren en la primera letra con mayor probabilidad; además de que si una palabra tiene un número insertado incorrectamente (en sustitución de una vocal acentuada por ejemplo) podría interferir en la obtención del esqueleto y por ende en la comparación con los esqueletos de sus correcciones potenciales.

   Después de obtener los esqueletos, se calcula el número de coincidencias entre los esqueletos de las candidatas con el de la palabra errónea. Se toma como corrección final la palabra cuyo esqueleto tenga mayor parecido, por ejemplo:

   0puestos y sus candidatas: *cuestas* y *opuestos*

p s t :: c s t coincidencias: 2 letras

p s t :: p s t coincidencias: 3 letras

En cuyo caso, la palabra seleccionada es *opuestos* porque tiene el mayor número de coincidencias.

Sin embargo, cuando el número de coincidencias es igual, como en el caso siguiente:

0tra y sus candidatas:*otra* y *otras*

t r :: t r coincidencias: 2 letras

t r :: t r s coincidencias: 2 letras

Se procede a comparar el tamaño de la palabra, en este ejemplo resultaría *otras* como la mejor candidata, dado que el tamaño es el mismo: 4 caracteres.

Pero, existen casos en que no es así, como en el siguiente ejemplo:

0puestos y sus candidatas: *expuesta* y *opuestos*

p s t :: x p s t coincidencias: 3 letras

p s t :: p s t coincidencias: 3 letras

En esta situación, el tamaño de las tres palabras es de 8 letras y las coincidencias son las mismas, por ello, se tienen que comparar los tamaños de los esqueletos, así, resulta como mejor candidata *opuestos*, porque el tamaño de su esqueleto es 3, al igual que el esqueleto de la palabra errónea *0puestos*, en contraste con *expuesta*, que tiene un esqueleto de 4 letras.

Si finalmente, después de aplicar estos tres procesos de desambiguación, no es posible obtener la palabra candidata correcta, el método CATMC toma como la corrección candidata final la primera candidata encontrada en el conjunto de correcciones potenciales y continúa con el procesamiento de la siguiente palabra errónea.

7.2.6 Postprocesamiento

En esta fase se intercambian las correcciones candidatas finales seleccionadas en la fase anterior (el conjunto PS), con las palabras erróneas detectadas. Además, se crea el archivo corregido final. La inserción se realiza tomando en cuenta el número de palabra que se asigna en la fase de preprocesamiento.

Posteriormente se restituyen las mayúsculas a las palabras del texto. Para ello, se toman los binomios construidos con las marcas de mayúsculas y las palabras del texto. En este momento, el conjunto PE tiene ya las correcciones anteriormente insertadas, de esta manera, ocupan el lugar de las palabras incorrectas y también toman las marcas que les fueron asignadas en el preprocesamiento. Por ejemplo: *[NC] [practic6]* se corrige como *[NC] [practicó]*

Durante la restitución se ignoran las partes de los binomios que tienen la marca *NC*, puesto que significan que no debe hacerse ninguna modificación a la palabra, como en el siguiente ejemplo:

[NC] [practicó] se convierte en *[practicó]*

A las palabras que tienen la marca *TMAY*, se les intercambian todas sus letras por letras mayúsculas. Por ejemplo:

[TMAY] [todas] se convierte en *[TODAS]*

Si se detecta la marca que indica la presencia de una mayúscula inicial (*MAY*), se toma el carácter inmediato siguiente y se compara con el carácter inicial de la palabra en cuestión. Si se encuentra que la letra inicial de la palabra marcada con *MAY* ha cambiado (gracias al proceso de corrección), se toma como letra inicial la primera letra de la palabra corregida en lugar de la letra de la marca. Por ejemplo:

*[MAY**N**] [**n**ació]* es convertido en *[MAY**N**] [**N**ació]*
*[MAY**P**] [valle]* es convertido en *[MAY**V**] [valle]*
con lo que se obtendrá *Valle* y no *Palle*

Una vez que se han restituido adecuadamente las mayúsculas, el método CATMC almacena las palabras del texto de entrada, la cuales incluyen ya las palabras corregidas y los signos de puntuación y mayúsculas originales.

En la siguiente sección, se expone el método de corrección automática de textos electrónicos propuesto en esta tesis, el CATGC.

7.3 Corrección Morfológica y Contextual Basada en la Asignación de Categorías Gramaticales

La propuesta que se hace en este trabajo es utilizar un método de corrección morfológica y contextual que se base en la asignación de categorías gramaticales a las palabras del texto de entrada, denominado CATCG. La diferencia con el método CATMC anteriormente descrito es que el método CATCG usa información morfosintáctica para crear el conjunto de correcciones potenciales para cada palabra identificada como errónea. De esta manera, determina las palabras sustitutas, seleccionándolas de un lexicón que tiene únicamente palabras de la misma categoría gramatical que la palabra incorrecta.

El fundamento de este método está en el concepto de paradigma, introducido en el capítulo 6, que nos indica que dos palabras pertenecen al mismo paradigma únicamente si pueden sustituirse (de forma válida dentro de la lengua) mutuamente en un mismo sintagma. Los paradigmas aquí utilizados se basan en la clasificación de las palabras del español en categorías gramaticales, también llamadas partes del discurso.

De modo general, el método CATCG realiza el proceso de corrección automática mediante los siguientes pasos:

1. **Etiquetamiento automático**. Determina, mediante un etiquetador automático de textos basado en TBL, las categorías gramaticales de las palabras del texto de entrada. Esto incorpora información morfosintáctica al corrector, pues se identifica cuál es el uso que tiene una palabra dentro del sintagma al que pertenece de acuerdo con su categoría gramatical.

2. **Detección contextual de errores**. Basándose en la clasificación de las palabras del texto obtenida en el etiquetamiento, lleva a cabo una detección de palabras erróneas mediante la búsqueda en los lexicones de categorías gramaticales correspondientes a cada palabra del texto de entrada y a través de un análisis de bigramas de palabras.

3. **Generación de correcciones potenciales**. Una vez que detecta todas las palabras incorrectas en el texto, realiza una búsqueda de las correcciones potenciales de cada palabra incorrecta. Lo anterior se realiza en los lexicones de categorías gramaticales correspondientes a cada

palabra incorrecta, según la categoría a la que pertenezca. De esta forma, se evita tomar como mejores correcciones aquellas palabras que no estén relacionadas (sintácticamente) con el sintagma al que pertenece la palabra incorrecta pero cuyo parecido sea demasiado.

4. **Corrección**. Cuando se tiene el conjunto de correcciones potenciales, se procede a seleccionar la más adecuada aplicando un análisis contextual y morfológico. Una vez determinadas las mejores candidatas, éstas sustituyen a las palabras erróneas en el texto de entrada.

En la figura 7.4 se muestra la arquitectura del método CATCG.

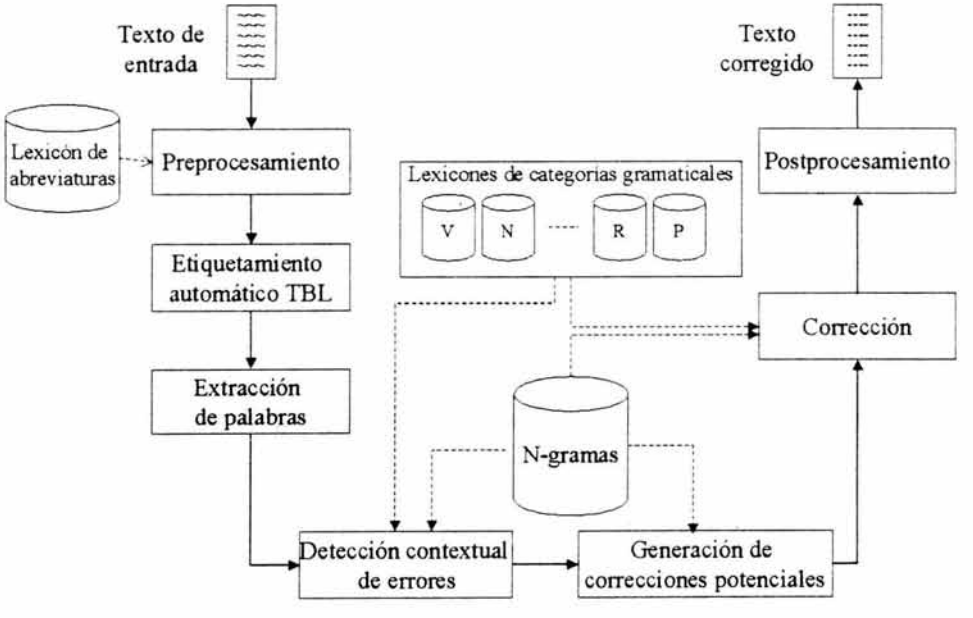


Figura 7.4: Método de corrección ortográfica automática de textos electrónicos basado en la identificación de las categorías gramaticales de las palabras (CATCG)

A continuación se presentan los componentes y funcionamiento del método CATCG.

7.3.1 Componentes del Método CATCG

El método CATCG utiliza los siguientes datos de entrada (descritos al principio del capítulo):

1. Lexicón de Verbos
2. Lexicón de Adverbios
3. Lexicón de Artículos
4. Lexicón de Conjunciones
5. Lexicón de Preposiciones
6. Lexicón de Pronombres
7. Lexicón de Adjetivos
8. Lexicón de Sustantivos
9. Lexicón de Interjecciones
10. Lexicón de Miscelánea
11. Modelo de lenguaje basado en bigramas

Se compone de los módulos siguientes:

1. **Módulo de preprocesamiento**. Se encarga de darle el formato adecuado al texto de entrada para que pueda ser etiquetado morfosintácticamente. Realiza las actividades siguientes:

   (a) Obtención del texto.

   (b) Expansión de abreviaturas y separación de sintagmas y palabras.

2. **Módulo de etiquetamiento automático**. Obtiene un texto etiquetado con las categorías gramaticales de las palabras que lo forman. Utiliza un anotador automático de textos basado en TBL.

3. **Módulo de extracción de palabras**. Este módulo separa las etiquetas morfosintácticas y la marcación de mayúsculas de las palabras del texto a corregir. Obtiene la lista de palabras del texto de entrada.

4. **Módulo de detección de errores**. Necesita como datos de entrada los lexicones de búsqueda de palabras y correcciones potenciales de cada categoría gramatical (lexicones de n-gramas de letras), y el lexicón de búsqueda del modelo de lenguaje LN (n-gramas de palabras). Este módulo busca las palabras incorrectas en el texto de entrada. Para ello realiza lo siguiente:

   - Búsqueda en los lexicones de categorías gramaticales.
   - Búsqueda en el modelo de lenguaje de todos los bigramas que pueden formarse con las palabras del texto.

5. **Módulo de generación de correcciones potenciales**. Crea el conjunto de correcciones potenciales (CP) para cada palabra detectada como incorrecta. Efectúa una búsqueda en el lexicón de n-gramas de la categoría gramatical correspondiente a cada palabra incorrecta.

6. **Módulo de selección de correcciones candidatas**. Su objetivo es obtener la corrección candidata más adecuada para cada palabra incorrecta. Para conseguirlo lleva a cabo las siguientes tareas:

   (a) Análisis de n-gramas de letras y palabras.

   (b) Cálculo de la mínima distancia de edición entre correcciones con igual ponderación morfológica.

   (c) Comparación de esqueletos.

   (d) Comparación de tamaño de palabras y esqueletos.

7. **Módulo de postprocesamiento**. Una vez que se han encontrado las correcciones finales para las palabras incorrectas, se sustituyen en el texto. Posteriormente se regresan las mayúsculas a las palabras correspondientes y se crea el archivo corregido final. Esto se realiza en los pasos que se mencionan a continuación:

   (a) Sustitución de correcciones candidatas.

   (b) Restitución de mayúsculas.

(c) Creación del archivo de texto corregido.

El método CATCG realiza el proceso de corrección en 7 etapas: preprocesamiento, etiquetamiento, extracción de palabras, detección de errores, generación de correcciones potenciales, selección de la corrección potencial final y postprocesamiento. A continuación se expone el funcionamiento de cada una de éstas.

7.3.2 Preprocesamiento

Esta etapa es similar a la fase de preprocesamiento descrita en la sección 7.2.2, sin embargo, en el modelo CATCG tiene como función únicamente preparar el texto de entrada para que sea procesado por la siguiente fase, para ello, le asigna el formato que se menciona a continuación:

- Los signos de puntuación deben estar separados de cada palabra.
- Debe haber un sólo sintagma por renglón.

Este paso es necesario debido a que el método CATCG utiliza el anotador automático basado en el algoritmo TBL, descrito en el capítulo 6, que requiere que el texto a etiquetar esté en el formato indicado.

Para darle al texto el formato antes mencionado, se lleva a cabo un proceso de expansión de abreviaturas y separación de palabras y sintagmas, el cual es mostrado en la figura 7.2 y consiste en lo siguiente:

1. Cargar en memoria el lexicón de abreviaturas (LA).
2. Expandir las abreviaturas que están presentes en el texto. El objetivo es eliminar los puntos de estas palabras, para prevenir que se tomen como un fin de sintagma y evitar que sean detectadas como erróneas indebidamente.
3. Se coloca un cambio de línea, es decir, se divide o crea otro sintagma, cuando existe cualquiera de los siguientes signos de puntuación: [. : ; ? !].
4. Se insertan espacios entre los signos de puntuación y las palabras del texto. Con ello, se realiza la separación de signos de puntuación de las palabras. Por ejemplo, los párrafos siguientes:

(a) La pobreza degrada y destruye, moral, social y biológicamente, al más grande milagro cósmico: "la vida humana".

(b) Pero (se podría preguntar): ¿hay tanta gramática en una lengua? Uno de los problemas de la consideración pública de las ciencias del lenguaje es la aparente contradicción entre la facilidad, la naturalidad del uso de la lengua (como ironizaba el poeta, "todos los niños de Francia sabían hablar francés"), y lo costoso de su formalización.

Serán convertidos a:

(a) La pobreza degrada y destruye , moral , social y biológicamente , al más grande milagro cósmico :

(b) " la vida humana " .

(c) Pero (se podría preguntar) :

(d) ¿ hay tanta gramática en una lengua ?

(e) Uno de los problemas de la consideración pública de las ciencias del lenguaje es la aparente contradicción entre la facilidad, la naturalidad del uso de la lengua (como ironizaba el poeta , " todos los niños de Francia sabían hablar francés ") , y lo costoso de su formalización .

Una vez que se tiene el texto formateado, se lleva a cabo el proceso de etiquetamiento automático, el cual se describe a continuación.

7.3.3 Etiquetamiento del Texto de Entrada

Como ya se ha indicado, el etiquetamiento morfosintáctico de un texto consiste en asignarle a cada una de las palabras la categoría gramatical que le corresponde según el uso que tiene dentro del sintagma al que pertenece. El método CATCG utiliza un anotador automático basado en TBL (descrito en el capítulo 6) que fue entrenado para etiquetar texto en español.

Como resultado del proceso de anotación automática, se tiene un texto con el siguiente formato: *<palabra>/<etiqueta morfosintáctica>*.

El etiquetario utilizado para hacer las anotaciones es el mismo que se empleó en el entrenamiento del etiquetador automático. El etiquetario es mostrado en el apéndice B.

De esta manera, el texto:

la relación entre transición política y ruptura filosófica

sería etiquetado de la manera siguiente:

la/*D00###SF* **relación**/*N00#~Sf* **entre**/*P00####N*
transición/*N00#~Sf* **política**/*A05#1SF* **y**/*C02###SN*
ruptura/*N00#~Sf* **filosófica**/*A05#1SF*

Donde a cada palabra le corresponde una cierta etiqueta morfosintáctica en función del uso que tenga dentro del sintagma. Para ilustrar lo anterior se muestra la tabla 7.16, en la cual se toman cuatro palabras del ejemplo antes citado.

relación	N00#~Sf	Sustantivo común singular femenino
entre	P00####N	Preposición simple
y	P00####N	Preposición simple
ruptura	N00#~Sf	Sustantivo común singular femenino

Tabla 7.16: Ejemplos de etiquetas morfosintácticas y sus significados.

Es de esta forma como se realiza la clasificación de las palabras del texto de entrada de acuerdo con sus categorías gramaticales, incluyendo las palabras incorrectas.

7.3.4 Extracción de Palabras

En esta tercera fase, el método CATCG carga en memoria los archivos de datos necesarios, separa las etiquetas y las mayúsculas del texto etiquetado y crea los lexicones de búsqueda. En primer término, carga en memoria el contenido de los siguientes archivos de texto plano:

1. El texto etiquetado (TE).
2. El modelo de lenguaje (LN).
3. Los lexicones de categorías gramaticales.

Después crea el archivo donde se almacenará el resultado de la corrección.

Posteriormente, se lleva a cabo un proceso de separación de las etiquetas y mayúsculas presentes en las palabras del texto a corregir con la finalidad de aislar las palabras que serán analizadas en las fases siguientes.

Durante la separación de palabras y etiquetas a cada inicio de sintagma se le inserta la pseudopalabra *<I>* que significa que la palabra que le sigue es la primera del sintagma. Esto se realiza para que la búsqueda en el modelo de lenguaje se efectúe de forma adecuada. A continuación, se identifican las etiquetas de cada palabra, buscando el patrón siguiente:

<palabra>/<primer carácter siguiente>

Por ejemplo:

Palabra	Etiqueta	Letra clave
Filosofía	N00#~Sf	N
humana	A05#1SF	A
de	P00###N	P

Tabla 7.17: Se toma la primera letra de las etiquetas morfosintácticas del texto.

De este modo, se tiene una sucesión de palabras (PA) y una sucesión de etiquetas (ETQ). A partir de esta última se determina en dónde se buscarán los conjuntos de correcciones potenciales para cada palabra que se detecte como incorrecta. Después de obtener las etiquetas de las palabras, se realiza la separación de mayúsculas. De forma similar a la fase de preprocesamiento del método CATMC, se forma una pareja con cada palabra y la letra inicial mayúscula que la forma. Hay que recordar que no existen palabras que tengan signos ortográficos inmediatos, sino que son separados en la fase de preprocesamiento, en la preparación del texto para su etiquetamiento, y aparecen como una palabra más. Los siguientes ejemplos ilustran el resultado del proceso de separación de mayúsculas:

Así = [MAYA] [así]
La = [MAYL] [la]
Valle = [MAYV] [valle]
: = [NC] :

Una vez que se tienen las palabras del texto, se procede a crear los lexicones de búsqueda de cada categoría gramatical y de n-gramas de palabras. Los lexicones de búsqueda y las etiquetas que los identifican son mostrados en la tabla 7.18.

En esta parte también se hace uso de los arreglos asociativos para crear los lexicones, utilizando como llave las categorías antes mencionadas. Así,

V	Verbo	P	Preposición
I	Interjección	C	Conjunción
D	Artículo	B	Adverbio
A	Adjetivo	R	Pronombre
N	Sustantivo	U	Numeral
F	Fecha	M	Miscelánea

Tabla 7.18: Los lexicones de búsqueda y sus etiquetas distintivas.

para los lexicones de búsqueda se forma una clave con la siguiente pareja de datos:

{Categoría gramatical}{Palabra}

Por ejemplo:

{R}{yo} (un pronombre)
{A}{mexicano} (un adjetivo)
{N}{gato} (un sustantivo)

En tanto que los lexicones de correcciones potenciales tienen las claves siguientes:

{Tamaño de palabra}{Categoría gramatical}{n-grama de búsqueda}

Y como valores, tienen la palabra y los n-gramas de letras correspondientes.

De la misma forma que en el método CATMC, el n-grama de búsqueda se integra según los siguientes criterios:

1. Palabras de tamaño *menor a cuatro letras*: el n-grama se integra con las x primeras letras, con $x = 1 \ldots 3$, así, cada letra diferente forma una llave. En la tabla 7.19 se muestra un ejemplo de esto.

2. Palabras de *cuatro letras*: el n-grama se forma con los dos únicos trigramas que posea. Cada uno creará un índice diferente, como se muestra en la tabla 7.20:

A	B	C
a:a	1,a	a:a
tú:t,ú	2,t	tú:t,ú
	2,ú	tú:t,ú
los:l,o,s	3,D,l	los:l,o,s
	3,D,o	los:l,o,s
	3,D,s	los:l,o,s

Tabla 7.19: N-gramas de búsqueda de los lexicones de categorías gramaticales para palabras de tres letras o menos. Las columnas contienen la información siguiente: A = Entrada del Lexicón de categorías gramaticales, B = Clave del lexicón de búsqueda y C = Valor de clave en el lexicón de búsqueda.

A	B	C
cual:cua,ual+cu,ua,al	4,R,cua	cual:cua,ual+cu,ua,al
	4,R,ual	cual:cua,ual+cu,ua,al

Tabla 7.20: Llaves de búsqueda para palabras de cuatro letras o más. Las columnas tienen la siguiente información: A = Entrada del Lexicón de categorías gramaticales, B = Clave del lexicón de búsqueda y C = Valor de clave en el lexicón de búsqueda (trigramas+bigramas)

3. De otra forma: el n-grama se integra con los tres primeros trigramas que posea la palabra. Cada uno creará un índice diferente. Por ejemplo, para la siguiente entrada del lexicón de categorías gramaticales:

   abanderar:aba,ban,and,nde,der,era,rar+ab,ba,an,nd,de,er,ra,ar

   Se producen las siguientes claves de búsqueda:

   {9,V,aba} =
   abanderar:aba,ban,and,nde,der,era,rar+ab,ba,an,nd,de,er,ra,ar
   {9,V,ban} =
   abanderar:aba,ban,and,nde,der,era,rar+ab,ba,an,nd,de,er,ra,ar
   {9,V,and} =
   abanderar:aba,ban,and,nde,der,era,rar+ab,ba,an,nd,de,er,ra,ar

Una vez que se han creado los lexicones de búsqueda de cada categoría gramatical, se crea el modelo de lenguaje, de la misma forma que en el método CATMC:

{palabra1} = *palabra2-frecuencia.*

Con esto, quedan integrados los conjuntos de palabras y los lexicones de búsqueda para comenzar con la fase de detección automática de errores en el texto.

7.3.5 Detección de Palabras Incorrectas

El proceso de detección de errores determina qué palabras del texto pueden contener errores. Una palabra del texto se considera errónea si no está entre las del lexicón de búsqueda de su categoría gramatical y si no se encuentra ninguno de sus bigramas en el modelo de lenguaje LN. Como resultado de la detección se obtiene el conjunto de palabras incorrectas (PE). Esto se lleva a cabo de forma análoga a la detección contextual y morfológica de errores del método CATMC. Para ello se utiliza el algoritmo 7.4.

Algoritmo 7.4. Detección Contextual de Errores en Palabras Etiquetadas
Variables de Entrada:

PA Lista de palabras del texto a corregir

ETQ Lista de las etiquetas morfosintácticas correspondientes a cada palabra del texto anotado

LN Lexicón de búsqueda del modelo de lenguaje (n-gramas)

LP Conjunto de lexicones de categorías gramaticales

Variables de Salida:

PE Lista de palabras incorrectas

Variables Locales:

$|PA|$ Número de palabras de PA

p Representa el índice de la palabra analizada en cada iteración

pa Es la palabra que se está analizando en cada iteración

ant Es la palabra anterior a pa

$post$ Es la palabra posterior a pa

Ca Almacena la categoría gramatical de la palabra pa

CA Almacena la categoría gramatical de la palabra ant

CP Almacena la categoría gramatical de la palabra $post$

LP_{Ca} Lexicón de palabras de la categoría gramatical Ca

$bandera$ Indica si no se han encontrado los bigramas de pa

Funciones Locales:

$ingresa(l, e)$ Añade el elemento e a la lista l

1. $p = 0$
2. Mientras $p < |PA|$ hacer
   (a) $ant = PA[p-1]$
   (b) $pa = PA[p]$
   (c) $post = PA[p+1]$
   (d) Si $pa \neq < I >$ entonces

i. Ca = categoría gramatical de la palabra *pa*

ii. CA = categoría gramatical de la palabra *ant*

iii. CP = categoría gramatical de la palabra *post*

iv. Si $pa \notin LP_{Ca}$ entonces

$ingresa(PE, pa)$

v. si no

Si $[ant, pa] \notin LN$ entonces

i. Si $ant \notin LP_{CA}$ entonces

$ingresa(PE, ant)$

ii. si no

$bandera = 1$

iii. Si $[pa, post] \notin LN$ entonces

Si $post \notin LP_{CP}$ entonces

$ingresa(PE, post)$

si no

Si $post \notin PE$ entonces

$bandera = 2$

iv. Si $bandera = 2$ entonces

$ingresa(PE, pa)$

v. Si $bandera = 1$ entonces

Si $ant \notin PE$ entonces

$ingresa(PE, pa)$

(e) $p = p + 1$

3. Regresar PE

A continuación se comenta el algoritmo 7.4.

Se busca cada palabra del texto (p) en el lexicón de la categoría gramatical que le corresponda LP_{Cp}. Para ello se utiliza como llave la categoría y la palabra buscada:

{categoría gramatical}{palabra}

La búsqueda se realiza tomando una palabra del conjunto PA a la vez. De no encontrarse una entrada en su lexicón correspondiente, se toma como palabra errónea.

En caso de que si se encuentre la palabra en turno, se construyen bigramas tomando en cuenta las palabras de posición anterior y posterior. Para las palabras iniciales se utilizan las pseudopalabras $<I>$ insertadas con anterioridad en la fase de extracción de palabras.

De esta forma, se buscan los bigramas construidos con la palabra buscada y su anterior y posterior en el lexicón de búsqueda del modelo de lenguaje LN.

Si no se encuentra alguno de los bigramas se verifica que las palabras anterior y posterior existan en sus respectivos lexicones. Hay que recordar que cada palabra del conjunto PA tiene asociada una categoría gramatical y por ende, se busca en un lexicón distinto. Si no se encuentra la palabra anterior y/o la posterior, la palabra actual se considera incorrecta y se ingresa al conjunto PE. De no ser así, se considera la palabra actual como incorrecta puesto que no existen bigramas en LN que concuerden con ella.

De esta forma, se crea el conjunto de palabras a corregir en las fases posteriores (PE). También se guardan los índices de las palabras para poder reinsertar las correcciones en el lugar correspondiente en el texto.

7.3.6 Generación de Correcciones Potenciales

Es en esta etapa donde se genera el conjunto de correcciones potenciales (CP) para cada palabra errónea detectada. Esto se realiza en los 4 pasos siguientes (para cada palabra de PE):

1. Se calcula el tamaño de la palabra errónea (el número de letras que la forman) $|P|$.

2. Se crean los n-gramas de letras de la palabra de la misma forma que en el método CATMC:

   (a) Si el tamaño de la palabra es menor o igual a tres caracteres: se toman todas las letras de la palabra para formar monogramas de letras. Por ejemplo: *a = a al = a,l ala = a,l,a*

   (b) De otro modo se realizan dos operaciones:

      i. Crear los trigramas de la palabra. Por ejemplo:

      cóndor = cón,ónd,ndo,dor

      almendras = alm,lme,men,end,ndr,dra,ras

      óseo = óse,seo

      ii. Crear los bigramas de la palabra. Por ejemplo:

      cóndor = có,ón,nd,do,or

      almendras = al,lm,me,en,nd,dr,ra,as

      óseo = ós,se,eo

   De tal forma, se crean las siguientes estructuras:

   palabra:monogramas

   palabra:trigramas+bigramas.

   Por ejemplo:

   el : e,l

   cóndor : cón,ónd,ndo,dor+có,ón,nd,do,or

   óseo : óse,seo+ós,se,eo

   almendras : alm,lme,men,end,ndr,dra,ras+al,lm,me,en,nd,dr,ra,as

   naci6 : nac,aci,ci6+na,ac,ci,i6

3. Con el fin de crear las llaves de búsqueda con las que se obtendrán las correcciones potenciales, se separan los monogramas, bigramas y trigramas obtenidos en el paso 2. Así, cada llave se forma con los primeros trigramas, bigramas o monogramas de la palabra errónea, además de la categoría gramatical de la palabra y los tamaños siguientes:$|P|$ (El tamaño de la palabra calculado en el paso 1), $(|P|+1)$ y $(|P|-1)$.

   Por ejemplo: para la palabra *cóndor*:

   $\{|P|\}\{categoría\}\{n\text{-}grama1\}\quad\{6\}\{S\}\{cón\}$
   $\{|P+1|\}\{categoría\}\{n\text{-}grama1\}\quad\{7\}\{S\}\{cón\}$
   $\{|P-1|\}\{categoría\}\{n\text{-}grama1\}\quad\{5\}\{S\}\{cón\}$

   $\{|P|\}\{categoría\}\{n\text{-}grama2\}\quad\{6\}\{S\}\{ónd\}$
   $\{|P+1|\}\{categoría$n-grama2$\}\quad\{7\}\{S\}\{ónd\}$
   $\{|P-1|\}\{categoría$n-grama2$\}\quad\{5\}\{S\}\{ónd\}$

   $\{|P|\}\{categoría\}\{n\text{-}grama3\}\quad\{6\}\{S\}\{ndo\}$
   $\{|P+1|\}\{categoría\}\{n\text{-}grama3\}\quad\{7\}\{S\}\{ndo\}$
   $\{|P-1|\}\{categoría\}\{n\text{-}grama3\}\quad\{5\}\{S\}\{ndo\}$

4. Con cada llave de búsqueda, se obtiene el conjunto de correcciones potenciales para cada palabra errónea CP. Esto se realiza comparando las llaves de las palabras erróneas, con las de las entradas del lexicón de búsqueda correspondiente. Por ejemplo, para la palabra *nacié*, se tendría el siguiente conjunto de correcciones candidatas:

   [nació, nacido, vaciar, saciar, nacías, nacían, nacía, nací, naces, naceré, nacerá, nacer, nacen, nace, aciar]

Hay que remarcar que, gracias a la asignación de categorías gramaticales, el conjunto de correcciones candidatas para cada palabra errónea detectada, pertenece a la misma categoría que ésta. En el ejemplo anterior, sólo se obtienen verbos como palabras candidatas para corregir un verbo.

En caso de no encontrarse ninguna corrección candidata para una palabra errónea, se marca con el identificador *NHS*, que indica que no existen correcciones candidatas para esa palabra. Dicha palabra será insertada sin

cambios en el texto original. Una vez que se tiene el conjunto de correcciones potenciales CP. Se procede a seleccionar la más adecuada para cada palabra incorrecta. A continuación se explica cómo se realiza esto.

7.3.7 Selección de Correcciones Potenciales

En esta fase se determina qué palabra del conjunto CP será insertada en el texto original en sustitución de la palabra errónea detectada. El funcionamiento de esta etapa es idéntico al del método CATMC, el cual fue descrito en la sección 7.2. Esta fase emplea los algoritmos 7.2 y 7.3 descritos con anterioridad. Por ello, sólo se recuerda que en la selección de correcciones potenciales se llevan a cabo dos procesos:

1. **La ponderación contextual y morfológica**, que obtiene un conjunto ponderado de correcciones potenciales CPP, las cuales serán comparadas en el paso siguiente para obtener la palabra sustituta que será la corrección final. Utiliza el algoritmo 7.2.

2. **Selección de la corrección potencial final**. Esta parte se lleva a cabo como se indica en el algoritmo 7.3 descrito anteriormente. Su objetivo es determinar cuáles palabras del conjunto CPP sustituirán a las palabras detectadas como incorrectas.

7.3.8 Postprocesamiento

En esta fase se realiza el intercambio entre las correcciones candidatas finales seleccionadas en la fase previa (el conjunto PS) y las palabras erróneas detectadas. Se efectúa también la restitución de mayúsculas y se escribe el archivo corregido final.

La restitución de mayúsculas que fueron separadas en la fase toma las marcas obtenidas durante la fase de detección para convertir las letras indicadas. Por ejemplo:

[MAYN] *[***n****ació]* → **N***ació*
[MAYL] *[***l****a]* → **L***a*
[MAYV] *[***v****alle]* → **V***alle*
[TMAY] *[enero]* → ENERO

Del mismo modo que en el método CATMC, si se detecta que la letra inicial de una palabra que tiene marca de mayúscula ha cambiado, se toma

como letra inicial la primera letra de la palabra corregida en lugar de la letra de la marca. Por ejemplo: *[MAYS] he* → **H***e* y no **S***e*.

Como última acción, el método CATCG escribe el archivo de texto con todas las palabras que fueron corregidas con sus mayúsculas respectivas. Hay que señalar que los signos de puntuación son considerados como palabras y están comprendidos dentro de la categoría Miscelánea. Por lo que también son escritos en el archivo final (incluidos como parte de la sucesión de palabras del texto).

7.4 Resumen

En el presente capítulo se hizo una descripción de los datos utilizados por los dos métodos de corrección automática que se presentaron en este trabajo. En primera instancia se indicaron las características del corpus de texto y del corpus de texto etiquetado morfosintácticamente. También se explicó la forma en que fue entrenado el etiquetador automático basado en el algoritmo TBL y se describió el etiquetario empleado para representar las categorías gramaticales de las palabras. Posteriormente se presentó el conjunto de prueba usado por los dos métodos de corrección ortográfica y se expusieron las características de los lexicones que los métodos utilizan, además se mencionaron las características del modelo de lenguaje basado en bigramas de palabras. En segundo término se presentaron dos métodos de corrección ortográfica automática: el método CATMC y el método CATCG. El primero proporciona un marco de comparación para evaluar los resultados del método de corrección propuesto en esta tesis (denominado CATCG), el cual se basa en la asignación de categorías gramaticales a las palabras del texto. En las secciones correspondientes a cada método se expusieron sus componentes, los datos que utilizan y su funcionamiento. De esta forma, se indicó que el método CATMC realiza la corrección ortográfica en 5 etapas: Preprocesamiento, detección de errores, generación de correcciones potenciales, selección de correcciones potenciales y postprocesamiento. Por su parte, el método CATCG lleva a cabo la corrección ortográfica en 7 fases: Preprocesamiento, etiquetamiento del texto de entrada, extracción de palabras, detección de palabras incorrectas generación de correcciones potenciales, selección de correcciones potenciales y postprocesamiento.

Capítulo 8

Evaluación del Método de Corrección Ortográfica Propuesto

8.1 Descripción del Experimento

En el capítulo 7 se expusieron dos métodos de corrección ortográfica: el método de Corrección Automática de Textos Morfológica y Contextual y el método de Corrección Automática de Textos basada en la asignación de Categorías Gramaticales. También se indicó que el segundo de ellos (el método CATCG) es el que se propone en este trabajo para corregir los errores insertados al texto por el reconocimiento automático de caracteres. Mientras que el método CATMC se desarrolló con el fin de tener un marco de comparación entre la corrección "tradicional" y la que se propone. En el presente capítulo se muestran los resultados de la comparación de ambos métodos.

Con el conjunto de prueba, descrito en el capítulo 8, se realizaron dos evaluaciones a los dos métodos de corrección, a saber:

1. **Corrección de errores insertados automáticamente**. Se buscó identificar el comportamiento de los métodos de corrección frente a errores simples de inserción, borrado y sustitución.

2. **Corrección de errores originados por un proceso de reconocimiento óptico de caracteres**. Se probaron los métodos de corrección en un caso real de errores provenientes de un reconocedor óptico de caracteres.

Por último, con el método CATCG se realizó una prueba más:

- **Corrección de errores en textos etiquetados manualmente**. Para esta evaluación se etiquetó manualmente el conjunto de prueba (que consiste en los textos con errores insertados automáticamente y los provenientes de un reconocedor óptico de caracteres).

La finalidad de esta prueba es tener un etiquetado "ideal", esto es, un etiquetamiento que asigne las categorías correctas al texto. Así, se verifica si un etiquetamiento preciso es de ayuda para incrementar la tasa de corrección de errores, como se establece en la hipótesis H_2.

En las tres evaluaciones anteriores se midió la tasa de corrección de cada método utilizando la fórmula 8.1 [66] .

$$c = \frac{p - i}{p} \tag{8.1}$$

Donde:

c Tasa de corrección de errores

p Palabras totales del texto

i Palabras incorrectas

Además, se realizó el conteo de los siguientes datos:

- Palabras corregidas (**C**). Se refiere a las palabras detectadas como errores que fueron corregidas.
- Errores detectados (**E**). Es el total de palabras que fueron tomadas como errores por el método evaluado.
- Errores no detectados (**e**). Son todos los errores que no identificó el método evaluado.
- Errores inducidos (**I**). Son aquellas palabras sustituidas indebidamente en el texto debido a errores falsos (palabras que sin ser errores fueron identificados como tales).
- Correcciones falsas (**F**). Son aquellos errores que no fueron corregidos, es decir, la corrección potencial elegida no corrigió el error.
- Errores originales (**o**). Son los errores iniciales del texto.

A continuación se muestran los resultados obtenidos en las tres evaluaciones.

8.2 Evaluación del Método CATMC

En esta sección se presentan los resultados de la corrección ortográfica y contextual.

En la tabla 8.1 se muestran los resultados de la evaluación de cada archivo. En la tabla 8.2 se muestran los resultados totales de la evaluación.

Archivo	p	o	i	C	E	e	I	F	c
1	79	7	3	4	7	0	0	3	0.962025316
2	92	7	3	5	7	1	1	1	0.967391304
3	126	13	1	12	13	0	0	1	0.992063492
4	150	14	3	12	15	0	1	2	0.98
5	147	15	3	13	15	1	1	1	0.979591837
6	260	25	11	16	25	2	2	7	0.957692308
7	310	30	12	20	29	3	2	7	0.961290323
8	591	55	22	35	53	4	2	16	0.962774958
9	1153	116	61	63	110	14	8	39	0.947094536
10	358	41	21	22	37	6	2	13	0.941340782

Tabla 8.1: Prueba 1. Textos con errores insertados automáticamente.

p	o	i	C	E	e	I	F	c
3266	323	140	202	311	31	19	90	0.957134109

Tabla 8.2: Resultados totales de la prueba 1. Textos con errores insertados automáticamente.

En la figura 8.1 se muestran, de forma gráfica, los resultados totales de la evaluación del método CATMC con textos que tienen errores insertados automáticamente.

La segunda prueba se realizó con textos que contenían errores provenientes de un proceso de reconocimiento óptico de caracteres. Cabe destacar que en el conteo de errores se tomaron en cuenta únicamente los que se encontraron dentro de las palabras del texto, esto es, aquellos errores que deforman las palabras que integran el texto y no aquellos que, por efecto de ruido, inserta el reconocedor antes y después del texto pero que no deforman las palabras.

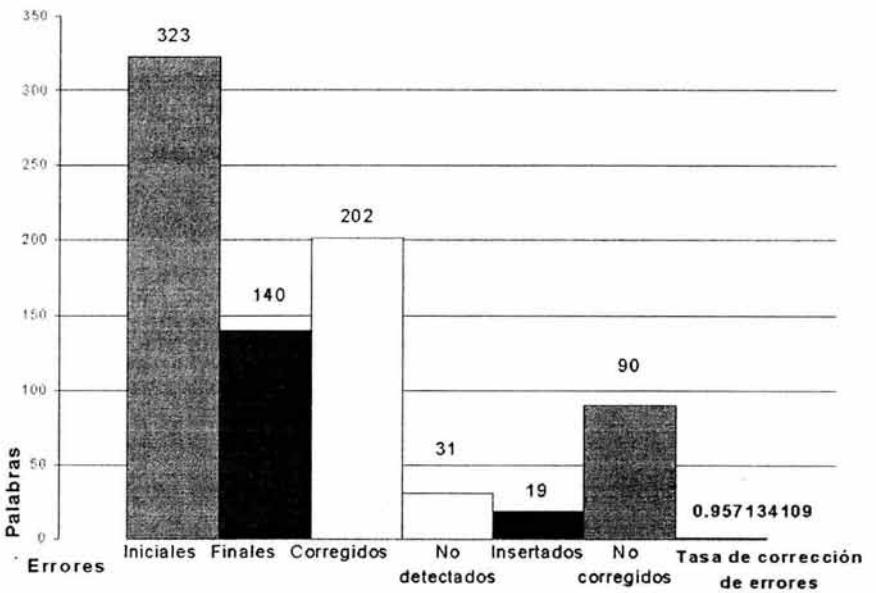


Figura 8.1: Resultados totales de la prueba 1 del método CATMC.

Los resultados de la evaluación de cada archivo se muestran en la tabla 8.3. Mientras que los resultados totales se muestran en la tabla 8.4

Archivo	p	o	i	C	E	e	I	F	c
1	80	5	3	4	7	0	2	1	0.9625
2	93	8	11	5	16	0	8	3	0.88172043
3	151	2	5	1	6	0	4	1	0.966887417
4	128	9	10	3	13	0	4	6	0.921875
5	153	4	10	1	11	0	7	3	0.934640523
6	260	28	13	19	29	3	5	5	0.95
7	310	10	4	7	11	0	1	3	0.987096774
8	587	52	19	30	46	3	2	14	0.967632027
9_1	743	20	12	8	17	3	3	6	0.98384926
9_2	405	9	9	1	9	1	1	7	0.977777778
10	355	15	8	7	15	0	1	7	0.977464789

Tabla 8.3: Prueba 2. Textos con errores originados por el reconocimiento óptico de caracteres.

p	o	i	C	E	e	I	F	c
3185	162	104	86	180	10	38	56	0.967346939

Tabla 8.4: Prueba 2. Resultados totales de la evaluación. Textos con errores insertados por el reconocimiento óptico de caracteres.

En la figura 8.2 se muestran los resultados totales de la evaluación del método CATMC con textos que tienen errores provenientes de un proceso de reconocimiento óptico de caracteres.

Como se puede ver, este modelo tiene un número alto de errores no detectados (e), lo que hace que su tasa de corrección disminuya pues la no detección de palabras incorrectas es un factor significativo para la inserción de errores.

Lo anterior se explica por el hecho de que los errores no detectados se presentaron principalmente en palabras de una o dos letras, como y, a, e, la, el, lo, no, ya que cuando se efectúa un borrado sobre ellas, desaparecen o se convierten en otras palabras válidas que afectan la estructura del sintagma al que pertenecen.

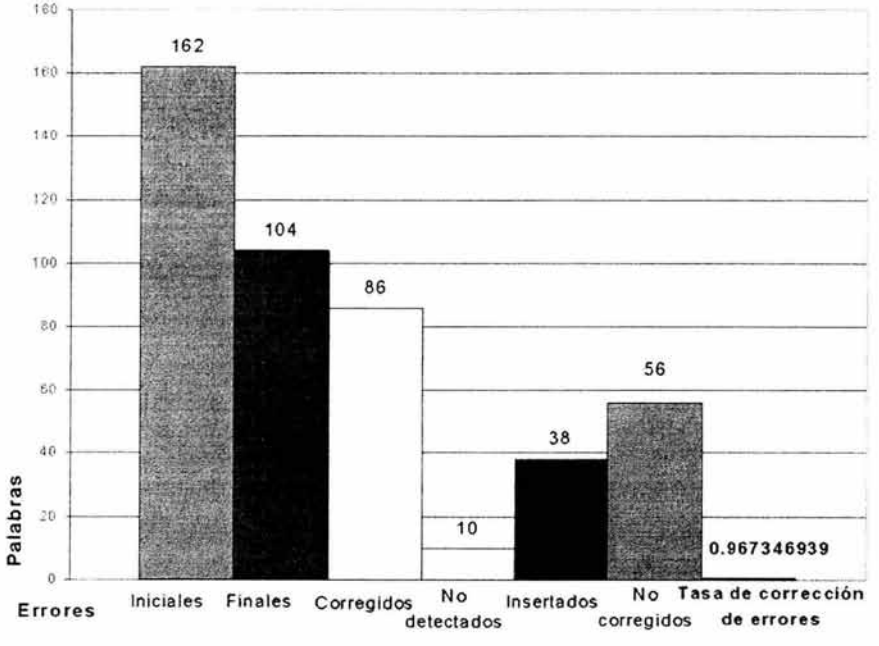


Figura 8.2: Resultados totales de la prueba 2 del método CATMC.

En el primer caso, la corrección contextual detecta que existe un error, sin embargo, al no estar la palabra en el sintagma, toma otra palabra como error. Así, el método CATMC toma como incorrecta la palabra anterior o posterior a la que fue borrada. Como consecuencia, se modifica una palabra que es correcta porque no se detectó adecuadamente el error.

En el segundo caso, el método falla en la detección de la palabra y toma como incorrecta otra, lo que se convierte en una no detección y en una inserción de errores. Esto se da principalmente cuando la palabra incorrecta está al inicio del sintagma o la palabra se reduce a una letra.

A continuación se muestra un ejemplo de una palabra incorrecta no detectado que provoca la inserción de otro error.

Fragmento del sintagma original:

la criada del cura

Fragmento del sintagma incorrecto:

a criada del cura

Esto provoca que el método de corrección no detecte el error y en su lugar tome otra palabra, insertando así un error al texto original, como a continuación se muestra:

No se encontró el bigrama {a,criada} palabra no. 138, **criada** *se toma como error*

Entonces, el sintagma es modificado por el corrector de la siguiente forma:

a **criadas** *del cura*

Por otra parte, el método CATMC presenta también un alto número de "falsas correcciones" (F), es decir, errores detectados pero cuya corrección potencial seleccionada no es la palabra adecuada. Esto se da porque dentro del conjunto de correcciones candidatas existen palabras que forman bigramas más probables que otras y en ocasiones desplazan a la corrección potencial más adecuada. A continuación se muestra un ejemplo:

Fragmento del sintagma original:

hubiese de ser la **norma** *para apreciar*

Fragmento del sintagma incorrecto:

hubiese de ser la **nAorma** *para apreciar*

En este caso, el corrector modificó el sintagma de la forma siguiente:

hubiese de ser la **forma** *para apreciar*

El resultado anterior se obtuvo gracias a que en el modelo de lenguaje se tienen los bigramas que se muestran a continuación

Frecuencia de forma 1.16666666666667
Frecuencia de norma 0.66666666666667

Con lo que se conserva **forma** como la mejor candidata y se produce así una falsa corrección.

En la sección siguiente se presentan los resultados del método CATCG.

8.3 Evaluación del Método CATCG

A continuación se presentan los resultados de la corrección morfológica y contextual basada en la asignación de categorías gramaticales, que es la propuesta que se hizo en esta tesis.

En la tabla 8.5 se muestran los resultados de la evaluación del método CATCG que usó textos con errores insertados automáticamente. Las cifras totales de esta evaluación se presentan en la tabla 8.6

En la figura 8.3 se muestran los resultados totales de la evaluación del método CATCG con textos que presentan errores insertados de forma automática.

Los resultados de la prueba 2, que usó textos con errores provenientes de un proceso de reconocimiento óptico de caracteres se muestran en la tabla 8.7 y los resultados totales de esta prueba se presentan en la tabla 8.8.

En la figura 8.4 se muestran los resultados totales de la evaluación del método CATCG con textos digitalizados mediante un reconocedor óptico de caracteres.

Como se puede apreciar, la tasa de corrección disminuyó con respecto al método CATCM. Lo anterior se debe a que el proceso de etiquetamiento no fue lo suficientemente preciso y se etiquetaron incorrectamente los textos. El etiquetamiento impreciso influye negativamente en dos aspectos:

Archivo	p	o	i	C	E	e	I	F	c
1	79	7	6	3	8	1	2	3	0.924050633
2	92	7	9	4	13	0	6	3	0.902173913
3	126	13	15	4	19	0	3	9	0.880952381
4	150	14	9	9	18	0	4	5	0.94
5	147	15	9	6	20	1	6	5	0.93877551
6	260	25	14	13	27	2	5	10	0.946153846
7	310	30	44	11	53	2	20	17	0.858064516
8	591	55	58	19	75	5	24	31	0.901861252
9	1153	115	156	32	178	14	73	70	0.864700781
10	358	41	50	11	63	3	24	27	0.860335196

Tabla 8.5: Resultados de la evaluación de cada archivo. Textos con errores insertados automáticamente.

p	o	i	C	E	e	I	F	c
3266	322	370	112	474	28	167	180	0.886711574

Tabla 8.6: Prueba 1. Textos con errores insertados automáticamente.

Archivo	p	o	i	C	E	e	I	F	c
1	80	5	3	4	7	0	2	1	0.9625
2	93	8	11	5	16	0	8	3	0.88172043
3	151	2	5	1	6	0	4	1	0.966887417
4	128	9	10	3	13	0	4	6	0.921875
5	153	4	10	1	11	0	7	3	0.934640523
6	260	28	16	17	28	3	5	8	.0.938461538
7	310	10	23	7	30	0	19	3	0.925806452
8	587	52	37	30	66	1	15	21	0.936967632
9_1	743	20	42	8	48	3	31	9	0.943472409
9_2	405	9	38	1	38	1	30	7	0.90617284
10	355	15	34	7	41	0	27	8	0.904225352

Tabla 8.7: Prueba 2. Textos con errores provenientes de un proceso de reconocimiento óptico de caracteres.

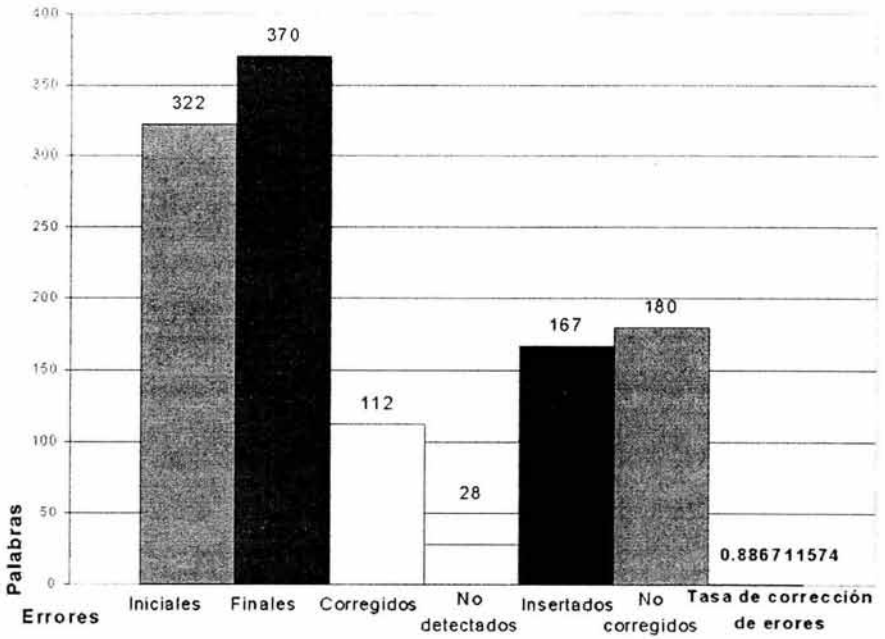


Figura 8.3: Resultados totales de la prueba 1 del método CATCG.

p	o	i	C	E	e	I	F	c
3185	162	229	84	304	8	152	70	0.928100471

Tabla 8.8: Cifras totales de la prueba 2 del método CATCG.

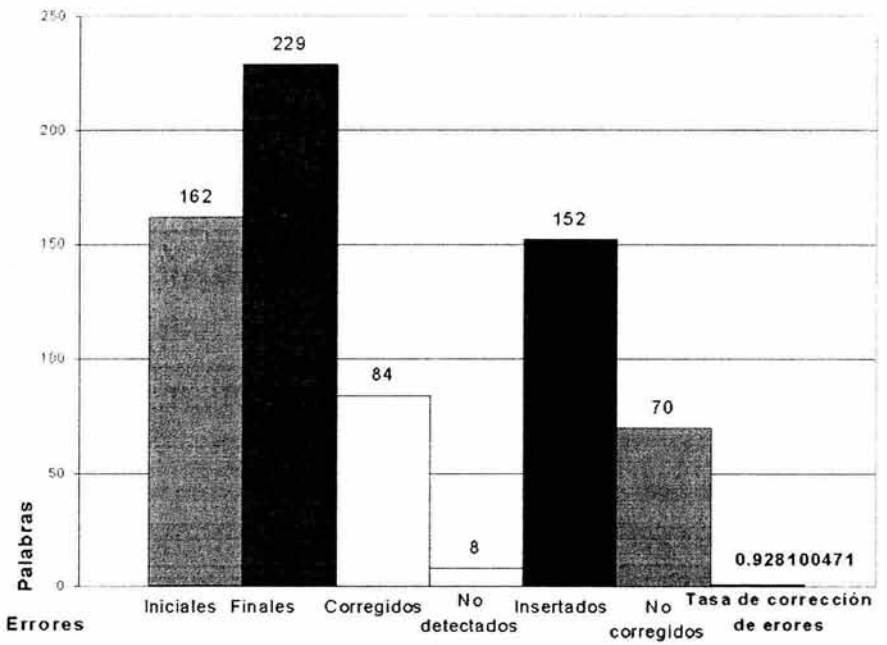


Figura 8.4: Resultados totales de la prueba 2 del método CATCG.

- Durante la detección de errores, la búsqueda de palabras en lexicones incorrectos trae como consecuencia que se tomen como errores palabras que son correctas.

- La generación de correcciones potenciales falla porque la búsqueda se realiza en lexicones que no corresponden a la categoría gramatical correcta de las palabras detectadas como incorrectas. De esta forma, cualquiera de las palabras candidatas será equivocada pues pertenece a una categoría gramatical distinta y por ende, no hay posibilidad de que la palabra adecuada esté dentro del conjunto de correcciones potenciales.

Por ello, se observa en los resultados un aumento en el número de errores insertados (I) y en el de correcciones falsas (F). A continuación se muestran dos ejemplos de esto.

- Inserción de errores:

  Fragmento del sintagma original:

  Antología del pensamiento hispánico **Juan** *Montalvo*

  El fragmento anterior se etiquetó como se muestra enseguida:

  Antología/N00#∼Sf del/P03###N pensamiento/N00#∼Sm hispánico/A05#1SM **Juan/VPePD6N** *Montalvo/N00#∼Sm*

  Como se puede ver, Juan es etiquetado como un verbo, lo que ocasiona que no se encuentre en el lexicón de verbos y se tome como error, de esta forma, se sustituye con la forma verbal *Rúan*:

  Antología del pensamiento hispánico **Rúan** *Montalvo*

- Generación de correcciones potenciales de un lexicón incorrecto:

  Fragmento del sintagma original:

  ¿cómo lo he **de** *negar?*

  Fragmento del sintagma incorrecto:

¿cómo lo he **se** *negar?*

El fragmento anterior se etiquetó como sigue a continuación:

¿/M06#### cómo/R22##LN lo/D00##SN he/VHaPD0N
se/R00##LN *negar/VImI~#N ?/M07####*

Como se observa, *se* es etiquetado como un pronombre, que es su categoría gramatical, sin embargo, contextualmente *he se negar* no es una construcción adecuada, por ello, acertadamente se detecta *se* como error, pero entre las correcciones candidatas no aparece *de* que es la palabra adecuada para corregir el error, ya que ésta es una preposición.

Por lo que el sintagma anterior queda de la forma siguiente:

¿cómo lo he **que** *negar?*

Es así como se obtiene una corrección falsa pese a que se detectó adecuadamente la palabra incorrecta.

En contraparte, el método CATCG presentó una ligera disminución de los errores no detectados, es decir, detectó errores que el método CATMC no fue capaz de identificar, aunque hay que señalar que estos errores también dieron origen a correcciones falsas. A continuación se presenta un ejemplo de esto.

Fragmento del sintagma original:

y *con tal fin*

Fragmento del sintagma incorrecto:

f *con tal fin*

El fragmento anterior se etiquetó como sigue a continuación:

f/NN *con/P00###N tal/A07#~SN fin/N00#~Sm*

Con ello, al buscar la letra **f** en el lexicón de sustantivos[1], no es encontrada y se detecta como un error.

Por último, se presentan los resultados de la prueba 3, la cual se llevó a cabo anotando manualmente los textos con errores insertados automáticamente y provenientes del reconocedor de caracteres. El objetivo de esta prueba es evaluar el método CATCG cuando el proceso de etiquetamiento automático de texto se realiza con un nivel de confiabilidad muy alto. En esta prueba, todos los textos están anotados con las etiquetas morfosintácticas correctas para cada una de sus palabras y a partir de ellos se realizan la detección de errores, la generación de correcciones potenciales y la selección de las palabras que sustituirán a las incorrectas. En las tablas 8.9 y 8.11 se presentan los resultados por cada archivo y en las tablas 8.10 y 8.12 se muestran los totales.

Archivo	p	o	i	C	E	e	I	F	c
1	79	7	1	6	7	0	0	1	0.987341772
2	92	7	4	5	8	1	2	1	0.956521739
3	126	13	5	10	15	0	2	3	0.96031746
4	150	14	2	12	14	0	0	2	0.986666667
5	147	15	6	9	8	3	0	3	0.959183673
6	260	25	9	16	24	1	0	8	0.965384615
7	310	30	12	20	27	3	2	7	0.961290323
8	591	55	21	37	48	5	3	13	0.964467005
9	1153	116	59	75	51	10	18	31	0.948829141
10	358	41	14	27	36	3	0	11	0.960893855

Tabla 8.9: Prueba 3a. Textos con errores insertados automáticamente.

p	o	i	C	E	e	I	F	c
3266	323	133	217	238	26	27	80	0.959277404

Tabla 8.10: Prueba 3a. Resultados totales de los textos con errores insertados automáticamente.

[1]La etiqueta NN significa que el etiquetador asignó la categoría sustantivo a esa palabra porque no encontró una etiqueta morfosintáctica. Esto, como se explicó en los capítulos 6 y 7, se realiza por omisión cuando el etiquetador se encuentra una palabra desconocida y las reglas que utiliza no le cambian su etiqueta.

En la figura 8.5 se muestran los resultados totales de la tercera prueba del método CATCG con textos anotados manualmente y errores insertados de forma automática.

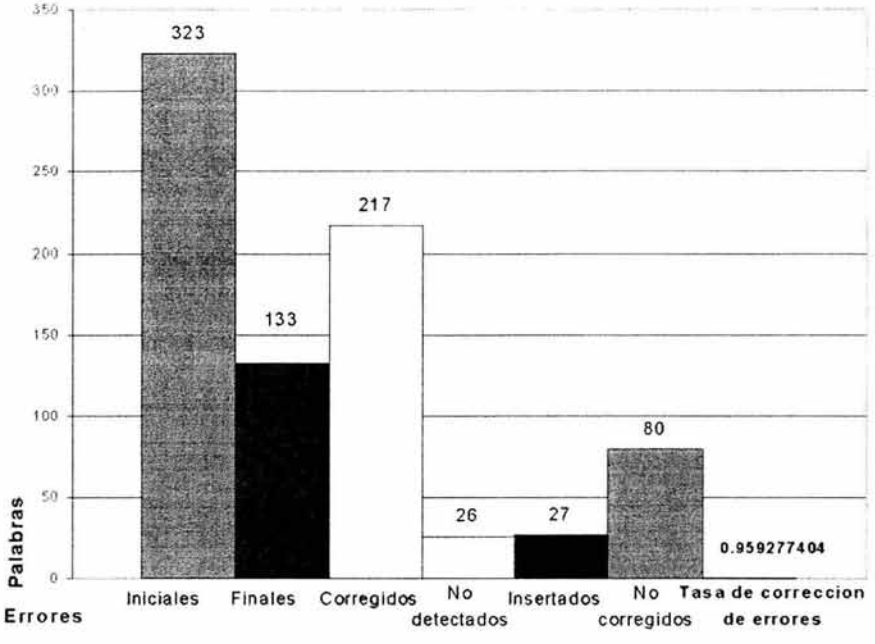


Figura 8.5: Resultados totales de la prueba 3a del método CATCG.

En la figura 8.6 se muestran los resultados totales de la tercera prueba del método CATCG con textos anotados manualmente y errores insertados por el reconocedor óptico de caracteres.

Como se puede observar, con esta última prueba el método CATCG logró una tasa de corrección superior a la del método CATMC, disminuyendo a su vez el número de errores insertados, errores no detectados y falsas correcciones. La razón de esta mejora radica en dos aspectos principales:

- Los errores que resultan en palabras no válidas son comparados con palabras de su misma categoría gramatical, por lo que su corrección potencial es encontrada no sólo por el parecido morfológico y la proba-

Archivo	p	o	i	C	E	e	I	F	c
1	82	5	0	1	1	0	9	0	1
2	93	8	1	6	1	0	0	1	0.989247312
3	151	2	1	1	1	0	0	1	0.993377483
4	128	9	5	6	11	1	1	3	0.9609375
5	153	4	2	2	4	0	0	2	0.986928105
6	260	28	5	21	26	2	0	5	0.980769231
7	310	10	6	7	12	1	2	3	0.980645161
8	587	52	15	36	49	2	2	11	0.974446337
9_1	743	20	19	13	51	1	11	7	0.974427995
9_2	405	9	13	3	15	1	8	5	0.967901235
10	355	15	7	8	15	0	0	7	0.98028169

Tabla 8.11: Prueba 3b. Textos con errores provenientes de un proceso de reconocimiento óptico de caracteres.

p	o	i	C	E	e	I	F	c
3267	162	74	104	186	8	33	45	0.97734925008

Tabla 8.12: Prueba 3b. Resultados totales de la evaluación del método CATCG. Textos con errores provenientes de un proceso de reconocimiento óptico de caracteres

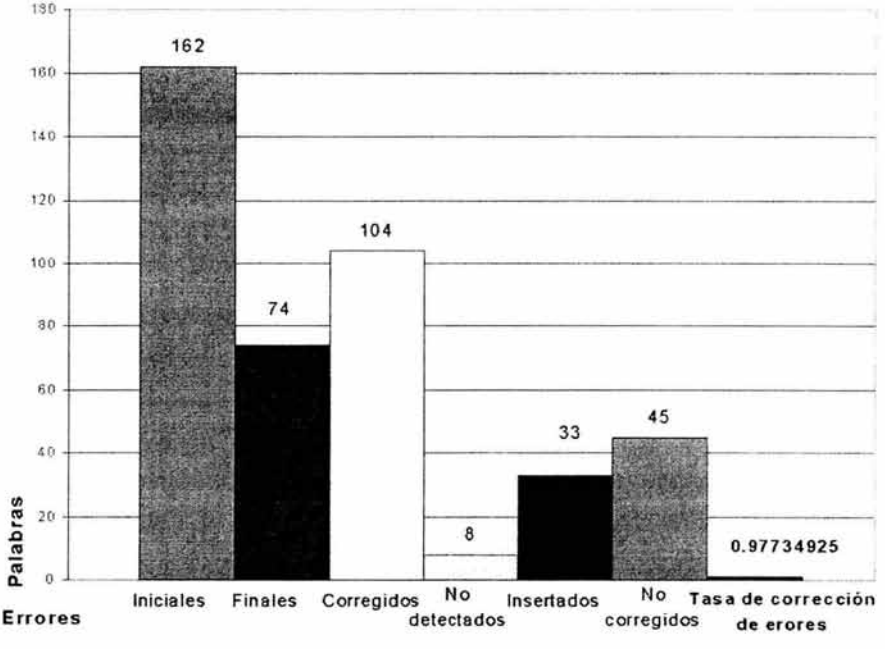


Figura 8.6: Resultados totales de la prueba 3b del método CATCG.

bilidad de sus bigramas, sino porque el espacio de búsqueda se restringe a palabras de su misma clase, con lo que se eliminan correcciones falsas.

- Los errores que resultan en palabras válidas son detectados porque su categoría gramatical es diferente a la de la palabra original que estaba en el texto. Además, las correcciones potenciales que se generan son validadas por el modelo de lenguaje. Con esto se incrementa la posibilidad de que se obtenga la candidata correcta.

8.4 Conclusiones

Los resultados presentados en las secciones anteriores muestran que el método de corrección ortográfica que utiliza técnicas de detección y corrección basadas en la morfología de las palabras, el análisis contextual y la asignación de categorías gramaticales (CATCG) tiene una tasa de corrección de errores mayor que el método de corrección ortográfica morfológica-contextual (CAMC) únicamente si se logra un etiquetamiento preciso del texto a corregir, de otra manera se presenta un gran número de inserciones de errores y falsas correcciones que hacen que sea más adecuado utilizar métodos de un sólo lexicón general.

Por ello, para mejorar el desempeño del método CATCG se plantea:

- Incrementar la precisión del etiquetador automático de textos. Se pueden tomar las siguientes opciones a este respecto:

  - Realizar un entrenamiento más intenso que el que se realizó para este experimento, utilizando un corpus más grande y con un léxico más variado.

  - Desarrollar una forma alterna de determinar las etiquetas de las palabras desconocidas.

  - Utilizar un etiquetador automático de textos que brinde mejores resultados para el idioma español.

  - Marcar las palabras del texto con más de una etiqueta morfosintáctica, según la probabilidad de éstas para cada palabra.

- Cambiar el contenido de los lexicones para que tengan sólo los lemas de las palabras y no la forma ortográfica completa. Esto sería de utilidad para tomar en cuenta las diferentes flexiones del idioma español de forma más completa.
- Mejorar las técnicas de detección y generación de correcciones potenciales para evitar en mayor medida la inserción de errores.

Capítulo 9

Conclusiones

En este trabajo se han investigado dos maneras de mejorar el desempeño de los lectores automáticos de texto: a través de un modelo de reconocimiento óptico de caracteres y mediante un método de corrección de los errores de un texto electrónico.

Por lo que respecta al modelo de reconocimiento óptico de caracteres, se presentó el diseño de una red neuronal convolucional modular. Dicha red introduce el concepto de modularidad de clases en las redes neuronales convolucionales con el propósito de reducir la tasa de error de reconocimiento. Como lo demuestran los resultados de las pruebas, el objetivo de reducción de errores se logró. No obstante, la tasa de rechazó se incrementó ligeramente en este modelo.

Sin embargo, el incremento de los rechazos es mínimo. Además, el contar con un modelo que en vez de cometer errores decide rechazar el carácter puede ayudar a la corrección posterior. De esta forma, la unión del modelo de red neuronal convolucional modular con un corrector de textos daría mejores resultados, ya que el reconocedor informaría al corrector sobre aquellos caracteres dudosos facilitando la corrección y minimizando la afectación del texto que no requiere correcciones.

Por lo que toca al método de corrección de textos electrónicos, se mostró que el diseño de un método que incluye el etiquetamiento del texto a corregir en las fases de detección de errores y generación de correcciones candidatas ayuda a aumentar la tasa de corrección siempre y cuando el etiquetamiento morfosintáctico ofrezca resultados muy precisos. En el caso de la detección de errores, un etiquetamiento incorrecto provoca que se inserten errores en el texto y que los errores originales no puedan ser corregidos. En tanto que,

la generación de correcciones potenciales no puede proponer adecuadamente palabras candidatas para los errores detectados si parte de un etiquetamiento impreciso.

La aplicación de ambas propuestas permite la reducción de la tasa de error de reconocimiento. Este beneficio se puede traducir en mejoras del rendimiento obtenido por los sistemas de lectura automática de textos. De esta manera, estos sistemas resultarán más útiles para la comunidad de invidentes y débiles visuales.

Finalmente, se espera que el presente estudio sirva de base a futuras investigaciones en el área gracias a la experiencia ganada en este trabajo.

Apéndice A

Implementación

En este apéndice se presenta la información referente a la implementación de los modelos propuestos en esta tesis, incluyendo el código fuente. De igual forma se describe el software y datos requeridos para la prueba de dichos modelos.

De manera general, los modelos de reconocimiento y los métodos de corrección de textos electrónicos se implementaron y probaron en una computadora con sistema operativo Red Hat Linux.

A.1 El Modelo de Red Neuronal Convolucional Modular

Por lo que toca al modelo de reconocimiento óptico de caracteres, la implementación tanto del modelo de red neuronal convolucional modular como del modelo de red neuronal convolucional general se realizó utilizando el lenguaje de programación Lush. Este lenguaje de programación se distribuye de forma gratuita y puede obtenerse en la siguiente dirección electrónica: http://lush.sourceforge.net/. El motivo por el cual se seleccionó este lenguaje de programación yace en que incluye librerías para el manejo de redes neuronales. De esta forma, tanto la implementación, como la prueba de los modelos resultó más simple, permitiendo concentrar los esfuerzos en la investigación y diseño del modelo.

A continuación, se muestra el código fuente del programa que se usó para entrenar a las redes neuronales convolucionales.

```
; Este programa sirve para entrenar una red neuronal convolu-
; cional sobre la base de datos MNIST (digitos escritos a ma-
; no). Este programa asume que los archivos de imágenes y eti-
; quetas MNIST se encuentran en:
;
; LUSHDIR/local/mnist/general
;
; Si están instalados en otra ruta, simplemente se debe indi-
; car la ruta así:
;
; (setq *mnist-img* <directorio-mnist-imágenes>)
; (setq *mnist-etq* <directorio-mnist-etiquetas>)
;
; Se asume que el entrenamiento es para una red general. Cuan-
; do se desee entrenar una red especializada, se debe especi-
; ficar:
;
; (setq etiqueta-red <valor de la etiqueta>)
;
; Otras dos inicializaciones antes de cargar el programa son:
;
; (setq *trabajo-dir* <directorio-trabajo>)
; (setq num-corrida <n>)

(libload "gblearn2/net-lenet5")
(libload "gblearn2/gb-trainers")
(libload "gblearn2/gb-meters")
(libload "dsource-mnist")
(libload "e-s-red")

(when (not *mnist-img*)
  (setq *mnist-img* (concat lushdir "/local/mnist/general")))

(when (not *mnist-etq*)
  (setq *mnist-etq* (concat lushdir "/local/mnist/general")))

(when (not etiqueta-red)
  (setq etiqueta-red -1))
```

```
; Función para cargar la base de datos MNIST
(de crear-bd-mnist (tam-entrena tam-valida img-dir etq-dir)
  (setq bd-entrena
    (new dsource-idx3l-permute
      (new dsource-idx3l-narrow
        (new dsource-mnist
          (load-matrix (concat-fname img-dir
                         "train-images-idx3-ubyte"))
          (load-matrix (concat-fname etq-dir
                         "train-labels-idx1-ubyte"))
          32 32 0 0.01)
        tam-entrena 0)))
  (setq bd-valida
    (new dsource-idx3l-narrow
      (new dsource-mnist
        (load-matrix (concat-fname img-dir
                       "v10k-images-idx3-ubyte"))
        (load-matrix (concat-fname etq-dir
                       "v10k-labels-idx1-ubyte"))
        32 32 0 0.01)
      tam-valida 0))
  ())

; Cargar los parámetros para el entrenamiento:
(leer-config *trabajo-dir*)

; Llenar la matriz con el código 1-de-n
(setq etiquetas (int-matrix nclases))
(setq objetivos (float-matrix (+ nclases 1)))
(objetivos () (/ (+ nclases 1)))

(if (= etiqueta-red -1)
  (for (i 0 (- nclases 1))
    (etiquetas i i))
  (etiquetas 0 etiqueta-red)
  (etiquetas 1 15))
```

```
; Crear los parámetros entrenables
(setq parametros (new idx1-ddparam 0 nparam))

; Crear la función de costo
(setq costo-rechazo (new idx1-ddparam 0 1))
(setq fun-costo (new mmi-cost etiquetas objetivos
                              1 1 costo-rechazo))

(if (> const-costo 0)
  (==> fun-costo set-junk-cost const-costo))

; Crear la red neuronal convolucional
(setq red-conv
  (new idx3-supervised-module
    (new-lenet5 32 32 5 5 2 2 5 5 2 2 120 nsalidas parametros)
    fun-costo
    (new mmi-classer etiquetas objetivos 1 1 costo-rechazo)))

; Crear el entrenador
(setq entrenador
  (new supervised-gradient red-conv parametros))

; Un classifier-meter mide errores de clasificación
(setq mtr-entrena (new classifier-meter))
(setq mtr-valida  (new classifier-meter))

; Inicializar los pesos de la red neuronal convolucional
(==> :red-conv:machine forget 2.4 1)

; Creación de las bases de datos. Los primeros 2 argumentos
; corresponden a los tamaños de los conjuntos de entrenamiento
; y validación. El tercer argumento indica el directorio donde
; se encuentra el archivo de imágenes MNIST. El cuarto argu-
; mento indica el directorio de las etiquetas MNIST.
(crear-bd-mnist 60000 5000 *mnist-img* *mnist-etq*)

(printf "Entrenando con %d muestras de entrenamiento"
  (==> bd-entrena size))
```

```
(printf " y %d muestras de validación\n" (==> bd-valida size))

; Función para realizar el entrenamiento de la red neuronal
(de entrenar (n)
  (setq error1 100)
  (setq bandera 1)
  (setq ciclo 0)

  (while (and (< ciclo n) (= bandera 1))
    ; Estimar la segunda derivada sobre 500 iteraciones usando
    ; mu = 0.02 y asignar los valores epsilon individuales
    (printf "Calculando la diagonal de la matriz de hessian ")
    (printf "y las tasas de aprendizaje...\n")
    (==> entrenador compute-diaghessian bd-entrena 500 0.02)

    (selectq ciclo
      ((0 1) (setq eta 0.0005))
      ((2 3 4) (setq eta 0.0002))
      ((5 6 7) (setq eta 0.0001))
      ((8 9 10 11) (setq eta 0.00005))
      (t (setq eta 0.00001)))

    (printf "Entrenando la red neuronal...\n")
    (==> bd-entrena shuffle)
    (repeat 60000
      (==> entrenador train-online bd-entrena
                                   mtr-entrena 1 eta 0)
      ())

    (printf "Resultados de entrenamiento: ") (flush)
    (==> entrenador test bd-entrena mtr-entrena)
    (==> mtr-entrena display)
    (printf "Resultados de validación: ") (flush)
    (==> entrenador test bd-valida mtr-valida)
    (==> mtr-valida display)

    (writing (open-append (concat-fname *trabajo-dir*
               (concat "resultados" (str num-corrida))))
```

```
      (print (==> mtr-entrena info))
      (print (==> mtr-valida info)))

    (setq error2 (nth 4 (==> mtr-valida info)))
    (if (< error1 error2)
      (setq bandera 0)
      (setq error1 error2)
      (==> :red-conv:machine save (concat-fname *trabajo-dir*
        (concat "parametros" (str num-corrida)))))

    (setq ciclo (+ ciclo 1))))

(entrenar 20)

(if (= bandera 0)
  (setq ciclo (- ciclo 1)))

(printf "Ciclos de entrenamiento requeridos:%d\n" ciclo)
(writing (open-append (concat-fname *trabajo-dir*
           (concat "resultados" (str num-corrida))))
  (print ciclo))
```

El programa de entrenamiento utiliza la siguiente función para cargar ciertos parámetros necesarios para el entrenamiento de la red neuronal. Dichos parámetros deben estar contenidos en el archivo config.txt dentro del directorio de trabajo.

```
; Esta función lee el archivo config.txt, que contiene los va-
; lores de inicio para el entrenamiento de la red neuronal
; convolucional. Su formato es el siguiente:
;   nclases - número de clases
;   nparam - número de parámetros entrenables de la red
;   const-costo - valor de costo para la clase basura. Si se
;                 desea que este valor sea aprendido, el valor
;                 asignado deberá ser 0; sino, se debe especi-
;                 ficar un número mayor a 0 (usamos el 333).
;   nsalidas - Número de neuronas en la capa de salida.
```

```
(de leer-config(trbj-dir)
  (let ((bandera 1)
    (aux ""))

    (setq nclases "")
    (setq nparam "")
    (setq const-costo "")
    (setq nsalidas "")

    (reading (open-read (concat-fname trbj-dir "config.txt"))
      (while (<> (skip-char) "\e")
        (setq aux (read))

        (selectq bandera
          (1 (setq nclases aux))
          (2 (setq nparam aux))
          (3 (setq const-costo aux))
          (4 (setq nsalidas aux)))

        (setq bandera (+ bandera 1))))))
```

Una vez entrenada la red neuronal se debe probar. Para ello, se desarrolló el siguiente programa de prueba, con el cual se puede evaluar tanto una red neuronal convolucional general como una red neuronal convolucional especializada de forma individual.

```
; Este programa sirve para evaluar a una red neuronal convolu-
; cional sobre la base de datos MNIST (digitos escritos a ma-
; no). Este programa asume que los archivos de imágenes y e-
; tiquetas MNIST se encuentran en:
;
; LUSHDIR/local/mnist/general
;
; Si están instalados en otra ruta, simplemente se debe indi-
; car la ruta así:
;
; (setq *mnist-dir* <directorio-mnist>)
;
```

```
; Se asume que la evaluación es para una red general. Cuando
; se desee evaluar una red especializada, se debe especificar:
;
; (setq etiqueta-red <valor de la etiqueta>)
;
; Otras dos inicializaciones antes de cargar el programa son:
;
; (setq *trabajo-dir* <directorio-trabajo>)
; (setq num-corrida <n>)

(libload "gblearn2/net-lenet5")
(libload "gblearn2/gb-trainers")
(libload "gblearn2/gb-meters")
(libload "dsource-mnist")
(libload "e-s-red")

(when (not *mnist-dir*)
  (setq *mnist-dir* (concat lushdir "/local/mnist/general")))

(when (not etiqueta-red)
  (setq etiqueta-red -1))

; Función para cargar la base de datos MNIST
(de crear-bd-mnist (inicio tam img-dir)
  (setq bd-prueba
    (new dsource-idx3l-narrow
      (new dsource-mnist
        (load-matrix (concat-fname img-dir
                                   "p10k-images-idx3-ubyte"))
        (load-matrix (concat-fname img-dir
                                   "p10k-labels-idx1-ubyte"))
        32 32 0 0.01)
      tam inicio))
  ())

; Cargar los parámetros para la red:
(leer-config *trabajo-dir*)
```

```
; Llenar la matriz con el código 1-de-n
(setq etiquetas (int-matrix nclases))
(setq objetivos (float-matrix (+ nclases 1)))
(objetivos () (/ (+ nclases 1)))

(if (= etiqueta-red -1)
  (for (i 0 (- nclases 1))
    (etiquetas i i))
  (etiquetas 0 etiqueta-red)
  (etiquetas 1 15))

; Crear los parámetros entrenables
(setq parametros (new idx1-ddparam 0 nparam))

; Crear la función de costo
(setq costo-rechazo (new idx1-ddparam 0 1))
(setq fun-costo (new mmi-cost etiquetas objetivos
                              1 1 costo-rechazo))

(if (> const-costo 0)
  (==> fun-costo set-junk-cost const-costo))

; Crear la red neuronal convolucional
(setq red-conv
  (new idx3-supervised-module
    (new-lenet5 32 32 5 5 2 2 5 5 2 2 120 nsalidas parametros)
    fun-costo
    (new mmi-classer etiquetas objetivos 1 1 costo-rechazo)))

; Cargar los parámetros almacenados
(==> :red-conv:machine load (concat-fname *trabajo-dir*
  (concat "parametros" (str num-corrida))))

; Esta es la parte que corre una imagen sobre la red
(de evaluar (n)
  ; Almacena el patrón de entrada
  (setq patron (new idx3-state 1 32 32))
```

```
; Almacena la etiqueta correspondiente al patrón
(setq etiqueta (new-index (int-storage 1) '()))

; Almacena el resultado asignado al patrón por la red
(setq salida (new class-state nclases))

; Almacena la matriz de confusión con los resultados
(setq resultados (int-matrix nclases (+ nclases 1)))

(for (i 0 (- nclases 1))
  (for (j 0 nclases)
    (resultados i j 0)))

(setq i 0)
(while (< i n)
  (crear-bd-mnist i 1 *mnist-dir*)
  (==> bd-prueba fprop patron etiqueta)
  (==> red-conv use patron salida)

  ; Probando una red neuronal general
  (if (= etiqueta-red -1)
    (if (= :salida:output-class -1)
      ; La red rechazó el patrón, se incrementan los rechazos
      (resultados (etiqueta) nclases
        (+ (resultados (etiqueta) nclases) 1))

      ; La red asignó una etiqueta válida para el patrón
      (resultados (etiqueta) :salida:output-class
        (+ (resultados (etiqueta) :salida:output-class) 1)))

    ; Probando la red especializada en <etiqueta-red>
    ; El patrón debe ser rechazado por la red
    (if (<> (etiqueta) etiqueta-red)
      (if (= :salida:output-class 15)
        ; La red rechazó el patrón: no es el digito buscado
        (resultados 1 1 (+ (resultados 1 1) 1))

        (if (= :salida:output-class -1)
```

```
            ; La red rechazó el patrón: no identificado como
            ; digito
            (resultados 1 nclases (+ (resultados 1 nclases) 1))

            ; El patrón fue clasificado incorrectamente como
            ; perteneciente a la clase de la red
            (resultados 1 0 (+ (resultados 1 0) 1))))

        ; El patrón debe ser clasificado como perteneciente a
        ; la clase
        (if (= :salida:output-class 15)
          ; El patrón fue clasificado incorrectamente como
          ; no perteneciente a la clase de la red
          (resultados 0 1 (+ (resultados 0 1) 1))

          (if (= :salida:output-class -1)
            ; La red rechazó el patrón: no identificado como
            ; digito
            (resultados 0 nclases (+ (resultados 0 nclases) 1))

            ; El patrón fue clasificado correctamente como
            ; perteneciente a la clase de la red
            (resultados 0 0 (+ (resultados 0 0) 1))))))

    (setq i (+ i 1)))

  ; Guardar el resultado
  (writing (open-write (concat-fname *trabajo-dir*
             (concat "evaluacion-gen" (str num-corrida))))
    (for (i 0 (- nclases 1))
      (for (j 0 nclases)
        (printf "%d:" (resultados i j)))
      (printf "\n")))
  ())
```

Ahora bien, para probar el modelo propuesto de red neuronal convolucional modular es necesario integrar las n redes neuronales convolucionales especializadas. Para ello, se realizó el siguiente programa.

```
; Este programa sirve para evaluar a la red neuronal convolu-
; cional modular sobre la base de datos MNIST (digitos escri-
; tos a mano). Este programa asume que los archivos de imáge-
; nes y etiquetas MNIST se encuentran en:
;
; LUSHDIR/local/mnist/general
;
; Si están instalados en otra ruta, simplemente se debe indi-
; car la ruta así:
;
; (setq *mnist-dir* <directorio-mnist>)
;
; Otras tres inicializaciones antes de cargar el programa son:
;
; (setq *trabajo-dir* <directorio-trabajo: .../0>)
; (setq num-corrida <n>)
; (setq *previo-dir* <directorio previo a *trabajo-dir*>)

(libload "gblearn2/net-lenet5")
(libload "gblearn2/gb-trainers")
(libload "gblearn2/gb-meters")
(libload "dsource-mnist")
(libload "e-s-red")

(when (not *mnist-dir*)
  (setq *mnist-dir* (concat lushdir "/local/mnist/general")))

; Función para cargar la base de datos MNIST
(de crear-bd-mnist (inicio tam img-dir)
  (setq bd-prueba
    (new dsource-idx3l-narrow
      (new dsource-mnist
        (load-matrix (concat-fname img-dir
                                   "p10k-images-idx3-ubyte"))
        (load-matrix (concat-fname img-dir
                                   "p10k-labels-idx1-ubyte"))
        32 32 0 0.01)
      tam inicio))
```

```
  ())

; Función para encontrar la posición del mínimo de un vector
(de minimo (vector n)
  (setq mini (vector 0))
  (setq pos 0)
  (setq ciclo 1)

  (while (< ciclo n)
    (if (< (vector ciclo) mini)
      (progn (setq mini (vector ciclo))
        (setq pos ciclo)))

    (setq ciclo (+ ciclo 1)))
  pos)

; Cargar los parámetros para la red:
(leer-config *trabajo-dir*)

; Esta función crea una red específica para el carácter etq
(de crear_red (etq)
  ; Llena la matriz con el código 1-de-n
  (setq etiquetas (int-matrix nclases))
  (setq objetivos (float-matrix (+ nclases 1)))
  (objetivos () (/ (+ nclases 1)))
  (etiquetas 0 etq)
  (etiquetas 1 15)

  ; Crear los parámetros entrenables
  (setq parametros (new idx1-ddparam 0 nparam))

  ; Crear la función de costo
  (setq costo-rechazo (new idx1-ddparam 0 1))
  (setq fun-costo (new mmi-cost etiquetas objetivos
                                1 1 costo-rechazo))

  (if (> const-costo 0)
    (==> fun-costo set-junk-cost const-costo))
```

```
  ; Crear la red neuronal convolucional
  (setq red-conv
    (new idx3-supervised-module
      (new-lenet5 32 32 5 5 2 2 5 5 2 2 120
                  nsalidas parametros)
      fun-costo
      (new mmi-classer etiquetas objetivos
                       1 1 costo-rechazo)))

  ; Cargar los parámetros almacenados
  (==> :red-conv:machine load (concat-fname *previo-dir*
    (concat (str etq)
      (concat "/parametros" (str num-corrida)))))
  ())

; Esta es la parte que corre una imagen sobre la red
(de evaluar (n)
  ; Almacena el patrón de entrada
  (setq patron (new idx3-state 1 32 32))

  ; Almacena la etiqueta correspondiente al patrón
  (setq etiqueta (new-index (int-storage 1) '()))

  ; Almacena el resultado asignado al patrón por la red
  (setq salida (new class-state nclases))

  ; Almacena la matriz de confusión con los resultados
  (setq resultados (int-matrix 10 11))

  ; Arreglo con las etiquetas válidas para los <n> patrones
  (setq etq-validas (int-matrix n))

  ; Almacena los resultados de cada red para cada uno de los
  ; <n> patrones
  (setq redes (int-matrix 10 n))

  (for (i 0 9)
```

```
    (for (j 0 10)
      (resultados i j 0)))

  (setq i 0)
  (while (< i 10)
    ; Crear la red <i>
    (crear_red i)

    ; Probar la red <i> con los <n> patrones
    (setq j 0)
    (while (< j n)
      (crear-bd-mnist j 1 *mnist-dir*)
      (==> bd-prueba fprop patron etiqueta)
      (==> red-conv use patron salida)

      ; Inicializar las etiquetas válidas
      (if (= i 0)
        (etq-validas j (etiqueta)))

      ; Almacenar la salida de la red <i> para el patrón <j>
      (redes i j :salida:output-class)

      (setq j (+ j 1)))

    (setq i (+ i 1)))

; Revisar los resultados dados por las redes para los <n>
; patrones
(setq j 0)
(while (< j n)
  (setq rechazos 0)

  ; Revisar el resultado dado por las 10 redes para el pa-
  ; trón <j>
  (setq i 0)
  (while (< i 10)
    (if (or (= (redes i j) 15) (= (redes i j) -1))
      ; La red <i> rechazó el patrón <j>
```

```
        (setq rechazos (+ rechazos 1))

        ; La red <i> le asignó su clase al patrón <j>
        (setq clase-salida (redes i j)))
      (setq i (+ i 1)))

    (if (= rechazos 9)
      ; 9 redes rechazaron el patrón y una lo identificó co-
      ; mo perteneciente a su clase
      (resultados (etq-validas j) clase-salida
        (+ (resultados (etq-validas j) clase-salida) 1))

      ; El patrón se rechaza: las 10 redes rechazaron el pa-
      ; trón o más de una red identificó al patrón como perte-
      ; neciente a su clase
      (resultados (etq-validas j) 10
        (+ (resultados (etq-validas j) 10) 1)))

    (setq j (+ j 1)))

  ; Guardar el resultado
  (writing (open-write (concat-fname *trabajo-dir*
             (concat "evaluacion" (str num-corrida))))
    (for (i 0 9)
      (for (j 0 10)
        (printf "%d:" (resultados i j)))
      (printf "\n")))
  ())
```

Como ya se mencionó anteriormente, para el entrenamiento y prueba de las redes neuronales se empleó la base de datos MNIST. Dicha base de datos se encuentra disponible de forma gratuita en la página personal de Yann LeCun: http://yann.lecun.com/exdb/mnist/index.html. Para poder segmentar los datos para el entrenamiento de las redes neuronales se desarrollaron unos programas en el lenguaje de programación C.

Se cuenta con un programa principal para la manipulación de la base de datos MNIST que permite desplegar las etiquetas contenidas en un archivo de etiquetas y extraer ciertas etiquetas para crear un nuevo archivo de etiquetas.

Además, se puede extraer una imagen de un archivo de imágenes y guardarla como imagen pgm y crear un nuevo archivo de imágenes a partir de otro.

Para entrenar a una red neuronal convolucional especializada se requiere que el archivo de etiquetas incluya sólo dos tipos de etiquetas: una para el carácter perteneciente a la clase y otra para los demás caracteres. Este archivo de etiquetas también se puede generar con el programa mencionado en el párrafo anterior. El código de dicho programa es el siguiente.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "memoria.h"
#include "e_s_imagen.h"

int main(int argc, char *argv[])
{
  Matriz_Double imagen;
  long inicio, cantidad, pos;
  int exito, caracter;
  char nombre_entrada[256], nombre_salida[256];
  char etiquetas[60000], opc[2], aux[10];

  // El número de argumentos es incorrecto
  if(argc < 5)
  {
    fprintf(stderr, "Uso:\n\tLeer:\n\t\t");
    fprintf(stderr, "%s '-e' archivo_etiquetas ", *argv);
    fprintf(stderr, "inicio cantidad\n\t\t");
    fprintf(stderr, "%s '-i' archivo_imagenes ", *argv);
    fprintf(stderr, "inicio archivo_pgm\n\n");
    fprintf(stderr, "\tCrear:\n\t\t");
    fprintf(stderr, "%s '-e' archivo_etiquetas ", *argv);
    fprintf(stderr, "inicio cantidad archivo_salida\n\t\t");
    fprintf(stderr, "%s '-i' archivo_imagenes inicio", *argv);
    fprintf(stderr, " archivo_salida cantidad\n\n");
    fprintf(stderr, "\tCrear Modificado:\n\t\t");
    fprintf(stderr, "%s '-e' archivo_etiquetas ", *argv);
    fprintf(stderr, "inicio cantidad archivo_salida ");
```

```
    fprintf(stderr, "etiqueta_deseada\n");
    exit(1);
  }
  else
  {
    //opc: -e trabajar con etiquetas, -i trabajar con imágenes
    strncpy(opc, *(argv + 1), 2);

    // Archivo de entrada
    strncpy(nombre_entrada, *(argv + 2), 256);

    // Apartir de que posición se empieza la lectura
    strncpy(aux, *(argv + 3), 10);
    inicio = atoi(aux);

    // Trabajar con el archivo de etiquetas
    if(strncmp(opc, "-e", 2) == 0)
    {
      // Cantidad de etiquetas por leer
      strncpy(aux, *(argv + 4), 10);
      cantidad = atoi(aux);

      /* Se lee la <cantidad> solicitada de etiquetas del ar-
         chivo <nombre_entrada>, empezando desde <inicio> y se
         guardan en <etiquetas> */
      cantidad = leer_etq(nombre_entrada, etiquetas,
                          inicio, cantidad);

      // Desplegar las etiquetas en pantalla
      if(argc == 5)
      {
        inicio = 0;
        while(inicio < cantidad)
        {
          fprintf(stderr, "Etq: %c\t", etiquetas[inicio]);
          ++inicio;
        }
```

```
      fprintf(stderr, "\n");
    }
    // Guardar las etiquetas en un archivo
    else
    {
      // Archivo de salida para las etiquetas
      strncpy(nombre_salida, *(argv + 5), 256);

      if(argc == 6)
        /* Crear un archivo de etiquetas a partir del arre-
           glo <etiquetas> sin alterar las etiquetas */
        crear_etq(nombre_salida, etiquetas, cantidad, -1);
      else
      {
        strncpy(aux, "\0\0\0\0\0\0\0\0\0\0", 10);
        strncpy(aux, *(argv + 6), 1);
        caracter = atoi(aux);

        /* Crear un archivo de etiquetas a partir del arre-
           glo <etiquetas> respetando <caracter>, y cambian-
           do las demás etiquetas por un 15 */
        crear_etq(nombre_salida, etiquetas,
                  cantidad, caracter);
      }
    }

    fprintf(stderr, "El número de etiquetas es: %ld\n",
            cantidad);
}
// Trabajar con el archivo de imágenes
else if(strncmp(opc, "-i", 2) == 0)
{
    // Archivo de salida
    strncpy(nombre_salida, *(argv + 4), 256);

    iniciar_matriz_double(&imagen);

    /* Leer una imagen del archivo <nombre_entrada> de la
```

```
   posición <inicio> y guardarla en <imagen> */
exito = leer_img(nombre_entrada, &imagen, inicio);

if(exito)
  if(argc == 5)
    // Guardar la imagen del carácter
    guardar_pgm(nombre_salida, &imagen);
  else
  {
    // Cantidad de imágenes por leer
    strncpy(aux, *(argv + 5), 10);
    cantidad = atoi(aux);

    // Crear el encabezado del archivo de imágenes
    exito = crear_cabeza_img(nombre_salida, cantidad,
                             imagen.rens, imagen.cols);

    if(exito)
    {
      pos = 0;
      while(pos < cantidad)
      {
        // Leer la siguiente imagen del archivo
        if(leer_img(nombre_entrada, &imagen, inicio))
          // Agregar la imagen al nuevo archivo
          agregar_img(nombre_salida, &imagen);
        else
        {
          eliminar_matriz_double(&imagen);
          fprintf(stderr, "Error al leer la imagen\n");
          exit(1);
        }

        ++pos;
        ++inicio;
      }
    }
    else
```

```
        {
          eliminar_matriz_double(&imagen);
          fprintf(stderr, "Error al crear el encabezado\n");
          exit(1);
        }
      }
    else
    {
      eliminar_matriz_double(&imagen);
      fprintf(stderr, "Error al leer la imagen\n");
      exit(1);
    }

    // Liberar memoria
    eliminar_matriz_double(&imagen);
  }
  // opc inválida
  else
  {
    fprintf(stderr, "Uso:\n\tLeer:\n\t\t");
    fprintf(stderr, "%s '-e' archivo_etiquetas ", *argv);
    fprintf(stderr, "inicio cantidad\n\t\t");
    fprintf(stderr, "%s '-i' archivo_imagenes ", *argv);
    fprintf(stderr, "inicio archivo_pgm\n\n");
    fprintf(stderr, "\tCrear:\n\t\t");
    fprintf(stderr, "%s '-e' archivo_etiquetas ", *argv);
    fprintf(stderr, "inicio cantidad archivo_salida\n\t\t");
    fprintf(stderr, "%s '-i' archivo_imagenes ", *argv);
    fprintf(stderr, "inicio archivo_salida cantidad\n\n");
    fprintf(stderr, "\tCrear Modificado:\n\t\t");
    fprintf(stderr, "%s '-e' archivo_etiquetas ", *argv);
    fprintf(stderr, "inicio cantidad archivo_salida ");
    fprintf(stderr, "etiqueta_deseada\n");
    exit(1);
  }
}

exit(0);
```

```
}
```

Las librerías memoria.h y e_s_imagen.h incluyen las definiciones de las funciones que utiliza este programa, cuyo código se halla en los correspondientes archivos fuente memoria.c y e_s_imagen.c. Las funciones de e_s_imagen.c permiten crear la imagen pgm y leer y crear los archivos de etiquetas e imágenes. El contenido de dicho archivo se muestra enseguida.

```
#include "e_s_imagen.h"

/* Guarda la imagen procesada en un archivo pgm */
void guardar_pgm(char nombre[], Matriz_Double *imagen)
{
  Matriz_Uchar imagen_u;
  FILE *arch_img;

  arch_img = fopen(nombre, "w");

  if(!arch_img)
  {
    fprintf(stderr,
      "¡Error: No se pudo crear el archivo %s!\n", nombre);
    exit(2);
  }

  fprintf(arch_img, "P5\n%d %d\n255\n",
          imagen->cols, imagen->rens);

  iniciar_matriz_uchar(&imagen_u);
  convertir_double2uchar(imagen, &imagen_u);
  fwrite(*imagen_u.datos, sizeof(unsigned char),
         imagen_u.tam, arch_img);

  eliminar_matriz_uchar(&imagen_u);
  fclose(arch_img);
}

/* Lee un máximo de <num_etq> etiquetas del archivo de etique-
```

```
   tas empezando en <inicio>, las almacena en <etiquetas> y
   devuelve el número de etiquetas leído. El formato del ar-
   chivo de etiquetas es:

[offset] [tipo]          [valor]          [descripción]
0000     32 bit integer  0x00000801(2049) magic number
0004     32 bit integer  60000            número de etiquetas
0008     unsigned byte   ¿?               etiqueta
0009     unsigned byte   ¿?               etiqueta
...
xxxx     unsigned byte   ¿?               etiqueta */
long leer_etq(char nom_arch[], char etiquetas[],
              long inicio, long num_etq)
{
  long tot_etq, pos;
  unsigned char aux;
  FILE *arch_etq;

  arch_etq = fopen(nom_arch, "r");

  if(!arch_etq)
  {
    fprintf(stderr, "¡Error: No se pudo abrir el archivo ");
    fprintf(stderr, "de etiquetas %s!\n", nom_arch);
    exit(2);
  }

  fread(&aux, sizeof(unsigned char), 1, arch_etq);
  tot_etq = (int) aux * 16777216;

  fread(&aux, sizeof(unsigned char), 1, arch_etq);
  tot_etq += (int) aux * 65536;

  fread(&aux, sizeof(unsigned char), 1, arch_etq);
  tot_etq += (int) aux * 256;

  fread(&aux, sizeof(unsigned char), 1, arch_etq);
  tot_etq += (int) aux;
```

```
// El número no corresponde a un archivo de etiquetas
if(tot_etq != 2049)
{
  fclose(arch_etq);
  return 0;
}

fread(&aux, sizeof(unsigned char), 1, arch_etq);
tot_etq = (int) aux * 16777216;

fread(&aux, sizeof(unsigned char), 1, arch_etq);
tot_etq += (int) aux * 65536;

fread(&aux, sizeof(unsigned char), 1, arch_etq);
tot_etq += (int) aux * 256;

fread(&aux, sizeof(unsigned char), 1, arch_etq);
tot_etq += (int) aux;

if(inicio < 0 || num_etq < 0 || tot_etq < inicio + num_etq)
{
  fclose(arch_etq);
  return 0;
}

fseek(arch_etq, inicio + 8, SEEK_SET);

pos = inicio;
while(pos < inicio + num_etq)
{
  fread(&aux, sizeof(unsigned char), 1, arch_etq);
  etiquetas[pos - inicio] = (char) (48 + (int) aux);

  ++pos;
}

fclose(arch_etq);
```

```
  return pos - inicio;
}

/* Extrae la imagen de la posición <num_img> del archivo de i-
   mágenes y la almacena en <imagen>. Devuelve 1 en caso de
   éxito, 0 en caso de error. El formato del archivo de imáge-
   nes es:

[offset] [tipo]          [valor]          [descripción]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            número de imágenes
0008     32 bit integer  28               número de renglones
0012     32 bit integer  28               número de columnas
0016     unsigned byte   ¿?               pixel
0017     unsigned byte   ¿?               pixel
...
xxxx     unsigned byte   ¿?               pixel */
int leer_img(char nom_arch[], Matriz_Double *imagen,
             long num_img)
{
  Matriz_Uchar imagen_u;
  long tot_img;
  int rens, cols;
  unsigned char aux;
  FILE *arch_img;

  arch_img = fopen(nom_arch, "r");

  if(!arch_img)
  {
    fprintf(stderr, "¡Error: No se pudo abrir el archivo ");
    fprintf(stderr, "de imágenes %s!\n", nom_arch);
    exit(2);
  }

  fread(&aux, sizeof(unsigned char), 1, arch_img);
  tot_img = (int) aux * 16777216;
```

```
fread(&aux, sizeof(unsigned char), 1, arch_img);
tot_img += (int) aux * 65536;

fread(&aux, sizeof(unsigned char), 1, arch_img);
tot_img += (int) aux * 256;

fread(&aux, sizeof(unsigned char), 1, arch_img);
tot_img += (int) aux;

// El número no corresponde a un archivo de imágenes
if(tot_img != 2051)
{
  fclose(arch_img);
  return 0;
}

fread(&aux, sizeof(unsigned char), 1, arch_img);
tot_img = (int) aux * 16777216;

fread(&aux, sizeof(unsigned char), 1, arch_img);
tot_img += (int) aux * 65536;

fread(&aux, sizeof(unsigned char), 1, arch_img);
tot_img += (int) aux * 256;

fread(&aux, sizeof(unsigned char), 1, arch_img);
tot_img += (int) aux;

if(num_img < 0 || num_img >= tot_img)
{
  fclose(arch_img);
  return 0;
}

fread(&aux, sizeof(unsigned char), 1, arch_img);
rens = (int) aux * 16777216;
```

```
  fread(&aux, sizeof(unsigned char), 1, arch_img);
  rens += (int) aux * 65536;

  fread(&aux, sizeof(unsigned char), 1, arch_img);
  rens += (int) aux * 256;

  fread(&aux, sizeof(unsigned char), 1, arch_img);
  rens += (int) aux;

  fread(&aux, sizeof(unsigned char), 1, arch_img);
  cols = (int) aux * 16777216;

  fread(&aux, sizeof(unsigned char), 1, arch_img);
  cols += (int) aux * 65536;

  fread(&aux, sizeof(unsigned char), 1, arch_img);
  cols += (int) aux * 256;

  fread(&aux, sizeof(unsigned char), 1, arch_img);
  cols += (int) aux;

  iniciar_matriz_uchar(&imagen_u);
  crear_matriz_uchar(rens, cols, &imagen_u);

  fseek(arch_img, num_img * imagen_u.tam + 16, SEEK_SET);
  fread(*imagen_u.datos, sizeof(unsigned char),
        imagen_u.tam, arch_img);
  fclose(arch_img);

  convertir_uchar2double(&imagen_u, imagen);
  eliminar_matriz_uchar(&imagen_u);

  return 1;
}

/* Crea el archivo de etiquetas <nom_arch> y devuelve el núme-
   ro de etiquetas creado. Cuando se desea conservar sólo una
   etiqueta se da su valor númerico en <caracter>, de otra
```

forma, se pasa -1. El formato del archivo de etiquetas es:

```
[offset] [tipo]          [valor]          [descripción]
0000     32 bit integer  0x00000801(2049) magic number
0004     32 bit integer  60000            número de etiquetas
0008     unsigned byte   ¿?               etiqueta
0009     unsigned byte   ¿?               etiqueta
...
xxxx     unsigned byte   ¿?               etiqueta */
long crear_etq(char nom_arch[], char etiquetas[],
               long num_etq, int caracter)
{
  long pos;
  unsigned char aux_char, hexadecimal[4];
  FILE *arch_etq;

  arch_etq = fopen(nom_arch, "w");

  if(!arch_etq)
  {
    fprintf(stderr, "¡Error: No se pudo abrir el archivo ");
    fprintf(stderr, "de etiquetas %s!\n", nom_arch);
    exit(2);
  }

  if(num_etq < 0)
  {
    fclose(arch_etq);
    return 0;
  }

  DecToHex4(2049, hexadecimal);

  pos = 0;
  while(pos < 4)
  {
    fwrite(&hexadecimal[pos], sizeof(unsigned char),
           1, arch_etq);
```

```
    ++pos;
  }

  DecToHex4(num_etq, hexadecimal);

  pos = 0;
  while(pos < 4)
  {
    fwrite(&hexadecimal[pos], sizeof(unsigned char),
           1, arch_etq);
    ++pos;
  }

  pos = 0;
  while(pos < num_etq)
  {
    aux_char = (unsigned char) etiquetas[pos] - 48;

    if(caracter != -1)
      if(aux_char != caracter)
        aux_char = 15;

    fwrite(&aux_char, sizeof(unsigned char), 1, arch_etq);

    ++pos;
  }

  fclose(arch_etq);

  return pos;
}

/* Crea el encabezado del archivo de imágenes <nom_arch>. De-
   vuelve 1 en caso de éxito, 0 en caso de error. El formato
   del encabezado es:

[offset] [tipo]          [valor]          [descripción]
0000     32 bit integer  0x00000803(2051) magic number
```

```
0004     32 bit integer  60000           número de imágenes
0008     32 bit integer  28              número de renglones
0012     32 bit integer  28              número de columnas */
int crear_cabeza_img(char nom_arch[], long num_img,
                     int renglones, int columnas)
{
  long pos;
  unsigned char hexadecimal[4];
  FILE *arch_img;

  arch_img = fopen(nom_arch, "w");

  if(!arch_img)
  {
    fprintf(stderr, "¡Error: No se pudo abrir el archivo ");
    fprintf(stderr, "de imágenes %s!\n", nom_arch);
    exit(2);
  }

  if(num_img < 1 || renglones < 1 || columnas < 1)
  {
    fclose(arch_img);
    return 0;
  }

  DecToHex4(2051, hexadecimal);

  pos = 0;
  while(pos < 4)
  {
    fwrite(&hexadecimal[pos], sizeof(unsigned char),
           1, arch_img);
    ++pos;
  }

  DecToHex4(num_img, hexadecimal);

  pos = 0;
```

```
  while(pos < 4)
  {
    fwrite(&hexadecimal[pos], sizeof(unsigned char),
           1, arch_img);
    ++pos;
  }

  DecToHex4(renglones, hexadecimal);

  pos = 0;
  while(pos < 4)
  {
    fwrite(&hexadecimal[pos], sizeof(unsigned char),
           1, arch_img);
    ++pos;
  }

  DecToHex4(columnas, hexadecimal);

  pos = 0;
  while(pos < 4)
  {
    fwrite(&hexadecimal[pos], sizeof(unsigned char),
           1, arch_img);
    ++pos;
  }

  fclose(arch_img);

  return 1;
}

/* Agrega <imagen> al archivo de imágenes <nom_arch>. Devuelve
   1 en caso de éxito, 0 en caso de error. El formato del ar-
   chivo de imágenes es:

[offset] [tipo]          [valor]          [descripción]
0000     32 bit integer  0x00000803(2051) magic number
```

```
0004     32 bit integer  60000           número de imágenes
0008     32 bit integer  28              número de renglones
0012     32 bit integer  28              número de columnas
0016     unsigned byte   ¿?              pixel
0017     unsigned byte   ¿?              pixel
...
xxxx     unsigned byte   ¿?              pixel */
int agregar_img(char nom_arch[], Matriz_Double *imagen)
{
  Matriz_Uchar imagen_u;
  FILE *arch_img;

  arch_img = fopen(nom_arch, "a");

  if(!arch_img)
  {
    fprintf(stderr, "¡Error: No se pudo abrir el archivo ");
    fprintf(stderr, "de imágenes %s!\n", nom_arch);
    exit(2);
  }

  iniciar_matriz_uchar(&imagen_u);
  convertir_double2uchar(imagen, &imagen_u);
  fwrite(*imagen_u.datos, sizeof(unsigned char),
         imagen_u.tam, arch_img);

  eliminar_matriz_uchar(&imagen_u);
  fclose(arch_img);

  return 1;
}

/* Convierte un número decimal en un número hexadecimal de 4
   bytes */
void DecToHex4(int decimal, unsigned char hexadecimal[])
{
  int pos, residuo1, residuo2;
```

```
  decimal = abs(decimal);
  pos = 3;

  while(pos >= 0)
  {
    residuo1 = decimal % 16;
    decimal /= 16;

    residuo2 = decimal % 16;
    decimal /= 16;

    hexadecimal[pos] = residuo1 + residuo2 * 16;

    --pos;
  }
}
```

Por su parte, el almacenamiento de la imagen se realiza a través del uso de memoria dinámica. Las funciones que gestionan la memoria con el sistema operativo se encuentran en el archivo memoria.c. A continuación se muestran dichas funciones.

```
#include "memoria.h"

/* Inicialización de la matriz de unsigned char */
void iniciar_matriz_uchar(Matriz_Uchar *m)
{
  m->datos = NULL;
  m->rens = m->cols = m->tam = 0;
}

/* Asignación de espacio en memoria para matriz de unsigned
   char */
void crear_matriz_uchar(int r, int c, Matriz_Uchar *m)
{
  int i;
  unsigned char *apuntador;
```

```
  m->rens = r;
  m->cols = c;
  m->tam = r * c;

  if(!(m->datos = calloc(r, sizeof(unsigned char *))))
    fprintf(stderr,
            "No se pudo reservar memoria para la matriz\n");

  if(!(m->datos[0] = calloc(m->tam, sizeof(unsigned char))))
    fprintf(stderr,
            "No se pudo reservar memoria para la matriz\n");

  apuntador = m->datos[0] + c;

  for(i = 1; i < r; i++, apuntador += c)
    m->datos[i] = apuntador;
}

/* Liberación de espacio en memoria para matriz de unsigned
   char */
void eliminar_matriz_uchar(Matriz_Uchar *m)
{
  if(m->datos)
  {
    if(m->datos[0])
      free(m->datos[0]);

    free(m->datos);
  }

  m->datos = NULL;
  m->rens = m->cols = m->tam = 0;
}

/* Inicialización de la matriz de double */
void iniciar_matriz_double(Matriz_Double *m)
{
  m->datos = NULL;
```

```
  m->rens = m->cols = m->tam = 0;
}

/* Asignación de espacio en memoria para matriz de double */
void crear_matriz_double(int r, int c, Matriz_Double *m)
{
  double *apuntador;
  int i;

  m->rens = r;
  m->cols = c;
  m->tam = r * c;

  if(!(m->datos = calloc(r, sizeof(double *))))
    fprintf(stderr,
            "No se pudo reservar memoria para la matriz\n");

  if(!(m->datos[0] = calloc(m->tam, sizeof(double))))
    fprintf(stderr,
            "No se pudo reservar memoria para la matriz\n");

  apuntador = m->datos[0] + c;

  for(i = 1; i < r; i++, apuntador += c)
    m->datos[i] = apuntador;
}

/* Liberación de espacio en memoria para matriz de double */
void eliminar_matriz_double(Matriz_Double *m)
{
  if(m->datos)
  {
    if(m->datos[0])
      free(m->datos[0]);

    free(m->datos);
  }
```

```
  m->datos = NULL;
  m->rens = m->cols = m->tam = 0;
}

/* Copia la matriz de double <original> en <copia> */
void copiar_matriz_double(Matriz_Double *original,
                          Matriz_Double *copia)
{
  double *apuntador, *fin, *aux;

  eliminar_matriz_double(copia);
  crear_matriz_double(original->rens, original->cols, copia);

  for(apuntador = *original->datos, fin = apuntador +
      original->tam, aux = *copia->datos; apuntador < fin;
      apuntador++, aux++)
    *aux = *apuntador;
}

/* Convierte una matriz de double en una matriz de unsigned
   char */
void convertir_double2uchar(Matriz_Double *d, Matriz_Uchar *u)
{
  register double *apuntador_d, *fin;
  register unsigned char *apuntador_u;

  eliminar_matriz_uchar(u);
  crear_matriz_uchar(d->rens, d->cols, u);

  for(apuntador_d = *d->datos, fin = apuntador_d + d->tam,
      apuntador_u = *u->datos; apuntador_d < fin;
      ++apuntador_d, ++apuntador_u)
    *apuntador_u = (unsigned char)*apuntador_d;
}

/* Convierte una matriz de unsigned char en una matriz de
   double */
void convertir_uchar2double(Matriz_Uchar *u, Matriz_Double *d)
```

```
{
  register double *apuntador_d;
  register unsigned char *apuntador_u, *fin;

  eliminar_matriz_double(d);
  crear_matriz_double(u->rens, u->cols, d);

  for(apuntador_u = *u->datos, fin = apuntador_u + u->tam,
      apuntador_d = *d->datos; apuntador_u < fin;
      ++apuntador_u, ++apuntador_d)
    *apuntador_d = (double)*apuntador_u;
}
```

Los tipos de datos Matriz_Uchar y Matriz_Double están definidos de la siguiente forma:

```
/* Estructura de tipo matriz para unsigned char */
typedef struct{
  unsigned char **datos;
  int rens, cols;
  long tam;
} Matriz_Uchar;

/* Estructura de tipo matriz para double */
typedef struct{
  double **datos;
  int rens, cols;
  long tam;
} Matriz_Double;
```

A.2 El Método de Corrección Ortográfica

Por su parte, los métodos de corrección de textos electrónicos fueron implementados utilizando el lenguaje de programación Perl (de Practical Extraction and Report Language), el cual es un software de libre distribución, bajo la licencia GNU (General Public License). Se utilizó la vesión 5.8.0 de su intérprete. La elección de este lenguaje se debe a que posee una amplia gama de funciones para el procesamiento de patrones, lo que permite implementar

funciones para el procesamiento de texto de forma más rápida, de esta manera se puede enfocar el esfuerzo en la investigación y el diseño del método de corrección y no tanto en la implementación del mismo. El intérprete puede obtenerse en la siguiente dirección electrónica: www.perl.com/download.csp A continuación se presenta el código fuente que se utilizó en los pasos siguientes.

A.2.1 Creación del modelo de lenguaje basado en bigramas

En primer término se muestran los programas utilizados para contar las frecuencias de los bigramas contenidos en el corpus de texto. Se asume que la estructura de los directorios es la siguiente:

- Directorio donde se encuentra el corpus:

  <directorio de trabajo>/corpus/Textos/CorpCompleto/

- Directorio donde están los programas que generan los bigramas de palabras:

  <directorio de trabajo>/Herramientas/

- Directorio donde se hallan los correctores ortográficos automáticos:

  <directorio de trabajo>/Corrector/

- Directorio donde se ubica el etiquetador automático:

  <directorio de trabajo>/Etiquetador/etq/Bin_and_Data/

A continuación se presenta el programa que obtuvo los bigramas de palabras del corpus de texto.

```
#!/usr/bin/perl
#
# CreaBigrs.pl
# Programa que crea los bigramas a partir de los
# archivos del corpus textual

# Carga el contenido del arhivo que contiene las funciones
# que calculan los bigramas de palabras.
```

```
require "bNgramasP2.pl";

if(@ARGV != 0){
  print "ERROR: No necesita argumentos\n";
  print "uso: CreaBigrs.pl\n";
  exit (0);
}

$total = 0;
$na = 0;

# Guarda los nombres de todos los archivos con extensión .txt
# que encuentre en el ditectorio indicado
@ARCHIVOS = <../corpus/Textos/CorpCompleto/*.txt>;

# Realiza el procesamiento por cada uno de los
# archivos encontrados en el directorio indicado.
# Crea un archivo de bigramas por cada archivo de texto.
foreach $arch_txt (@ARCHIVOS){
  print "\nProcesando $arch_txt\n";
  open(ARCH, $arch_txt) ||
      die("ERROR: no pudo abrirse el archivo $arch_txt\n");
  @lineas = <ARCH>;
  close ARCH;
  &bigramas;
  print "Bigramas de $arch_txt obtenidos\n";
  open(ARCHS, ">../corpus/Ngramas/BiGRAMAS$na") ||
      die("ERROR: no pudo crearse el archivo BiGRAMAS$na\n");
  &imprimeArch;
  $na++;
}
# Renombra el último archivo de bigramas y elimina
# los archivos de bigramas generados anteriormente
# con excepción del último, el cual contiene todos
# los bigramas del corpus completo.
$na--;
system("mv ../corpus/Ngramas/BiGRAMAS$na
       ../corpus/Ngramas/cBiGRAMAS.TODO");
```

```
system("rm -f ../corpus/Ngramas/BiGRAMAS*");

undef %ngrs;
```

Enseguida se muestra el archivo bNgramasP2.pl

```
#!/usr/bin/perl
#
# bNgramasP2.pl
#
# Contiene las funciones que calcula bigramas de palabras,
# genera una lista (almacenada en el archivo de salida) con
# la frecuencia de cada bigrama encontrado en el texto.

# Guarda en un archivo los bigramas encontrados
sub imprimeArch{
  $total = keys %ngrms;
  foreach(sort keys (%ngrms)){
    print ARCHS "$_|$ngrms{$_}\n";
  }
  print "No. de n-gramas: $total\n";
  print "Se ha creado el archivo: BiGRAMAS$na\n";
  close ARCHS;
}

# Obtiene monogramas de cada archivo
sub unigramas{
  foreach(@lineas){
    # Elimina los espacios en blanco y cambia las
    # letras mayúsculas por minúsculas
    s/^\s+//;
    s/\s+$//;
    s/  / /g;
    tr/A-Z/a-z/;
    tr/Á/á/;
    tr/É/é/;
    tr/Í/í/;
    tr/Ú/ú/;
```

```
    tr/Ő/ó/;
    tr/Ú/ú/;
    tr/Ü/ü/;
    tr/Ñ/ñ/;
    # Si no se eliminó la línea, divide las palabras
    # de ésta y obtiene sus monogramas
    if(!/^$/){
      @palabras = split(/\s+/, $_);
      for($cont = 0;  $cont <= $#palabras; ++$cont){
        ++$ngrms{$palabras[$cont]};
      }
    }
  }
}

# Obtiene los bigramas de palabras de cada archivo
sub bigramas{
  # Elimina los espacios en blanco y cambia las
  # letras mayúsculas por minúsculas
  foreach(@lineas){
    s/^\s+//;
    s/\s+$//;
    tr/A-Z/a-z/;
    tr/Á/á/;
    tr/É/é/;
    tr/Í/í/;
    tr/Ú/ú/;
    tr/Ő/ó/;
    tr/Ú/ú/;
    tr/Ü/ü/;
    tr/Ñ/ñ/;
    # Si no se eliminó la línea, divide las palabras
    # de ésta y obtiene sus bigramas
    if(!/^$/){
      @palabras = split(/\s+/, $_);
      # Inserta la pseudopalabra <I> al inicio
      # para que forme el primer bigrama
      if($palabras[0] ne ""){
```

```
        ++$ngrms{"<I>". " " . $palabras[0]};
      }
      for($cont = 1; $cont <= $#palabras; ++$cont){
        if($palabras[$cont - 1] ne ""){
          ++$ngrms{$palabras[$cont - 1]." ".$palabras[$cont]};
        }
      }
    }
  }
}

# Obtiene trigramas de palabras de cada archivo
sub trigramas{
  # Elimina los espacios en blanco y cambia las
  # letras mayúsculas por minúsculas
  foreach(@lineas){
    s/^\s+//;
    s/\s+$//;
    s/  / /g;
    tr/A-Z/a-z/;
    tr/Á/á/;
    tr/É/é/;
    tr/Í/í/;
    tr/Ú/ú/;
    tr/Ó/ó/;
    tr/Ú/ú/;
    tr/Ü/ü/;
    tr/Ñ/ñ/;
    # Si no se eliminó la línea, divide las palabras
    # de ésta y obtiene sus trigramas
    if(! /^$/){
      @pals = split(/\s+/, $_);
      # Inserta la pseudopalabra <I> al inicio
      # para que forme el primer trigrama
      if($pals[0] ne ""){
        ++$ngrms{"<I> ". $pals[0]." " .$pals[1]};
      }
      for($k = 1;  $k <= $#pals-1; ++$k){
```

```
        if($pals[$k - 1] ne "" && $pals[$k + 1] ne ""){
          ++$ngrms{$pals[$k-1]." ".$pals[$k]." ".$pals[$k+1]};
        }
      }
    }
  }
}
1;
```

Por último, se muestra a continuación el programa utilizado para calcular las probabilidades de los bigramas de palabras mediante el descuento Good-Turing. Este programa trabaja sobre el archivo de bigramas creado por el programa CreaBigrs.pl, presentado anteriormente.

```
#!/usr/bin/perl
#
# goodTuring.pl
# Programa que calcula el descuento de good-turing
# para un lexicón de bigramas. Utiliza el proceso siguiente:
#
# Si c > k
#   c* = c
# si no
#   c* = (c + 1) * (N[c + 1] / N[c])
#
# c = no. de veces que aparece un n-grama en el corpus
# c* = no. de veces que aparece un n-grama en el corpus
#      después de aplicar el descuento
# N[c] = no. de n-gramas que aparecen c veces en el corpus
# N[c+1] = no. de n-gramas que aparecen c+1 veces en el corpus
# k = umbral de descuento, determina si se calcula el descuen-
#     to para un n-grama que aparece c veces en el corpus

$k = 5;

if(@ARGV  < 1){
  print "ERROR: Necesita argumentos\n";
  print "uso: goodTuring.pl arch_n-gramas\n";
```

```
  exit(0);
}

$total = 0;
$na = 0;
$archNgrs = $ARGV[0];

# Abre el archivo de bigramas y crea un archivo
# para guardar los bigramas y el registro de operación
open(ARCH, "$archNgrs") ||
    die("ERROR: no puede abrirse el archivo $archNgrs\n");
open(LOG, ">LogGT") ||
    die("ERROR: no puede abrirse el archivo LogGT\n");
print "\nProcesando $archNgrs\n";

@NGramas = <ARCH>;
close ARCH;

# Toma los bigramas y sus frecuencias
foreach $ngram (@NGramas){
  $ngram =~ s/\s+$//;
  ($bigr, $frec) = split(/\|/, $ngram);
  $frecNgram{$frec}++;
  $ngrs{$bigr} += $frec;
}

print LOG "N-gramas (n-grama|frecuencia)\n";
print LOG "Original\t\t\tDescontado\n";

# Aplica el descuento a cada bigrama
foreach $ngram (keys %ngrs){
  $fngram = $ngrs{$ngram};
  print LOG "$ngram|$fngram\t";
  if(!$frecNgram{$fngram}){
    $fngram = 1/$#NGramas;
  }
  elsif($fngram < $k){
    $cm1 = $fngram + 1;
```

```
    &buscaCm1;
    $fngram = $cm1 * ($frecNgram{$cm1} / $frecNgram{$fngram});
    $ngrs{$ngram} = $fngram;
  }
  else{
    $fngram /= $#NGramas;
  }
  print LOG "\t\t$fngram\n";
}

close LOG;

# Esta función busca la frecuencia siguiente de los bigramas
# que se están calculando. Esto con el fin de evitar que se
# asigne una frecuencia superior inexistente.
sub buscaCm1{
  if($frecNgram{$cm1} eq ""){
    $cm1++;
    &buscaCm1;
  }
}
```

A.2.2 Entrenamiento del Etiquetador Automático TBL

Como se mencionó anteriormente, en esta tesis se utilizó un etiquetador automático basado en el algoritmo de aprendizaje llamado *Transformation Based Learning* (TBL). Dicho etiquetador puede ser obtenido de las direcciones siguientes:

- http://research.microsoft.com/~brill/
- http://www.cs.jhu.edu/~brill/
- http://www.cs.jhu.edu/~brill/RBT1_14.tar.Z
- http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/parsing/taggers/brill/0.html
- ftp.cs.jhu.edu:/pub/brill/Programs/

- ftp.cs.jhu.edu:/pub/brill/Papers/

A continuación se presenta el *script* utilizado para entrenar el etiquetador automático basado en TBL, el cual utiliza las herramientas que la misma distribución del etiquetador incluye.

```
# Entrena.sh
# Dividir el corpus etiquetado de texto en dos partes
cat CORPUS.ESP | perl ../Utilities/divide-in-two-rand.prl \
 CORPUS1 CORPUS2

# Crear el corpus no etiquetado
cat CORPUS1 | perl ../Utilities/tagged-to-untagged.prl \
 > CORPUSNE

# Crear archivos: bigramas, listas y lexicones
cat CORPUSNE | perl ../Utilities/wordlist-make.prl | \
 sort +1 -rn | awk '{print $1}' > LGDEPAL
cat CORPUS1 | perl ../Utilities/word-tag-count.prl | \
 sort +2 -rn > LPEQPALETQ
cat CORPUSNE | perl ../Utilities/bigram-generate.prl | \
 awk '{print $1,$2}' > LGDEBIGRS

# Ejecutar el programa de aprendizaje
perl ../Learner_Code/unknown-lexical-learn.prl LGDEPAL \
 LPEQPALETQ LGDEBIGRS 30 ARCHREGLEX

# Seguir entrenando
perl ../Learner_Code/unknown-lexical-learn-continue.prl \
 LGDEPAL LPEQPALETQ LGDEBIGRS 30 ARCHREGLEX ARCHVREGLEX

# Concatenar las reglas anteriores con las nuevas
cat ARCHREGLEX >> ARCHVREGLEX

# Crear el lexicón de entrenamiento para aprender
# reglas contextuales
cat CORPUS1 | perl ../Utilities/make-restricted-lexicon.prl \
 > LEXICON.ENTR
```

```
cat CORPUS.ESP |perl ../Utilities/make-restricted-lexicon.prl\
 > LEXICON.FINAL
cat CORPUS2 | perl ../Utilities/tagged-to-untagged.prl\
 > CORPUSNE2

# Ejecutar el etiquetador
./tagger LEXICON.ENTR CORPUSNE2 LGDEBIGRS ARCHREGLEX \
 /dev/null -w LGDEPAL -i TONTOCorpus

# Aprender las reglas contextuales
./contextual-rule-learn CORPUS2 TONTOCorpus ARCHREGCONTX \
 LEXICON.ENTR

# comando para etiquetar morfosintáctiamente un texto
#./tagger LEXICON.ENTR CorpNE LGDEBIGRS ARCHREGLEX /dev/null\
 -w LGDEPAL -i TONTOCorpNE
```

A.2.3 Correctores automáticos CATMC y CATCG

A continuación se presenta el código principal de los correctores automáticos (programa analizar.pl) y posteriormente se muestra el código de las funciones que cada uno de ellos utiliza (bSinEtq.pl y bConEtq.pl).

Los dos correctores pueden se ejecutados desde el programa analizar.pl mediante una opción que se incluye como parámetro al invocar al programa ya que están definidos como funciones. En primer lugar aparece el código del corrector ortográfico automático que realiza un análisis morfológico y contextual de las palabras del texto a corregir CATMC. Enseguida se lista el código del corrector ortográfico automático que realiza un análisis morfológico y contextual basado en las categorías gramaticales de las palabras del texto a corregir CATCG.

Por último se muestran las funciones comunes a los dos correctores ortográficos implementados en esta tesis: CATMC y CATCG. Estas funciones, que se utilizan para realizar las comparaciones y la selección de la mejor corrección potencial para cada palabra detectada como incorrecta, están contenidas en el archivo bMorfolog.pl

Cabe aclarar que se muestran aquellas funciones que son relevantes para el proceso de corrección, esto es, las que crean los lexicones de búsqueda y las que influyen en la detección, generación de correcciones potenciales y en

la selección de éstas.

```
#!/usr/bin/perl
#
# analizar.pl
# Este programa ejecuta uno de los dos correctores
# ortográficos automáticos: CATMC y CATCG.
#
# Requiere una opción y los archivos de entrada y salida:
#   analizar.pl <opción> <arch_entrada> <arch_salida>
#

require "./archivos.cfg"; # archivo de configuración
require "./bMorfolog.pl"; # funciones de análisis morfológico

$opc = $ARGV[0];
$arche = $ARGV[1];
$archs = $ARGV[2];
$logErr = $archs;
$logErr =~ s/corr$/LOG/;

open(STDERR, ">$logErr");

$mensaje = "Uso:\tperl analizar.pl [S|E] <arch_entrada.txt>
<arch_salida.txt>\n\tDonde:\n\t\tS: Corrección Sin etiquetado
morfológico\n\t\tE: Corrección con Etiquetado morfológico.\n";

if(!$opc || !$arche || !$archs){
  print "$mensaje";
  close STDERR;
  exit;
}

# Si la opción es S ejecuta el corrector CATMC
if($opc eq "S"){
  print STDERR "Corrigiendo el archivo $arche.
 El log es $logErr, opción $opc\n";
```

```
  # Hace un llamado al archivo que contiene las funciones
  # usadas por el corrector CATMC
  require "./bSinEtq.pl";
  # Ejecuta el corrector CATMC
  &cSinEtq;
}
# Si la opción es E ejecuta el corrector CATCG
elsif($opc eq "E"){
  print STDERR "Corrigiendo el archivo $arche.
 El log es $logErr, opción $opc\n";

  # Hace un llamado al archivo que contiene las funciones
  # usadas por el corrector CATMC
  require "./bConEtq.pl";
  # Ejecuta el corrector CATMC
  &cConEtq;
}
else{
  print "La opción $opc no es correcta\n";
  print "$mensaje";
  exit;
}

close STDERR;

sub cSinEtq{
  # FASE I PREPROCESAMIENTO
  # Aquí se lleva a cabo el proceso de formateado del texto
  system("perl ../Herramientas/prepTexto.pl ./$arche");
  $arche =~ s/\.txt$/\.pre/i;
  open(ARCHT, "$arche");
  open(BIGRS, "$archbigramas");
  open(DGEN, "<$tbGen");
  open(ARCHS, ">$archs");

  # Se cargan en memoria el texto de entrada, el modelo
  # de lenguaje y las abreviaturas
  @texto = <ARCHT>;
```

```
  @ngramas = <BIGRS>;
  &creaAbrev;

  # FASE II DETECCIÓN CONTEXTUAL Y MORFOLÓGICA DE ERRORES
  # Crea la lista de palabras a partir del texto de entrada
  @palabras = &tomaPalTxt(@texto);
  @palabras2 = @palabras;

  # Crea el lexicón de búsqueda
  @TribigG = <DGEN>;
  %tribig = &creaLexCve(@TribigG);
  undef @TribigG;

  # Estructura el lexicón de búsqueda del modelo de lenguaje
  &hazBigramas;
  # Detecta los errores contextual y morfológicamente
  &detectaErroresCntx;

  # FASE III GENERACIÓN DE CORRECCIONES POTENCIALES
  &buscaCandid;

  # FASE IV CORRECCIÓN
  &selecCandid;

  # FASE V POSTPROCESAMIENTO
  &sustituye;
  &regresaPalTxt(@mayusTxt);

  close ARCHS;
}

sub cConEtq{
  # FASE I PREPROCESAMIENTO
  # Aqui se lleva a cabo el proceso de formateado del
  # texto previo al etiquetamiento
  system ("perl ../Herramientas/prepTexto.pl $arche");
  $archp = $arche;
  $archetq = $arche;
```

```
$archp =~ s/\.txt$/\.pre/i;
$archetq =~ s/\.txt$/\.etq/i;

# FASE II ETIQUETAMIENTO
# Aqui se lleva a cabo el proceso de etiquetamiento
system ("../Etiquetador/etq/Bin_and_Data/etiquetar.sh
$archp $archetq");

# FASE III EXTRACCIÓN DE PALABRAS
# Se crea el archivo de salida y se carga en memoria el ar-
# chivo etiquetado y el contenido de cada lexicón de catego-
# rías gramaticales. Cada lexicón es de una categoría grama-
# tical distinta
open(ARCHT, "$archetq");
open(ARCHS, ">$archs");
open(BIGRS, "$archbigramas");

open(DVERB, "<$tbVerb");
open(DADJ, "<$tbAdj");
open(DADV, "<$tbAdv");
open(DART, "<$tbArt");
open(DCON, "<$tbConj");
open(DPRE, "<$tbPrep");
open(DPRO, "<$tbPron");
open(DSUS, "<$tbSust");
open(DNUM, "<$tbNum");

# Se obtiene el contenido del archivo de entrada
# (etiquetado) y el modelo de lenguaje
@texto = <ARCHT>;
@ngramas = <BIGRS>;

# Se crean los lexicones de búsqueda de palabras,
# y del modelo de lenguaje
@TribigVerb = <DVERB>;
@TribigAdj = <DADJ>;
@TribigAdv = <DADV>;
@TribigArt = <DART>;
```

```
@TribigCon = <DCON>;
@TribigPre = <DPRE>;
@TribigPro = <DPRO>;
@TribigSus = <DSUS>;
@TribigNum = <DNUM>;

close ARCHT; close DVERB; close DADV;
close DART; close DCON; close DPRE;
close DPRO; close DSUS; close DADJ;

# Crea la lista de palabras a partir del texto de entrada
@palabras = &tomaPalTxtEtq(@texto);
@palabras2 = @palabras;

# Crea los lexicones en donde se realizará la busqueda
# V   Verbo            P   Preposición
# I   Interjección     C   Conjunción
# D   artículo         B   adverBio
# A   Adjetivo         R   pRonombre
# N   sustaNtivo       U   nUmeral
# F   Fecha

&creaLexCveEtq("V", @TribigVerb);
&creaLexCveEtq("A", @TribigAdj);
&creaLexCveEtq("B", @TribigAdv);
&creaLexCveEtq("D", @TribigArt);
&creaLexCveEtq("C", @TribigCon);
&creaLexCveEtq("P", @TribigPre);
&creaLexCveEtq("R", @TribigPro);
&creaLexCveEtq("N", @TribigSus);
&creaLexCveEtq("U",@TribigNum);

# Crea el lexicón de búsqueda del modelo de lenguaje
&hazBigramas;

# FASE IV DETECCIÓN CONTEXTUAL Y MORFOLÓGICA DE ERRORES
&detectaErroresCntxEtq;
```

```
  # FASE V GENERACIÓN DE CORRECCIONES POTENCIALES
  &buscaCandidEtq;

  # FASE VI CORRECCIÓN
  &selecCandid;

  # FASE VII POSTPROCESAMIENTO
  &sustituyeEtq;
  &regresaMAYTxtEtq;

  close ARCHS;
}
```

A continuación se muestra el contenido del archivo bSinEtq.pl, el cual contiene las funciones utilizadas por el corrector CATMC.

```
#!/usr/bin/perl
#
# bSinEtq.pl
#

sub detectaErroresCntx{
  ### Realiza la detección de errores mediante la búsqueda de
  ### los bigramas de las palabras del texto en el lexicón de
  ### bigramas y en el lexicón de palabras
  $p = 1;
  $y = 0;
  $band = 0;
  while($p <= $#palabras){ # 1
    ($inicio, $palabraA) = split(/\s+/, $palabras[$p - 1]);
    ($inicio, $palabra) = split(/\s+/, $palabras[$p]);
    ($inicio, $palabraP) = split(/\s+/, $palabras[$p + 1]);
    if($palabra ne "<I>"){ # 2
      # Buscar si tiene un error ortográfico
      if($diccionario{$palabra} eq ""){
        $errores{$p} = "$palabra";
        $y++;
      }
```

```
else{ # 3
  @BipalA = &buscaBiGramas($palabraA, $palabra);
  if($BipalA[0] eq "NHCdB"){ # 4
    # Si No se encontró el bigrama palabraA,palabra
    if($diccionario{$palabraA} eq ""){
      #PalabraP SI tiene errores ortográficos
      $errores{$p - 1} = "$palabraA";
      $y++;
      $band = 0;
    }
    else{
      #Si se encontró A en el lexicón la
      # palabra {$palabraA}
      $band = 1;
    }
    # Verificando bigramas p,P {$palabra, $palabraP}
    @BipalP = &buscaBiGramas($palabra, $palabraP);
    if($BipalP[0] eq "NHCdB"){
      # No se encontró el bigrama palabraA,palabra
      if($diccionario{$palabraP} eq ""){
        # palabraP SI tiene errores ortográficos
        $errores{$p + 1} = "$palabraP";
        $y++;
      }
      else{
        #verifica que la palabra posterior
        #no sea incorrecta
        if($errores{$p + 1} ne $palabraP){
          $band = 2;
        }
      }
    }
    # $palabra es incorrecta, no se encontró
    # algún bigrama
    if($band == 2){
      $errores{$p} = "$palabra";
      $y++;
    }
```

```
          # no se encontró el bigrama p,P
          if($band == 1){
            # verifica que la palabra anterior
            # no sea incorrecta
            if($errores{$p - 1} ne $palabraA){
              $errores{$p} = "$palabra";
            }
          }
        } # if 4
      } # else 3
    } # if 2
    $p++;
  } # while 1

  # Aquí se crea el arreglo con las palabras
  # incorrectas y sus índices
  @Ierroneas = sort keys(%errores);
  foreach $ie (@Ierroneas){
    $Erroneas[$ie] = $errores{$ie};
  }
  $ind = 0;
  foreach $er (@Erroneas){
    $er =~ s/^[0-9]+$//;
    $er =~ s/^[\(\)\[\]\{\}\"\'\?\¿\!\¡\:\.\,\;]$//;
    if($er ne ""){
      push(@erroneas, $er);
      push(@ierroneas, $ind);
    }
    $ind++;
  }
}

sub buscaCandid{
  #### Realiza la generación y ponderación de palabras
  #### sustitutas para cada palabra errónea
  foreach $pal (@erroneas){
    $tampal = length($pal);
    push(@sustitutas, &creaSustitutas);
```

```
  }
}

# Toma las palabras del texto, separa mayúsculas y minúsculas.
sub tomaPalTxt{
  local(@lineas) = @_;
  $x = 0;
  foreach $lin (@lineas){
    # Inserta en las palabras la marca de inicio de sintagma
    $lin = "<I> $lin";
    # Expande abreviaturas
    &expandeAbrev;
    (@palabs, @resto) = split(/\s+/, $lin);
    $pmay = 0;
    foreach (@palabs){
      if(!/<I>/){
        $tp = length($_);
        $tamsPal{$tp}++;
        if((/^([A-ZÁÉÍÓÚÑ])([A-ZÁÉÍÓÚÑ])([a-záéíóúñü\d]+)$/)){
          $may2 = $2;
          $may2 =~ tr/A-Z/a-z/;
          $may2 =~ tr/Á/á/;
          $may2 =~ tr/É/é/;
          $may2 =~ tr/Í/í/;
          $may2 =~ tr/Ó/ó/;
          $may2 =~ tr/Ú/ú/;
          $may2 =~ tr/Ő/ó/;
          $may2 =~ tr/Ú/ú/;
          $may2 =~ tr/Ü/ü/;
          $may2 =~ tr/Ñ/ñ/;
          $_ = "$1"."$may2"."$3";
        }
        if(/^([A-ZÁÉÍÓÚÑÜ]+)$/){
          $_ =~ tr/A-Z/a-z/;
          $_ =~ tr/Á/á/;
          $_ =~ tr/É/é/;
          $_ =~ tr/Í/í/;
          $_ =~ tr/Ő/ó/;
```

```
            $_ =~ tr/Ú/ú/;
            $_ =~ tr/Ő/ó/;
            $_ =~ tr/Ú/ú/;
            $_ =~ tr/Ü/ü/;
            $_ =~ tr/Ñ/ñ/;
            $mayusTxt[$x] = "TMAY $_ ";
          }
          elsif((/^([A-ZÁÉÍÓÚÑ])([a-záéíóúñü\d]+)$/)){
            $may = $1;
            $letraM = "MAY$1";
            $may =~ tr/A-Z/a-z/;
            $may =~ tr/Á/á/;
            $may =~ tr/É/é/;
            $may =~ tr/Í/í/;
            $may =~ tr/Ő/ő/;
            $may =~ tr/Ú/ú/;
            $may =~ tr/Ő/ó/;
            $may =~ tr/Ú/ú/;
            $may =~ tr/Ü/ü/;
            $may =~ tr/Ñ/ñ/;
            $mayusTxt[$x] = "$letraM $may$2";
          }
          else{
            $mayusTxt[$x] = "NC $_";
          }
        }
        else{
          $mayusTxt[$x] = "NC $_";
        }
        $x++;
      }
    }
    return(@mayusTxt);
}

sub sustituye{
    # Realiza la sustitución de las correcciones
    # en el texto de entrada
```

```
  $i = 0;
  foreach $ip (@ierroneas){
    ($inicio, $medio) = split(/ /, $palabras[$ip]);
    $mayusTxt[$ip] = "$inicio $correctas[$i]";
    $i++;
  }
}

sub regresaPalTxt{
  # Restituye el aspecto original de las palabras del texto
  local (@triadas) = @_;

  foreach (@triadas){
    $palabraf = "";
    ($letraM,$pala) = split(/\s/, $_);
    if(/NC/){
      $letraM =~ s/NC//g;
    }
    if(/TMAY/){
      $letraM =~ s/TMAY//g;
      $pala =~ tr/a-z/A-Z/;
      $pala =~ tr/á/Á/;
      $pala =~ tr/é/É/;
      $pala =~ tr/í/Í/;
      $pala =~ tr/ó/Ó/;
      $pala =~ tr/ú/Ú/;
      $pala =~ tr/ü/Ü/;
      $pala =~ tr/ñ/Ñ/;
    }
    if(/(MAY)([A-Z])\s/){
      $letraM =~ s/MAY//;
      $alap = reverse($pala);
      $maycand = chop($alap);
      $maycand =~ tr/a-z/A-Z/;
      $maycand =~ tr/á/Á/;
      $maycand =~ tr/é/É/;
      $maycand =~ tr/í/Í/;
      $maycand =~ tr/ó/Ó/;
```

```
      $maycand =~ tr/ú/Ú/;
      $maycand =~ tr/ü/Ü/;
      $maycand =~ tr/ñ/Ñ/;
      # en el caso de que la letra inicial
      # de la palabra sustituta cambie
      if($maycand ne $letraM){
        $letraM = $maycand;
      }
      $pala = reverse($alap);
      $pala = $letraM.$pala;
    }
    else{
      $palabraf .= $letraM;
    }
    $palabraf .= $pala;
    $palabraf .= " ";
    $palabraf =~ s/<I>/\n/;
    #$palabraf =~ s/<F>//;
    $textoF .= $palabraf;
    #print "$palabraf";
  }
  $textoF =~ s/^\s+//g;
  $textoF =~ s/ ([\.\,\;\]\)\}\?\!])/$1/g;
  $textoF =~ s/([\[\(\{\¿\¡]) /$1/g;
  print ARCHS "$textoF";
}

sub creaSustitutas{
  # Crea el conjunto de palabras sustitutas para cada palabra
  # incorrecta detectada. El formato final es:
  # palabraIncorrecta|corrección1:trigramas+bigramas&
  # corrección2:trigramas+bigramas&corrección3:trigramas+
  # bigramas& etc.

  # El diccionario de tribigramas está vacío
  # o no se introdujo una palabra
  if(%tribig == 0 || $pal eq ""){
    return 0;
```

```
}
else{
  # Crea los tribigramas de la palabra no encontrada
  $tbgrpal = &hazTriBigr($pal);
  $palsust1 = "$pal\:$tbgrpal|";
  ($trgrpal, $bigrpal) = split(/\+/, $tbgrpal);
  @tgrpal = split(/,/, $trgrpal);
  @bgrpal = split(/,/, $bigrpal);
  # Para el caso de palabras que tienen solo bigramas
  if(@bgrpal <= 2){
    @bgrpal = split(/,/, $trgrpal);
  }
  if(length($pal) == 4){
    @tgrpal = @bgrpal;
    @bgrpal = split(/,/, $trgrpal);
  }
  $longit[0] = $tampal - 1;
  $longit[1] = $tampal;
  $longit[2] = $tampal + 1;

  $palsust2 = "";
  for($p = 0; $p < 3; $p++){
    $llavepal = "$tgrpal[$p]";
    foreach $lp (@longit){
      # Obtiene las candidatas del lexicón de
      # correcciones potenciales
      (@tbgcand) = split(/ /, $tribig{$lp}{$llavepal});
      if($llavepal){
        foreach (@tbgcand){
          # separa los tribigramas en indice y tribigramas
          # $itrib y $ctrib de cada palabra respectivamente
          ($itrib, $ctrib) = split(/:/, $_);
          $ctrib =~ s/(\s)$//;
          ($trgr, $bigr) = split(/\+/, $ctrib);
          (@tgrs) = split(/,/, $trgr);
          (@bgrs) = split(/,/, $bigr);
          #Evita que se repitan las correcciones
          if(!$ctribun{$itrib}){
```

```
                $ctribun{$itrib}++;
                $palsust2 .= "$itrib\:$ctrib&";
              }
            }
          }
        }
      }
      undef %ctribun;
    }

    if("$palsust2" ne ""){
      $palsust2 =~ s/(\&)+$//;
    }
    else{
      $palsust2 = "NHS";
    }
    $palsust = "$palsust1"."$palsust2";
    return $palsust;
}

# Crea los lexicones de búsqueda de palabras
sub creaLexCve{
    local(@DiTrBiGr) = @_;
    foreach (@DiTrBiGr){
      # Elimina el cambio de linea de cada n-grama del archivo
      chop($_);
      ($pal, $resto) = split(/:/, $_);
      ($tam, $pal) = split(/\|/, $pal);
      # Palabras menores a 4 letras
      if(/:(\D{1,1}),{0,1}(\D{0,1}),{0,1}(\D{0,1})$/){
        $indtbg1 = "$1";
        $indtbg2 = "$2";
        $indtbg3 = "$3";
        $diccionario{$pal}++;
        $ngram = $pal.":".$resto;
        $tribig{$tam}{$indtbg1}.="$ngram ";
        $tribig{$tam}{$indtbg2}.="$ngram ";
        $tribig{$tam}{$indtbg3}.="$ngram ";
```

```
    }
    # Palabras de 4 letras
    elsif(/:\D{3,3},\D{3,3}\+(\D{2,2}),(\D{2,2}),(\D{2,2})/){
      $indtbg = "$1";
      $indtbg2 = "$2";
      $indtbg3 = "$3";
      $diccionario{$pal}++;
      $ngram = $pal.":".$resto;
      $tribig{$tam}{$indtbg} .= "$ngram ";
      $tribig{$tam}{$indtbg2} .= "$ngram ";
      $tribig{$tam}{$indtbg3} .= "$ngram ";
    }
    # Palabras mayores a 4 letras
    elsif(/:(\D{3,3}),(\D{3,3}),(\D{3,3}),{0,1}/){
      $indtbg = "$1";
      $indtbg2 = "$2";
      $indtbg3 = "$3";
      $diccionario{$pal}++;
      $ngram = $pal.":".$resto;
      $tribig{$tam}{$indtbg} .= "$ngram ";
      $tribig{$tam}{$indtbg2} .= "$ngram ";
      $tribig{$tam}{$indtbg3} .= "$ngram ";
    }
}
# Añade al lexicón los signos de puntuación
$diccionario{":"}++;
$diccionario{";"}++;
$diccionario{"."}++;
$diccionario{","}++;
$diccionario{"."}++;
$diccionario{"-"}++;
$diccionario{"\""}++;
$diccionario{"\'"}++;
$diccionario{"\?"}++;
$diccionario{"\¿"}++;
$diccionario{"\!"}++;
$diccionario{"\¡"}++;
$diccionario{"<I>"}++;
```

```
  return (%tribig);
}

# Crea un lexicón de abreviaturas
sub creaAbrev{
  $archab = "../corpus/Diccionarios/abrevSig.txt";
  open(ARCHAB, "$archab");
  @abreviats = <ARCHAB>;
  close ARCHAB;
  foreach (@abreviats){
    ($sigAb, $signif) = split(/\|/, $_);
    #sólo para el caso de que el texto esté todo en minúsculas
    $signif =~ s/\s+$//;
    $sAbrev{$sigAb} = "$signif";
  }
}

# Esta función realiza la expansión de abreviaturas
sub expandeAbrev{
  @palas = split(/\s+/, $lin);
  foreach $pala (@palas){
    $pala =~ s/[,;:"'\/{}\(\)\[\]\/\>\<]+//g;
    $sigAb = $sAbrev{$pala};
    if($sigAb ne ""){
      $lin =~ s/$pala/$sigAb/;
    }
  }
}

1;
```

Enseguida se presenta el contenido del archivo bConEtq.pl, el cual contiene las funciones utilizadas por el corrector CATCG.

```
#!/usr/bin/perl
# bConEtq.pl

sub detectaErroresCntxEtq{
```

```
# Realiza la detección de errores mediante la búsqueda de
# los bigramas de las palabras del texto en el lexicón de
# bigramas y en el lexicón de palabras
$p = 1;
$y = 0;
$band = 0;
while($p <= $#palabras){ # 1
  ($inicio, $palabraA) = split(/\s+/, $palabras[$p - 1]);
  ($inicio, $palabra) = split(/\s+/, $palabras[$p]);
  ($inicio, $palabraP) = split(/\s+/, $palabras[$p + 1]);
  if($palabra ne "<I>"){ # 2
    # Buscar si tiene un error ortográfico
    if($diccionario{$catego[$p]}{$palabra} eq ""){
      $errores{$p} = "$palabra";
      $y++;
    }
    else{ # 3
      @BipalA = &buscaBiGramas($palabraA, $palabra);
      if($BipalA[0] eq "NHCdB"){ # 4
        # No se encontró el bigrama {palabraA, palabra}
        if($diccionario{$catego[$p - 1]}{$palabraA} eq ""){
          # palabraP SI tiene errores ortográficos
          $errores{$p - 1} = "$palabraA";
          $y++;
          $band = 0;
        }
        else{
          # Si se encontró A en el dicc. {$palabraA}
          $band = 1;
        }
        # Verificando bigr p,P $palabra, $palabraP
        @BipalP = &buscaBiGramas($palabra, $palabraP);
        if($BipalP[0] eq "NHCdB"){
          # no se encontró el bigrama palabraA,palabra
          if($diccionario{$catego[$p+1]}{$palabraP} eq ""){
            # palabraP SI tiene errores ortográficos
            $errores{$p + 1} = "$palabraP";
            $y++;
```

```
            }
            else{
              # verifica que la palabra posterior
              # no sea incorrecta
              if ($errores{$p+1} ne $palabraP){
                $band = 2;
              }
            }
          }
          # palabra es incorrecta, no se encontró
          # ningún bigrama
          if($band == 2){
            # No se encontraron bigramas para
            # $palabraA|$palabra|$palabraP se toma como error
            $errores{$p} = "$palabra";
            $y++;
          }
          # no se encontró el bigrama p,P
          if($band == 1){
            # verifica que la palabra anterior
            # no sea incorrecta
            if($errores{$p - 1} ne $palabraA){
              # No se encontró el bigrama A,p,
              # {$palabraA,$palabra} $palabra es el error
              $errores{$p} = "$palabra";
            }
          }
        } # 4
      } # 3
    } # 2
    $p++;
} # 1
# Aquí se crea el arreglo con las palabras incorrectas
# y sus índices
@Ierroneas = sort keys(%errores);
foreach $ie (@Ierroneas){
  $Erroneas[$ie] = $errores{$ie};
}
```

```
  $ind = 0;
  foreach $er (@Erroneas){
    $er =~ s/^[0-9]+$//;
    $er =~ s/^[\(\)\[\]\{\}\"\'\?\¿\!\¡\:\.\,\;]$//;
    if($er ne ""){
        push(@erroneas, $er);
        push(@ierroneas, $ind);
    }
    $ind++;
  }
  print STEDERR "Palabras incorrectas:<<<<<@erroneas>>>>>\n";
}

sub buscaCandidEtq{
  # Realiza la generación y ponderación de palabras
  # sustitutas para cada palabra errónea
  $i = 0;
  foreach $pal (@erroneas){
    $tampal = length($pal);
    $Cat = $catego[$ierroneas[$i]];
    push(@sustitutas, &creaSustitutasEtq);
    $i++;
  }
}

# Toma las etiquetas del cada palabra
# y separa las mayúsculas y minúsculas
sub tomaPalTxtEtq{
  local(@lineas) = @_;
  $x = 0;
  foreach $lin (@lineas){
    # Inserta la pseudopalabra al inicio de cada sintagma
    $lin = "<I> $lin";
    (@palabs, @resto) = split(/\s+/, $lin);
    foreach (@palabs){
      $_ =~ s/\/NNP/\/N/i;
      $_ =~ s/\/NN/\/N/i;
      #toma las categorías gramaticales de cada palabra
```

```
    if(!/<I>/){
      if(/([a-zA-ZáéíóúñüÑÁÉÍÓÚÜ0-9]+
              |[\.\,\"\'\?\:\;\!\¿\¡\(\)\[\]\{\}])
              \/([a-zA-ZáéíóúñÑÁÉÍÓÚ0-9\#\~]{1,1})/){
        ($palabras[$x], $categoComp[$x]) = split (/\//, $_);
        $catego[$x] = $2;
      }
      $_ = $palabras[$x];
      if (/^([A-ZÁÉÍÓÚÑÜ]+)$/){
        $_ =~ tr/A-Z/a-z/;
        $_ =~ tr/Á/á/;
        $_ =~ tr/É/é/;
        $_ =~ tr/Í/í/;
        $_ =~ tr/Ó/ó/;
        $_ =~ tr/Ú/ú/;
        $_ =~ tr/Ó/ó/;
        $_ =~ tr/Ú/ú/;
        $_ =~ tr/Ü/ü/;
        $_ =~ tr/Ñ/ñ/;
        $mayusTxt[$x] = "TMAY $_ ";
      }
      elsif(/^([A-ZÁÉÍÓÚÑ])([a-záéíóúñü\d]+)/){
        $may = $1;
        $pal = $2;
        $May = "MAY$1";
        $may =~ tr/A-Z/a-z/;
        $palabras[$x] = "$may$2";
        $mayusTxt[$x] = "$May $may$2";
      }
      else{
        $mayusTxt[$x] = "NC $_";
      }
    }
    else{
      $mayusTxt[$x] = "NC $_";
    }
    $x++;
  }
```

```
  }
  return(@mayusTxt);
}

sub sustituyeEtq{
  # Realiza la sustitución de las correcciones
  # en el texto de entrada
  $i = 0;
  foreach $ip (@ierroneas){
    ($inicio, $pal) = split(/ /, $mayusTxt[$ip]);
    $mayusTxt[$ip] = "$inicio $correctas[$i]";
    $i++;
  }
}

# Regresa el aspecto original a las palabras del texto
sub regresaMAYTxtEtq{
  foreach (@mayusTxt){
    $palabraf = "";
    ($letraM, $pala) = split(/\s/, $_);
    if(/NC/){
      $letraM =~ s/NC//g;
    }
    if(/TMAY/){
      $letraM =~ s/TMAY//g;
      $pala =~ tr/a-z/A-Z/;
      $pala =~ tr/á/Á/;
      $pala =~ tr/é/É/;
      $pala =~ tr/í/Í/;
      $pala =~ tr/ó/Ó/;
      $pala =~ tr/ú/Ú/;
      $pala =~ tr/ü/Ü/;
      $pala =~ tr/ñ/Ñ/;
    }
    if(/(MAY)([A-Z])\s/){
      $letraM =~ s/MAY//;
      $alap = reverse($pala);
      $maycand = chop($alap);
```

```
      $maycand =~ tr/a-z/A-Z/;
      $maycand =~ tr/á/Á/;
      $maycand =~ tr/é/É/;
      $maycand =~ tr/í/Í/;
      $maycand =~ tr/ó/Ó/;
      $maycand =~ tr/ú/Ú/;
      $maycand =~ tr/ü/Ü/;
      $maycand =~ tr/ñ/Ñ/;
      # en el caso de que la letra inicial
      # de la palabra sustituta cambie
      if($maycand ne $letraM){
        $letraM = $maycand;
      }
      $pala = reverse($alap);
      $pala = $letraM.$pala;
    }
    else{
      $palabraf .= $letraM;
    }
    $palabraf .= $pala;
    $palabraf .= " ";
    $palabraf =~ s/<I>/\n/g;
    $textoF .= $palabraf;
  }
  $textoF =~ s/^\s+//g;
  $textoF =~ s/ ([\.\,\;\]\)\}\?\!])/$1/g;
  $textoF =~ s/([\[\(\{\¿\¡]) /$1/g;
  print ARCHS "$textoF";
}

# Crea el conjunto de correcciones potenciales para
# cada palabra incorrecta
sub creaSustitutasEtq{
  # El diccionario de tribigramas está vacío o no se
  # introdujo una palabra ($pal)
  if(%tribig == 0 || $pal eq ""){
    return 0;
  }
```

```
else{
  # crea los tribigramas de la palabra no encontrada
  $tbgrpal = &hazTriBigr($pal);
  $palsust1 = "$pal\:$tbgrpal|";
  ($trgrpal, $bigrpal) = split(/\+/, $tbgrpal);
  @tgrpal = split(/,/, $trgrpal);
  @bgrpal = split(/,/, $bigrpal);
  # para el caso de palabras que tienen solo bigramas
  if(@bgrpal <= 2){
    @bgrpal = split (/,/, $trgrpal);
  }
  if(length($pal) == 4){
    @tgrpal = @bgrpal;
    @bgrpal = split(/,/, $trgrpal);
  }
  $longit[0] = $tampal - 1;
  $longit[1] = $tampal;
  $longit[2] = $tampal + 1;
  $palsust2 = "";

  for($p = 0; $p < 3; $p++){
    $llavepal = "$tgrpal[$p]";
    foreach $lp (@longit){
      (@tbgcand) = split(/ /,$tribig{$lp}{$Cat}{$llavepal});
      if($llavepal){
        foreach (@tbgcand){
          # Separa los bigramas en indice y tribigramas
          # ($itrib y $ctrib) de cada palabra candidata
          ($itrib, $ctrib) = split(/:/, $_);
          $ctrib =~ s/(\s)$//;
          ($trgr, $bigr) = split(/\+/, $ctrib);
          (@tgrs) = split(/,/, $trgr);
          (@bgrs) = split(/,/, $bigr);
          if(!$ctribun{$itrib}){
            $ctribun{$itrib}++;
            $palsust2 .= "$itrib:$ctrib&";
          }
        }
```

```
        }
      }
    }
    undef %ctribun;
  }
  if("$palsust2" ne ""){
    $palsust2 =~ s/(\&)+$//;
  }
  else{
    # No se encontraron sustitutas para la palabra
    $palsust2 = "NHS";
  }
  $palsust = "$palsust1"."$palsust2";
  return $palsust;
}

# Crea los lexicones de búsqueda y de correcciones potenciales
# de cada categoría gramatical. Recibe la categoría gramatical
# como parámetro
sub creaLexCveEtq{
  local($cate, @DiTrBiGr) = @_;
  if($cate){
  foreach (@DiTrBiGr){
    # elimina el cambio de linea de cada n-grama del archivo
    chop($_);
    ($pal, $resto) = split(/:/, $_);
    ($tam, $pal) = split (/\|/, $pal);
    # Palabras menores a 4 letras
    if(/:(\D{1,1}),{0,1}(\D{0,1}),{0,1}(\D{0,1})$/){
      $indtbg1 = "$1";
      $indtbg2 = "$2";
      $indtbg3 = "$3";
      $diccionario{$cate}{$pal}++;
      $ngram = $pal.":".$resto;
      $tribig{$tam}{$cate}{$indtbg1} .= "$ngram ";
      $tribig{$tam}{$cate}{$indtbg2} .= "$ngram ";
      $tribig{$tam}{$cate}{$indtbg3} .= "$ngram ";
    }
```

```
    # Palabras de 4 letras
    elsif(/:\D{3,3},\D{3,3}\+(\D{2,2}),(\D{2,2}),(\D{2,2})/){
      $indtbg1 = "$1";
      $indtbg2 = "$2";
      $indtbg3 = "$3";
      $diccionario{$cate}{$pal}++;
      $ngram = $pal.":".$resto;
      $tribig{$tam}{$cate}{$indtbg1} .= "$ngram ";
      $tribig{$tam}{$cate}{$indtbg2} .= "$ngram ";
      $tribig{$tam}{$cate}{$indtbg3} .= "$ngram ";
    }
    # Palabras mayores a 4 letras
    elsif(/:(\D{3,3}),(\D{3,3}),(\D{3,3}),{0,1}/){
      $indtbg1 = "$1";
      $indtbg2 = "$2";
      $indtbg3 = "$3";
      $diccionario{$cate}{$pal}++;
      $ngram = $pal.":".$resto;
      $tribig{$tam}{$cate}{$indtbg1} .= "$ngram ";
      $tribig{$tam}{$cate}{$indtbg2} .= "$ngram ";
      $tribig{$tam}{$cate}{$indtbg3} .= "$ngram ";
    }
    # Crea el lexicón de búsqueda de la categoría Miscelánea
    $diccionario{"M"}{":"}++;
    $diccionario{"M"}{";"}++;
    $diccionario{"M"}{"."}++;
    $diccionario{"M"}{","}++;
    $diccionario{"M"}{"."}++;
    $diccionario{"M"}{"-"}++;
    $diccionario{"M"}{"\""}++;
    $diccionario{"M"}{"\'"}++;
    $diccionario{"M"}{"\?"}++;
    $diccionario{"M"}{"\¿"}++;
    $diccionario{"M"}{"\!"}++;
    $diccionario{"M"}{"\¡"}++;
    $diccionario{"M"}{"<I>"}++;
    return(%dicci, %tribig);
}
```

```
}

1;
```

Finalmente, se muestra el código de las funciones de análisis morfológico utilizadas en los dos correctores anteriormente presentados. Estas funciones están en el archivo bMorfolog.pl.

```
#!/usr/bin/perl
#
# bMorfolog.pl
# Biblioteca de funciones de comparación morfológica

sub selecCandid{
  # Realiza la selección de la mejor candidata
  # para cada palabra errónea
  $ie = 0;
  foreach $sustit (@sustitutas){
    $ip = $ierroneas[$ie];
    ($marca, $palA) = split(/ /, $palabras[$ip-1]);
    ($marca, $palP) = split(/ /, $palabras[$ip+1]);
    $corr = &selecCorrecta(&compMorf($sustit));
    push(@correctas, $corr);
    $ip++;
    $ie++;
  }
}

# Crea el lexicón de búsqueda del modelo de lenguaje
sub hazBigramas{
  foreach (@ngramas){
    /(\D+) (\D+)\|(\d+)/;
    $bigramas{$1} .= "$2\-$3 ";
  }
}

# Realiza la búsqueda del ngrama formado con sus parámetros
# $p1 y $p2 dentro lexicón  de búsqueda del modelo de lenguaje
```

```
sub buscaBiGramas{
  local ($p1, $p2) = @_;
  @biGramas = split (/\s+/, $bigramas{$p1});
  $mayor = 0;
  if(@biGramas == 0){
    # No hay bigramas para [$p1, $p2]
    return("NHCdB", 0);
  }
  foreach $bi (@biGramas){
    ($palc, $frec) = split(/\-/, $bi);
    # Si encuentra un bigrama que coincida
    # con la palabra $p2 termina la búsqueda
    if($palc eq $p2)
    {
      return($palc, $frec);
    }
  }
  return ("NHCdB", 0);
}

# Aquí se lleva a cabo la comparación morfológica de las
# correcciones potenciales y las palabras incorrectas
# El formato de salida de las corr. potenc. ponderadas es:
#
# palabraIncorrecta|N=sustituta&N=sustituta2 etc.
#
#Donde N es la ponderación morfológica de la
#corrección potencial
sub compMorf{
  local($linea) = @_;
  # separa la palabra(erronea)-tribigramas
  # y la(s) sustituta(s)-tribigramas
  ($pala, $trig) = split(/\|/, $linea);

  # Si no hay correcciones potenciales para
  # la palabra en turno, termina la ejecución
  if("$trig" eq "NHS"){
    return $linea;
```

```
}

# separa cada palabra-tribigramas de la sustituta
@tribgrs = split(/&/, $trig);

# separa la palabra erronea de sus tribigramas
($palerr, $ptbg) = split(/:/, $pala);

# separa los trigramas y bigramas de la palabra erronea
($paltrgr, $palbigr) = split(/\+/, $ptbg);
# separa cada trigrama
@paltgr = split(/\,/, $paltrgr);
# y bigrama de la palabra erronea
@palbgr = split(/\,/, $palbigr);

$numco = "$palerr|";
# Para cada palabra sustituta
foreach (@tribgrs){
  ($tribac, $tbgrac) = split(/:/, $_);
  if($tribac eq $palerr){
    $numco .= "0=$tribac&";
  }
  else{
    $coinciden = 0;
    # separa trigramas y bigramas
    # de la palabra sustituta actual
    ($trgr, $bigr) = split(/\+/, $tbgrac);

    # Aqui se verifica cuál candidata
    # tiene mayor probabilidad de
    # aparición en el modelo de lenguaje,
    # con las palabras anterior y
    # posterior a la palabra incorrecta
    @BigpalA=&buscaBiGramas($palA,$tribac);
    @BigpalP=&buscaBiGramas($tribac,$palP);
    if($BigpalA[0] ne "NHCdB"){
      $coinciden += 1;
    }
```

```
    if($BigpalP[0] ne "NHCdB"){
      $coinciden += 1;
    }

    # Para cada TRIGRAMA de la
    # palabra erronea
    foreach $ptgr (@paltgr){
      # y para cada trigrama de la
      # palabra sustituta
      $coi = /($ptgr)+/;
      if($coi){
        $coinciden++;
      }
    }
    # Para soportar comparaciones de
    # subcadenas adecuadamente
    # utilizando trigramas (en palabras
    # mayores a 3 letras)
    $max = length($tribac);
    $tmp = length($palerr);
    # Ppara cada BIGRAMA de la palabra erronea
    foreach $pbgr (@palbgr){
      $coi = /($pbgr)+/;
      if($coi){
        $coinciden++;
      }
    }
    if($tmp > 3){
      if($max < $tmp){
        $max = $tmp;
      }
      $coinciden = $coinciden / ($max);
    }

    undef(@tgrs);
    undef(@bgrs);
    $numco .= "$coinciden=$tribac&";
  }
```

```
  }
  undef(@paltgr);
  undef(@palbgr);
  $numco =~ s/(\&)+$//;
  return $numco;
}

# Esta función compara la frecuencia de dos bigramas para de-
# terminar cuál es la corrección potencial más adecuada
sub cmpBigramasAP{
  # Busca los bigramas Anterior y Posterior para las
  # palabras candidatas empatadas
  @respalA = &buscaBiGramas($palA, $pal);
  @respalP = &buscaBiGramas($pal, $palP);
  @rescandidA = &buscaBiGramas($palA, $candid);
  @rescandidP = &buscaBiGramas($candid, $palP);
  # Quita una unidad al peso de la palabra menos frecuente en
  # el caso de que encuentre ambos bigramas, 0.5 por cada uno
  # si las dos frecuencias son iguales, se comparan morfológi-
  # camente porque los pesos no varían
  # Probando bigrama anterior
  if($respalA[0] ne "NHCdB"){
    if($rescandidA[0] ne "NHCdB"){
      if($respalA[1] > $rescandidA[1]){
        $pesoCand -= 0.5;
      }
      if($respalA[1] < $rescandidA[1]){
        $pesoPal -= 0.5;
      }
    }
    else{
      $pesoCand -= 0.5;
    }
  }
  else{
    if($rescandidA[0] ne "NHCdB"){
      $pesoPal -= 0.5;
    }
```

```
  }
  # Probando bigrama posterior
  if($respalP[0] ne "NHCdB"){
    if($rescandidP[0] ne "NHCdB"){
      if($respalP[1] > $rescandidP[1]){
        $pesoCand -= 0.5;
      }
      if($respalP[1] < $rescandidP[1]){
        $pesoPal -= 0.5;
      }
    }
    else{
      $pesoCand -= 0.5;
    }
  }
  else{
    if($rescandidP[0] ne "NHCdB"){
      $pesoPal -= 0.5;
    }
  }
}

# Esta función compara los esqueletos de dos correcciones
# potenciales empatadas con respecto a la palabra incorrecta
sub cmpEsqueletos{
  $c_pa_pl = 0;
  $c_pa_ca = 0;
  @esq_pa = &hazEsqueleto($pala);
  @esq_pl = &hazEsqueleto($pal);
  @esq_ca = &hazEsqueleto($candid);
  # se cuentan las coincidencias del esqueleto
  foreach $l_pl (@esq_pl){
    foreach $l_pa (@esq_pa){
      if("$l_pa" eq "$l_pl"){
        $c_pa_pl++;
      }
    }
  }
```

```
foreach $l_ca (@esq_ca){
  foreach $l_pa (@esq_pa){
    if("$l_pa" eq "$l_ca"){
      $c_pa_ca++;
    }
  }
}
# Si es mas parecida la candidata ($candid) que la palabra
# actual ($pal), se queda $candid como la  mejor candidata,
# de lo contrario queda $pal.
if($c_pa_pl < $c_pa_ca){
  $candid = $pal;
}
elsif($c_pa_pl == $c_pa_ca){
  # Comparación de tamaño de palabra
  $t_pl = length($pal);
  $t_c = length($candid);

  $d_pa_pl = $t_pa - $t_pl;
  $d_pa_c = $t_pa - $t_c;

  if($d_pa_pl < 0){$d_pa_pl *= -1;}
  if($d_pa_c < 0){$d_pa_c *= -1;}

  if($d_pa_pl < $d_pa_c){
    $candid = $pal;
  }
  if($d_pa_pl == $d_pa_c){
    # Comparación de tamaño de esqueleto
    $te_pa = @esq_pa;
    $te_pl = @esq_pl;
    $te_c = @esq_ca;

    $de_pa_pl = $te_pa - $te_pl;
    $de_pa_c = $te_pa - $te_c;
    if($de_pa_pl < 0){$de_pa_pl *= -1;}
    if($de_pa_c < 0){$de_pa_c *= -1;}
    if($de_pa_pl < $de_pa_c){
```

```
        $candid = $pal;
      }
    }
  }
}

# Realiza la comparación entre las mínimas distancias de edi-
# ción entre dos correcciones potenciales y la palabra
# incorrecta
sub cmpMDE{
  $mdepa_pl = &mde2($pala, $pal);
  $mdepa_c = &mde2($pala, $candid);
  if($mdepa_pl < $mdepa_c){
    $candid = $pal;
  }
}

# Selecciona la corrección potencial final
# para cada palabra incorrecta
sub selecCorrecta{
  local($linea) = @_;
  # $pala representa a la palabra analizada
  # $pal representa a la palabra candidata actual
  # $candid es la palabra candidata anterior de mas parecido
  ($pala, $cand) = split(/\|/, $linea);
  if("$cand" eq "NHS"){
    ($pal, $basura) = split(/:/, $pala);
    return $pal;
  }
  $t_pa = length($pala);
  @candidatas = split(/\&/, $cand);
  $pesoCand = 0;
  $candid = "";
  @candidatas = sort(@candidatas);
  @candidatas = reverse(@candidatas);
  foreach (@candidatas){
    ($pesoPal, $pal) = split(/\=/, $_);
    if($pesoPal > $pesoCand){
```

```
      $pesoCand = $pesoPal;
      $candid = $pal;
    }
    else{
      if($pesoPal == 0 && @candidatas == 1){
        $candid = $pal;
      }
      elsif($pesoPal == $pesoCand){
        &cmpBigramasAP;
        if($pesoPal > $pesoCand){
          $pesoCand = $pesoPal;
          $candid = $pal;
        }
        elsif($pesoPal == $pesoCand){
          # se intenta desambiguar con la MDE entre la palabra
          # incorrecta, la candidata actual y la candidata
          # anterior
          &cmpMDE;
          if($mdepa_pl == $mdepa_c){
            &cmpEsqueletos;
          }
        }
      }
    }
  }
  return $candid;
}

# Crea los trigramas y bigramas de una palabra
sub hazTriBigr{
  local($pa) = @_;
  (@pala) = split(//, $pa);
  $trig = "";

  # Para las palabras con 3 o menos letras,
  # se regresa la misma palabra
  if(@pala <= 3){
    $trig .= "$pala[0],"."$pala[1],"."$pala[2]";
```

```
    $trig =~ s/(\,)+$//;
  }
  else{
    # crea trigramas de todas las palabras mayores a 3 letras
    for($le = 0; $le <= @pala - 3; $le++){
      $trip = $pala[$le].$pala[$le+1].$pala[$le+2];
      $trig .= $trip . ",";
    }
    $trig =~ s/(\,)+$//;
    $trig .= "+";
    # crea bigramas de todas las palabras
    for($le = 0; $le <= @pala - 2; $le++){
      $trip = $pala[$le] . $pala[$le + 1];
      $trig .=  $trip . ",";
    }
  }
  $trig =~ s/(\,)+$//;
  return $trig;
}

# Crea el esqueleto de un palabra
sub hazEsqueleto{
  local($palabra) = @_;

  @cars = split(//, $palabra);

  $esq = "";
  foreach (@cars){
    if(!/[aeiouAEIOUÁÉÍÓÚáéíóú0-9]/){
      $esqueleto{$_}++;
      if($esqueleto{$_} == 1){
        $esq .= "$_";
      }
    }
  }
  undef %esqueleto;
  return split(//, $esq);
}
```

```
# Calcula la mínima distancia de edición entre dos palabras
# con el costo de sustitución con 2
sub mde2{
  # borrado e inserción tiene un costo de 1
  # sustitución si cadena1[x] = cadena2[y] = 0,
  # si no el costo es 2
  local($pala1, $pala2) = @_;

  $tp1 = length($pala1);
  $tp2 = length($pala2);

  @pal1 = split(//, $pala1);
  @pal2 = split(//, $pala2);

  $c[0][0] = 0;
  for($y = 1; $y <= $tp2; $y++){
    $c[0][$y] = $y;
    for($x = 1; $x <= $tp1; $x++){
      $c[$x][0] = $x;
      if($pal1[$x - 1] eq $pal2[$y - 1]){$sus = 0;}
      else{$sus = 2;}
      $minimo = &mini($c[$x-1][$y]+1,
                      $c[$x][$y-1]+1, $c[$x-1][$y-1]+$sus);
      $c[$x][$y] = $minimo;
    }
  }
  return $c[$tp1][$tp2];
}

# Devuelve el menor de tres números $x, $y, $z
sub mini{
  local($x, $y, $z) = @_;
  if($x < $y){
    if($x < $z){return $x;}
    elsif($y < $z){return $y;}
    return $z;
  }
```

```
  elsif ($y < $z){return $y;}
  return $z;
}

1;
```

Apéndice B

Etiquetario de Categorías Gramaticales

El etiquetario se compone de 12 clases: Verbo, Miscelánea, Interjección, Artículo, Adjetivo, Sustantivo, Preposición, Conjunción, Adverbio, Pronombre, Numeral y Fecha. Cada etiqueta tiene una longitud de 7 caracteres, y cada carácter tiene un significado según su posición. En las siguientes tablas se muestran las clases y sus componenentes.

Posición	Atributo	Valor	Código
1	Clase	VERBO	V
2-3	Subclase	Personal	Pe
		Impersonal	Im
		Haber	Ha
		Ser	Se
		Estar	Es
		No especificado	..
4	Tiempo	Presente	P
		Perfecto	R
		Imperfecto	M
		Futuro	F
		Condicional	C
		Infinitivo	I
		Gerundio	G
		Participio	T
		No especificado	.
		No existe	#
5	Modo/Grado	Indicativo	D
		Imperativo	O
		Subjuntivo	S
		No especificado	.
		No existe	#
6	Persona/Número	Primera singular	0
		Segunda singular	1
		Tercera singular	2
		Primera plural	3
		Segunda plural	4
		tercera plural	5
		seg/terc plural	6
		No existe	#
7	Orden/Género	No género	N
		Enclítico	E

Tabla B.1: Categoría gramatical Verbo.

Posición	Atributo	Valor	Código
1	Clase	MISCELÁNEA	M
2-3	Subclase	Expresión extranjera	00
		Identificador	01
		Abreviatura	02
		Elemento no analizable	03
		Punto	04
		Coma	05
		InterrogAbre	06
		InterrogCierra	07
		AdmiraAbre	08
		AdmiraCierra	09
		Punto_y_coma	10
		Dos_puntos	11
		Guión	12
		ComillaAbre	13
		comillaCierra	14
		apóstrofeAbre	15
		apóstrofeCierra	16
		parént/corchetAbre	17
		parént/corchetCierra	18
		Sporcentaje	19
		Spesos	20
		ampersand	21
		arroba	22
		Snúmero	23
		asterisco	24
		Smás	25
		diagonal/	26
		Sigual	27
		letra	28
		no especificado	29
4	Tiempo	No existe	#
5	Modo/grado	No existe	#
6	Persona/número	No existe	#
7	Orden/género	No existe	#

Tabla B.2: Categoría gramatical Miscelánea.

Posición	Atributo	Valor	Código
1	Clase	PREPOSICIÓN	P
2-3	Subclase	Simple	00
		Con artículo	03
		No especificado	..
4	Tiempo	No existe	#
5	Modo/Grado	No existe	#
6	Persona/Número	No existe	#
7	Orden/Género	No género	N

Tabla B.3: Categoría gramatical Preposición.

Posición	Atributo	Valor	Código
1	Clase	INTERJECCIÓN	I
2-3	Subclase	No especificado	..
4	Tiempo	No existe	#
5	Modo/Grado	No existe	#
6	Persona/Número	Invariable	L
		No especificado	.
7	Orden/Género	No género	N

Tabla B.4: Categoría gramatical Interjección.

Posición	Atributo	Valor	Código
1	Clase	CONJUNCIÓN	C
2-3	Subclase	Copulativa/disyuntiva	02
		Distributiva	04
		Que	06
		Final	07
		Temporal	08
		Causal	09
		Consecutiva	10
		Condicional/modal	11
		Concesiva	13
		Adversativa	19
		Ilativa	20
		No especificado	..
4	Tiempo	No existe	#
5	Modo/Grado	No existe	#
6	Persona/Número	Singular	S
		Plural	P
		Invariable	L
7	Orden/Género	No género	N

Tabla B.5: Categoría gramatical Conjunción.

Posición	Atributo	Valor	Código
1	Clase	ARTÍCULO	D
2-3	Subclase	Definido	00
		Indefinido	01
		Relativo	02
		Todo/a/os/as	03
		Demostrativo	08
		No especificado	..
4	Tiempo	No existe	=
5	Modo/Grado	No existe	=
6	Persona/Número	Singular	S
		Plural	P
		Invariable	L
7	Orden/Género	Masculino	M
		Femenino	F
		Neutro	N
		No especificado	.

Tabla B.6: Categoría gramatical Artículo.

Posición	Atributo	Valor	Código
1	Clase	ADVERBIO	B
2-3	Subclase	Lugar	00
		Tiempo	01
		Modo	03
		Relativo	06
		Interrogativo	07
		Cantidad	08
		Negación/Afirmación	09
		Cualitativo	21
		No especificado	..
4	Tiempo	No especificado	.
		Comparativo/superlativo	0
5	Modo/Grado	No especificado	.
		Comparativo	2
		Superlativo	4
		Superlativo_relativo	5
6	Persona/Número	Singular	S
		Plural	P
		Invariable	L
7	Orden/Género	No género	N

Tabla B.7: Categoría gramatical Adverbio.

Posición	Atributo	Valor	Código
1	Clase	ADJETIVO	A
2-3	Subclase	Posesivo	06
		Indefinido	07
		Demostrativo	08
		Interrogativo	09
		Calificativo	05
		Distributivo	04
		No especificado	..
4	Tiempo	No existe	#
5	Modo/Grado	No especificado	.
		No existe	#
		Comparativo	2
		Superlativo	3
		Positivo	1
6	Persona/Número	Singular	S
		Plural	P
		Invariable	L
7	Orden/Género	Masculino	M
		Femenino	F
		Neutro	N

Tabla B.8: Categoría gramatical Adjetivo.

Posición	Atributo	Valor	Código
1	Clase	PRONOMBRE	R
2-3	Subclase	Personal sujeto	00
		Personal objeto tónico	01
		Personal átono	02
		Posesivo	05
		Demostrativo	11
		Indefinido	14
		Numeral cardinal	17
		Numeral ordinal	19
		Relativo	20
		Interrogativo	22
		No especificado	..
4	Tiempo	No existe	#
5	Modo/Grado	No existe	#
6	Persona/Número	Singular	S
		Plural	P
		Invariable	L
		Primera singular	I
		Segunda singular	U
		Tercera singular	H
		Primera plural	W
		Segunda plural	Y
		Tercera plural	T
		Seg_Sing_Cortesía	C
7	Orden/Género	Masculino	M
		Femenino	F
		Neutro	N

Tabla B.9: Categoría gramatical Pronombre

Posición	Atributo	Valor	Código
1	Clase	SUSTANTIVO	N
2-3	Subclase	Común	00
		Propio	01
		Común/Propio	02
		No especificado	..
4	Tiempo	No existe	#
5	Modo/Grado	Población	1
		Apellido	2
		Accidente_geográfico	3
		Siglas	4
		De_pila	5
		No especificado	.
6	Persona/Número	Singular	S
		Plural	P
		Invariable	L
7	Orden/Género	Masculino	m
		Femenino	f
		Neutro	n
		Pendiente	p

Tabla B.10: Categoría gramatical Sustantivo.

Posición	Atributo	Valor	Código
1	Clase	NUMERAL	U
2-3	Subclase	Nombre	00
		Pronombre	01
		Adverbio	02
		Fraccionario	03
		Ordinal	04
		Cardinal	05
		Romano	06
		Hora	07
		Número/letras	08
		Operación_aritmética	09
		NúmLet/signo_aritm	10
4	Tiempo	No existe	#
5	Modo/Grado	No existe	#
6	Persona/Número	Singular	S
		Plural	P
		Invariable	L
7	Orden/Género	Masculino	M
		Femenino	F

Tabla B.11: Categoría gramatical Numeral.

Posición	Atributo	Valor	Código
1	Clase	FECHA	F
2-3	Subclase	20_de_febrero_de_2004	01
		20_de_febrero_del_2004	02
		20/02/04	03
		20-02-04	04
		20.02.04	05
		20/02/2004	06
		20-02-2004	07
		20.02.2004	08
		Febrero_de_2004	09
		Febrero_del_2004	10
		Feb	11
		Febrero	12
		2004	13
		20/II/04	14
		20-II-04	15
		20.II.04	16
		20/II/2004	17
		20-II-2004	18
		20.II.2004	19
4	Tiempo	No existe	#
5	Modo/Grado	No existe	#
6	Persona/Número	No existe	#
7	Orden/Género	No existe	#

Tabla B.12: Categoría gramatical Fecha.

Apéndice C

Matriz de Confusión

En la siguiente tabla se muestra la matriz de confusión utilizada para insertar errores al conjunto de prueba.

Carácter	Caracteres sustitutos
,	.
.	, <B>
0	o O
1	í i I l f
6	á é ó
5	S
a	e f g i m o s w
b	b f h o
c	d e g o t x é
d	c s k ó
e	a c g s o
f	a b i k t y l
g	a c e i m p q r s
h	b i n
i	a f g h j l n p t u I l í
j	i l
k	d f x
l	l i j o r t I í
m	a g n o r s w rn ri
n	h i m ñ p q r u ri
ñ	n
o	a b c e l m p q 0
p	g i n o
q	g n o
r	g l m n v
s	a d e g m t
t	c f i l s
u	i n v x
v	r u y
w	a m
x	c k u
y	f v
z	z

Tabla C.1: Matriz de confusión utilizada para crear el conjunto de prueba. El símbolo <B> representa el espacio en blanco.

Carácter	Caracteres sustitutos
I	l i 1
M	H N
O	0
U	ü u
~	ñ
á	a ó 6
é	e c 6
í	i 1 l
ó	o d á 6
ú	u
ü	ii u U
ii	ü
co	cr
rn	m rr ri
ri	m n rn rr
rr	m ir rn

Tabla C.2: Matriz de confusión utilizada para crear el conjunto de prueba (continuación).

Bibliografía

[1] Allebach, J. P. (2000). Image scanning, sampling, and interpolation. In A. Bovik (Ed.), *Handbook of image and video processing* (pp. 629–643). San Diego, California: Academic Press.

[2] Alvarez Cabán, J. M. (2000). Alternatives for access and use of Spanish language assistive technology equipment by individuals with visual disabilities. In California State University Northridge (Comp.), *Proceedings of the Technology and Persons with Disabilities Conference, 2000.* Los Ángeles: California State University Northridge. Retrieved from http://www.csun.edu/cod/conf/2000/proceedings/0187Alvarez_Caban.htm

[3] Ávila, J. L., Fuentes, C. y Tuirán, R. (2001). *Índices de marginación, 2000.* México: Consejo Nacional de Población.

[4] Bao-Qing Li, & Baoxin Li. (1999). Building pattern classifiers using convolutional neural networks. In *International Joint Conference on Neural Networks, 1999* (Vol. 5, pp. 3081–3085). Los Alamitos, California: IEEE Press.

[5] Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: Fundamentals, computing, design, and application. *Journal of Microbiological Methods, 43*(1), 3–31.

[6] Beitzel S. M., Jensen E. C. & Grossman D. A. (2002, August) Retrieving OCR Text: A Survey of Current Approaches. Invited to the *Workshop on OCR Information Retrieval at SIGIR-2002.* Information Retrieval Laboratory, Department of Computing Science, Illinois Institute of Technology.

[7] Belaïd, A. (1998). OCR: Print. In R. Cole, J. Mariani, H. Uszkoreit, G. Battista Varile, A. Zaenen, & A. Zampolli (Eds.), *Survey of the state of the art in human language technology* (pp. 81–85). Cambridge, Massachusetts: Cambridge University Press and Giardini.

[8] Beristáin, H. (1984). *Gramática estructural de la lengua española.* México: UNAM.

[9] Bernal, J. D. (1967). *The social function of science.* Cambridge, Massachusetts: The MIT Press.

[10] Brill E. (1995). Unsupervised learning of disambiguation rules for part of speech tagging. In D. Yarowsky & K. Church (Eds.), *Proceedings of the third ACL Workshop on Very Large Corpora (WVLC-95).* (pp. 1–13). Somertes, New Jersey: Asociation for Computational Linguistics.

[11] Brokshear J. G. (1995). *Introducción a las ciencias de la computación.* (Trad. R. Escalona) (4a. Ed.). México: Addison-Wesley Iberoamericana.

[12] Cabré M. T. (s. f.). *Propuesta de marcaje gramatical del corpus en castellano.* Obtenido en octubre 10, 2003, Instituto Universitario de Lingüística Aplicada, Universidad Pompeu Fabra, Barcelona, página del Proyecto Corpus textual especializado plurilingüe:
http://www.iula.upf.es/corpus/
http://www.iula.upf.es/corpus/etqfrmes.htm

[13] Cabrerizo, E. (2001, Octubre). El braille, mucho más que un sistema de lectura para los ciegos. *UTLAI Punto Doc.* Obtenido de http://www.nodo50.org/utlai/lucer11.htm

[14] Cairó O. y Guardati S. B. (1993). *Estructuras de datos.* México: McGraw-Hill.

[15] Costa Romero de Tejada, M. (2001). *Sistemas de lectura digital y libros electrónicos.* Barcelona: Fundació Privada Catalana per a l'Orientació i Suport Tecnològic dels Cecs Manuel A. Caragol. Obtenido de http://www.funcaragol.org/html/equildsl.htm

[16] de Oliveira Jr., J. J., Veloso, L. R., & de Carvalho, J. M. (2000). Interpolation/decimation scheme applied to size normalization of character

images. In *Proceedings of the 15th International Conference on Pattern Recognition, 2000* (Vol. 2, pp. 577–580). Los Alamitos, California: IEEE Press.

[17] Desarrollan prototipo de impresora de código braille: Este año podría salir al mercado; la producción, totalmente mexicana. (2003, Enero). *Gaceta UNAM*, p. 9.

[18] Dreyfus, H. L. (1992). *What computers still can't do: A critique of artificial reason* (Rev. ed). Cambridge, Massachusetts: The MIT Press.

[19] Ducrot, O. & Todorov T. (1974). *Diccionario enciclopédico de las ciencias del lenguaje.* (Trad. E. Pezzont) (12a. Edición). México: Siglo XXI Editores. (Trabajo original publicado en 1972).

[20] Firebaugh M. W. (1988). *Artificial intelligence. A knowledge based approach.* Boston: PWS-KENT.

[21] Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (2000). *Connected components labeling.* Edinburgh, Scotland: The University of Edinburgh, Division of Informatics, Department of Artificial Intelligence. Retrieved from http://www.dai.ed.ac.uk/HIPR2/label.htm

[22] Freeman, J. A., & Skapura, D. M. (1991). *Neural networks: Algorithms, applications, and programming techniques.* Reading, Massachusetts: Addison-Wesley.

[23] Fundació Privada Catalana per a l'Orientació i Suport Tecnològic dels Cecs Manuel A. Caragol. (2002). *Estudio comparativo de programas OCR-2002.* Barcelona: Autor. Obtenido de http://www.funcaragol.org/html/cmocr2sl.htm

[24] Furui, S. (2001). *Digital speech processing, synthesis, and recognition* (2nd. ed.). New York: Marcel Dekker.

[25] Glymour, C., Ford, K., & Hayes, P. (1995). The prehistory of android epistemology. In G. F. Luger (Ed.), *Computation and intelligence: Collected readings* (pp. 3–21). Menlo Park, California: American Association for Artificial Intelligence.

[26] Golding, A. R. & Roth, D. (1999). Applying winnow to context-sensitive spelling correction. *Machine Learning. 34* 107–130.

[27] Gonzalez, R. C., & Woods, R. E. (1992). *Digital image processing.* Reading, Massachusetts: Addison-Wesley.

[28] González Araña C. y Herrero Aísa M. C. (1997). *Manual de gramática española. Gramática de la palabra, de la oración y del texto.* Madrid: Castalia.

[29] González Reyna S. (1990). *Manual de redacción e investigación documental.* (4a. Ed.). México: Trillas.

[30] Grishman R. (1991). *Introducción a la lingüística computacional.* (Trad. de A. Moreno Sandoval). Madrid: Visor Distribuciones. (Trabajo originalmente publicado en 1966).

[31] Hassoun, M. H. (1995). *Fundamentals of artificial neural networks.* Cambridge, Massachusetts: The MIT Press.

[32] Haugeland, J. (1988). *La inteligencia artificial.* México: Siglo Veintiuno.

[33] Havlik, J. M. (Comp.). (2000). *Informática y discapacidad: Fundamentos y aplicaciones.* Buenos Aires: Novedades Educativas.

[34] Hecht-Nielsen, R. (1990). *Neurocomputing.* Reading, Massachusetts: Addison-Wesley.

[35] Hertz, J. A., Krogh, A. S., & Palmer, R. G. (1991). *Introduction to the theory of neural computation.* Redwood City, California: Addison-Wesley.

[36] Instituto Nacional de Estadística, Geografía e Informática. (2002). *Agenda estadística de los Estados Unidos Mexicanos* (34a. ed.). México: Autor.

[37] Instituto Nacional de Estadística, Geografía e Informática (Ed.). (2002). *Directorio nacional de asociaciones de y para personas con discapacidad* (5a. Ed.). Aguascalientes, Aguascalientes: Instituto Nacional de Estadística, Geografía e Informática.

[38] Jiménez Pozo A. (1999). *Adaptación y mejora de un sistema de preprocesamiento y categorización gramatical.* Obtenido en octubre 10, 2003, Speech Technology Group, página personal del profesor Juan Manuel Montero Martínez:
http://lorien.die.upm.es/~juancho/pfcs.htm
http://lorien.die.upm.es/~juancho/pfcs/AJP/

[39] Jin R., Hauptmann A. G., & Xiang Z. C. (2003, January). A content-based probabilistic correction model for ocr document retrieval. In T. Kanungo, E. H. Barney Smith, H. Jianying, P. B. Kantor (Eds.) *Document Recognition and Retrieval X (Proceedings of SPIE/IS&T).* (pp. 128–135). Sta. Clara, California: The Society for Imaging Science and Technology.

[40] Jones M. P. & Martin J. (1997). Contextual spelling correction using latent semantic analysis. In *Proceedings of the Fifth Conference on Applied Natural Language Processing.* (pp. 166–173). USA: Washington DC.

[41] Jurafsky D. & Martin J. H. (2000). *Speech and laguage processing. An introduction to natural language processing, computational linguistics and speech recognition.* USA: Prentice-Hall.

[42] Kanai, J. (1998). Character recognition. In R. Cole, J. Mariani, H. Uszkoreit, G. Battista Varile, A. Zaenen, & A. Zampolli (Eds.), *Survey of the state of the art in human language technology* (pp. 435–438). Cambridge, Massachusetts: Cambridge University Press and Giardini.

[43] Kipnis, D. (1990). *Technology and power.* New York: Springer-Verlag.

[44] Klein, S. T., & Kopel, M. (2002, august). A voting system for automatic OCR correction. In K. Järvelin (General Chair), *Information Retrieval and OCR: From Converting Content to Grasping Meaning.* Workshop conducted at the 25th ACM SIGIR Conference, 2002, Tampere, Finland.

[45] Knight, K. (1990). Connectionist ideas and algorithms. *Communications of the ACM, 33*(11), 59–74.

[46] Konno, A., & Hongo, Y. (1993). Postprocessing algorithm based on the probabilistic and semantic method for Japanese OCR. In *Proceedings of the Second International Conference on Document Analysis and Recognition, 1993* (pp. 646–649). Los Alamitos, California: IEEE Press.

[47] Kukich K. (1992). Techniques for automatically correcting words in text. *ACM Computing Survey. 24*, 377–439.

[48] Lara Luis Fernando. (s. f.). *Diccionario del español usual en México.* Obtenido el 23 de abril de 2004, de Diccionario del español usual en México, Biblioteca Virtual Miguel de Cervantes: http://www.cervantesvirtual.com/servlet/SirveObras/06811650999196173088968/index.htm?na=26041

[49] LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 255–257). Cambridge, Massachusetts: The MIT Press.

[50] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In D. A. Touretzky (Ed.), *Advances in neural information processing systems 2 (NIPS 89)* (pp. 396–404). San Mateo, California: M. Kaufman.

[51] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (2001). Gradient-based learning applied to document recognition. In S. S. Haykin, & B. Kosko (Eds.), *Intelligent signal processing* (pp. 306–351). New York: IEEE Press.

[52] LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient backprop. In G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade* (pp. 9–50). Heidelberg, Germany: Springer-Verlag.

[53] Luger, G. F., & Stubblefield, W. A. (1993). *Artificial intelligence: Structures and strategies for complex problem solving* (2nd. ed.). Redwood City, California: Benjamin/Cummings.

[54] Malmberg B. (1989). *Los nuevos caminos de la lingüística.* (Trad. de J. Almela). México: Siglo XXI Editores. (Trabajo original publicado en 1967).

[55] Manning C. D. & Schütze H. (2001). *Foundations of statistical natural language processing.* Cambridge, Massachusetts: The MIT Press.

[56] Maravall Gómez-Allende, D. (1994). *Reconocimiento de formas y visión artificial.* Wilmington, Delaware: Addison-Wesley Iberoamericana.

[57] Màrquez L. (2002). Aprendizaje automático y procesamiento del lenguaje natural. En M. A. Martí, J. Llisterri (Eds.), *Tratamiento del lenguaje natural: Tecnología de la lengua oral y escrita* (pp. 133–188). Barcelona: Universidad de Barcelona.

[58] Martín de Santa Olalla Sánchez, A. (1999). *Una propuesta de codificación morfosintáctica para corpus de referencia en lengua española.* Obtenido en octubre 10, 2003, Estudios de lingüística española (ELiEs): http://elies.rediris.es/elies3/index.htm

[59] Michie, D., Spiegelhalter D. J. & Taylor C. C. (Eds.). (1994). *Machine learning, neural and statistical classification.* Great Britain: Ellis Horwood.

[60] Moreno Sandoval, A. (1998). *Lingüística Computacional.* Madrid: Síntesis.

[61] Mori, S., Nishida, H., & Yamada, H. (1999). *Optical character recognition.* New York: John Wiley & Sons.

[62] Morse, B. S. (2000). *Lecture 9: Shape Description (Regions).* Provo, Utah: Brigham Young University, Department of Computer Science. Retrieved from http://rivit.cs.byu.edu/550/w02/lectures/lect09.html

[63] Morse, B. S. (2003). *Lecture 5: Level operations (Part I).* Provo, Utah: Brigham Young University, Department of Computer Science. Retrieved from http://rivit.cs.byu.edu/450/w03/lectures/lect-05.php

[64] Moure T. y Llisterri J. (1996). Lenguaje y nuevas tecnologías: El campo de la lingüística computacional. En Milagros Fernández (Coord). *Avances en lingüística aplicada* (pp. 147–211). Santiago de Compostela, Universidad: Servicio de publicaciones e intercambio científico.

[65] Myka Andreas. (n.d.). *Putting paper documents in the World-Wide Web.* Retrieved August 18, 2003, from Electronic Proceedings of the "Second World Wide Web Conference '94: Mosaic and the Web"! Developer's Day: http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/myka/node8.html#SECTION00042000000000000000

[66] Nartker, T. A., Taghva, K., Young, R., Borsack, J., & Condit, A. (2003). OCR correction based on document level knowledge. In T. Kanungo, E. H. Barney Smith, J. Hu, & P. B. Kantor (Eds.), *Document Recognition and Retrieval X: Proceedings of the IS&T/SPIE's 15th Annual Symposium on Electronic Imaging Science and Technology, 2003* (Vol. 5010, pp. 103–110). Bellingham, Washington: The International Society for Optical Engineering.

[67] Nartker, T. A., & Young, R. (2002). *OCR accuracy produced by the current DOE document conversion system* (Tech. Rep. 2002–06). Las Vegas: University of Nevada, Information Science Research Institute.

[68] Nebauer, C. (1998). Evaluation of convolutional neural networks for visual recognition. *IEEE Transactions on Neural Networks, 9*(4), 685–696.

[69] O'Gorman, L., & Kasturi, R. (1995). *Document image analysis.* Los Alamitos, California: IEEE Computer Society Press.

[70] Oh, I.-S., & Suen, C. Y. (2002). A class-modular feedforward neural network for handwriting recognition. *Pattern Recognition, 35*(1), 229–244.

[71] Organización Mundial de la Salud. (1997). *Blindness and visual disability* (Fact Sheet No. 142). Ginebra: Author.

[72] Parker, J. R. (1996). *Algorithms for image processing and computer vision.* New York: John Wiley & Sons.

[73] Pérez Guerra, J. (1998). *Introducción a la lingüística de corpus: Un ejercicio con herramientas informáticas aplicadas al análisis textual.* Santiago de Compostela: Torculo Ediciones.

[74] Perez-Cortes, J.C., Amengual, J.C., Arlandis, J., & Llobet, R. (2000). *Stochastic error-correcting parsing for ocr post-processing.* International Conference on Pattern Recognition ICPR-2000. (pp. 4405–4408). Barcelona, Spain.

[75] Ping Zhang, Lihui Chen, & Alex, K. C. (2000). Text document filters using morphological and geometrical features of characters. In *Proceedings of the 5th International Conference on Signal Processing, 2000* (Vol. 1, pp. 472–475). Los Alamitos, California: IEEE Press.

[76] Pollock J. J. & Zamora A. (1984, april) Automatic spelling correction in scientific and scholarly text. In *Communication of the ACM. 27*, 358–368.

[77] Prechelt, L. (1994). *PROBEN1: A set of benchmarks and benchmarking rules for neural network training algorithms* (Tech. Rep. 21/94). Karlsruhe, Germany: Universität Karlsruhe, Fakultät für Informatik.

[78] Ramstad, T. A. (1998). Still image compression. In V. K. Madisetti, & D. B. Williams (Eds.), *The digital signal processing handbook* (pp. 52-1–52-27). Boca Raton, Florida: CRC Press.

[79] Reilly, R. (2001). Finding your way in the dark. *Multimedia Schools, 8*(3), 73–76.

[80] Rich, E. (1983). *Artificial intelligence.* Republic of Singapore: McGraw-Hill.

[81] Russell, S. J., & Norvig, P. (1996). *Inteligencia artificial: Un enfoque moderno.* México: Prentice Hall Hispanoamericana.

[82] Schmidt, A. (1996). *A modular neural network architecture with additional generalization abilities for high dimensional input vectors.* Master's thesis, Manchester Metropolitan University, England.

[83] Sinha, R.M.K. (1993). On using syntactic constraints in text recognition. In *Proceedings of the Second International Conference on Document Analysis and Recognition.* Tsukuba Science City, Japan, pp. 858–861.

[84] Sonka, M., Hlavac, V., & Boyle, R. (1999). *Image processing, analysis, and machine vision* (2nd. ed.). Pacific Grove, California: PWS Publishing Company.

[85] Sproat, R., & Olive, J. (1998). Text-to-speech synthesis. In V. K. Madisetti, & D. B. Williams (Eds.), *The digital signal processing handbook* (pp. 46-1–46-11). Boca Raton, Florida: CRC Press.

[86] Srihari, S. N., & Lam, S. W. (1995). *Character recognition* (Tech. Rep. CEDAR-TR-95-1). Amherst: State University of New York at Buffalo, Center of Excellence for Document Analysis and Recognition.

[87] Srihari, S. N., & Srihari, R. K. (1998). Written language input: Overview. In R. Cole, J. Mariani, H. Uszkoreit, G. Battista Varile, A. Zaenen, & A. Zampolli (Eds.), *Survey of the state of the art in human language technology* (pp. 71–76). Cambridge, Massachusetts: Cambridge University Press and Giardini.

[88] Taghva, K., & Stofsky, E. (2001). OCRSpell: An interactive spelling correction system for OCR errors in text. *International Journal on Document Analysis and Recognition*, *3*(3), 125–137.

[89] Tong, X., & Evans, D. A. (1996). A statistical approach to automatic OCR error correction in context. *Proceedings of the Fourth Workshop on Very Large Corpora* (pp. 88–100)

[90] Luz Abril Torres Méndez. (1998). *Viterbi algorithm in text recognition.* Retrieved April 8, 2003, from McGill University, Centre for Intelligent Machines, term proyect for course page: http://linas.org/mirrors/www.cim.mcgill.ca/2002.09.19/~latorres/Viterbi/va_applet.htm

[91] Vemuri, V. R. (1992). *Artificial neural networks: Concepts and control applications.* Los Alamitos, California: IEEE Computer Society Press.

[92] Wang, J., & Jean, J. (1993). Multiresolution neural networks for omnifont character recognition. In *IEEE International Conference on Neural Networks*, 1993 (Vol. 3, pp. 1588–1593). Los Alamitos, California: IEEE Press.

[93] Winograd T. (1983). *Language as a cognitive process.* USA: Addison-Wesley.

[94] Winston, P. H. (1992). *Artificial intelligence* (3rd. ed.). Reading, Massachusetts: Addison-Wesley.