Universidad Nacional Autónoma de México

Facultad de Ciencias

IMPLEMENTACIÓN DE RECONOCIMIENTO DE VOZ CON APRENDIZAJE PROFUNDO

T E S I S

Que para obtener el título de

Licenciado en Ciencias de la computación

P R E S E N T A :

Benjamin Torres Saavedra

T U T O R

Dra. Verónica Esther Arriola Ríos

Ciudad universitaria, Ciudad de México, 2017

Dedico este trabajo a mi familia, la cual supo guiarme y darme todas las herramientas necesarias a lo largo de los años para poder concluir con mi formación en aquello que me apasiona.

Agradecimientos

A mi tutora Verónica Arriola Ríos, por orientarme para la realización de esta tesis, por la paciencia que eso requiere y por haber enriquecido mi educación a cada paso del camino.

A mis profesores en la carrera: por haberme instruido en cada paso de ella y contribuir con mi desarrollo académico, entre ellos al Dr. David Flores Peñaloza, y a la Dra. Amparo López Gaona, además de mis antiguos profesores que inspiraron mi deseo de estudiar esta área de la ciencia que tanto disfruto: Diego González y Antonio Granja.

A Sandy García, Brisa Castro, Eduardo Villa, Paulina Carranco, Roberto Piche, Tonathiu Maldonado y Ulises Cervantes, pues cada vez que mi voluntad se resistía a continuar, estuvieron para escucharme y apoyarme, sin ustedes el camino hubiera sido mucho mas difícil de recorrer y obviamente mucho menos alegre.

A Claudia Rojas, por haberme guiado con su luz a través del momento más sombrío de mi vida y gracias a la cual aprendí a hacer los cambios necesarios en ella.

Finalmente: Al proyecto PAPIME [1] "Laboratorio de aprendizaje de máquina multimodal" por el financiamiento otorgado para la conclusión de esta tesis.

[1] clave PE109116

Índice general

Índice de figuras

Capítulo 1

Motivación

En la presente tesis se hará una evaluación de la posibilidad de la implementación de un sistema de reconocimiento de voz haciendo uso de lo que se conoce como aprendizaje profundo, el cual tiene varias definiciones, pero, entre ellas:

> Una clase de técnicas de aprendizaje automático que explotan muchas capas de procesamiento no lineal de información para la extracción y transformación supervisadas o no supervisadas de características, y para el análisis y clasificación de patrones.[1, p. 109-201]

Lo anterior debido a que durante los últimos años ha existido un aumento en las aplicaciones del aprendizaje profundo en diversas áreas como el reconocimiento de imágenes, voz y de patrones en general,atribuido en gran medida a el incremento en la capacidad de cómputo, sobre todo por la popularización del uso de GPUs[1, p. 208](y la reducción de sus costos) para realizar cálculos en paralelo; por otro lado: el surgimiento de Big Data, que abre las puertas a realizar procesos de entrenamiento con datos reales a los cuales poder ajustar un modelo. Aunado a lo anterior existe un motivo que por su simplicidad demuestra que la utilidad de un método es determinante en su adopción: Deep learning logro mejoras en los sistemas de reconocimiento de voz por encima de otras técnicas, lo cual hizo que esta área de la inteligencia artificial recibiera renovada atención alrededor del 2009.[2]

Aunque el plan de estudios de ciencias de la computación en la facultad de ciencias incluye las materias de inteligencia artificial y de redes neuronales, en

éstas no se alcanza a desarrollar un sistema de tal complejidad como podría ser un reconocedor de voz, en parte por que requiere de amplios conocimientos no fundamentales como el análisis de audio y extracción de características, lo anterior sin contar el límite de tiempo que se tiene para adquirirlos durante un semestre y, al mismo tiempo recolectar datos de entrenamiento y realizar su implementación. Lo anterior incita a proveer a los alumnos de la facultad la oportunidad de aprender cómo funcionan las técnicas para realizar estos trabajos, con el fin de mantener a la vanguardia el plan de estudios de la carrera, el cual en estos momentos no contempla la necesidad de estudiar ni implementar un sistema similar.

Por tales motivos en la presente tesis me he dado a la tarea de reunir diversas fuentes que implementan sistemas similares para guiar a alumnos de semestres superiores de la carrera en la implementación de un sistema reconocedor de voz, con lo cual sería posible que lo realicen en un periodo corto de tiempo, comparado con el requerido para reunir y analizar por ellos mismos la información disponible.

1.1. Objetivo

Para esta tesis buscare evaluar la factibilidad de utilizar aprendizaje profundo para reconocer un conjunto predefinido de comandos de voz en español para un robot. Lo anterior se hará desde la etapa de captura de datos hasta la implementación del sistema propuesto.

1.2. Hipótesis

Con los conocimientos adquiridos durante los últimos semestres de la carrera de ciencias de la computación en inteligencia artificial, redes neuronales y algunos conocimientos adicionales en análisis del habla es posible construir un sistema reconocedor de voz básico que sea capaz de identificar un conjunto reducido de comandos aptos para dirigir un robot.

1.3. Introducción

En los siguientes capítulos se expondrán las tareas requeridas para poder implementar un sistema de reconocimiento de voz teniendo como principal instrumento las redes neuronales. Los primeros capítulos presentarán las herramientas necesarias para la captura y procesamiento de voz y darán una breve explicación del uso que se les dará a lo largo de esta tesis.

En concreto, en el capítulo 2 se brinda una explicación de lo que será usado cómo materia prima para el trabajo: audio y características junto con una explicación del problema central: del reconocimiento de voz, además en el capitulo 3 se dará un breve recordatorio de lo que es y como funciona una red neuronal, con lo cual quienes no tengan experiencia alguna en el área podrán entrar en materia rápidamente y entender, en un inicio, cuál será la línea de trabajo que se sigue para la implementación de redes neuronales (recolección de datos, preprocesamiento, entrenamiento, etc), posteriormente se introducirían los modelos ocultos de Márkov en el capitulo 4, los cuales formaran parte fundamental de la solución final de transcripción del audio en texto.

En el capítulo 5 se comenzará propiamente con el desarrollo del sistema reconocedor de voz desde su parte más esencial: la recolección de datos y su procesamiento. Aunque esta tarea pudiera pensarse sencilla, quien desee implementar el sistema deberá tener en cuenta ciertas formas de organización en sus datos para que, en un futuro, pueda manipular fácilmente un volumen considerable de información. Es importante notar que el sistema propuesto tiene como objetivo reconocer voz en español, lo cual conlleva el problema de encontrar un corpus de datos en nuestro idioma, de ahí que sea necesario mostrar cómo se realiza la captura de datos.

En el capítulo 6 se expone un modelo propuesto para la clasificación de los trifonos extraídos del audio y la justificación de dicho modelo. Es destacable el hecho de que, aunque el modelo que se usará para realizar la tarea no es alguno que se pueda decir forma parte del *estado del arte*, éste es suficiente para la magnitud de la tarea que se busca resolver y, posiblemente un modelo mas complejo requeriría una gran cantidad de datos de entrenamiento, los cuales no se encuentran fácilmente disponibles en español y requeriría una gran cantidad de tiempo para poder ser recolectarlo.

En el capítulo 7 se explicará cómo reconstruir las frases completas que fueron

capturadas en el audio, pues, previo a este momento lo que se tiene son únicamente trifonos que componen palabras, además, se hará una comparación con el el uso de modelos ocultos de Márkov.

Finalmente se muestran los resultados en el capitulo 8 de este trabajo y se habla brevemente del trabajo futuro que podría ser realizado para mejorar la precisión de este sistema.

Capítulo 2

El problema del reconocimiento de voz

Cuando hablamos de reconocimiento de voz automático nos referimos a la tarea de una computadora en convertir discurso humano a su forma de texto[3], sin que esto signifique en algún momento que el sistema comprenda el significado que tienen las palabras que esta transcribiendo. Esta tarea tiene sentido en una gran variedad de entornos que van desde tareas cotidianas como pueden ser las transcripciones a correos electrónicos, notas de alguna clase o listas de compras, pasando por la asistencia a personas con discapacidades físicas que les impidan escribir; dar los comandos mediante voz a sistemas tales como los robots y por su puesto la traducción de un idioma a otro de algún contenido audiovisual. El fin último del reconocimiento automático de voz es lograr que una computadora reconozca en tiempo real, con precisión del 100 % todas las palabras dichas por cualquier persona independientemente del ruido o de las características particulares de la voz del interlocutor.[4, p. 48]

Dado que es una tarea compleja, en un principio se ha dividido el problema en distintas variedades y suelen clasificarse en las siguientes:[4][5]

1. *Reconocimiento de palabras aisladas*, en el cual se busca reconocer una única palabra y el *reconocimiento de secuencias de palabras*.

2. Reconocimiento de *discurso espontáneo*, que suele contener rupturas a la estructura gramatical del idioma y *reconocimiento de discurso escrito*, el

cual sigue las reglas gramaticales del idioma que está siendo reconocido y es un problema más sencillo que el reconocimiento de un discurso espontáneo.

3. Reconocimiento de voz *dependiente* e *independiente* del hablante.

4. Reconocimiento de voz de *vocabulario cerrado*, donde todas las palabras a reconocer se considera que forman parte de un diccionario y el de *vocabulario abierto* que no realiza tal suposición.

5. Reconocimiento de voz *cercana* y *distante* el cual considera el problema de separar la voz humana del entorno en el cual se encuentran los locutores.

Esta tesis tendrá como objetivo la implementación de un sistema que reconozca una secuencia de palabras predefinidas e independiente del hablante con grabaciones realizadas de forma cercana al individuo. El conjunto de palabras predefinidas corresponderá a una lista de comandos que sean útiles para brindar instrucciones a un robot y se definirá en detalle en el capitulo 5.

Los sistemas de reconocimiento de voz suelen ser modulares y separan el proceso de la clasificación del sonido en varias tareas principales[6] (ver figura[1] 2.1): La captura y preprocesamiento de la señal,el cual recibe los datos recolectados por un micrófono, esta señal suele pasar a través de algún método de mejoramiento del audio (como la disminución del ruido de fondo o amplificación) con la finalidad de maximizar las posibilidades de que el siguiente módulo sea capaz de extraer características útiles de la grabación; la extracción de características, que se encarga de eliminar datos innecesarios de la señal y enfatizar los datos que son de utilidad para distinguir una señal de otra[7], con lo cual el modulo de clasificación tendrá una cantidad menor de datos que procesar y la ventaja de que los datos serán útiles en la clasificación; El modelo acústico, que se encarga de asignar una etiqueta a las características de las muestras recolectadas en algún elemento básico como puede ser un fono,trifono o palabra. Finalmente el modelo de lenguaje se encarga de delimitar y reconstruir, las palabras y frases dichas por el interlocutor de manera tal que se obtenga la traducción de audio en texto.[4]

La extracción de características se realiza con el fin de simular los datos que recibe el sistema auditivo humano[8, p. 1739] bajo la idea de que un sistema que emule al cerebro (como una red neuronal) debería ser capaz de reconocer el habla si tiene los mismos datos que obtiene el ser humano. El módulo de clasificación

[1] Imagen tomada y traducida de http://recognize-speech.com/

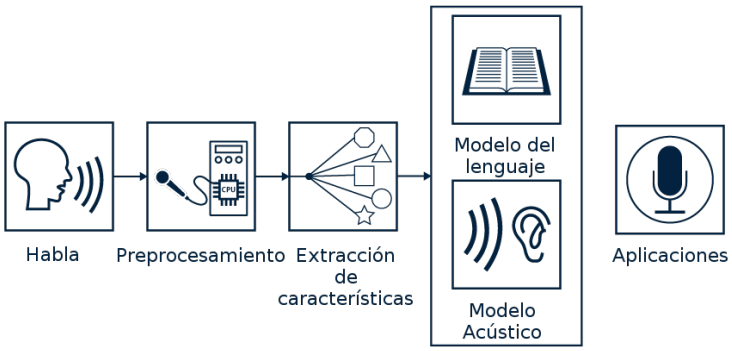


Figura 2.1: Diagrama del proceso de reconocimiento de voz. Se muestran los pasos que realiza el sistema: preprocesamiento de la señal, extracción de características, clasificación y reconstrucción en texto con las palabras del audio transcritas.

corresponde con la función del cerebro, el cual se encarga de analizar la señal recibida y asignarle una etiqueta, ya sea en palabras del orador o en sonidos ambientales.

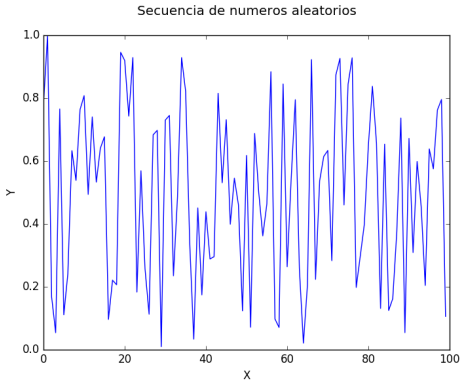
En el caso de esta tesis el preprocesamiento es realizado por el mismo programa que realiza la extracción de características (HTK) y consiste en una ventana de Hamming a la señal para realzar la parte central del audio, la cual esta definida como [9, p. 62] :

$$v(n) = 0.54 - 0.46cos(\frac{2\pi(n-1)}{N-1}),$$ con N el número de muestras de la señal.

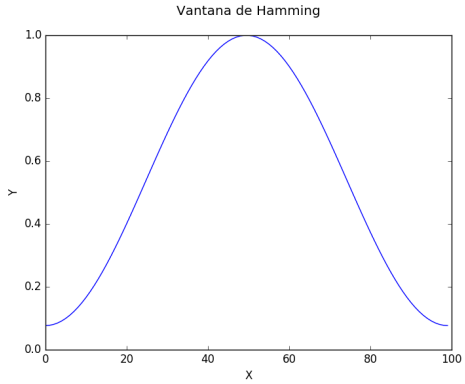
Esta ventana,de tamaño variable es multiplicada por la señal original y lo que logra es incrementar la magnitud del sonido en su parte central,un ejemplo de este efecto puede verse en la figura 2.2.

En cuanto a las características seleccionadas: serán utilizados los coeficientes PLP (Perceptual Linear Prediction,de los cuales se habla en la siguiente sección). Para el modelo acústico sera usada una red neuronal que se encargara de clasificar segmentos del sonido capturado y al ultimo estos segmentos serán usados por una cadena oculta de Márkov para obtener la frase que fue enunciada por el locutor.

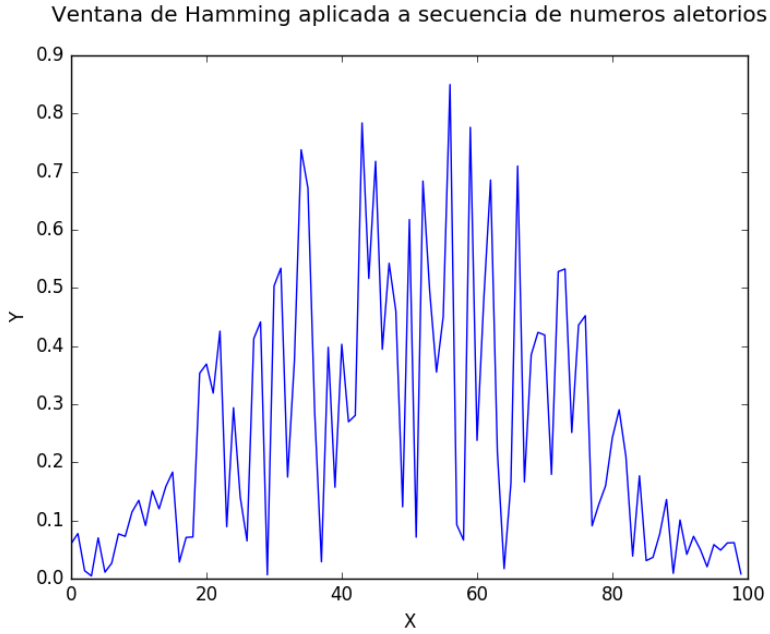
Figura 2.2: Ejemplo de señal aleatoria y el resultado al aplicar ventana de Hamming.



(a) En esta figura se muestra una señal de 100 muestras con valores aleatorios en el intervalo (0,1).



(b) Ventana de Hamming de tamaño 100



(c) Al multiplicar la ventana de Hamming por la señal elemento a elemento se obtiene esta nueva señal,la cual muestra valores mayores en su parte central, dando énfasis a esta parte de la señal.

Para la reconstrucción de las frases se usara un proceso similar al mostrado en la figura 2.3 [2]. En la parte inferior de la imagen se observa la capa P, la cual corresponde a los trifonos que serán identificados por el modelo acústico. A continuación, en la capa superior, estos trifonos son usados por un modelo de Márkov que tiene información sobre cuales son las componentes del trifono (capa L) para poder averiguar cual fue el **fono** que fue dicho (las etiquetas de ambas capas no coinciden por que sencillamente no es necesario,en la capa P puede usarse cualquier identificador para cada trifono siempre que se pueda recordar cuales son sus componentes en la capa L).En la capa Q, dado que los trifonos ya han aportado información contextual, se puede afirmar que hay un trifono determinado (notar que de todos los fonos en cada trifono de P,solo se conservo el intermedio) para poder reconstruir una palabra o frase ,lo cual se realiza en la capa W.

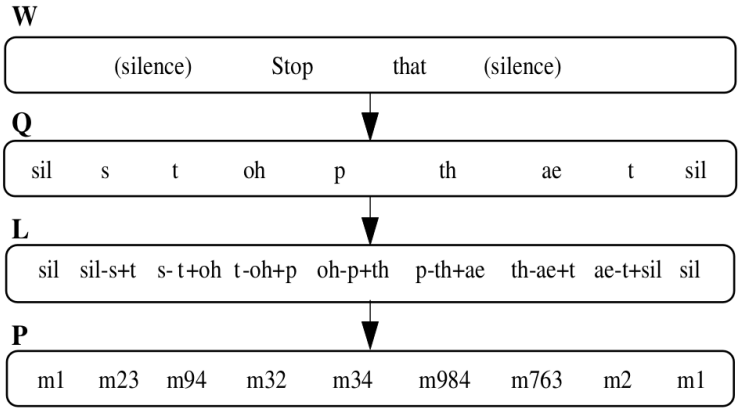


Figura 2.3: Diagrama de reconstrucción de trifonos a palabras.

A lo largo de los siguientes capítulos se abordaran los temas que guardan relación directa con la implementación de estos sistemas: redes neuronales, características, y modelos ocultos de Márkov.

[2]Imagen tomada de [10, p. 207]

2.1. Audio y características

Ya que la red neuronal que será implementada operará sobre sonidos es indispensable conocer cómo los datos de audio se encuentran codificados en la computadora.

En el mundo físico el sonido es la perturbación mecánica de algún medio, ya sea algún gas (como el aire), un líquido o un sólido causada por la aplicación de una fuerza sobre él[11], en el caso del sentido del oído en los seres humanos y animales, esta perturbación mecánica altera el tímpano, haciéndolo vibrar. Esta perturbación puede ser entendida como una onda, donde las crestas y valles se corresponden con variaciones de la intensidad de la fuerza aplicada. Con la breve descripción anterior es natural comprender al sonido como un fenómeno continuo a lo largo del tiempo, es por tal que para su manejo digital el sonido es discretizado en una secuencia de intensidades muestreadas a intervalos fijos de tiempo. Así pues, es posible conceptualizar al sonido como un simple arreglo de números que se encuentran en algún rango (por ejemplo: entre 0 y 1) y que puede ser reproducido emitiendo tales intensidades a una velocidad similar a la cual fue capturada (del orden de decenas de millones de veces por segundo, es decir MHz).

Las grabaciones de sonidos que corresponden al habla humana tienen características que varían de una persona a otra dependiendo de factores tales como: el género del hablante, su idioma natal, el idioma en el cual esté hablando al momento de la grabación y el estado de ánimo (entre otras). Además las grabaciones son distintas entre si aunque el hablante sea el mismo y se hayan realizado en momentos cercanos en el tiempo, lo cual se puede atribuir a diferencias entre los dispositivos de grabación o ruido en el ambiente en el cual se estén realizando las capturas de audio.

Es por lo anterior que es necesario encontrar una forma de minimizar las diferencias entre sonidos, de tal manera que grabaciones diciendo la misma palabra (o frase) por distintas personas en momentos diferentes y capturadas con dispositivos distintos sean aproximadamente similares. Es posible realizar esta tarea procesando el audio y extrayendo características PLP(Perceptual linear prediction), las cuales realizan justo la tarea deseada[8]: eliminar diferencias entre sonidos, realizando además un filtrado importante de información que el humano no es capaz de percibir, con lo cual se estará proporcionando a una red neuronal (o a cualquier otro método) una entrada que es similar a la que percibimos los

humanos cuando interpretamos el habla de otra persona. Para el proceso de extracción de estas características usaremos HTK3, un conjunto de herramientas que permiten (entre otras cosas) realizar esta tarea.

Finalmente definiremos lo que será la entrada de la red neuronal: **trifonos**.

Los trifonos son la unión de tres fonos, donde un **fono** se entiende como: "Cada uno de los segmentos de características acústicas particulares y con duración típica en que podemos dividir la secuencia sonora." [12] Es decir, que la red neuronal recibirá un fragmento de audio que contenga tres sonidos "inseparables" de una grabación de voz. Por ejemplo: la palabra "agarra" se puede descomponer en los fonos: a, g, a, rr, a, por lo que los trifonos que componen a la palabra son: aga, garr y rra, este último con la peculiaridad de que se le añade un fono "silencio" (únicamente conceptual) para que la red pueda identificar los trifonos en cualquier momento que hayan sido capturados, incluyendo aquellos que estaban por terminar en la ventana de audio actual.El uso de trifonos se debe a que hay estudios que indican que su uso permite mejorar el rendimiento de los sistemas de reconocimiento de voz gracias al contexto que brindan.[13][14]

2.1.1. Perceptual Linear Prediction (PLP)

Antes de entrar en materia y revisar detalles sobre cómo se realiza el proceso de extracción de características PLP, es prudente realizar una breve introducción sobre predicción lineal.

La *predicción lineal* es una operación matemática donde los valores futuros de una señal discreta son estimados como una función lineal de las muestras anteriores.[15]

Es decir:

$$\hat{x}(x) = \sum_{i=1}^{p} a_i x(n-i) \tag{2.1}$$

donde $\hat{x}$ es el valor que buscamos predecir y $x(n-i)$ son los valores anteriores de la señal.

Otro proceso importante para comprender PLP es la descomposición de señales usando la *transformada discreta de Fourier(DFT)*. A su vez, es necesario que el

lector recuerde algunos conocimientos de álgebra lineal para comprender la transformada, en particular es necesario recordar los conceptos de *combinación lineal* y *base ortogonal*, ya que, esencialmente, la transformada de Fourier encuentra una forma de descomponer una señal (vector) en sus componentes dentro del espacio de frecuencias del sonido. La propiedad más importante de las bases ortogonales, es que permiten encontrar de manera simple los coeficientes para expresar a un vector X como combinación lineal de los elementos de ella, es decir:

$$\text{si } \beta = e_1, ..., e_n \text{ es una base ortogonal}$$

$$a_i = \langle x, e_i \rangle$$

$$X = \sum_{i=1}^{n} a_i e_i$$

Ahora bien:

La transformada discreta de Fourier de una secuencia de N números complejos esta dada por [16]:

$$x_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \text{ con } k = 0, ..., N-1 \tag{2.2}$$

con i la unidad imaginaria y $e^{-\frac{2\pi i}{N}}$ la N-ésima raíz de la unidad.

Nótese que los elementos $e^{\frac{2\pi i}{N} kn}$ forman una base ortogonal sobre el cuerpo de los vectores complejos N-dimensionales, por lo que, si deseamos reconstruir la señal original podemos utilizar los coeficientes obtenidos de la manera siguiente:

$$X_n = \frac{1}{N} \sum_{k=0}^{N-1} a_k e^{\frac{2\pi i}{N} kn} \text{ con } k = 0, ..., N-1 \tag{2.3}$$

Así que, ahora, las señales pueden ser vistas como combinación lineal de elementos de funciones exponenciales, lo cual nos permite realizar una operación vital para el análisis del sonido: eliminar componentes del audio que no aporten información útil, es decir: remover el ruido.

La eliminación del ruido se puede realizar disminuyendo (o estableciendo a 0) el coeficiente de los componentes que corresponden a frecuencias que no se encuentren en el rango que deseamos escuchar claramente. En nuestro caso, sabemos

que el ser humano tiene un rango de audición de entre 20 Hz y los 20 kHz por lo que, en realidad podemos descartar todas las componentes inferiores y superiores a esas frecuencias en el audio.

Y es justo ahora que es posible comenzar a estudiar cómo es que son calculadas las características PLP. Aunque la introducción de este proceso se realiza en el articulo "Perceptual linear predictive (PLP) analysis of speech"[8], a continuación se repasará de manera simple cuál es la serie de pasos usados para calcular dichos datos.

El proceso está descrito en el siguiente diagrama de bloques 2.4:

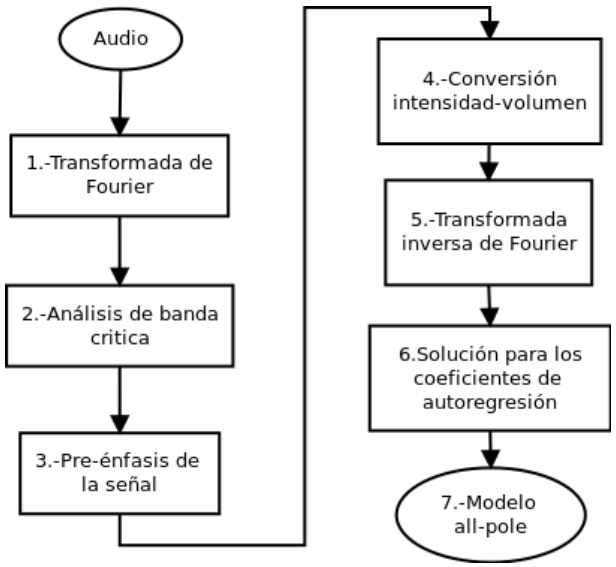


Figura 2.4: Diagrama de bloques del proceso de análisis de voz PLP

1. Antes de proceder con el tratamiento de la señal es necesario conocer cuales son los componentes de los que consta, lo cual se logra aplicando la transformada de Fourier al audio a analizar, la transformada nos dará como resultado las frecuencias presentes en el audio.

2. A continuación se realiza un análisis de *banda critica*, el cual se encarga de descomponer la señal en un conjunto discreto de muestras de espectros cuya información es similar a la que recibe el sistema auditivo.

3. El siguiente paso consiste en un proceso de pre-enfatización de la señal, que se encarga de incrementar aquellas frecuencias del sonido que sabemos que corresponden a las que son audibles por el ser humano aunque su intensidad sea baja.

4. La señal es procesada usando la "ley de potencia de Steven", que asigna una nueva magnitud subjetiva a la intensidad que provoca el estímulo sonoro en :el oído humano[17], dicha operación esta definida como:

$$\psi(I) = kI^a \tag{2.4}$$

   donde I es la magnitud del estimulo físico, a es un exponente que depende del tipo de estimulo (en nuestro caso 0.33 [8, p. 1740]) y k es una constante de proporcionalidad dependiendo de las unidades usadas.

   Este procesamiento tiene como finalidad dar importancia a las frecuencias que son notadas con mayor facilidad por las personas independientemente si "deberían" escucharlas igual.

5. Los datos son devueltos a una señal usando la transformada inversa de Fourier (recordemos que en los pasos anteriores estábamos trabajando con frecuencias de la señal, no con la señal en si).

6. Se usa un modelo auto-regresivo para poder obtener los coeficientes que hacen que la señal se pueda ver en función de sus valores temporales anteriores.

7. Finalmente se usan los coeficientes del paso anterior para encontrar una base que sea capaz de expresarlos con un modelo "all pole"', la cual resulta conveniente para expresar la voz y permite hacer un mejor uso de los datos anteriormente calculados.

La salida del proceso son una serie de *coeficientes cepstrales*, los cuales son una serie de valores para la representación del habla que tratan de emular lo que percibe un humano de las señales auditivas de una manera compacta, por lo cual son usados para la tarea del reconocimiento de voz.[18] [19]

Capítulo 3

Redes neuronales

Al ser el modelo principal de este trabajo, procederé a dar una revisión de lo que es una red neuronal y cómo se realizan sus procedimientos esenciales: evaluación y entrenamiento. Primero, la definición de lo que es una red neuronal: "En su sentido más amplio una red neuronal es una máquina diseñada para modelar la manera en la cual el cerebro realiza una tarea particular o una función de interés." [20]

3.1. Usos

Las redes neuronales tienen diversas aplicaciones, entre las cuales se encuentran:

- Clasificación de datos

  En este caso una red neuronal se encarga de recibir datos y asignar a cada uno de los ejemplares de entrada una etiqueta correspondiente a la clase a la que pertenece. Entre los ejemplos de este tipo de usos se encuentran: Clasificación de correos electrónicos (spam y correo deseado), sistemas de reconocimiento óptico de caracteres (OCR) y la de detección e identificación de rostros y objetos.

- Aproximación de funciones

Estas redes son entrenadas para poder simular el comportamiento de una función de la cual únicamente se conoce su resultado. Ejemplo de este tipo de redes es la que permite calcular el valor de recompensa al realizar una acción, como en el caso de la usada en el artículo Playing Atari with Deep Reinforcement Learning. [21]

- Procesamiento de datos

  Las redes encargadas de realizar operaciones sobre datos pueden tener diversos fines tales como comprimir (ya sea con o sin pérdida) o filtrar información. Uno de los ejemplos más llamativos son las redes neuronales que permiten obtener una imagen de mayor calidad a partir de una imagen de baja calidad, esto, para ahorrar tiempo y ancho de banda en la transferencia datos, sobre todo en dispositivos móviles. [22]

3.1.1. Estructura de las redes neuronales

- Neurona y red neuronal

  La neurona es la unidad de procesamiento de información fundamental para una red neuronal [20, p. 11]. Una esquematización de una neurona podría ser la mostrada en la figura 3.1.

  Sus componentes, de izquierda a derecha, son: Señales de entrada, las cuales están etiquetadas por X_i, pesos de cada una de las entradas, en W_k, sesgo de la neurona en b, suma de las entradas $\sum$ y finalmente una función de activación φ que da lugar a la salida de la neurona, denominada y.

  Cada una de las entradas X_i de la neurona es codificada como números flotantes, al igual que el valor de los pesos (w) y el sesgo (b).

  Una función de activación es una que se encarga de delimitar cuando una neurona proporciona una salida "positiva" o no. La función más simple que podría usarse es la llamada función escalón, la cual se aplica a la combinación lineal de entradas y pesos.[20, p. 13]

$$\varphi(x) = \begin{cases} 1 & si \quad x \geq 0 \\ 0 & si \quad x < 0 \end{cases}$$

  Otras funciones útiles son:

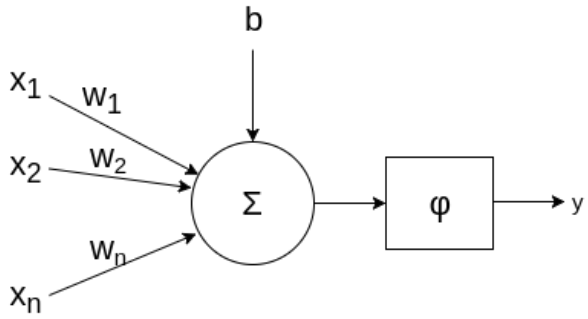


Figura 3.1: Esquema de una neurona

- Sigmoide [23, p. 197]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

- Softmax[24]

$$f(x)_i = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_j}} \tag{3.2}$$

  Donde x_j es la j-ésima entrada del vector de características del ejemplar X.

- ReLu [25]

$$\varphi(x) = max(0, x) \tag{3.3}$$

En particular: la función softmax tiene especial relevancia en la tarea de clasificación con redes neuronales debido a que proporciona una interpretación probabilística de la salida de la evaluación de un elemento por la red neuronal, forzando a que todas las salidas de una determinada capa sumen 1

De acuerdo con [20, p. 10],para calcular el valor de la salida de una única

neurona se utiliza la siguiente fórmula:

$$y = \varphi\left(\sum_{j=1}^{m}(w_j x_j) + b\right) \tag{3.4}$$

Que, como se dijo antes, corresponde a aplicar la función de activación φ a la combinación lineal de los pesos con la entrada sumada a b. Ahora: si vemos a la entrada como una matriz de m x 1 entradas, a los pesos como matriz de 1 x m entradas, con m el número de características de cada ejemplar, podríamos expresar el valor de salida con un producto de matrices:

$$y = \varphi(WX + b) \tag{3.5}$$

Adicionalmente, podemos eliminar el valor b, agregando un nuevo peso a la neurona con el índice 0 y estableciendo siempre en su entrada un 1, lo cual dejaría el valor de salida únicamente como:

$$y = \varphi(WX) \tag{3.6}$$

Ahora que contamos con un modelo para una sola neurona es posible extender el modelo a una serie de neuronas que reciben una entrada y que, en conjunto, producen una o más salidas, tal como ilustra la figura 3.2. Donde:

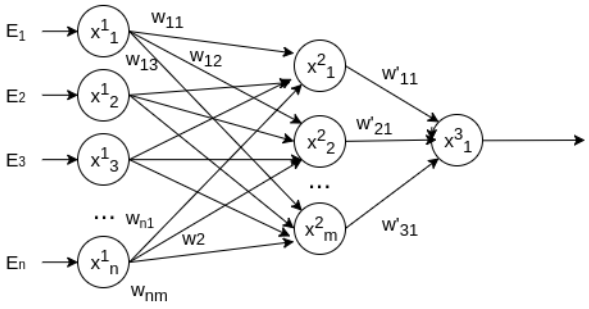


Figura 3.2: Extensión del modelo

E_i es el dato de entrada que recibe la neurona.

x_i, w_{ij} es el peso de la conexión entre la neurona i de la capa de entrada y la neurona j de la capa oculta, estos pesos, en conjunto forman W^1, la última capa puede consistir en una neurona como la del modelo original (si es que solo se desea producir una salida) o en otra capa con más neuronas.

En el caso que deseemos evaluar en una sola operación múltiples ejemplares, podemos crear una matriz X de m x n, con m igual al número de ejemplares a introducir a la red y n el número de datos para cada uno de ellos. Para calcular la salida en este caso usaremos:

$$A = \varphi(XW^1) \tag{3.7}$$

Que da como resultado una matriz de m x n que ahora puede ser considerada como una entrada a la siguiente capa realizando una transposición, resultando en A^T = n x m. Para calcular el valor final de la red únicamente repetimos la operación con W^2 (los pesos de la segunda capa).

$$B = \varphi(A^T W^2)W^2 \text{matriz de m x k} \tag{3.8}$$

Resultando en B una matiz de n x k entradas.

Donde cada una de las entradas contendrá las K salidas de la red neuronal para cada uno de los n elementos.

Algunos otros detalles sobre el cálculo de los valores de salida pueden ser consultados en "Neural Networks and Learning Machines" de Simon Haykin.[20]

Al igual que con las funciones de activación, existen varias funciones de error que pueden ser usadas para entrenar la red, dos de ellas que suelen ser usadas con frecuencia son:

1. Error cuadrático medio (Mean squared error)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) \tag{3.9}$$

2. Logística o entropía cruzada

$$L = -\frac{1}{n}\sum_{n=1}^{n}[y_n log(\hat{y_n}) + (1 - y_n) log(1 - \hat{y_n})] \tag{3.10}$$

En cuanto a la función logística, ésta puede tener dos formas: aquella que evalúa la clasificación en m clases y la que únicamente considera 2 clases posibles (llamada binaria).

- Entrenamiento

  La sección anterior únicamente nos permite calcular el valor de salida de uno o elementos al ser procesados por la red neuronal, lo cual es substancialmente diferente a que tales valores sean los que esperamos obtener como solución a nuestro problema.

  Para poder obtener los resultados deseados de una red neuronal, ésta primero debe pasar por un proceso que se encarga de calibrar los pesos de las conexiones entre neuronas para dar mayor o menor importancia a los datos que se van obteniendo con cada capa, tal proceso se denomina entrenamiento. El entrenamiento que será usado para poder realizar el reconocimiento de voz es denominado entrenamiento supervisado (pues conocemos de antemano el valor que deseamos obtener). Aunque existen muchas variedades de este proceso conocido como *descenso por el gradiente*, en términos generales todos siguen pasos similares a lo siguiente: [20, p. 34] [26, p. 734]

  1. Alimentar a la red neuronal con los datos que se desean procesar.
  2. Obtener el resultado que la red proporciona con los pesos actuales.
  3. Contrastar el resultado con la salida que debería haberse producido y haciendo uso de una función de error, asignar un valor de error a cada salida.
  4. Con base en las características de la red (en particular de las funciones de error) y el error obtenido calcular el gradiente de la función de error ∇Wy actualizar los pesos de la red de acuerdo a la siguiente regla:

$$W = W - \gamma * \nabla W \tag{3.11}$$

  γ es conocido como taza de aprendizaje.

Este proceso se realiza también con la primera capa, pero esta vez se calcula el gradiente con respecto a los pesos de esa capa y se actualizan los pesos y se repite lo anterior con los datos disponibles para entrenar a la red hasta que el error se encuentre en un umbral aceptable.

En la sección 6 se hará uso del método *AdaDelta* [27], el cual es una variación del algoritmo anterior en dos puntos clave: El establecimiento del valor γ y el hecho de que este podría no ser útil durante todo el proceso de entrenamiento.

Esencialmente *AdaDelta* se encarga de ajustar el parámetro γ durante todo el proceso de entrenamiento realizando un decaimiento exponencial de su valor. Entre las ventajas de este método de optimización está el hecho de que mientras más cerca de la solución se encuentre un modelo, el ajuste realizará cada vez una alteración menor, además de que el valor de la taza de aprendizaje puede ser arbitrariamente colocado, pues este será calculado a un mejor valor en una iteración siguiente.

Capas dropout

Fruto del entrenamiento al que son sometidas las redes neuronales eventualmente generan las salidas esperadas para cada uno de los ejemplares de entrenamiento. El problema que suele presentarse es que justamente el entrenamiento calibra los pesos de la red neuronal para poder dar respuesta a los datos que le fueron proporcionados y no siempre es capaz de responder correctamente ante datos nuevos, lo cual reduce en buena medida la utilidad de una red neuronal pues escencialmente solo clasifica los datos que ya tenemos etiquetados para el propio aprendizaje. A este problema se le conoce como *overfitting*, en español: "sobreajuste".

Ante el problema anterior existen varias estrategias de entrenamiento que permiten detectar y evitar el fenómeno, entre otras se encuentran:

- Incremento del número de elementos de entrenamiento
- Detener el entrenamiento de la red neuronal en cuanto el rendimiento en el conjunto de validación comience a decrecer[28].

- Partición de los datos de entrenamiento disponibles en 3[29]: Conjunto de entrenamiento, el cual sera usado para ajustar los parámetros del modelo; Conjunto de validación: el cual permitirá ajustar los hiperparámetros del modelo (por ejemplo: capas, número de neuronas, funciones de activación, etc); Conjunto de prueba: el cual permitirá evaluar el rendimiento de cualquiera de los posibles modelos usados.

- Regularización de los pesos[30]: Se suelen preferir los modelos que son simples, una medida de la simplicidad de la red neuronal es la diversidad de magnitudes de sus pesos, con esta técnica se modifica la función objetivo para penalizar los modelos con pesos grandes.

- Dropout[28]: Consiste en eliminar de manera aleatoria pesos de la red durante una época de entrenamiento.

Las estrategias anteriores suelen ser de utilidad, aunque no son siempre posibles, sobre todo en el caso del incremento en la cantidad de datos de entrenamiento, los cuales no siempre están disponibles. No es el caso con las capas dropout, ya que para aplicar esta técnica únicamente es necesario crear una capa (o cambiar la función de activación, según la implementación) en la red neuronal entre otras dos y eliminar de manera aleatoria las contribuciones de una capa a la otra. Según Nitish Srivastava: "Previene el sobreajuste y proporciona una manera aproximada de combinar un numero exponencial de arquitecturas de redes neuronales de manera eficiente"[28].

En las secciones 4 y 5.1 de "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" se define formalmente la operación:

> Asi pues, para implementar dropout es necesario establecer un parámetro p, el cual sera la probabilidad de eliminar la contribución de una neurona en una época de entrenamiento, con lo cual si se considera una red neuronal de L capas $1, \ldots, L$ y sea $z^{(l)}$ denota el vector de de entradas a la capa l, $y^{(l)}$ denota el vector de salidas de la capa l ($y^{(0)} = x$ es la entrada). $W^{(l}$ y $b^{(l)}$ son los pesos y sesgos en la capa l. La operación *feed-forward* en una red neuronal estándar puede ser descrita (para $l \in 0, \ldots, L-1$) como:

$$z_i^{l+1} = w_i^{(l+1)} y^l + b_i(l+1), \tag{3.12}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}), \tag{3.13}$$

Donde f es cualquier función de activación, por ejemplo $f(x) = 1/(1 + exp(-x)$. Con dropout, la operación de *feed-forward* se convierte en:

$$r_j^{(l)} \sim Bernoulli(p), \tag{3.14}$$

$$\tilde{y}^{(l)} = r^{(l)} * y^{(l)}, \tag{3.15}$$

$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^l + b_i^{(+1)}, \tag{3.16}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{3.17}$$

One-hot

Aunque ya se han definido las redes neuronales, sus operaciones de evaluación y entrenamiento y revisamos algunos tipos de funciones de activación, aun queda un punto a tratar antes de poder implementar una red neuronal: la codificación de sus datos de salida.

Anteriormente se dijo que las salidas de una red neuronal son comparadas con un valor objetivo para poder realizar el ajuste de los pesos, pero no siempre es útil proporcionar como etiquetas de salida los valores que esperamos obtener de la red neuronal. Un caso donde esto puede llevar a problemas es en el que se espera que la red sea capaz de clasificar sus entradas en alguna categoría, el cual es el uso que se va a dar a las redes neuronales en esta tesis.

Para tareas de clasificación, donde la salida de la red es una clase en vez de un valor numérico son utilizadas *variables categóricas*. Estas variables pueden tomar uno de un número limitado y fijo de posibles valores en base a una alguna propiedad cualitativa[31].

Ejemplo:

1. Género: hombre o mujer

2. Tipos de sangre: A, B, AB u O.

3. Especies de mascotas: gatos, perros, peces, etc

4. Estados de la república: México, Monterrey, Aguascalientes.

5. Colores: rojo, azul, verde, amarillo, etc

Es evidente que las variables anteriores no tienen ninguna restricción en su representación como números que aquella que nos pueda resultar conveniente según el tipo de problema que busquemos resolver. En el caso de querer asignar una etiqueta meramente numérica a cada una de esas variables puede llevar a resultados extraños en la predicción de una red neuronal[32],por el hecho de que la red neuronal podría comenzar a asumir alguna relación de orden entre las clases del estilo:

$$\text{Rojo} > \text{Azul} > \text{Verde} > \text{Amarillo}$$

sencillamente por que asignamos 4=amarillo, 3=verde, 2=azul y 1= rojo.

Entonces si buscamos aprender una una relación entre ciertas características $C = [x_1, \ldots, x_n]$ y sus resultados $G = \{1, 2, 3, 4\}$,podríamos diseñar la red para que su salida sean tales dígitos en formato binario, es decir '$G = \{001, 010, 011, 100\}$.Durante el proceso de entrenamiento (incluso al final de tal) podrían aparecer valores de salida como 111 el cual no tiene una correspondencia clara a alguna de las clases que definimos, ya que 101 podría ser interpretado como 001 o 100, en ambos casos con un error.

Lo anterior lleva a la necesidad de codificar las etiqueta de cada clase de manera que sea clara la distinción entre una clase y otra; para lo cual usaremos la codificación *one-hot*

Para obtenerla podemos valernos de las etiquetas que ya teníamos en el conjunto G y de su cardinalidad N (en este caso 4) de la siguiente manera:

1. Todas las etiquetas tendrán una longitud de N dígitos.

2. Cada etiqueta tendrá establecido en 1 el bit que corresponda con su etiqueta numérica Ejemplo: 1=[1000], 2=[0100], 3=[0010] y 4=[0001]

Si además nuestra red neuronal hace uso de la función softmax en su última capa, esta representación tiene una interpretación muy útil: cada entrada del arreglo de números representa la probabilidad de que la clase del elemento evaluado corresponda a la i-esima entrada. Por ejemplo: [0.8 0.05 0.05 0.1] nos indicaría que es 80 % probable que el elemento evaluado corresponda a la primera clase, 5 % a la segunda, 5 % a la tercera y 10 % a la cuarta.

Capítulo 4

Modelo oculto de Márkov

Una vez que la red neuronal sea capaz de clasificar correctamente los trifonos de las instrucciones dadas, es necesario recopilar la información de fragmentos sucesivos de audio para poder averiguar la palabra y, en última instancia, la frase que fue dicha en la grabación.

Para lo anterior se puede hacer uso de un modelo oculto de Márkov, el cual requiere algunas definiciones previas antes de ser explicado.

Propiedad de Márkov:[33] En teoría de la probabilidad y estadística, la propiedad de Márkov se refiere a la propiedad de ciertos procesos estocásticos por la cual "carecen de memoria", lo cual significa que la distribución de probabilidad del valor futuro de una variable aleatoria depende únicamente de su valor presente, siendo independiente de la historia de dicha variable. A los procesos que satisfacen esta condición se les conoce como procesos de Márkov.

Proceso de Márkov:[34] En la teoría de la probabilidad y en estadística, un proceso de Márkov, llamado así por el matemático ruso Andréi Márkov, es un fenómeno aleatorio dependiente del tiempo para el cual se cumple una propiedad específica: la propiedad de Márkov.

Finalmente se define una **Cadena de Márkov**[35]. Una cadena de Márkov es una secuencia $X_1, X_2, X_3, ...$ de variables aleatorias. El dominio de estas variables es llamado espacio estado; el valor de X_n es el estado del proceso en el tiempo n. Si la distribución de probabilidad condicional de X_{n+1} en estados pasados es una función de X_n por sí sola, entonces:

$$P(X_{n+1} = x_{n+1}|X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1) = P(X_{n+1} = x_{n+1}|X_n = x_n) \tag{4.1}$$

Donde x_i es el estado del proceso en el instante i. La identidad mostrada es la propiedad de Márkov.

El ejemplo trata de predecir cuál es la secuencia de climas con relación a las actividades que realiza una persona cuando se dan determinados climas.

Gráficamente, un modelo se puede representar como en la figura4.1:

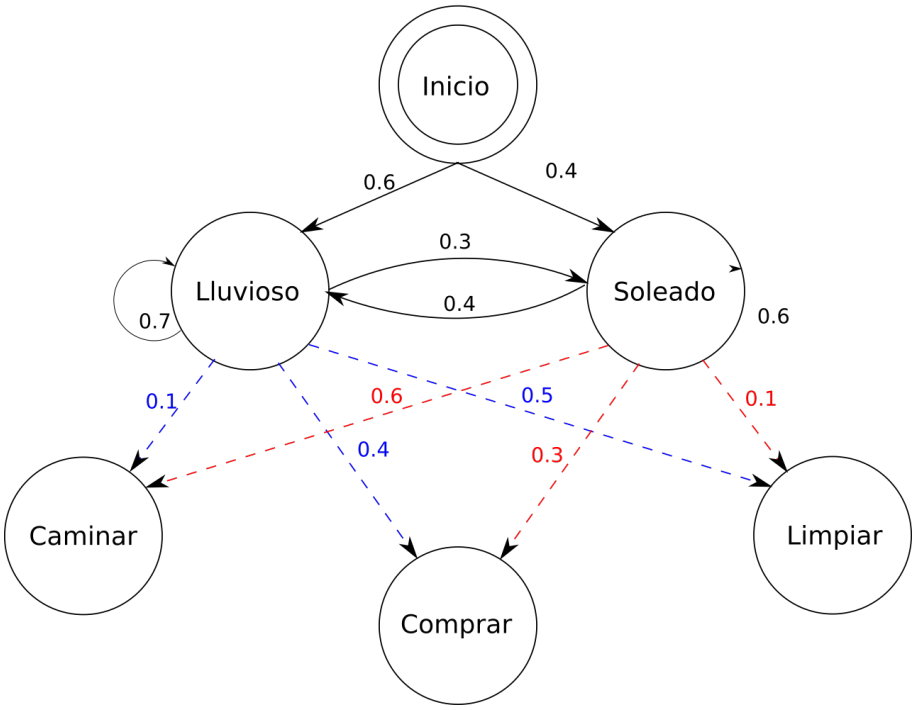


Figura 4.1: Representación de un modelo oculto de Márkov.

Donde:

Inicio, lluvioso y soleado son los estados observables del clima (lo que se trata de predecir con el modelo del ejemplo)

Caminar, comprar y limpiar son las acciones que la persona realiza en base al estado del tiempo.

Además, el modelo requiere de varios datos adicionales: la probabilidad de que se pase del clima lluvioso al soleado, la de cambio del clima de soleado a lluvioso y las probabilidades de que dado alguno de los dos climas, el siguiente día tenga el mismo clima. Junto con lo anterior, se debe definir cuáles son las probabilidades de que en determinado clima se realicen alguna de las tres acciones (las cuales se obtienen de manera experimental y reciben el nombre de observaciones).

Formalmente un modelo oculto de Márkov queda definido por la tupla (Q,V,ϕ,A,B), de ahora en adelante denotada por λ:

Donde:

Q es el conjunto de estados

V es el conjunto de posibles valores $v_1, \ldots, v_n$

$\pi = \pi_i$ son las probabilidades iniciales, es decir, la probabilidad de que el primer estado sea Q_i

A es el conjunto de probabilidades $A = a_{ij}$ de transiciones entre estados

B es el conjunto de probabilidades de las observaciones, es decir:

$$b_j(v_k) = P(o_t = v_k | q_t = j) \tag{4.2}$$

la probabilidad de observar v_k cuando se esta en el estado j en el instante t

La secuencia de observaciones se denota por $O = (o_1, ...o_T)$.

Luego con el algoritmo de *búsqueda de Viterbi* se podrá estimar cuál fue la secuencia de las actividades que fueron realizadas por la persona dado algún clima y, eventualmente, predecir cuales serán las actividades que se realizaran en momentos futuros.

El algoritmo de Viterbi se encarga de encontrar la secuencia de estados que más se ajusta a la serie de observaciones obtenida [26, p. 877], en este caso por la red neuronal.

Brevemente: el algoritmo de Viterbi se encarga de calcular cuáles son las observaciones que son posibles en un momento determinado del tiempo. Para el primer elemento de la secuencia hace uso de ϕ_i y la probabilidad de emisión de cada trifono, de tal manera que se obtiene, para cada uno de los trifonos, la probabilidad de haber sido observado en la primera posición de la secuencia. Para el resto de los elementos se realiza un proceso similar: se realiza la estimación de la probabilidad de observar un trifono, dado que se vio o_{t-1}, y se busca el elemento que maximiza la probabilidad de transitar desde el elemento del paso anterior al que acaba de ser observado y se considera a tal elemento el siguiente en la secuencia. Debe recalcarse el hecho de que únicamente se continuará con los elementos de mayor probabilidad, por lo que no se tomarán en cuenta todas las posibles secuencias.

Una vez que repetimos los pasos anteriores con toda la secuencia de observaciones, obtenemos la secuencia correcta (o al menos probable) que debió haber sido dicha en una grabación.

4.1. Algoritmo de Viterbi:

[6]

[1]

Inicialización:

Sea:

$$\delta_1 = \pi_i b_i(o_1) \text{ donde: } i \in [1, N] \tag{4.3}$$

$$\varphi_1(i) = 0 \tag{4.4}$$

Recursión:

$$\delta_{t+1} = \max_i [\delta_t(i) a_{ij}] b_j(o_{t+1}) \qquad \text{donde: t=2...,T y } j \in [1, N] \tag{4.5}$$

$$\varphi_{t+1} = \operatorname*{argmax}_{1 \le i \le N} [\delta_t(i) a_{ij}] \tag{4.6}$$

[1] http://recognize-speech.com/acoustic-model/hidden-markov-models/viterbi-algorithm

Terminación:

$$q_T^* = \underset{1 \leq i \leq N}{\operatorname{argmax}}[\delta_T(i)] \tag{4.7}$$

Y para reconstruir la secuencia de estados probables:

$$q_t^* = \varphi_{t+1}(q_{t+1}^*) \text{ donde: } t = T-1, T-2, ..., 1 \tag{4.8}$$

Antes de explicar el algoritmo de Baum-Welch que realiza la estimación de los parametros del HMM, es necesario introducir el calculo de dos variables: Forward (α) y Backward (β).

4.2. Forward

[36, p. 18]

Consideramos la variable $\alpha_t(i)$ como:

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, q_t = i|\lambda) \tag{4.9}$$

Dado el modelo λ, $\alpha_t(i)$ es la probabilidad de observar $o_1, o_2, \ldots, o_t$ y estar en el instante de tiempo t y en el estado i

Cálculo hacia adelante de la probabilidad de una secuencia de observaciones.

Inicialización:

$$\alpha_1(i) = \pi_i b_i(o_1), i \in [1, N] \tag{4.10}$$

Recurrencia:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}), \text{ con: } t \in [1, T-1] \text{ y } j \in [1, N] \tag{4.11}$$

Terminación:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i). \tag{4.12}$$

4.3. Backward

[36, p. 19]

Consideramos la variable $\beta_t(i)$.

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T|q_t = i, \lambda) \tag{4.13}$$

Dado el modelo λ, $\beta_t(i)$ es la probabilidad de la secuencia de observación desde el instante de tiempo t+1 hasta el final, cuando el estado en el instante de tiempo t es i.

Inicialización:

$$\beta_T(i) = 1, i \in [1, N] \tag{4.14}$$

Recurrencia:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\beta_{t+1}(j)b_j(o_{t+1}) \text{ con } \text{ t=T-1,T-2...1 y con } 1 \leq \ i \leq N \tag{4.15}$$

Terminación:

$$P(O|\lambda) = \sum_{i=1}^{N} \beta_1(i)\pi_i b_i(o_1) \tag{4.16}$$

4.4. Algoritmo de Baum-Welch

[36, p. 25][37] Dada una secuencia de observaciones $O = (o_1, o_2, \ldots, o_T)$, el al-

goritmo de Baum y Welch permite estimar los parámetros λ de un HMM que maximizan la probabilidad de dicha secuencia, es decir, $P(O|\lambda)$.

Es necesario conocer:

1. El número esperado de transiciones desde el estado i en O
2. El número esperado de transiciones desde el estado i al estado j en O

Para ello definimos previamente $\xi_t(i,j)$ como la probabilidad de estar en el estado i en el instante t y en el estado j en el instante t+1, dado una observación O y el modelo λ.

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O,\lambda) \tag{4.17}$$

$$\xi_t(i,j) = \frac{P(q_t = i, q_{t+1} = j|O,\lambda)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \tag{4.18}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{k=1}^{N}\sum_{l=1}^{N}\alpha_t(k)a_{kl}b_l(o_{t+1})\beta_{t+1}(l)} \tag{4.19}$$

Y finalmente definimos también $\gamma_t(i)$ como la probabilidad de estar en el estado i en el instante t,

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j) \tag{4.20}$$

De tal manera que, sumando cada $\gamma_t(i)$ en cada instante de tiempo obtenemos:

1. El número esperado de transiciones desde el estado i en la observación O

$$\sum_{t=1}^{T-1}\gamma_t(i) \tag{4.21}$$

2. Y haciendo lo mismo con cada $\xi_t(i,j)$ obtenemos el número esperado de transiciones desde el estado i al estado j en la observación O.

$$\sum_{t=1}^{T-1} \xi_t(i,j) \tag{4.22}$$

El funcionamiento del procedimiento iterativo es el siguiente:

- Se parte de un modelo inicial que puede seleccionarse aleatoriamente
- Se realiza el cálculo de las transiciones y símbolos de emisión que son mas probables según el modelo inicial escogido.
- Se construye un nuevo modelo que incrementa la probabilidad de las transiciones y símbolos determinados en el paso anterior. Para la secuencia de observaciones en cuestión, el modelo tendrá ahora una probabilidad mayor que el modelo anterior.
- Repetir los dos pasos anteriores varias veces hasta que no exista mejora entre un modelo y el siguiente.

La probabilidad de estar en el estado i en el instante de tiempo t=1 (es decir, la probabilidad inicial):

$$\overline{\pi}_i = \gamma_1(i), i \in [1, N] \tag{4.23}$$

Para reestimar las probabilidades de transición:

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \text{ con } i,j \in [1,N] \tag{4.24}$$

Y finalmente para las probabilidades de emisión:

$$\overline{b_j}(o_k) = \frac{\sum_{t=1:o_t=o_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \text{ con } j,k \in [1,N] \tag{4.25}$$

El numerador representa el número esperado de veces que se pasa por el estado j y que se observa o_k, y el denominador representa el número de veces que se pasa por el estado j.

Capítulo 5

Preparación de datos

5.1. Herramientas

- Lenguaje Python
  - Pydub Esta API sera usada para manipular fragmentos del audio de manera sencilla.
  - Seqlearn Es una biblioteca de Python que permite la implementación de modelos ocultos de Márkov, lo cual sera útil en la ultima fase de implementación.
- Tensorflow De reciente publicación, esta herramienta es un conjunto de bibliotecas enfocadas a la implementación de sistemas de aprendizaje automatizado y en especial de redes neuronales.
- Keras Es una API que implementa envoltorios para el diseño rápido de redes neuronales en TensorFlow y Theano. Con esta API se pueden realizar rápidamente cambios a un modelo de red, con lo cual se reduce el tiempo de implementación y es posible enfocarse en los parámetros de ella en vez de en cómo se realizan los procesos necesarios para implementarla.
- Audacity Este editor de audio fue elegido por ser de uso libre e implementar una función fundamental para el posterior desarrollo del proyecto: la posibilidad de etiquetar segmentos de audio. item HTK Su utilidad en este trabajo radica en su capacidad de extraer características PLP del audio

capturado. Este programa es de código abierto, por lo cual es posible realizar cambios para mejorarlo y adaptarlo de acuerdo a las necesidades del usuario.

Mas información sobre la instalación de estas herramientas se encuentra en el apéndice A.

5.2. Definición del vocabulario

El siguiente paso para poder implementar el sistema de reconocimiento de voz es realizar la captura de muestras de audio, las cuales serán usadas para entrenar la red neuronal, que entrenaremos para reconocer trifonos.

Para realizar la captura de datos es requerido contar con:

- Lista de comandos a reconocer.
- Archivo con la lista de todas las palabras usadas en los comandos.
- Diccionario de pronunciación del español con las palabras usadas en los comandos.
- Voluntarios para realizar el lote de grabaciones.
- Computadora con micrófono instalado y funcional.
- Audacity.

En nuestro caso la lista de comandos será:

- Agarra la esponja
- Agarra el dado rojo
- Agarra el dado verde
- Agarra el dado azul
- Alto
- Avanza
- Busca el dado azul
- Busca la esponja
- Busca el dado rojo
- Busca el dado verde
- Gira a la derecha
- Gira a la izquierda

- Retrocede
- Saluda
- Ve al dado azul
- Ve al dado rojo
- Ve al dado verde
- Ve a zona amarilla
- Ve a zona naranja
- Ve a zona negra
- Ve a la esponja

Y con esa lista de comandos deberá crearse una lista adicional con todas las palabras (sin repeticiones) usadas:

La lista deberá estar en orden alfabético y tener una palabra por renglón. A lo largo de este documento la lista se considerará almacenada en el archivo words-sorted.list. El diccionario de fonemas se puede obtener de diversas fuentes e inclusive puede ser generada manualmente, el requisito es que cumpla con el formato aceptado por HTK, es decir que se forme de la siguiente manera:

- Una palabra por renglón (sensible a capitalización).
- Todas las palabras deberán encontrarse en orden alfabético ascendente.
- Después de cada palabra se debe dejar un espacio y escribir cada uno de los fonemas que compone a la palabra, separando cada uno por un espacio.

Nuestro diccionario [dict] es el siguiente:

```
1  A a
2  AGARRA a g a rr a
3  ALTO a l t o
4  AMARILLA a m a r i ll a
5  AVANZA a b a n z a
6  AZUL a z u l
7  BUSCA b u s k a
8  DADO d a d o
9  DERECHA d e r e ch a
10 EL e l
11 ESPONJA e s p o n j a
12 GIRA j i r a
13 IZQUIERDA i z k i e r d a
14 LA l a
15 NARANJA n a r a n j a
16 NEGRA n e g r a
17 RETROCEDE rr e t r o z e d e
```

```
18 ROJO rr o j o
19 SALUDA s a l u d a
20 SENT-END [] sil
21 SENT-START [] sil
22 VE b e
23 VERDE b e r d e
24 ZONA z o n a
```

Listing 5.1: Ejemplo de un diccionario en español, a la izquierda las palabras dentro de él y a la derecha su descomposición en fonos.
Notese la inclusión de SENT-END y SENT-START, las cuales son marcas requeridas por HTK para procesar el inicio y final de cada oración.

5.3. Generación del listado de fonemas

La lista de fonos que deberá aprender la red neuronal será generada automáticamente a partir del diccionario de fonemas y el listado de palabras del paso anterior usando HTK, para esto bastará con ejecutar:

```
1 HDMan -m -w words-sorted.list -n phones.list -l
2 dlog phones.dict dict
```

donde:

words-sorted.list será el resultado de cada una de las palabras descompuestas en fonos presentes en el diccionario.

phones.list es la lista de fonemas usados.

dlog será el log del proceso, donde pueden encontrarse (en caso de haber) la palabras no presentes en el diccionario que están en el listado.

dict será el diccionario del español que fue descargado o generado.

Una vez realizado obtendremos phones.list, archivo que servirá de ayuda para saber qué fonos deberemos etiquetar en nuestras muestras de audio.

5.4. Captura de datos

Por cada una de las frases de nuestra lista de comandos y para cada uno de los voluntarios disponibles se debe grabar un archivo en formato .wav. Aunque

HTK usa únicamente audio a 16, 000 KHz y en un canal, recomiendo usar los ajustes predeterminados (44100 KHz y 2 canales) en caso de que en un futuro sea posible usar ambos para realizar el análisis PLP, con lo cual podríamos obtener mas información para entrenamiento y un mejor rendimiento de la red neuronal. La captura únicamente consiste en oprimir el botón rojo en la interfaz para iniciar la grabación, dictar la instrucción y oprimir el cuadro azul de STOP para finalizar la grabación (ver figura 5.1) Se recomienda agrupar todas las muestras de una instrucción en una misma carpeta y guardar los archivos con nombres que den una idea de cual es su contenido sin necesidad de reproducirlo.

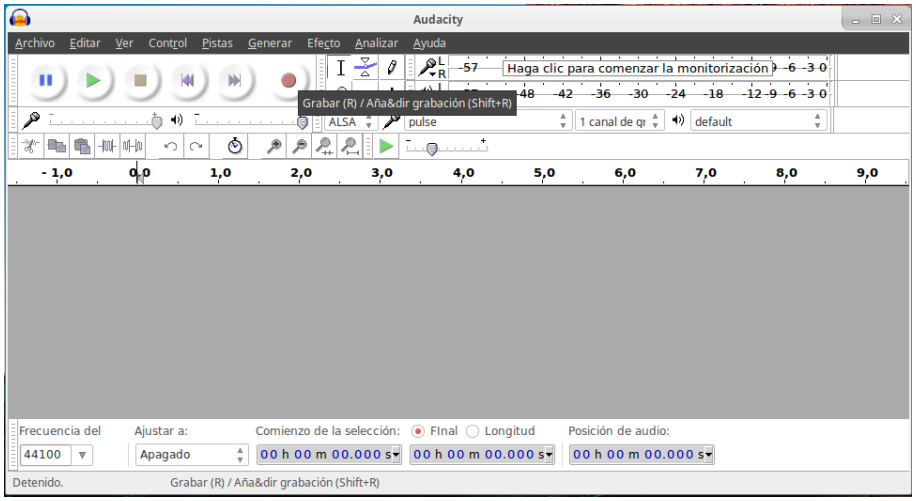


Figura 5.1: Interfaz de Audacity

Ejemplo:

Agarra la esponja –>agesp1.wav

Que será la grabación de la instrucción indicada realizada por el voluntario número 1.

5.5. Etiquetado de fonos

Ahora, con ayuda del listado generado en phones.list se puede etiquetar el audio capturado de los voluntarios. El archivo phones.list tiene una apariencia similar al de la imagen5.2:

Éste archivo (phones.list) es generado con el conjunto de palabras mencionado anteriormente, tiene 104 grupos de fonos (algunos de ellos bifonos o fonos simples) y trifonos, con los cuales se puede tener una referencia de los trifonos que debemos etiquetar en cada grabación.

```
a+sp
a-sp
a+g
a-g+a
g-a+rr
a-rr+a
rr-a+sp
a+l
a-l+t
l-t+o
t-o+sp
o-sp
a+m
a-m+a
m-a+r
a-r+i
r-i+ll
i-ll+a
```

Figura 5.2: Fragmento del archivo phones.list donde se muestran los fonos usados en el diccionario "dict", es de especial interés observar las lineas de la mitad de la imagen, pues en ellas se observan trifonos, los cuales pueden ser usados como guia para el etiquetado. *Nota:sp=silencio.

El proceso se realiza también usando Audacity.

- Seleccionar Archivo >Abrir... y seleccionar el archivo de audio que se desea etiquetar.
- Seleccionar Pistas>Añadir nueva>Pista de Etiqueta.
- Usando el mouse se selecciona el segmento de audio a etiquetar y se escribe el nombre del fono contenido en ese segmento.
- Repetimos a lo largo de todo el audio hasta etiquetar los fonos correspondientes a la orden dada, para dudas sobre la separación de fonos en una palabra se puede consultar el archivo phones.list donde se encontrarán las descomposiciones de cada palabra.
- Una vez etiquetados todos los fonos de una grabación deben guardarse el archivo de etiquetas, esto seleccionando: Archivo>Exportar etiquetas y asignando un nombre al archivo. Se recomienda que el nombre de las etiquetas sea similar al archivo que etiqueta, con el fin de poder encontrar

fácilmente la grabación correspondiente.

El audio etiquetado deberá verse de manera similar en Audacity (figura 5.3).

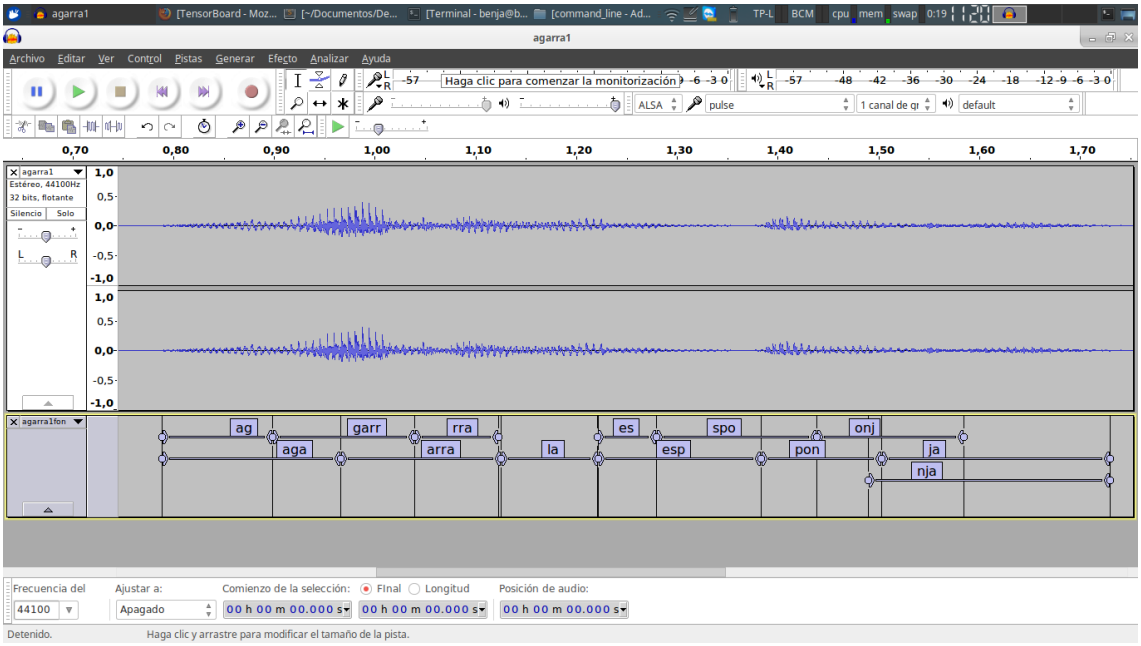


Figura 5.3: Captura de una muestra de audio etiquetada en Audacity. En la parte inferior de la captura se observa el inicio, final y etiqueta de cada trifono.

Cada uno de los archivos de etiquetas cumple con el formato:
[inicio en segundos] [final en segundos] [etiqueta correspondiente]

5.6. Separación de fragmentos de audio

Una vez que tenemos las grabaciones de audio completadas podemos proceder a separar de cada una de ellas los fragmentos de audio que contengan únicamente determinado fono, para lo cual usaremos las etiquetas creadas con Audacity anteriormente.
Esta tarea es sencilla de realizar utilizando PyDub en python, para lo cual es aconsejable crear un script con las siguientes funciones:

- Recibir nombre de archivo de audio y nombre de archivo de etiquetas.

- Abrir el archivo de etiquetas y multiplicar cada uno de los números del audio x1000 para obtener los tiempos de cada fono en milisegundos.
- Crear un segmento de audio de PyDub con el archivo de audio.
- Recorrer la lista de etiquetas y generar nuevos segmentos de audio usando los milisegundos como índices de inicio y final.
- Guardar cada uno de los nuevos segmentos, se recomienda guardar usando el nombre del archivo y el fono como nombre del archivo.

Un ejemplo simple de como cargar un archivo usando PyDub es el siguiente:

```
1  from pydub import AudioSegment
2  audio=AudioSegment.from_wav("prueba.wav")
3  #Una vez creado el AudioSegment podemos manipular el objeto como
4  #una lista y los indices corresponden a milisegundos del archivo
5  duracionAudio=len(audio)
6  #separar el archivo en dos audios de la mitad de duracion del
       original
7  mitadInicial=audio[:duracionAudio/2]
8  mitadFinal=audio[duracionAudio/2:]
9  #guardarmos los archivos
10 mitadInicial.export("mitadA.wav",format="wav")
11 mitadFinal.export("mitaB.wav",format="wav")
```

5.7. Calculo de características PLP

A continuación realizaremos la extracción de características PLP de los archivos de fonos usando una de las herramientas de HTK: HCopy.
PLP, por sus siglas Perceptual Linear Prediction, extrae información del sonido para, entre otras cosas, eliminar las frecuencias inaudibles al oído humano y eliminar la dependencia de hablante de la grabación, lo cual es ideal para un sistema que se espera pueda reconocer comandos de distintas personas.
Para la extracción será necesario un archivo de configuración de HTK como el siguiente: [1]

```
1  SOURCEKIND = WAVEFORM
2  SOURCEFORMAT = WAVE
```

[1] Obtenido de `http://labrosa.ee.columbia.edu/matlab/rastamat/mfccs.html`

```
 3 SOURCERATE = 625
 4 TARGETKIND = PLP_0
 5 TARGETRATE = 100000.0
 6 WINDOWSIZE = 250000.0
 7 USEHAMMING = T
 8 PREEMCOEF = 0.97
 9 NUMCHANS = 20
10 CEPLIFTER = 22
11 NUMCEPS = 12
12 USEPOWER = T
13 LPCORDER = 12
```

el cual se considerara guardado en config.plp en el directorio de trabajo.
Donde:

SOURCEKIND Indica la fuente de donde espera obtener datos (en nuestro caso un archivo)

SOURCEFORMAT Es el codec en el cual estará codificado el archivo de entrada

SOURCERATE Velocidad de muestreo de la fuente en múltiplos de 100ns

TARGETKIND Es el tipo de archivo de destino

TARGETRATE Velocidad de muestreo del destino en múltiplos de 100ns

WINDOWSIZE Tamaño de la ventana de análisis en muestras de 100ns

USEHAMMING Valor booleano, indica si se deberá aplicar una ventana de hamming al archivo de entrada

PREEMCOEF Establece el coeficiente de pre-énfasis

NUMCHANS Numero de canales de filtros

CEPLIFTER Coeficiente de elevación cepstral

NUMCEPS Número de parámetros cepstrales

USEPOWER Valor booleano, indica si se usara la energia y no la magnitud para el analisis

LPCORDER Orden del analisis LPC

Ahora, por cada archivo de fono se deberá realizar:

```
1 HCopy -C config.plp [nombre de archivo entrada]
2 [caracteristicas plp]
```

que dará como resultado un archivo binario[2] con los valores de los coeficientes PLP, los cuales pueden ser vistos con:

```
1 HList [caracteristicas plp]
```

```
benja@benjaminPavilion-14:/media/archivos/Audio$ HCopy -C config agarra1.wav agarra1.plp
benja@benjaminPavilion-14:/media/archivos/Audio$ HList agarra1.plp
------------------------------------ Samples: 0->-1 ------------------------------------
0:      -1.235   0.152  -1.172   0.159  -2.120  -1.233  -0.731   0.021   0.289   0.276
         0.856   0.127   2.177
1:      -1.264   0.162  -1.534  -0.110  -2.190  -1.241  -0.620   0.118   0.120   0.145
         1.130   0.309   2.129
2:      -1.351   0.117  -1.384  -0.618  -2.264  -1.112  -0.277  -0.046   0.116   0.107
         0.688   0.313   2.091
3:      -1.246   0.048  -1.347  -0.410  -2.102  -1.244  -0.320   0.147   0.047   0.097
         0.254   0.071   2.132
4:      -1.465  -0.162  -1.698  -0.405  -1.917  -1.073  -0.587  -0.090   0.185   0.527
         0.533  -0.041   2.070
5:      -1.425  -0.048  -1.402  -0.126  -1.805  -0.876  -0.480  -0.082   0.629   0.575
         0.817  -0.092   2.098
6:      -1.271   0.088  -1.449  -0.237  -1.888  -0.840  -0.191   0.044   0.392   0.676
         0.812  -0.094   2.111
7:      -1.403  -0.056  -1.434  -0.113  -1.995  -0.768  -0.146  -0.083  -0.060   0.195
         0.512  -0.050   2.090
8:      -1.284   0.044  -1.267   0.406  -1.797  -1.234  -0.374   0.135   0.098   0.331
         0.504  -0.184   2.164
```

Figura 5.4: Resultado de HList a un archivo de características extraídas con HCopy, en la imagen podemos ver los coeficientes PLP del archivo.

En la figura 5.4, se muestra la salida HCopy, en ella se muestran una serie de valores numéricos correspondientes con las características PLP extraídas del audio (las cuales fueron revisadas en la seccion 2.1.1).
Para generar las entradas a la red neuronal se usarán los archivos de audio capturado, se dividirán en archivos de audio que únicamente contengan un fono y, posteriormente se tomará cada archivo con un fono y se dividirá en ventanas de 100 ms, tal división será generada de manera que dos ventanas consecutivas únicamente difieran en un milisegundo en su inicio. El motivo de que estas ventanas reusen información del fragmento anterior (el milisegundo de desfase) tiene como finalidad proporcionar información contextual para poder distinguir los trifonos en cualquier momento, ya sea que apenas comience a sonar o ya a haya terminado de ser emitido.[38]
Este valor de 100 ms por cada fono se debe a la duración promedio de los trifonos[38], el cual es visible de manera durante el etiquetado con audacity,

[2]Una alternativa a la salida binaria es posible modificando el archivo HShell.c de HTK, para imprimir números flotantes.

por lo que supuse que el hecho de que un humano pudiera reconocer el inicio de un trifono y su final de manera visual, entonces la red neuronal podría aprender más fácilmente a distinguir entre varias muestras de esa duración. Posteriormente, cada una de esas ventanas deberá ser procesada con HCopy para extraer sus características y, finalmente, crear un arreglo con todas las características PLP de los fragmentos de fono extraídos. El archivo de características deberá contener un arreglo de numpy con las dimensiones [N,338]. El 338 es un número aproximado de las salidas que se obtienen de cada uno de los fragmentos de los trifonos, en ocasiones se presentan archivos que tienen un número mayor de características pero estos son bastante raros al igual que el caso contrario (archivos con menos de 338 caracteristicas), por lo que este valor abarca la gran mayoría de los casos. En cuanto a las etiquetas, estas deberán reunirse también en un arreglo de numpy (y posteriormente en archivos .npy) de dimensiones n,num_fonos con n igual al número de muestras y una peculiaridad en cuanto a la segunda dimensión: aunque cada muestra tiene únicamente una etiqueta, ésta se debe encontrar codificada usando *one-hot* .
Para codificar en one-hot, se debe primero averiguar cuántos elementos forman al conjunto de muestras, en nuestro caso son 72 fonos, así, una etiqueta para cada fono consta de 72 dígitos binarios, donde el dígito correspondiente a la n-ésima posición deberá ser establecido en 1 para representar al n-ésimo fono.
Ejemplo 1: Codificar los números 1 al 3 en one-hot

1–>[1, 0, 0]
2–>[0, 1, 0]
3–>[0, 0, 1]

Ejemplo 2: Codificar los números 0 al 3 en one-hot

0–>[1, 0, 0, 0]
1–>[0, 1, 0, 0]
2–>[0, 0, 1, 0]
3–>[0, 0, 0, 1]

Esta codificación es sencilla de obtener usando la biblioteca keras.utils, más concretamente la función to_categorical().

5.8. Generación de estados para HMM

Tomando en cuenta que se busca implementar un modelo oculto de Márkov, además de etiquetas de los trifonos (como en la sección 5.5) deberemos etiquetar

también los *estados* del modelo, los cuales se pueden obtener con una variación de las etiquetas antes mencionadas, la mayoría de los estados corresponden a trifonos, salvo algunas excepciones: los trifonos "a" y "la". La explicación de lo anterior es sencilla: ambos sonidos (aunque son el mismo) están en momentos distintos de frases distintas, en la frase "agarra la esponja","la" aparece entre los trifonos "rra" y "esp", mientras que en "Gira a la derecha" y "Gira a la izquierda" aparece entre los "a" y "izq" o "der". Caso similar con "a", que puede aparecer en las frases "Ve a zona amarilla" y "Gira a la izquierda". El grafo con los estados etiquetados se muestra en la figura 5.5 y en ella se muestran las transiciones esperadas entre estados.

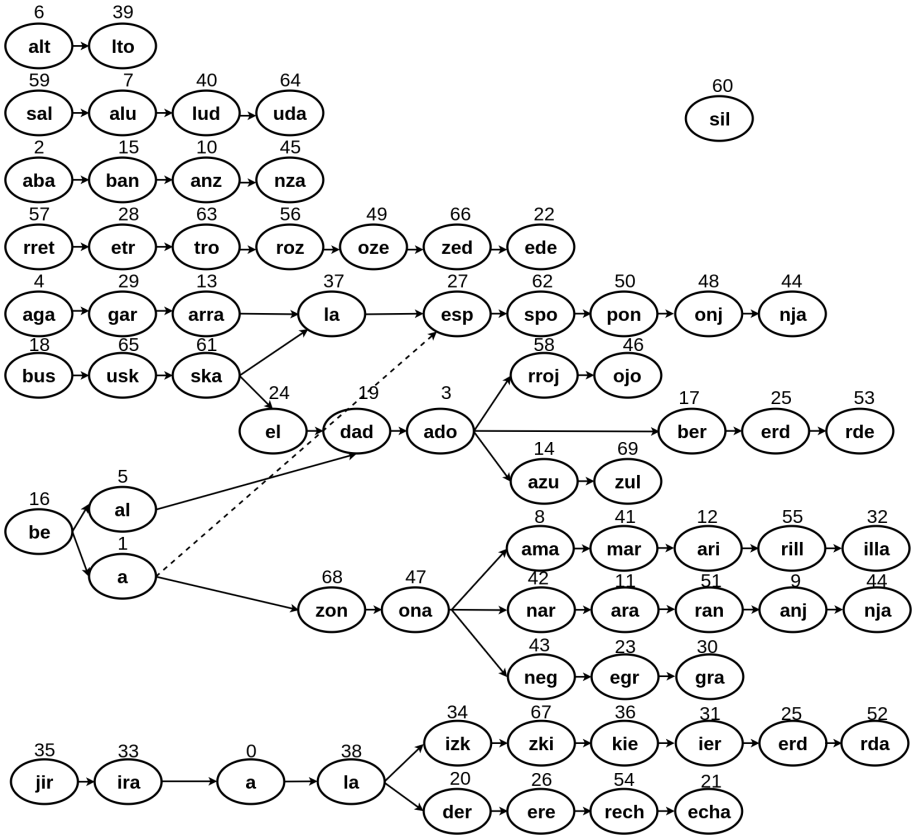


Figura 5.5: Grafo con las transiciones permitidas entre estados.(Estados numerados alfabéticamente)

Entonces: Etiquetar los estados puede ser hecho sencillamente tomando los archivos de etiquetas de trifonos y cambiando consistentemente los nombres de

etiquetas por un identificador de estado,por ejemplo, su numero en la lista de trifonos ordenada de manera alfabética o cambiar a mayúsculas la etiqueta, en el caso de los “a” y “la” asignando identificadores distintos a los usos en frases distintas.

5.9. Generación de conjuntos de entrenamiento

Finalmente, antes de comenzar con la implementación de la red neuronal para reconocer los trifonos, se deben generar 3 grupos de archivos: las ventanas de entrenamiento, de validación y prueba, todos con sus correspondientes etiquetas. Aunque para esta actividad se suele recomendar [39] una distribución de los archivos de:
70 % para conjunto de entrenamiento.
20 % para conjunto de validación.
10 % para conjunto de prueba.
Dado el conjunto de datos que tuve disponible opté por proporcionar la mayor cantidad posible de elementos al conjunto de entrenamiento, siendo aproximadamente un 51 % de datos para tal conjunto, un 32 % al de validación y el restante 16.8 % al conjunto de prueba. Este ultimo conjunto no pudo haber sido mas pequeño para poder cumplir con el tamaño usual del 10 %, pues en tal caso hubieran existido instrucciones que no serian evaluadas.
Los conjuntos deben generarse procurando que cada una de las palabras sea dicha por personas distintas en conjuntos distintos. En la siguiente imagen se muestran las instrucciones capturadas en cada renglón (figura 5.6), el ultimo digito de cada celda indica hace referencia al hablante que fue grabado en el audio.
Se asumirán los nombres entrenamiento.npy, entrenamientoEtiquetas.npy, validación.npy, validacionEtiquetas.npy, prueba.npy y pruebaEtiquetas.npy para los archivos de ventanas y etiquetas.
Asi pues, el archivo entrenamiento.npy contendrá únicamente los archivos mostrados en la figura 5.7 y serán procesados de acuerdo al diagrama de la figura 5.8.

Frase								
Agarra la esponja	agarra1	agarra41	agarra41P	agarra61	agarra31	agarra51		agarra21
Agarra el dado rojo	agarra2	agarra22	agarra32	agarra62	agarra42P	agarra52		agarra42
dado verde	agarra3	agarra43	agarra43P	agarra53	agarra23	agarra63		agarra33
azul	agarra4	agarra24	agarra64		agarra34			agarra54
Alto	alto1	alto2	alto5		alto3	alto4	alto6	alto4P
Avanza	avanza1	avanza3	avanza6		avanza2	avanza4	avanza5	avanza4P
Busca el dado azul	dadoa4	dadoa5			dadoa1	dadoa6		dadoa4P
la esponja	esponja1	esponja4P	esponja5		esponja6			esponja4
el dado rojo	dador4	dador6			dador1	dador5		dador4P
el dado verde	dadov4	dadov5			dadov4p	dadov6		dadov1
Gira a la derecha	gder2	gder4	gder4P	gder6	gder1	gder5		gder3
Gira a la izquierda	gizq1	gizq4P	gizq5		gizq3	gizq4	gizq6	gizq2
Retrocede	retrocede1	retrocede2	retrocede6		retrocede3			retrocede5
Saluda	saluda2	saluda3	saluda5		saluda4	saluda4P	saluda6	saluda1
Ve al dado azul	vedad3	vedad4	vedad4P	vedada6	vedada2	vedada5		vedada1
Ve al dado rojo	vedadr1	vedadr5	vedadr3		vedadr2			vedadr6
Ve al dado verde	vedav1	vedav2			vedav3	vedav5		vedav6
Ve a zona amarilla	veam1	veam2	veam4P	veam5	veam6	veam4		veam3
Ve a zona naranja	vena3	vena4			vena1*			vena2*
Ve a zona negra	veneg2	veneg3	veneg4	veneg6	veneg5			veneg1
Ve a la esponja	veesp2	veesp3			veesp1	veesp6		veesp5

Figura 5.6: Ejemplo de distribución de elementos: Amarillo entrenamiento, verde validación y morado prueba

Frase				
Agarra la esponja	agarra1	agarra41	agarra41P	agarra61
Agarra el dado rojo	agarra2	agarra22	agarra32	agarra62
dado verde	agarra3	agarra43	agarra43P	agarra53
azul	agarra4	agarra24	agarra64	
Alto	alto1	alto2	alto5	
Avanza	avanza1	avanza3	avanza6	
Busca el dado azul	dadoa4	dadoa5		
la esponja	esponja1	esponja4P	esponja5	
el dado rojo	dador4	dador6		
el dado verde	dadov4	dadov5		
Gira a la derecha	gder2	gder4	gder4P	gder6
Gira a la izquierda	gizq1	gizq4P	gizq5	
Retrocede	retrocede1	retrocede2	retrocede6	
Saluda	saluda2	saluda3	saluda5	
Ve al dado azul	vedad3	vedad4	vedad4P	vedada6
Ve al dado rojo	vedadr1	vedadr3	vedadr5	
Ve al dado verde	vedav1	vedav2		
Ve a zona amarilla	veam1	veam2	veam4P	veam5
Ve a zona naranja	vena3	vena4		
Ve a zona negra	veneg2	veneg3	veneg4	veneg6
Ve a la esponja	veesp2	veesp3		

Figura 5.7: Elementos que serán usados como conjunto de entrenamiento.

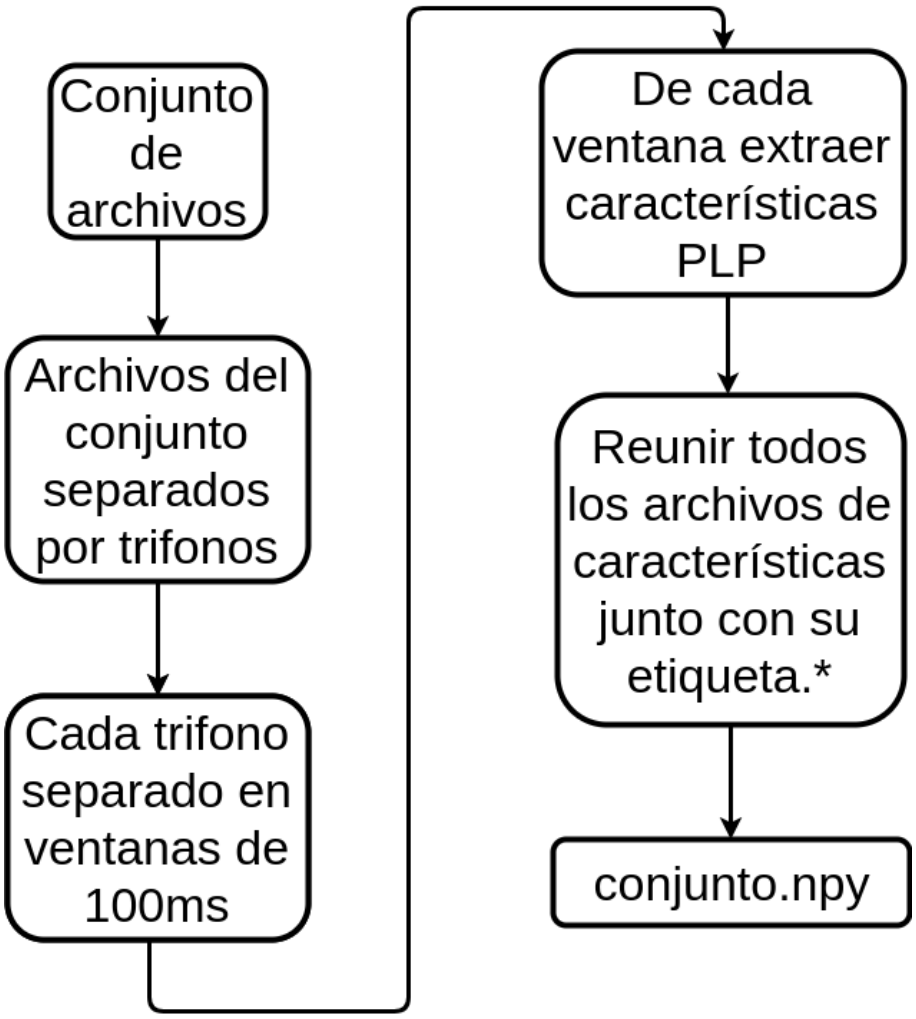


Figura 5.8: Secuencia de pasos para generar los archivos de cada conjunto.* recordar que las etiquetas deberán estar codificadas en one-hot.

Capítulo 6

Red neuronal

6.1. Estructura

Antes de introducir el modelo principal que será usado para la clasificación de los trifonos, hace falta una pequeña introducción de lo que son los *autoencoders*. Los autoencoders son redes neuronales utilizadas para el aprendizaje de codificaciones de datos de manera no supervisada (es decir: no requieren que sus datos de entrada se encuentren clasificados)[40] [41], es decir: estas redes aprenden una representación de los datos de entrada de tal manera que sea posible reconstruir el dato original a partir de ella.
Un *autoencoder* consiste en: neuronas de entrada, una serie de capas ocultas y una capa final del mismo tamaño que la primera capa. La red está entonces compuesta por dos partes:

1. La primera capa se encarga de transformar la entrada en algún código (Codificador o *Encoder*).

2. La segunda capa deberá transformar el código de la capa anterior de vuelta en la entrada que se tenia originalmente (Decodificador o *Decoder*).

En el caso simple (autoencoder) la red tiene un aspecto similar al mostrado en la figura 6.1.
El objetivo del autoencoder es aprender una función que a primera vista podría parecer trivial: la identidad. Esto significa que si la entrada de un autoencoder es un vector de datos $\overline{x}$ se espera que la salida y:

$$y(x) = x \tag{6.1}$$

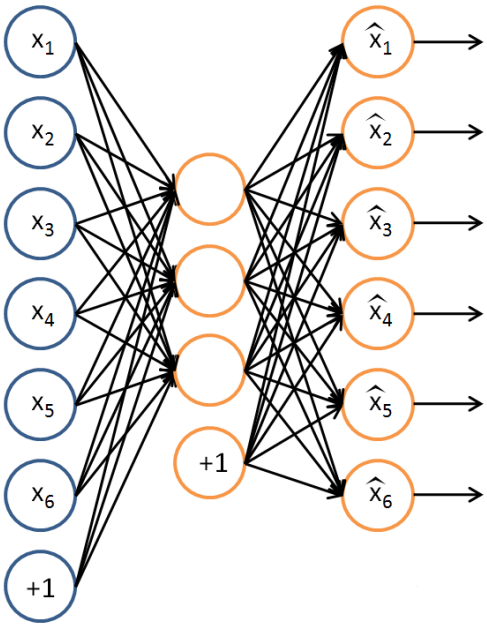


Figura 6.1: Autoencoder simple. De izquierda a derecha se muestran: neuronas de entrada, que reciben la información de la cual se busca aprender; en medio se muestra la capa correspondiente a la codificación de la información; a la derecha se muestran las neuronas que deben reconstruir la entrada.

Un "Deep autoencoder"(DA) se basa en la arquitectura del autoencoder pero, a diferencia de éste último, el DA procura hacer un procesamiento más exhaustivo de su entrada. Para convertir un autoencoder en un "deep autoencoder" hace falta colocar encoders de manera sucesiva y posteriormente agregar alguna cantidad de encoders. De la manera antes descrita lo que se esta construyendo es un encoder multicapa y un decoder multicapa con lo cual se espera que se aprendan codificaciones de mas alto nivel de la entrada.[42]
El objetivo del DA (al igual que encoder clásico) es que su salida sea igual a su entrada, para lo cual se vale de una serie de capas que encuentran patrones dentro de la señal que entra, cuando esta red neuronal logra encontrar esos patrones reconstruye la entrada original. Como efecto del entrenamiento, el DA aprende a tener cierta sensibilidad para reconstruir su entrada, así que, en vez de permitir que realice su tarea, colocamos una capa que se encargue de clasificar aquellos patrones aprendidos en alguno de los trifonos que deseamos clasificar usando una capa softmax[43] frente a los encoders y descartamos el resto de la red.
A continuación se muestra la implementación del autoencoder simple en Keras[44][1]:

```
from keras.layers import Input, Dense
from keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32  # 32 floats
# this is our input placeholder
input_data = Input(shape=(134,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_data)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_data, decoded)
encoder = Model(input_img, encoded)
As well as the decoder model:
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='SGD', loss='binary_crossentropy')
autoencoder.fit(datos_entrada,etiquetas_entrada)
```

[1]Obtenido de:https://blog.keras.io/building-autoencoders-in-keras.html

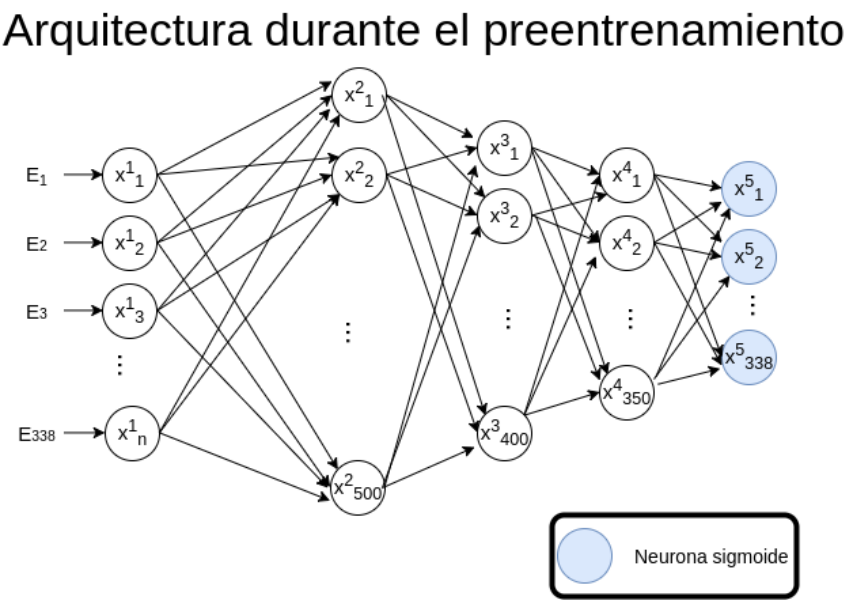
Además, se hará uso de un tipo especial de operación sobre las conexiones de la red neuronal que fue revisada en el capitulo 3:*Dropout*. Ésta operación se encarga de eliminar los valores de entrada o salida de una capa a otra de manera aleatoria. La finalidad de esta operación es forzar a que la red sea capaz de reconstruir (o identificar) los valores de entrada aunque estos se encuentren incompletos, además, permite evitar el sobreajuste a los datos al crear una variedad del conjunto de entrenamiento ligeramente diferente con cada época de entrenamiento. El entrenamiento de la red neuronal se lleva a cabo en dos etapas:

Preentrenamiento : Durante la cual se entrenara la red tratándola como un DA, es decir, que tendrá el mismo número de neuronas de entrada (338) que de salida y su objetivo será que su salida sea lo mas similar posible a su entrada.

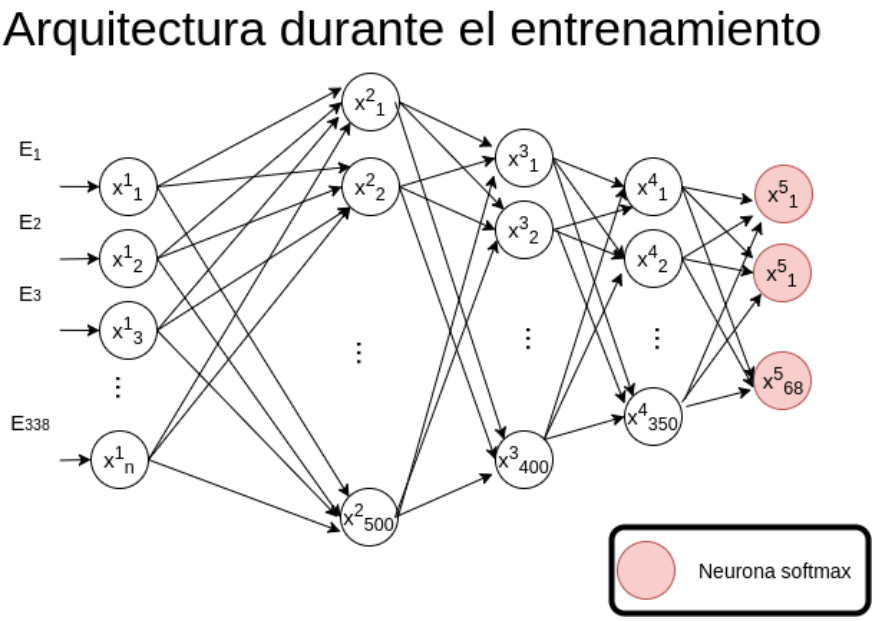
Entrenamiento : En esta etapa se reemplazara la ultima capa de la red por una capa de 68 neuronas (cada una correspondiente a un trifono) que usaran la función de activación softmax y cuyo objetivo sera clasificar correctamente cada entrada en alguna de las 68 clases de trifonos.

La red neuronal para identificar los trifonos será implementada con Keras y las características serán:

- La estructura inicial será la de un *deep autoencoder*.
- 4 capas de neuronas de tamaños: 500, 400, 350, 338 (ésta última es la reconstrucción de los datos originales durante el preentrenamiento).
- La función de activación de las neuronas de las capas 1 a la 3 será $ReLu$, mientras que las neuronas de la cuarta capa(338) que tendrán la función sigmoide.
- Entre cada capa se usara un *dropout* que conserve el 70 % de las entradas de manera aleatoria.
- Se usara el optimizador AdaDelta con los valores predeterminados de Keras.
- La función de error sera *entropía cruzada*.
- Durante la etapa de entrenamiento se reemplaza la capa de 338 neuronas por una de 68 con la función de activación $Softmax$



(a) Arquitectura de la red neuronal durante la fase de DA, en color azul se muestra la capa de salida, la cual corresponderá a la reconstrucción de los elementos de la entrada. (No se muestran las capas *Dropout* para conservar la simplicidad del diagrama.)



(b) Arquitectura de la red neuronal que sera usada como clasificador, en color rojo se muestra la capa que reemplaza a la azul de la imagen anterior, esta capa utiliza la función de activación softmax. (No se muestran las capas *Dropout* para conservar la simplicidad del diagrama.)

Figura 6.2: Arquitectura de la red en las dos etapas.

La cantidad de neuronas en cada capa es decreciente debido a que se espera que la representación de cada capa sea de mas "alto nivel" que la se su predecesora. La excepción a lo anterior únicamente es la capa con 500 elementos los cuales tienen como finalidad encontrar primero una representación mas amplia sobre la cual aprender las siguientes características.
Gráficamente, la red neuronal tiene una arquitectura similar a la mostrada en la figura 6.2a durante el preentrenamiento de la red y una arquitectura como la mostrada en la figura 6.2b cuando se esta entrenando para realizar clasificación de trifonos .
Durante ambos procesos (preentrenamiento y entrenamiento) es irrelevante el orden en que tomemos los trifonos de las frases, pues únicamente estamos enseñando a la red neuronal a reconocerlos, no a establecer una relación de orden sobre ellos, lo único que se debe procurar es que los ejemplares de entrenamiento en un lote no correspondan a una misma clase para evitar el sobreajuste de la red únicamente a tal clase.

6.1.1. Preentrenamiento

El preentrenamiento corresponde a la creación de un *Deep autoencoder* con las capas antes mencionadas, este modelo únicamente tiene como entrada las características PLP (de una ventana de 100ms de audio) y deberá ser entrenado para reconstruir su entrada, es decir: recibirá un vector de 338 valores, mismos que deberá reconstruir después de pasar por todas las capas intermedias. Lo anterior se logra usando AdaDelta (que es una variante de descenso por el gradiente) y proporcionando los mismos vectores PLP como valores esperados. Este proceso se realiza durante 30 epocas en lotes de 2048 elementos, donde cada elemento corresponde a una ventana de características PLP de 100ms (recordar diagrama 5.8).
Mediante este preentrenamiento se busca ajustar todos los pesos de las conexiones de la red a valores que proporcionen cierta sensibilidad a la presencia de trifonos (de ahí que al terminar el preentrenamiento sea posible recuperar cierta parte de las entradas).

6.1.2. Entrenamiento

Para el entrenamiento deberá eliminarse la última capa de la red (la que reconstruye el vector de características PLP) y en su lugar se deberá colocar una capa softmax de 68 elementos (correspondientes a los 68 trifonos que serán identifica-

dos).
Durante esta fase se ocupara AdaDelta como algoritmo de entrenamiento durante 30 épocas y durante cada una de ellas se proporcionaran lotes de 2048 elementos del conjunto de entrenamiento. En esta fase se requiere que cada una de las muestras este clasificada y que ese dato sea proporcionado durante el entrenamiento en formato one-hot.

6.2. Resultados

La red neuronal y los parámetros antes descritos fueron fruto de múltiples pruebas en redes que aumentaban o disminuían el número de capas o neuronas en cada capa. Estas variantes pueden ser consultadas en el apéndice B.
Para los parámetros descritos los resultados obtenidos son, de acuerdo a cada etapa del proceso de entrenamiento, los que se explican en las secciones siguientes.

6.2.1. Entrenamiento del deep autoencoder

Inicialmente se entrena el DA procurando que la salida sea similar a su entrada, es decir, que en este momento aún no se realiza una predicción de trifonos en el audio, sino que únicamente se están extrayendo características de la entrada, lo cual lleva a los pesos de la red a tener datos útiles para la identificación de cada uno de los sonidos que son introducidos a la red.
Sobre los datos de entrenamiento, el DA llega a alcanzar hasta un 12 % de precisión (figura 6.3) y un error de 0.0002 % en 30 epocas (figura 6.4). Es importante notar que, aunque el error en determinado momento llega a descender muy lentamente, este nunca se incrementa, lo cual indica que la red aun es capaz de aprender datos nuevos si es que el conjunto de entrenamiento incrementa su tamaño.
Al revisar el conjunto de validación obtenemos resultados que, aunque muestran una tendencia a mejorar su calidad, son notablemente menores que sobre los datos de entrenamiento
En la figura 6.5 es visible que la precisión del las predicciones en datos nunca antes vistos se va incrementando con el número de epocas de entrenamiento, lo cual es indicador de que la red está aprendiendo a generalizar correctamente la manera de reconstruir los datos que se le proporcionan. De igual manera en la figura 6.6 se puede observar que el error continúa disminuyendo a lo largo del tiempo. La precisión máxima alcanzada en estos datos es de un 11 % de aciertos.
En cuanto al error es evidente que el comportamiento tiende a imitar al error en

Figura 6.3: Precisión de la reconstrucción a lo largo de las epocas de entrenamiento. Eje X: número de épocas, eje Precisión de la predicción.

Figura 6.4: Error de la reconstrucción a lo largo de las epocas de entrenamiento. Eje X: número de épocas, eje Perdida de reconstrucción.

el conjunto de entrenamiento.

6.2.2. DA + softmax

Como fue explicado anteriormente, una vez realizado el entrenamiento del DA, la última capa es retirada y en su lugar se coloca una capa softmax de 68 neuronas, lo cual da lugar a una red que ya puede clasificar, en base a las características aprendidas por el encoder, el trifono presente en cada una de las muestras de entrenamiento.
Al observar el comportamiento de la red aprendiendo trifonos en la figura 6.7 nos damos cuenta que, aunque en la etapa de entrenamiento anterior únicamente fue posible reconstruir el 11 % de la entrada, eso fue suficiente para que, en 10

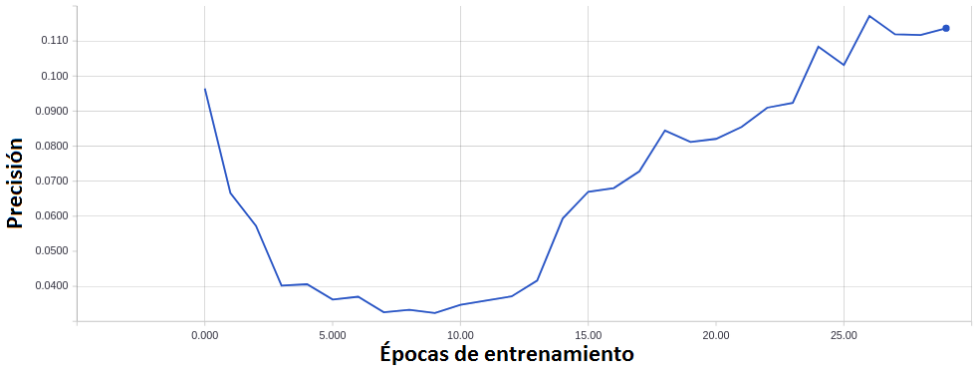
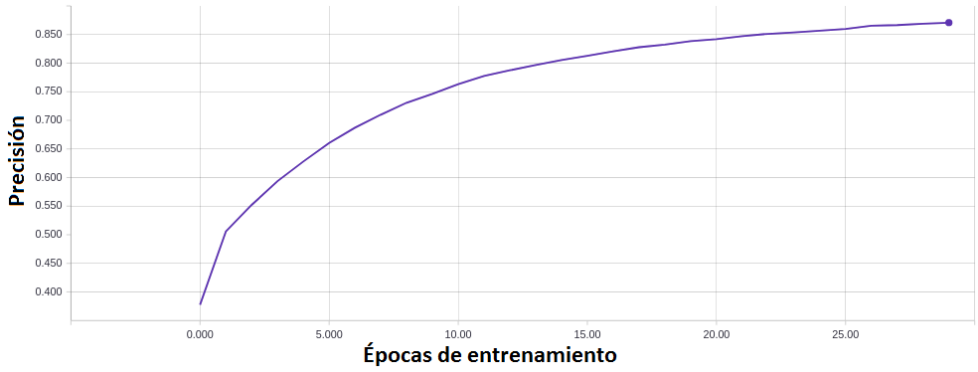


Figura 6.5: Precisión de la reconstrucción a lo largo de las epocas de entrenamiento medida en el conjunto de validación. Eje X: número de épocas, eje Precisión de la predicción.

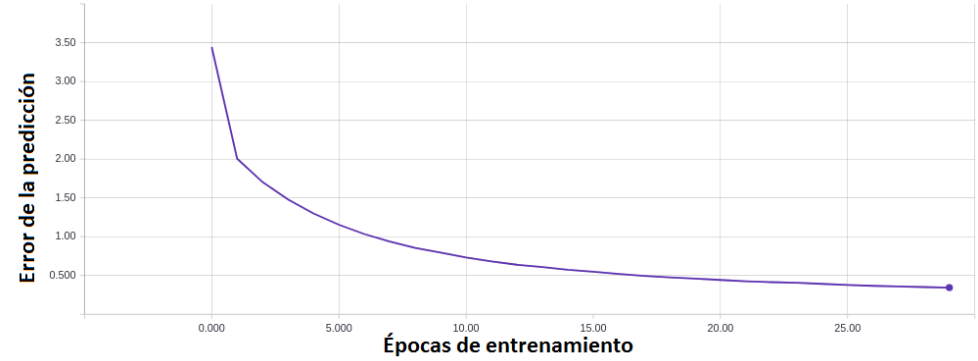
Figura 6.6: Error de la reconstrucción a lo largo de las epocas de entrenamiento medida en el conjunto de validación. Eje X: número de épocas, eje Perdida de reconstrucción.

iteraciones, ésta fuera capaz de alcanzar un 75 % de precisión en el conjunto de entrenamiento y, al llegar al total de 30 iteraciones un 85 %. El error, graficado en la figura 6.8, al igual que en el caso del entrenamiento del DA, nos indica que el hecho de aumentar el tamaño del conjunto de entrenamiento podría llevar a un incremento del aprendizaje de esta red,pues continua disminuyendo con el paso del tiempo.
Finalmente, en los datos de validación:
Podemos ver en la figura 6.9 que la precisión del modelo fue en aumento a lo largo del tiempo, alcanzando 63 % de aciertos, mientras que en la figura 6.10



netAccEnt

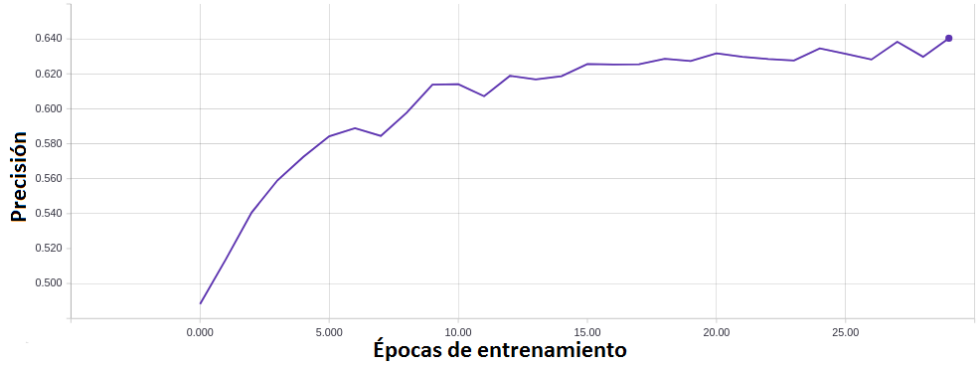
Figura 6.7: Precisión de la predicción de trifonos a lo largo de las epocas de entrenamiento medida en el conjunto de entrenamiento. Eje X: número de épocas, eje Y Precisión de la predicción.



netLossEnt

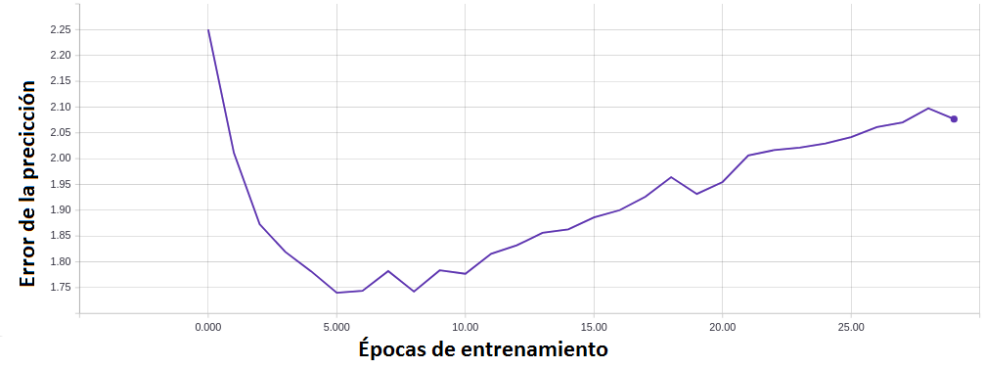
Figura 6.8: Error de la predicción a lo largo de las epocas de entrenamiento medida en el conjunto de entrenamiento. Eje X: número de épocas, eje Y Perdida de reconstrucción.

vemos el error de reconstrucción también en el conjunto de validación. En esta ultima figura podemos notar fácilmente que a partir de la séptima época de entrenamiento el error de reconstrucción comenzó a incrementar, lo cual indica un sobreajuste de la red y por consiguiente una menor capacidad de generalización sobre datos nuevos. Es posible que incrementando el tamaño del conjunto de entrenamiento sea posible realizar el entrenamiento durante un periodo más largo de tiempo sin que este fenómeno ocurra y permitiendo que las predicciones de la red neuronal mejoren.



netAccVal

Figura 6.9: Precisión de la predicción de trifonos a lo largo de las epocas de entrenamiento medida en el conjunto de validación. Eje X: número de épocas, eje Y Precisión de la predicción.



netLossVal

Figura 6.10: Error de la predicción a lo largo de las epocas de entrenamiento medida en el conjunto de validación. Eje X: número de épocas, eje Y Perdida de reconstrucción.

6.2.3. Mejora de la calidad de la predicción

El *Deep autoencoder* expuesto anteriormente alcanza una precisión del 63 % usando todos los datos de entrenamiento disponibles, por lo cual es necesario que en el futuro se extienda este conjunto de datos para mejorar la calidad de la predicción. Como justificación de que la afirmación anterior es válida, fueron realizadas pruebas con el mismo modelo que únicamente cambian la cantidad de datos de entrenamiento que fueron proporcionadas como entrada. Los valores elegidos fueron: 60, 70 y 80 % de datos.
A continuación se muestran las gráficas correspondientes al entrenamiento con

los porcentajes de datos propuestos, además del contraste con la red neuronal que tuvo acceso al 100 % de datos.

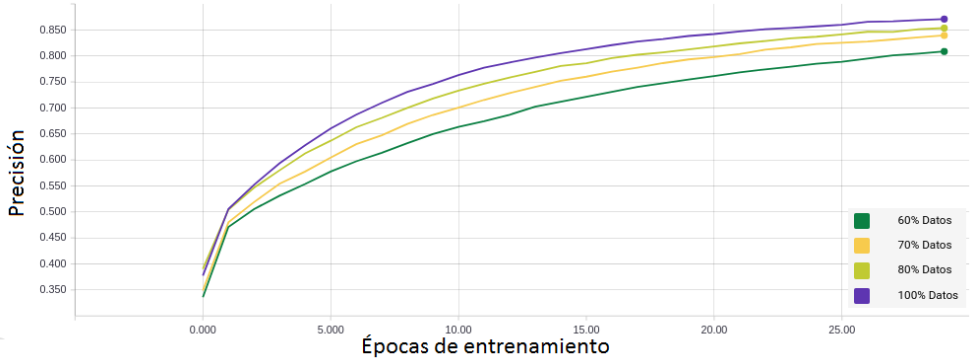


Figura 6.11: Comparativa de la precisión de los modelos variando el tamaño del conjunto de entrenamiento (Conjunto de entrenamiento). Eje X número de épocas y eje precisión

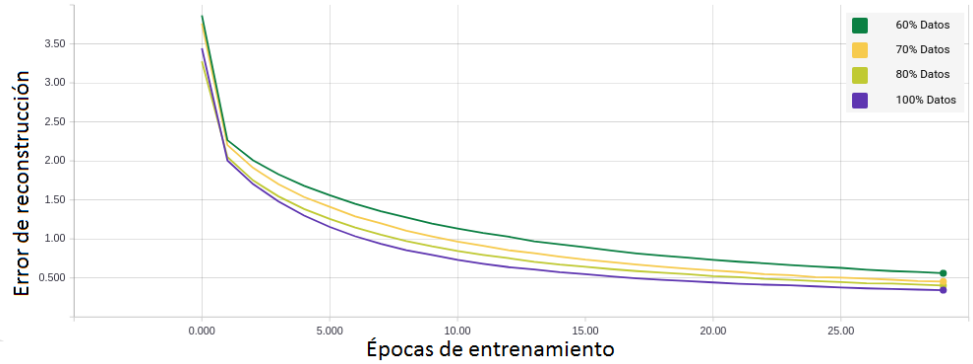


Figura 6.12: Error de reconstrucción de los modelos variando el tamaño del conjunto de entrenamiento (Conjunto de entrenamiento). Eje X número de épocas y eje Y error de reconstrucción

Es posible ver en las figuras 6.11 y 6.12 que el hecho de proporcionar el 60 % de los datos de entrenamiento provocó que la red acertara únicamente el 80 % de las ocasiones a identificar los trifonos, así como el sucesivo "incremento" de los datos provoco un incremento constante en el rendimiento de la red.
Al observar los resultados de la precisión (figura 6.13) en el conjunto de validación (cuyo tamaño no vario para estas pruebas) se obtienen resultados consistentes con lo anterior: a mayor cantidad de datos durante la fase de entrenamiento mayor precisión de la precisión.

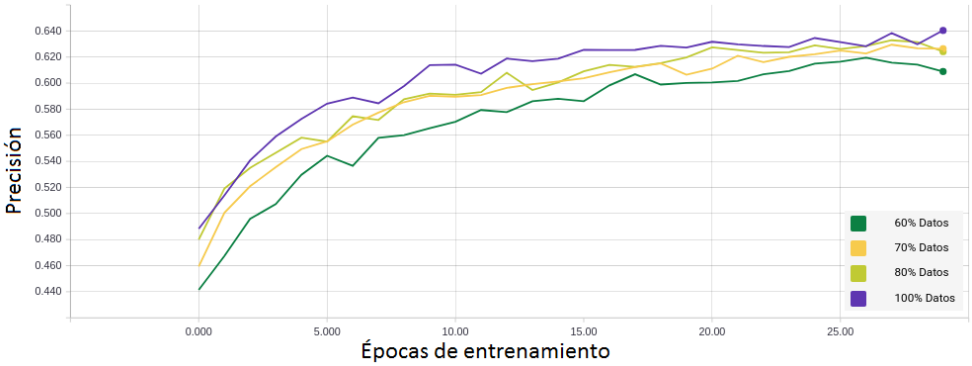


Figura 6.13: Comparativa de la precisión de los modelos variando el tamaño del conjunto de entrenamiento (Conjunto de validación). Eje X número de épocas y eje precisión

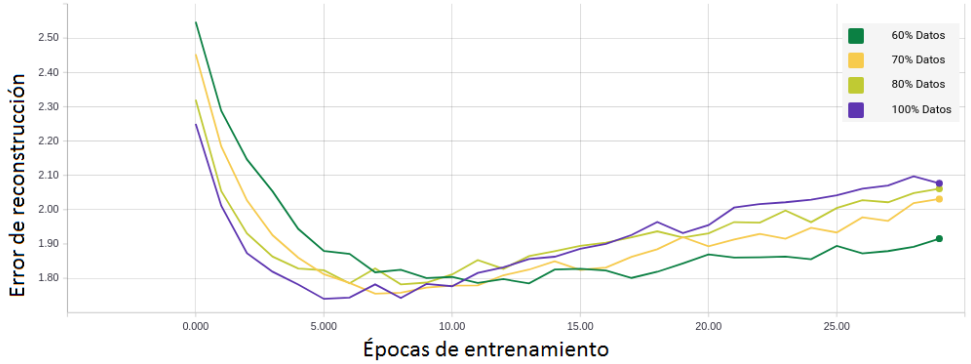


Figura 6.14: Error de reconstrucción de los modelos variando el tamaño del conjunto de entrenamiento (Conjunto de validación). Eje X número de épocas y eje Y error de reconstrucción

Por lo cual se puede concluir que un incremento en el número de ejemplares de entrenamiento, en este caso, llevaría a una mejora de de la capacidad de clasificación de la red neuronal.

Capítulo 7

Predicción de frases

7.1. Modelo oculto de Márkov

Esta sección mostrará brevemente el uso de los modelos ocultos de Márkov (HMM) para realizar la reconstrucción de los trifonos en frases, sin embargo este método no resultó ser efectivo debido, esencialmente, a que la red neuronal aún no alcanza un rendimiento lo suficientemente alto como para clasificar correctamente todos los trifonos del audio. Este capitulo tiene la finalidad de servir de guía para la implementación de los HMM debido a que es la manera mas eficiente que se conoce para realizar la reconstrucción de las frases y permite reconocer un conjunto amplio de ellas (a diferencia de la técnica usada en la sección "Reconocimiento de frases"), esto para que en un futuro que sea posible reunir una suficiente cantidad de datos y logre mejorar la eficiencia de la red neuronal descrita en el capitulo anterior se pueda implementar esta técnica con mejores resultados.
En esencia, para el reconocimiento de voz, una cadena de Márkov podría ayudar a modelar cómo es que los distintos trifonos aparecen en una secuencia de audio, dependiendo únicamente del fono que haya sido reconocido en la posición anterior de la secuencia.[33]
Para poder entrenar una cadena oculta de Márkov debemos:

1. Generar el primer conjunto(A) que corresponde a las secuencias de estados que identifican el trifono de cada ventana del conjunto de entrenamiento.

2. Crear el segundo conjunto(observaciones) que corresponde a las etiquetas asignadas por la red neuronal a partir del conjunto de entrenamiento.

Cabe resaltar que para generar el conjunto A de estados, únicamente se requiere procesar las etiquetas del archivo de audio para generar un arreglo que tenga, a saltos de 1ms y en ventanas de 100ms el trifono que se encuentra presente en tal fragmento, es decir, no se requieren en ningún momento las características PLP o el audio original para tal proceso, este conjunto contiene *estados* de la cadena oculta de Márkov y no corresponde exactamente con los trifonos capturados [1] .
Para el conjunto de observaciones: cada ventana del audio es procesada (según su orden temporal) por la red neuronal y se obtiene el resultado, el cual deberá ser transformado en el identificador del trifono mas probable, en este caso, las observaciones individuales corresponden unívocamente con alguno de los 68 trifonos.
Proponer un ejemplo concreto del etiquetado de un audio real seria un despropósito, pues el numero de etiquetas de un archivo sencillo con una sola palabra como "saluda" tiene alrededor de 1700 etiquetas que poco aportarían a la comprensión de la forma de etiquetar. Sin embargo, si se omiten los tiempos que deberían abarcar las etiquetas y únicamente consideramos un numero reducido de ellas podemos llegar al siguiente ejemplo de etiquetado de la muestra de audio que contiene la frase"Agarra el dado azul".

Agarra el dado
azul=[aga,aga,aga,garr,garr,arra,el,el,dad,dad,dad,ado,ado,azu,azu,azu,zul,zul]

Mientras que la clasificación de las muestras por red neuronal podría ser:

observaciones=[aga,ve,aga,aga,garr,garr,garr,arra,arra,el,ado,ado,dad,ado,ado]

En el ejemplo hay que notar varias cosas:

1. Existen repeticiones de trifonos debido a que cada trifono permanece sonando un periodo de tiempo.

2. Existen elementos que no son estrictamente trifonos (como el caso de "el"), los cuales considere un caso especial para facilitar la reconstrucción de las frases.

Una vez que se tengan todas las observaciones de las frases y sus correspondientes estados, es posible implementar una cadena de Márkov oculta usando Seqlearn en un script en Python como el siguiente:

[1] Recordar explicación de la sección 5.8

```
from seqlearn.hmm import MultinomialHMM

#procesar cada archivo con la red neuronal y concatenar
#los resultados en un único archivo
#donde cada renglón sea la secuencia de observaciones
#de un archivo
observaciones=np.load('observaciones.npy')
A=np.load('etiquetasCorrectas.npy')

#observaciones obtiene la longitud de cada secuencia observada
#y devuelve una lista
longitudes=obtenerLongitudesObservaciones(observaciones)

HMM=MultinomialHMM()
HMM.fit(observaciones,A,longitudes)
```

Seqlearn se encargará de ejecutar el algoritmo de Baum-Welch usando la secuencia de estados A y la secuencia de trifonos obtenidos por la red neuronal para crear un modelo oculto de Márkov capaz de reconstruir la secuencia correcta de estados presentes en el audio para el cual no tengamos la "etiqueta correcta", esto aunque no todas las predicciones realizadas por la red sean correctas.Cabe destacar que este algoritmo trata de crear un modelo de Márkov que maximice las probabilidades de las observaciones dadas, por lo cual es necesario proporcionar la mayor cantidad de muestras posibles durante el entrenamiento.
Para reconstruir la frase dicha de una grabación nueva, bastará con llamar al método predict() con la secuencia de observaciones, con lo cual se ejecutara el algoritmo de Viterbi, que dará como resultado una lista de fonos, que debe corresponder aproximadamente a la concatenación de los fonos de alguna de las frases que fue entrenada para reconocer.

```
resultado=HMM.predict(nuevasObservaciones)
```

Dado el etiquetado y la forma de procesar el audio de cada archivo de audio tendremos como resultado un arreglo de varios miles de etiquetas, debido a que es frecuente que se realicen transiciones de un estado a si mismo, por lo que es recomendable implementar una función que se encargue de eliminar estas redundancias y deje únicamente las etiquetas que se encuentran de manera única en un tramo de la predicción, como resultado de esta eliminación se debería obtener un resultado similar a 7.1 donde se muestran las frases reconstruidas a partir de los trifonos clasificados por la red neuronal en el siguiente formato: Después del simbolo = aparece la frase que fue dicha en el audio y en los renglones subsecuentes aparecen las secuencias reconstruidas (es decir, alguna de las frases

definidas en 5.2) que mas se ajustan a la observación. El resto de los resultados pueden ser consultados en C

```
Frase:simpleEntrenamiento/tmpagarra41P.wav.npy =AGARRA LA ESPONJA
('Viterbi:', array(['sil', 'nja', 'aga', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra',
       'la', 'esp', 'spo', 'pon', 'onj', 'nja', 'gra', 'sil'],
      dtype='|S4'))
**************************
Frase:simpleEntrenamiento/tmpagarra41.wav.npy =AGARRA LA ESPONJA
('Viterbi:', array(['sil', 'nja', 'garr', 'aga', 'garr', 'arra', 'la', 'esp', 'la',
       'esp', 'spo', 'pon', 'onj', 'nja', 'sil', 'gra', 'sil', 'gra', 'sil'],
      dtype='|S4'))
**************************
Frase:simpleEntrenamiento/tmpagarra43P.wav.npy =AGARRA EL DADO VERDE
('Viterbi:', array(['sil', 'nja', 'aga', 'anj', 'aga', 'garr', 'aga', 'garr', 'arra',
       'garr', 'la', 'ran', 'aga', 'ran', 'aga', 'anj', 'el', 'dad', 'ado',
       'pon', 'ber', 'erd', 'rde', 'sil'],
      dtype='|S4'))
**************************
Frase:simpleEntrenamiento/tmpagarra43.wav.npy =AGARRA EL DADO VERDE
('Viterbi:', array(['sil', 'nja', 'anj', 'aga', 'anj', 'aga', 'garr', 'arra', 'be',
       'ere', 'gra', 'ere', 'be', 'a', 'dad', 'ado', 'spo', 'ber', 'rde',
       'erd', 'rde', 'gra', 'sil'],
      dtype='|S4'))
**************************
Frase:simpleEntrenamiento/tmpagarra53.wav.npy =AGARRA EL DADO VERDE
('Viterbi:', array(['sil', 'aga', 'sil', 'aga', 'ado', 'aga', 'garr', 'aga', 'garr',
       'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr',
       'garr', 'aga', 'garr', 'arra', 'garr', 'arra', 'ska', 'alt', 'a',
       'alt', 'a', 'el', 'a', 'el', 'alu', 'uda', 'alu', 'uda', 'dad',
       'ado', 'dad', 'ado', 'rroj', 'ber', 'erd', 'rde', 'sil'],
      dtype='|S4'))
**************************
Frase:simpleEntrenamiento/tmpagarra61.wav.npy =AGARRA LA ESPONJA
('Viterbi:', array(['aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga',
       'sal', 'aga', 'sal', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr',
       'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'arra', 'aga',
       'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr',
       'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr',
       'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'aga',
       'arra', 'aga', 'la', 'esp', 'spo', 'pon', 'onj', 'nja', 'arra',
       'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska',
       'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'sil'],
      dtype='|S4'))
**************************
Frase:simpleEntrenamiento/tmpagarra62.wav.npy =AGARRA EL DADO ROJO
('Viterbi:', array(['sil', 'rech', 'echa', 'rech', 'echa', 'rech', 'sil', 'nja', 'aga',
       'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr',
       'aga', 'garr', 'aga', 'garr', 'arra', 'sil', 'el', 'be', 'kie',
       'zed', 'kie', 'be', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'aba',
       'ado', 'aba', 'ado', 'rroj', 'ado', 'rroj', 'ojo', 'ska', 'sil'],
      dtype='|S4'))
```

Figura 7.1: Algunos resultados de la predicción usando el modelo oculto de Márkov. Después del símbolo = aparece la frase que fue dicha en el archivo procesado y en la siguiente linea la secuencia mas probable de trifonos a partir de las observaciones.

En la primera frase podemos ver cómo es que el comando “agarra la esponja” se encuentra como subsecuencia de la traducción obtenida, es decir: se pueden eliminar algunos trifonos para que al final se encuentren únicamente los que componen a la frase correcta.

7.2. Reconocimiento de frases

Debido al alcance de este trabajo y a la cantidad de frases a reconocer (que, en la practica es bastante pequeña) esta tarea puede reducirse a una especie de comparación de listas, concretamente a la contención de una serie de trifonos presentes en la instrucción con los que se encuentran presentes en el audio procesado.
Propongo el siguiente algoritmo:

1. Obtener, de cada frase a reconocer (listadas en 5.2) la lista de los trifonos que lo componen, por ejemplo de la frase "Ve a zona amarilla" obtendríamos: [be, a, zon, ona, ama, mar, rill, illa].

2. Luego, para cada trifono en una frase a clasificar, buscar dentro del audio procesado el primer trifono y guardar si fue encontrado o no, en caso de no ser encontrado devolver -1, en caso de encontrarlo, pasar al siguiente trifono de la frase, solo que, en esta ocasión únicamente buscaremos desde la posición donde fue encontrado el trifono anterior.

3. Repetir el paso anterior hasta que se haya terminado de consumir la lista de trifonos en la frase o en el audio procesado.

4. Una vez terminado el proceso, obtener el porcentaje de trifonos de la frase presentes en el audio procesado. Este porcentaje indicará cuánta certeza existe de que la frase evaluada sea la dicha en la grabación.

5. Pasar a la siguiente frase

Una vez recorridas todas las frases se podrá obtener fácilmente cuál es la que tiene mayor posibilidad de haber sido reconocida.
Es importante resaltar que la búsqueda de cada trifono de la frase continúa justo después de donde fue encontrado el anterior y por tanto, únicamente se recorre tanto la grabación como la frase una sola vez, además esta forma de evaluación permite saber si los trifonos fueron encontrados en el orden correcto, lo cual aun es necesario debido al rendimiento de la red neuronal, la cual podría estar prediciendo trifonos incorrectos en posiciones incorrectas.
Aunque este proceso requiere revisar todas las frases es posible reducir el tiempo de ejecución con distintas técnicas que podría implementar el alumno:

1. Se podría verificar, en paralelo, cada una de las frases con el algoritmo anterior, lo cual reduciría el tiempo de ejecución de la verificación.

2. Es posible que, en vez de una lista de trifonos para cada frase, se cree un árbol de trifonos común para toda la comprobación, con lo cual es posible verificar frases que comiencen con las mismas palabras al mismo tiempo, lo cual reduciría la complejidad de la comprobación.

3. Detener el algoritmo cuando se encuentre una predicción de 90 % de puntuación o mas.

Finalmente, este método toma en cuenta varios factores esenciales para poder decidir si una secuencia corresponde a una frase o no:

- Únicamente evalúa de manera positiva las secuencias que tiene los trifonos en el orden correcto, limitando así el problema de la identificación falsa de la red neuronal.
- Da prioridad a secuencias que contengan todos los trifonos de una frase dada.
- Da cierta habilidad de tolerancia al reconocimiento en caso de que los trifonos no sean reconocidos

A raíz de la implementación del algoritmo sugerido, se obtuvieron los siguientes resultados:

- En 57 grabaciones del conjunto de entrenamientos:
  - 52 fueron reconocidas exitosamente con la frase de mayor puntuación,
  - 5 exitosamente reconocidas en su segunda alternativa.
- En las 40 grabaciones del conjunto de validación:
  - 13 fueron clasificadas correctamente en su primer opción
  - 6 fueron clasificadas correctamente en la segunda opción
  - 6 más fueron clasificadas en la tercera opción
  - 15 no fueron clasificadas en la tercera opción y fueron consideradas error.

El resultado en extenso se puede consultar en el apéndice E.
Para poder apreciar la calidad de la clasificación lograda por este algoritmo es conveniente introducir el concepto de matriz de confusión y dos medidas asociadas a esta: la precisión y la exhaustividad (del inglés: precisión and recall).

VP=52	FP=5
FN=5	

Precisión=0.9122
Exhaustividad=0.9122

Tabla 7.1: Matriz confusión condensada(Entrenamiento).VP= verdaderos positivos,FN=Falsos negativos y FP=Falsos positivos

q

"Una matriz de confusión resume el desempeño de un clasificador con respecto a algún conjunto de datos. Es una matriz bidimensional, indexada en una dimensión por las clases verdaderas de un objeto y en la otra por las clases que el clasificador asigna."[45]

La precisión y la exhaustividad son medidas usadas en el campo de recuperación de la información para medir que tan bien un sistema recupera los documentos relevantes solicitados por un usuario. Las medidas se definen de la siguiente manera: [46]

Precisión :

$$\frac{\text{Número total de documentos devueltos que son relevantes}}{\text{numero de documentos devueltos}} \tag{7.1}$$

Exhaustividad :

$$\frac{\text{Número de documentos devueltos que son relevantes}}{\text{el número de documentos en la base de datos}} \tag{7.2}$$

Es decir, que el dato de precisión nos indica cuantas veces el sistema esta logrando clasificar correctamente un ejemplar, mientras que el valor de exhaustividad nos indica, dada una clase concreta "X" cuantas veces fue correctamente clasificada (es decir,el énfasis se encuentra esta vez en la clase completa, no en un ejemplar particular).[47]
De los datos antes mencionados se calcularon las matrices de confusión en ambos conjuntos:
En la matriz de confusión del conjunto de entrenamiento 7.2 es apreciable que, la mayoría de los elementos fue clasificado en la dia lo cual indica que los ejemplares de las clases prácticamente siempre fueron clasificadas en su clase real.La precisión y exhaustividad del modelo (tabla 7.1) fueron de 0.9122.
En la matriz del conjunto de validación (figura 7.3) se aprecia que la clasificación asignada por este sistema comete errores al clasificar las frases con datos nunca

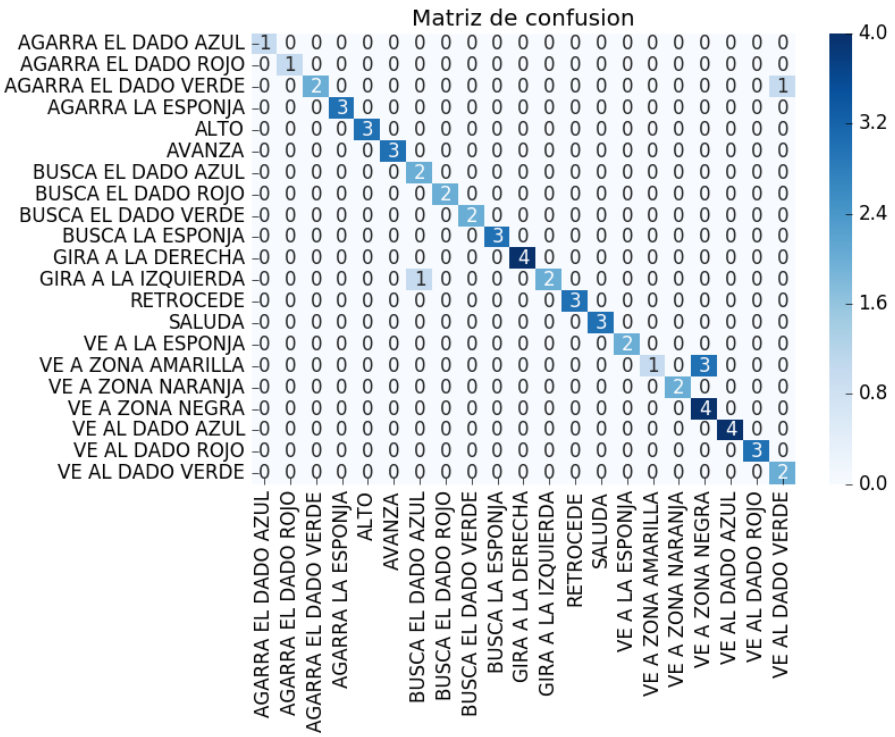


Figura 7.2: Matriz de confusión en el conjunto de entrenamiento, de manera horizontal se muestra la frase predicha y de manera vertical la frase real.

antes vistos. La precisión y exhaustividad del modelo (tabla 7.2) fueron de 0.35. Finalmente, para el caso del conjunto de prueba, fueron realizadas capturas de audio adicionales para poder evaluar de manera objetiva al c,debido a que en etapas anteriores de estos experimentos este conjunto únicamente contaba con un ejemplar de cada frase,lo cual impedía valorar si el modelo se comportaba como era esperado (pues un error significaba un error del 100 %). Estos datos adicionales no fueron etiquetados por trifonos y solo se utilizaron durante esta ultima etapa del experimento.

A continuación se muestran la matriz de confusión en la figura 7.4 y la correspondiente tabla condensada en 7.3, de la cual se desprende que la precisión y exhaustividad tienen un valor de 0.29.

También es conveniente para el análisis de estos datos calcular los valores antes mencionados para cada clase, los cuales se muestran en la tabla 7.4.

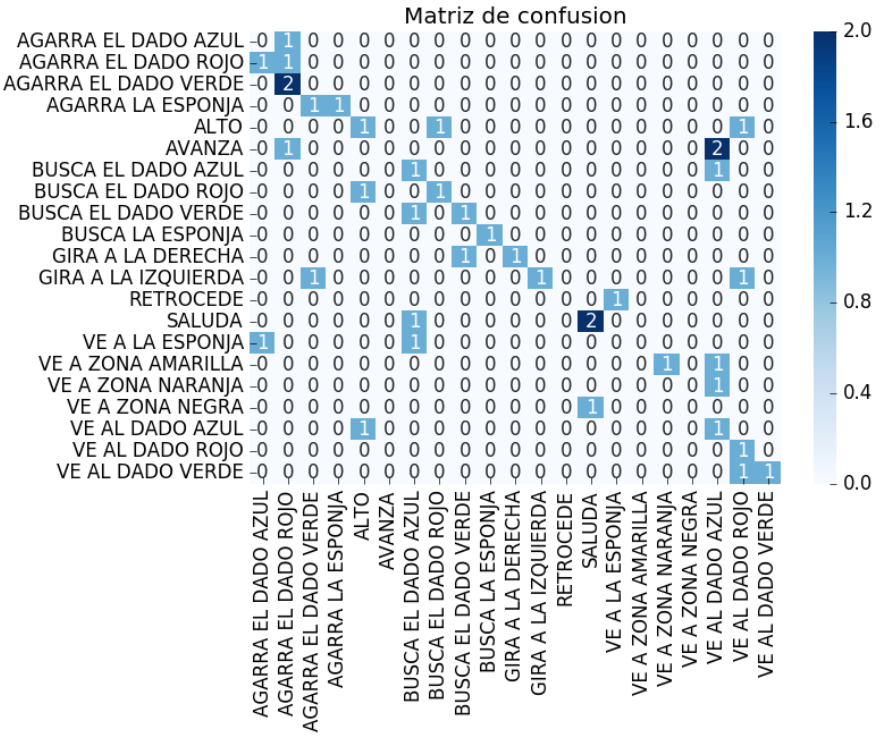


Figura 7.3: Matriz de confusión en el conjunto de validación, de manera horizontal se muestra la frase predicha y de manera vertical la frase real.

VP=14	FP=26
FN=26	

Precisión=0.35
Exhaustividad=0.35

Tabla 7.2: Matriz confusión condensada(Validación). VP verdaderos positivos, FN Falsos negativos y FP Falsos positivos

VP=31	FP=74
FN=74	

Precisión=0.2952
Exhaustividad=0.2952

Tabla 7.3: Matriz confusión condensada(Prueba). VP= verdaderos positivos, FN=Falsos negativos y FP=Falsos positivos

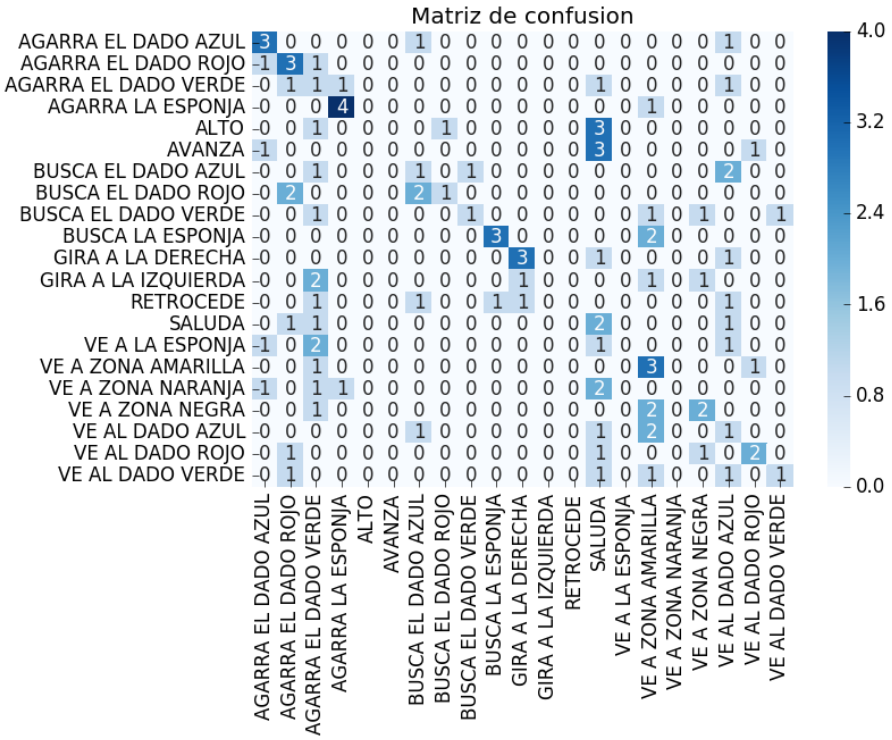


Figura 7.4: Matriz de confusión en el conjunto de prueba, de manera horizontal se muestra la frase predicha y de manera vertical la frase real.

Tabla 7.4: Precisión y exhaustividad del algoritmo propuesto

Frase	Precisión (Entrenamiento)	Exhaustividad (Entrenamiento)	Precisión (Validación)	Exhaustividad (Validación)	Precisión (Prueba)	Exhaustividad (Prueba)
AGARRA EL DADO AZUL	1	1	0.3333	0	0	-
AGARRA EL DADO ROJO	0.6666	1	0.3333	0.33	0	-
AGARRA EL DADO VERDE	1	1	0	1	0.6	0.4285
AGARRA LA ESPONJA	1	1	0.5	1	0.8	0.6666
ALTO	1	1	0	0	0	-
AVANZA	1	1	0.6666	-	0.4	0.125
BUSCA EL DADO AZUL	1	1	1	0.28	0.6	0.75
BUSCA EL DADO ROJO	1	1	0.5	0.5	0.6	0.3333
BUSCA EL DADO VERDE	1	1	0.5	1	0.2	0.1
BUSCA LA ESPONJA	0.6666	1	0	1	0.2	0.0714
GIRA A LA DERECHA	1	1	0	1	0	-
GIRA A LA IZQUIERDA	1	0.6666	0.5	1	0.2	0.5
RETROCEDE	1	1	0	-	0	-
SALUDA	1	0.6666	0.5	0.5	0.2	0.1666
VE A LA ESPONJA	1	1	0.5	1	0.2	0.5
VE A ZONA AMARILLA	1	1	0.5	-	0.6	0.6
VE A ZONA NARANJA	1	1	1	0	0.4	0.5
VE A ZONA NEGRA	1	1	0	0	0	-
VE AL DADO AZUL	1	1	0.5	0.16	0.2	0.5
VE AL DADO ROJO	1	0.5714	0	0.33	0.4	0.4
VE AL DADO VERDE	0.25	1	0	0.5	0.6	0.2307
[promedio]	0.9325	0.9478	0.3492	0.4627	0.2952	0.3914

De estos datos podemos sacar en conclusión que el reconocedor suele alcanzar una buena taza de éxitos de clasificacion en tiempo de entrenamiento (en torno al 90 %) y que el rendimiento en datos de validación es aun pobre, pues mientras hay ciertas clases predichas correctamente la mayoría del tiempo (como: “Avanza”) aun hay ejemplares que nunca fueron correctamente predichos (tal es el caso de “alto”) al igual que hay clases que nunca fueron correctamente predichas (con exhaustividad 0).

Capítulo 8

Conclusión

Una vez implementados todos los componentes antes descritos se obtiene un sistema capaz de reconocer un conjunto reducido de frases con una probabilidad de acierto de alrededor del 30 % (en datos nuevos) y es posible utilizar las técnicas descritas para crear un reconocedor de voz con frases distintas dado que las frases elegidas son meramente ilustrativas y en la practica se puede definir un conjunto apropiado para alguna aplicación particular. Además es necesario encontrar alguna manera más eficiente de etiquetar los trifonos de los clips de audio capturados, probablemente con algún módulo que implemente de manera unificada el etiquetado de múltiples archivos al mismo tiempo e inclusive una red neuronal que se encargue de distinguir el inicio y el final de un trifono, con lo cual únicamente se deberá etiquetar el fragmento identificado.
En lo que respecta a la hipótesis de esta tesis: dados los resultados presentados es posible afirmar que es factible implementar un sistema reconocedor de voz utilizando la técnica de redes neuronales desde su etapa mas temprana: la captura de audio y su posterior clasificación. El inconveniente mas grande radica en la gran cantidad de información necesaria para el entrenamiento y tiempo para su procesamiento, el cual debe hacerse de manera manual, por lo cual es recomendable para quien trate de seguir los pasos descritos en esta tesis que cuente con apoyo humano en las primeras etapas del desarrollo.

Apéndice A

Instalación de las herramientas

Para la implementación del sistema de se eligieron las siguientes herramientas:

- Lenguaje Python

  Este lenguaje fue seleccionado debido en buena parte a que cuenta con una curva de aprendizaje relativamente corta y gracias a la existencia de un conjunto de herramientas desarrolladas por Google para la implementación de redes neuronales: Tensorflow. Además cuenta con otras bibliotecas dedicadas especialmente a realizar tareas de tratamiento de datos, tales como sci-py y numpy.

- Pydub

  Esta API sera usada para manipular fragmentos del audio de manera simple, para su instalación solo es necesario ejecutar:

```
1 pip install pydub
```

  La documentación de esta herramienta se puede encontrar en: http://pydub.com/

- Seqlearn

  Biblioteca de Python que permite la implementación de modelos ocultos de Márkov, lo cual sera útil en la ultima fase de implementación.

  Bastará con:

```
1 sudo -H pip install seqlearn
```

para completar la instalación de esta herramienta, cuya documentación se encuentra disponible en:Seqlearn

- Tensorflow

  De reciente publicación, esta herramienta es un conjunto de bibliotecas enfocadas a la implementación de sistemas de aprendizaje automatizado y en especial de redes neuronales. Entre sus ventajas cuenta con la posibilidad de aprovechar el uso de GPUs para realizar rápidamente cálculos típicos de las redes neuronales como el cálculo de producto de matrices y de funciones de activación sobre estas.

  Para la instalación simple de Tensorflow en Xubuntu debemos realizar lo siguiente en terminal:

```
1 sudo apt-get install python-pip python-dev
2 sudo -H pip install tensorflow
```

  En cuyo caso la instalación deberá estar finalizada únicamente para realizar computo en CPU, pero, de hecho, es recomendable que en caso de contar con acceso a una tarjeta gráfica NVIDIA con capacidades para usar CUDA, reemplace la ultima linea por

```
1 pip install tensorflow-gpu
```

  Con el detalle de ser necesario realizar la instalación adicional de CUDA y CuDNN en el equipo, lo cual se puede realizar de distintas maneras, por lo que quedaría a cargo del del lector la instalación de tal requerimiento por su cuenta.

  TensorFlow puede ser instalado (al momento de la redacción) en, Windows y Mac OS X de una manera similar, para lo cual se puede consultar:

  Instalación de Tensorflow[48]

- Keras

  Es un API que implementa envoltorios para el diseño rápido de redes neuronales en TensorFlow y Theano. Con esta API se pueden realizar rápidamente cambios a un modelo de red, con lo cual se reduce el tiempo de implementación y es posible enfocarse en los parámetros de ella en vez de en cómo se realizan los procesos necesarios para implementarla.

Esta API requiere de numpy, scipy, yaml y hp5y, requisitos que pueden ser satisfechos con la siguiente instrucción

```
1 sudo -H pip install numpy scipy scipython-yaml pandas
```

Y finalmente ejecutar:

```
1 sudo -H pip install keras
```

- Audacity

  Este editor de audio fue elegido por ser de uso libre e implementar una función fundamental para el posterior desarrollo del proyecto: la posibilidad de etiquetar segmentos de audio.

  Este software para grabación y edición de audio puede ser fácilmente instalado en Xubuntu al ejecutar

```
1 sudo apt-get install audacity
```

- HTK

  Es uno de los programas que por medio de sus módulos permite realizar reconocimiento de voz usando modelos de mezcla gaussiana y cadenas de Márkov, sin embargo su utilidad en este proyecto radica en su capacidad de extraer características PLP del audio capturado. Este programa es de código abierto, por lo cual es posible realizar cambios para mejorarlo y adaptarlo de acuerdo a las necesidades del usuario. Dicho lo anterior, el programa se usará con una pequeña modificación propia que simplifica su salida, en aras de ser usado por los demás componentes del proyecto.

  Para la instalación de este programa hará falta el registro dentro de la pagina del proyecto en : http://htk.eng.cam.ac.uk/register.shtml

  Una vez llenados los datos solicitados, procederemos a satisfacer el único requerimiento de la herramienta, que es libx11, esto con:

```
1 sudo apt-get install libx11-devel g++-multilib
```

  Ahora la descarga e instalación: Asumiendo que el nombre del archivo descargado es HTK-3.3.tar.gz

```
1 tar xzf HTK-3.3.tar.gz
2 cd htk
3 ./configure --disable-hslab -disable-hlmtools
4 Make clean
5 Make all
6 Sudo make install
```

Con lo cual HTK3 quedará completamente instalado.

- Xubuntu Linux 16.05 (Sugerido)

  Esta distribución ha demostrado ser útil para la implementación del sistema y, aunque todos las herramientas anteriormente mencionadas están disponibles para los sistemas Windows y Linux, este último tiene la ventaja de permitir la instalación de manera más rápida a través de línea de comandos y sobretodo por que varias de las utilidades necesarias para compilar (cuando lo requiera) Tensorflow y HTK se encuentran preinstaladas. Adicionalmente en este sistema operativo es muy fácil implementar scripts en Bash, con lo cual se podrá realizar un manejo más eficiente de los archivos capturados y procesados.

Apéndice B

Variantes del modelo propuesto

En este apéndice se muestran los modelos precedentes a la red neuronal propuesta en el capitulo 6. Para ello haré uso de una notación que simplifica la manera de ver la arquitectura de la red neuronal y la cual explico a continuación:

- La red neuronal comienza a ser descrita por capas desde su entrada a la izquierda hacia la derecha, donde se obtiene el resultado final.
- Un número entero N significa una capa de N neuronas completamente conexa con la capa anterior, es decir que cada neurona tiene N pesos con la capa anterior, la función de activación siempre es ReLu.
- La letra D simboliza el uso de una capa dropout, sobre la cual se pueden leer algunos detalles en el capitulo 3.

Asi pues, la tabla de los modelos probados (ordenados de manera descendente según su rendimiento) es la mostrada en la figura B.1.
Cabe destacar que todos los modelos tienen en su ultima etapa una capa de 338 neuronas durante el entrenamiento del deep autoencoder(preentrenamiento de la red) , además de que esta última capa es eliminada y reemplazada por una de 70 neuronas para realizar la clasificación en trifonos en la etapa final del entrenamiento(entrenamiento de la red general) en donde la función de activación es reemplazada por softmax.
Por ejemplo:
El modelo 1 que se encuentra codificado por:

338 D 1200 1200 1200

	Pruebas de DA + Softmax					
Variacion número:	Estructura	Precisión entrenamiento %	Error entrenamiento	Precisión validacion %	Error validacion	Tiempo de entrenamiento (Min:segs)
1	338 D 1200 1200 1200	93.27	0.1247	62.15	2.41	09:52
2	338 D 800 800 800	93.11	0.1329	61.75	2.401	05:45
3	338 D 400 400 400	92.8	0.1548	62.44	2.67	03:17
4	338 400 400 400	92.9	0.1411	60.5	2.692	03:01
5	338 800 800 800	93.48	0.1138	62.1	2.309	05:22
6	338 D 400 D 400 D 400	83.6	0.4558	62.86	1.978	04:37
7	338 1200 1200 1200	93.58	0.1098	64.07	2.167	09:13
8	338 D 800 D 800 D 800 D	91.44	0.1968	63.84	2.306	07:57
9	338 D 1200 D 1200 D 1200 D	92.29	0.1655	64.73	2.277	13:36
10	338 D 800 D 400 D 300 D	87.46	0.3295	64.02	2.058	06:09
11	338 D 500 D 400 D 300 D	85.79	0.386	63.16	2.031	04:21
A	338 D 500 D 400 D 350 D	86.68	0.3572	64.04	2.072	05:52

Figura B.1: Tabla con los modelos probados, en color verde se muestra la red utilizada para el desarrollo de esta tesis y en color amarillo las siguientes dos redes con mejor rendimiento.

Consta, durante el entrenamiento del deep autoencoder, de la siguiente arquitectura:

338 D 1200 1200 1200 338

Donde:

338 es el número de elementos de entrada.

D corresponde a una capa dropout entre la capa anterior y la siguiente de 1200 elementos.

1200 corresponde con 3 capas consecutivas de 1200 elementos cada una usando ReLu como función de activación.

338 el número de neuronas para reconstruir la entrada durante la fase del entrenamiento del deep autoencoder, esta capa usa la función de activación sigmoide.

Y, posteriormente, para el aprendizaje de la clasificación adquiere esta nueva arquitectura:

338 D 1200 1200 1200 70

338 es el número de elementos de entrada.

D corresponde a una capa dropout entre la capa anterior y la siguiente de 1200 elementos.

1200 corresponde con 3 capas consecutivas de 1200 elementos cada una usando ReLu como función de activación.

70 el número de neuronas que indican el resultado de la clasificación de la muestra de entrada durante la fase del entrenamiento de toda la red, la función de activación de esta capa es softmax sigmoide.

En los casos de las variaciones 7 y 9 se observa que ambas superan en rendimiento al modelo A por menos de un punto porcentual en la precisión de la clasificación en el conjunto de validación, pero tienen dos grandes desventajas: ambas cuentan con 3600 elementos frente a los 1250 del modelo A y toman alrededor del doble de tiempo en ser entrenadas[1], motivo por el cual fueron descartadas para las siguientes etapas de la tesis.
En la figura B.2 observamos que las redes 7 y 9 obtuvieron rápidamente un mejor rendimiento que el modelo A, a costa de mantenerlo estancado a partir de la cuarta iteración en el caso de la variación 7 y de la decimoquinta época en el caso de la variación 9, mientras que el modelo A tiene un rendimiento de alrededor de 6 % menor.
En la figura B.3 es apreciable que el error de todas las redes disminuyo de manera progresiva y estable a lo largo del tiempo.
Ahora bien, en el caso del conjunto de validación observamos importantes diferencias en el proceso aprendizaje de cada una de las redes en contraste con el modelo A. Los modelos 7 y 9 mostraron una considerable variación en su rendimiento a lo largo del tiempo, mientras que el modelo A se mantuvo siempre mejorando, demostrando con esto que la red es capaz de generalizar la información aprendida del conjunto de entrenamiento, lo cual no es el caso de el resto de modelos.
Finalmente se muestra la gráfica del error en el conjunto de validación de las tres redes, donde en todo momento el modelo A mantiene un error menor a las otras 2. En el caso de las red 9, desde la segunda época de entrenamiento el error continuo aumentando de manera progresiva. En la red 7 el error tiene una tendencia creciente, mientras que la variación A comienza a aumentar su error a partir de la séptima época.

[1]Tiempo medido entrenando el sistema con una tarjeta gráfica NVIDIA 840M con 384 núcleos CUDA

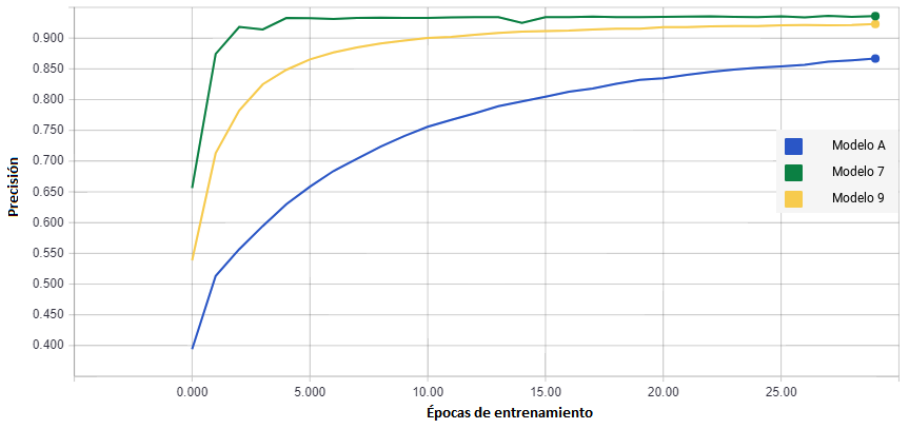


Figura B.2: Precisión de las redes 7,9 y A en el conjunto de entrenamiento. En el eje X se muestran las épocas de entrenamiento y en el eje Y la precisión del modelo.

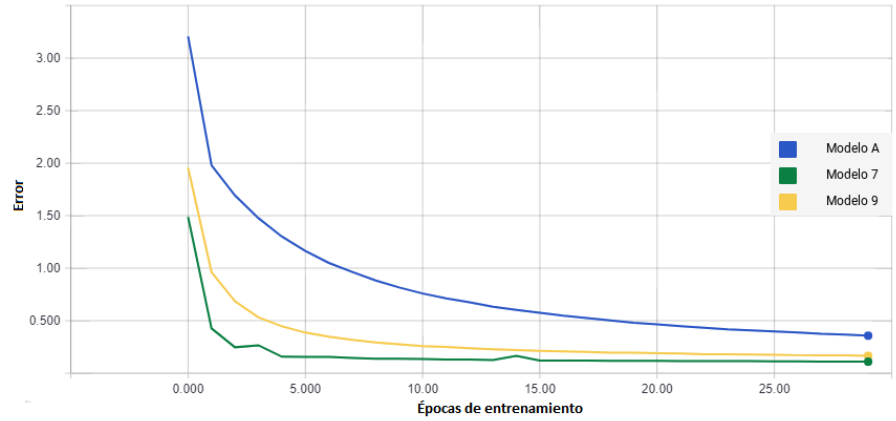


Figura B.3: Error de las redes neuronales 7, 9 y A en el conjunto de entrenamiento. En el eje X se muestran las épocas de entrenamiento y en el eje Y la precisión del modelo.

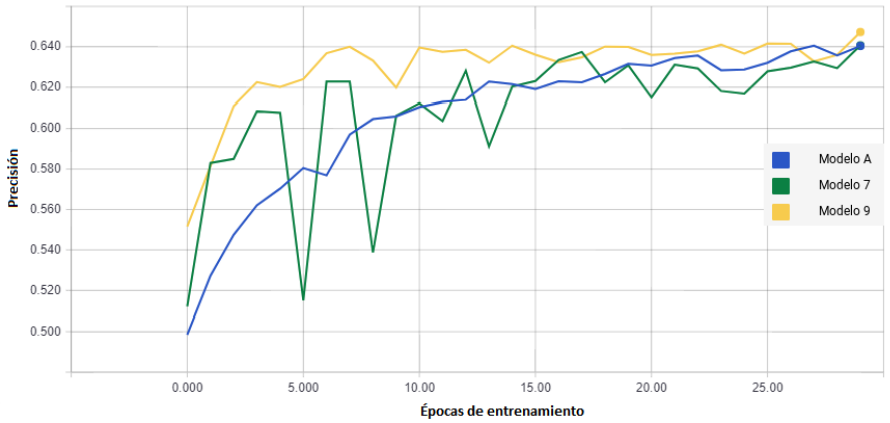


Figura B.4: Precisión de las redes 7,9 y A en el conjunto de validación. En el eje X se muestran las épocas de entrenamiento y en el eje Y el error del modelo.

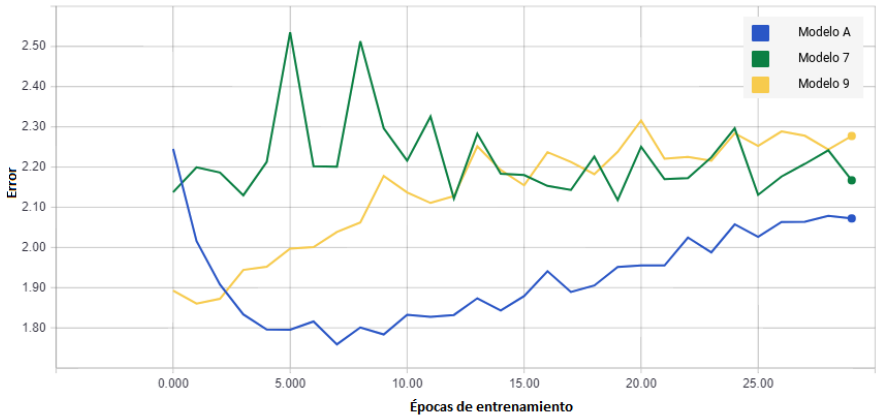


Figura B.5: Error de las redes 7,9 y A en el conjunto de validación. En el eje X se muestran las épocas de entrenamiento y en el eje Y el error del modelo.

B.1. Otros cambios

Algunas otras modificaciones hechas a la red neuronal A fueron la ubicación de capas que hacen uso de Dropout, los resultados de estos cambios no resultaron ser substancialmente diferentes a la red propuesta en la sección anterior. En la tabla se encuentran las predicciones realizadas sobre trifonos y únicamente se considera acierto o fallo en conjuntos de validación y entrenamiento.

Descripción	Aciertos entrenamiento(%)	Aciertos validación(%)
Dropout 30 % entre cada capa, excepto entre el ultimo encoder y softmax	93.20	63.95
Dropout únicamente en la entrada 30 %	93.21	62.02
Dropout 30 % entre todas las capas(incluida softmax)	89.76	61.84

Tabla B.1: Comparación de modelos basados en la variante A.

Apéndice C

Predicciones usando seqlearn

En este apéndice se muestran los resultados de las reconstrucciones de secuencia de seqlearn (HMM) Cada uno de los archivos de entrenamiento muestran después del símbolo igual la frase que fue dicha.
Para todas se utilizo la red neuronal completamente entrenada y se eliminaron secuencias continuas de elementos iguales para facilitar la lectura.
La lectura de estos datos es sencilla: En el primer renglón, después del signo igual se muestra cual fue la frase dicha por el hablante y en el siguiente renglón se muestra la secuencia de fonos obtenida por seqlearn usando el algoritmo de Viterbi sobre los trifonos clasificados por la red neuronal. Cada archivo de entrenamiento esta separado por
Aunque estas secuencias ya contienen los trifonos que corresponden a una frase, aun muestran trifonos incorrectos.
Frase:simpleEntrenamiento/tmpagarra41P.wav.npy =AGARRA LA ESPONJA ('Viterbi:', array(['sil', 'nja', 'aga', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'la', 'esp', 'spo', 'pon', 'onj', 'nja', 'gra', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpagarra41.wav.npy =AGARRA LA ESPONJA ('Viterbi:', array(['sil', 'nja', 'garr', 'aga', 'garr', 'arra', 'la', 'esp', 'la', 'esp', 'spo', 'pon', 'onj', 'nja', 'sil', 'gra', 'sil', 'gra', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpagarra43P.wav.npy =AGARRA EL DADO VERDE ('Viterbi:', array(['sil', 'nja', 'aga', 'anj', 'aga', 'garr', 'aga', 'garr', 'arra', 'garr', 'la', 'ran', 'aga', 'ran', 'aga', 'anj', 'el', 'dad', 'ado', 'pon', 'ber', 'erd', 'rde', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpagarra43.wav.npy =AGARRA EL DADO VERDE ('Viterbi:', array(['sil', 'nja', 'anj', 'aga', 'anj', 'aga', 'garr', 'arra', 'be', 'ere', 'gra', 'ere', 'be', 'a', 'dad', 'ado', 'spo', 'ber', 'rde', 'erd', 'rde', 'gra', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpagarra53.wav.npy =AGARRA EL DADO VERDE ('Viterbi:', array(['sil', 'aga', 'sil', 'aga', 'ado', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'arra', 'garr', 'arra', 'ska', 'alt', 'a', 'alt', 'a', 'el', 'a', 'el', 'alu', 'uda', 'alu', 'uda', 'dad', 'ado', 'dad', 'ado', 'rroj', 'ber', 'erd', 'rde', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpagarra61.wav.npy =AGARRA LA ESPONJA ('Viterbi:', array(['aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'sal', 'aga', 'sal', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'arra', 'aga', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'aga', 'arra', 'aga', 'la', 'esp', 'spo', 'pon', 'onj', 'nja', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'ska', 'arra', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpagarra62.wav.npy =AGARRA EL DADO ROJO ('Viterbi:', array(['sil', 'rech', 'echa', 'rech', 'echa', 'rech', 'sil', 'nja', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'aga', 'garr', 'arra', 'sil', 'el', 'be', 'kie', 'zed', 'kie', 'be', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'aba', 'ado', 'aba', 'ado', 'rroj', 'ado', 'rroj', 'ojo', 'ska', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpagarra64.wav.npy =AGARRA EL DADO AZUL ('Viterbi:', array(['sil', 'be', 'sil', 'rde', 'zul', 'ska', 'aga', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'sil', 'be', 'sil', 'spo', 'be', 'rech', 'neg', 'ere', 'der', 'ere', 'der', 'ere', 'der', 'el', 'der', 'el', 'ede', 'el', 'ede', 'zed', 'ede', 'dad', 'ado', 'azu', 'zul', 'azu', 'zul', 'azu', 'zul', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpalto1.wav.npy =ALTO ('Viterbi:', array(dtype='—S3'))

Frase:simpleEntrenamiento/tmpalto2.wav.npy =ALTO ('Viterbi:', array(['sil', 'ona', 'aga', 'alt', 'nja', 'lto', 'ojo', 'nja', 'lto', 'nja', 'lto', 'nja', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpalto5.wav.npy =ALTO ('Viterbi:', array(['sil', 'sal', 'al', 'a', 'al', 'a', 'alt', 'al', 'alt', 'al', 'alt', 'al', 'alt', 'al', 'alt', 'al', 'alt', 'al', 'alt', 'lto', 'ado', 'lto', 'ado', 'ojo', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpavanza1.wav.npy =AVANZA ('Viterbi:', array(dtype='—S4'))

Frase:simpleEntrenamiento/tmpavanza3.wav.npy =AVANZA ('Viterbi:', array(dtype='—S3'))

Frase:simpleEntrenamiento/tmpavanza6.wav.npy =AVANZA ('Viterbi:', array(['sil', 'sal', 'aga', 'sal', 'aga', 'arra', 'aba', 'arra', 'aba', 'arra', 'aba', 'arra', 'aba', 'garr', 'arra', 'garr', 'ban', 'anz', 'nza', 'esp', 'spo', 'esp', 'spo', 'ska', 'spo', 'ska', 'arra', 'dad', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpdadoa4.wav.npy =BUSCA EL DADO AZUL ('Viterbi:', array(['sil', 'gra', 'nja', 'gra', 'nja', 'sil', 'be', 'bus', 'be', 'bus', 'usk', 'ska', 'echa', 'el', 'ira', 'arra', 'ira', 'arra', 'garr', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'a', 'azu', 'zul', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpdadoa5.wav.npy =BUSCA EL DADO AZUL ('Viterbi:', array(['sil', 'bus', 'usk', 'zul', 'usk', 'zul', 'usk', 'sil', 'ska', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'ska', 'el', 'be', 'el', 'be', 'el', 'la', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'dad', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'ado', 'ama', 'azu', 'zul', 'azu', 'zul', 'azu', 'zul', 'azu', 'zul', 'azu', 'zul', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpdador4.wav.npy =BUSCA EL DADO ROJO ('Viterbi:', array(['sil', 'be', 'bus', 'usk', 'ska', 'nar', 'ara', 'el', 'arra', 'el', 'arra', 'el', 'aga', 'garr', 'aga', 'garr', 'dad', 'ado', 'ber', 'ado', 'ber', 'ado', 'ber', 'ado', 'ber', 'ado', 'rroj', 'ojo', 'pon', 'nja', 'sil', 'nja', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpdador6.wav.npy =BUSCA EL DADO ROJO ('Viterbi:', array(['sil', 'be', 'sil', 'be', 'sil', 'ska', 'sil', 'ska', 'sil', 'ska', 'be', 'bus', 'usk', 'ska', 'arra', 'ska', 'arra', 'sil', 'be', 'oze', 'zed', 'el', 'be', 'dad', 'ado', 'rroj', 'ado', 'rroj', 'ojo', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpdadov4.wav.npy =BUSCA EL DADO VERDE ('Viterbi:', array(['sil', 'bus', 'usk', 'sil', 'ska', 'sil', 'ska', 'sil', 'ska', 'bus', 'ska', 'zul', 'bus', 'usk', 'zul', 'usk', 'oze', 'ska', 'zul', 'ber', 'zul', 'ber', 'zul', 'ber', 'dad', 'ber', 'ira', 'dad', 'ira', 'el', 'ira', 'el', 'ado', 'dad', 'ado', 'arra', 'ado', 'arra', 'garr', 'zul', 'ber', 'erd', 'rde', 'sil', 'rde', 'sil', 'rde', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpdadov5.wav.npy =BUSCA EL DADO VERDE ('Viterbi:', array(['sil', 'ska', 'sil', 'ska', 'bus', 'ska', 'lto', 'el', 'be', 'el', 'la', 'dad', 'ado', 'rroj', 'ona', 'ber', 'erd', 'ber', 'erd', 'rde', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpesponja1.wav.npy =BUSCA LA ESPONJA ('Viterbi:', array(['sil', 'be', 'bus', 'usk', 'ska', 'nja', 'ska', 'erd', 'ira', 'arra', 'garr', 'arra', 'la', 'esp', 'spo', 'pon', 'onj', 'lto', 'nja', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpesponja4P.wav.npy =BUSCA LA ESPONJA ('Viterbi:', array(['sil', 'be', 'bus', 'usk', 'ska', 'spo', 'nar', 'el', 'la', 'garr', 'arra', 'neg', 'la', 'a', 'la', 'a', 'esp', 'spo', 'esp', 'spo', 'pon', 'nja', 'gra', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpesponja5.wav.npy =BUSCA LA ESPONJA ('Viterbi:', array(['sil', 'ska', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'ska', 'arra', 'al', 'mar', 'alu', 'la', 'esp', 'la', 'esp', 'spo', 'esp', 'spo', 'esp', 'spo', 'pon', 'spo', 'pon', 'onj', 'nja', 'onj', 'nja', 'ska', 'nja', 'ska', 'nja', 'ska', 'nja', 'ska', 'nja', 'ska',

'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpgder2.wav.npy =GIRA A LA DERECHA ('Viterbi:', array(['sil', 'jir', 'ira', 'a', 'aga', 'al', 'mar', 'aga', 'la', 'aga', 'la', 'azu', 'al', 'sal', 'al', 'sal', 'al', 'sal', 'al', 'sal', 'nja', 'al', 'nja', 'alu', 'al', 'alu', 'al', 'alu', 'al', 'alu', 'al', 'alu', 'al', 'der', 'ere', 'rech', 'echa', 'nja', 'rda', 'nja', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpgder4P.wav.npy =GIRA A LA DERECHA ('Viterbi:', array(['sil', 'echa', 'jir', 'ira', 'a', 'el', 'a', 'la', 'garr', 'la', 'der', 'ere', 'echa', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpgder4.wav.npy =GIRA A LA DERECHA ('Viterbi:', array(['sil', 'jir', 'ira', 'ara', 'ran', 'el', 'la', 'el', 'la', 'el', 'la', 'a', 'la', 'a', 'la', 'el', 'a', 'la', 'a', 'la', 'der', 'ere', 'rech', 'echa', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpgder6.wav.npy =GIRA A LA DERECHA ('Viterbi:', array(['sil', 'zki', 'rech', 'echa', 'kie', 'jir', 'ira', 'gra', 'aga', 'gra', 'arra', 'aba', 'aga', 'aba', 'a', 'la', 'der', 'el', 'der', 'el', 'ere', 'rech', 'ere', 'rech', 'echa', 'ere', 'rech', 'ere', 'rech', 'ere', 'rech', 'el', 'rech', 'el', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpgizq1.wav.npy =GIRA A LA IZQUIERDA ('Viterbi:', array(['sil', 'jir', 'ira', 'al', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'arra', 'garr', 'aga', 'garr', 'aga', 'garr', 'la', 'izk', 'zki', 'kie', 'ier', 'rda', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpgizq4P.wav.npy =GIRA A LA IZQUIERDA ('Viterbi:', array(['sil', 'rde', 'sil', 'ona', 'zul', 'ona', 'zul', 'sil', 'jir', 'ira', 'aga', 'dad', 'a', 'dad', 'a', 'erd', 'a', 'erd', 'a', 'erd', 'a', 'erd', 'a', 'la', 'izk', 'zki', 'azu', 'kie', 'ier', 'erd', 'zul', 'nja', 'ona', 'nja', 'rda', 'arra', 'rroj', 'ojo', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpgizq5.wav.npy =GIRA A LA IZQUIERDA ('Viterbi:', array(['sil', 'jir', 'ira', 'a', 'ira', 'a', 'aga', 'ama', 'aga', 'ama', 'aga', 'ama', 'aga', 'ama', 'a', 'ama', 'a', 'al', 'la', 'ber', 'la', 'ber', 'la', 'ber', 'la', 'ber', 'la', 'ber', 'la', 'ari', 'rill', 'illa', 'izk', 'zki', 'kie', 'ier', 'erd', 'rde', 'rda', 'rde', 'rda', 'rde', 'rda', 'rde', 'rda', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpretrocede1.wav.npy =RETROCEDE ('Viterbi:', array(['sil', 'be', 'rret', 'etr', 'el', 'alu', 'rde', 'tro', 'rde', 'la', 'roz', 'usk', 'oze', 'zed', 'ede', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpretrocede2.wav.npy =RETROCEDE ('Viterbi:', array(['sil', 'rret', 'etr', 'tro', 'roz', 'oze', 'roz', 'oze', 'roz', 'oze', 'roz', 'oze', 'zed', 'ede', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpretrocede6.wav.npy =RETROCEDE ('Viterbi:', array(['sil', 'echa', 'ska', 'rret', 'etr', 'rret', 'aga', 'tro', 'aga', 'tro', 'roz', 'oze', 'zed', 'ede', 'etr', 'ede', 'etr', 'ede', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpsaluda2.wav.npy =SALUDA ('Viterbi:', array(

dtype='—S3'))

Frase:simpleEntrenamiento/tmpsaluda3.wav.npy =SALUDA ('Viterbi:', array(dtype='—S3'))

Frase:simpleEntrenamiento/tmpsaluda5.wav.npy =SALUDA ('Viterbi:', array(['sil', 'sal', 'al', 'sal', 'al', 'sal', 'al', 'sal', 'alu', 'lud', 'uda', 'lud', 'uda', 'lud', 'uda', 'lud', 'uda', 'rda', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpveam1.wav.npy =VE A ZONA AMARILLA ('Viterbi:', array(['sil', 'be', 'ber', 'be', 'ber', 'dad', 'garr', 'aga', 'a', 'zon', 'ona', 'sil', 'spo', 'nja', 'alt', 'ama', 'mar', 'ari', 'rill', 'illa', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpveam2.wav.npy =VE A ZONA AMARILLA ('Viterbi:', array(['sil', 'be', 'ira', 'a', 'zon', 'ado', 'ama', 'ona', 'mar', 'ona', 'ama', 'mar', 'ama', 'aga', 'ama', 'aga', 'mar', 'aga', 'mar', 'ari', 'rill', 'illa', 'rda', 'nja', 'rda', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpveam4P.wav.npy =VE A ZONA AMARILLA ('Viterbi:', array(['sil', 'be', 'ira', 'a', 'zon', 'ona', 'ama', 'mar', 'la', 'el', 'rill', 'illa', 'ira', 'jir', 'echa', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpveam5.wav.npy =VE A ZONA AMARILLA ('Viterbi:', array(['sil', 'be', 'a', 'zon', 'ona', 'ama', 'mar', 'ari', 'rill', 'illa', 'rill', 'illa', 'rill', 'illa', 'rill', 'illa', 'rill', 'illa', 'rill', 'illa', 'rill', 'illa', 'rill', 'illa', 'rde', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpvedada3.wav.npy =VE AL DADO AZUL ('Viterbi:', array(['sil', 'be', 'ber', 'a', 'ber', 'a', 'al', 'anz', 'arra', 'anz', 'arra', 'anz', 'arra', 'anz', 'arra', 'al', 'arra', 'al', 'arra', 'ado', 'dad', 'ado', 'rroj', 'ojo', 'azu', 'zul', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpvedada4P.wav.npy =VE AL DADO AZUL ('Viterbi:', array(['sil', 'zul', 'ska', 'be', 'ira', 'dad', 'ira', 'dad', 'a', 'al', 'ado', 'al', 'ado', 'dad', 'arra', 'dad', 'arra', 'dad', 'zul', 'dad', 'zul', 'dad', 'ado', 'azu', 'zul', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpvedada4.wav.npy =VE AL DADO AZUL ('Viterbi:', array(['sil', 'bus', 'usk', 'bus', 'usk', 'bus', 'usk', 'bus', 'sil', 'ska', 'oze', 'be', 'oze', 'be', 'ira', 'be', 'ira', 'erd', 'ira', 'erd', 'ira', 'erd', 'a', 'erd', 'a', 'erd', 'al', 'a', 'al', 'a', 'al', 'ado', 'al', 'ado', 'al', 'ado', 'al', 'ado', 'zul', 'ado', 'zul', 'dad', 'ado', 'zon', 'ado', 'zon', 'nja', 'onj', 'nja', 'onj', 'nja', 'azu', 'zul', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpvedada6.wav.npy =VE AL DADO AZUL ('Viterbi:', array(['sil', 'be', 'sil', 'be', 'ska', 'a', 'al', 'rde', 'al', 'rde', 'al', 'rde', 'al', 'rde', 'al', 'ado', 'dad', 'ado', 'azu', 'zul', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpvedadr1.wav.npy =VE AL DADO ROJO ('Viterbi:', array(['sil', 'be', 'ira', 'al', 'ira', 'al', 'dad', 'zul', 'dad', 'ado', 'dad', 'ado', 'pon', 'dad', 'pon', 'dad', 'pon', 'ado', 'rroj', 'ado', 'rroj', 'ado', 'pon', 'aba', 'pon', 'aba', 'rroj', 'aba', 'rroj', 'aba', 'rroj', 'nja', 'rroj', 'nja', 'rroj', 'ojo', 'sil'], dty-

pe='—S4'))

Frase:simpleEntrenamiento/tmpvedadr3.wav.npy =VE AL DADO ROJO ('Viterbi:', array(['sil', 'be', 'echa', 'ier', 'al', 'erd', 'rde', 'dad', 'ado', 'rroj', 'ojo', 'rroj', 'ojo', 'rroj', 'ojo', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpvedadr5.wav.npy =VE AL DADO ROJO ('Viterbi:', array(['sil', 'ska', 'be', 'rret', 'al', 'la', 'dad', 'ado', 'rroj', 'ado', 'rroj', 'ojo', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpvedav1.wav.npy =VE AL DADO VERDE ('Viterbi:', array(['sil', 'be', 'ira', 'erd', 'ira', 'erd', 'al', 'zul', 'bus', 'dad', 'ado', 'pon', 'ber', 'rret', 'rde', 'rret', 'rde', 'be', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpvedav2.wav.npy =VE AL DADO VERDE ('Viterbi:', array(['sil', 'be', 'ira', 'be', 'ira', 'be', 'ber', 'al', 'ber', 'al', 'el', 'al', 'rde', 'be', 'bus', 'be', 'dad', 'ado', 'dad', 'ado', 'nja', 'lto', 'rroj', 'ber', 'erd', 'ber', 'erd', 'ede', 'etr', 'rde', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpveesp2.wav.npy =VE A LA ESPONJA ('Viterbi:', array(['sil', 'be', 'ira', 'a', 'garr', 'aga', 'garr', 'al', 'aga', 'al', 'aga', 'al', 'aga', 'al', 'aga', 'al', 'aga', 'garr', 'ber', 'al', 'ber', 'al', 'ber', 'al', 'ber', 'al', 'ber', 'al', 'ber', 'al', 'erd', 'el', 'esp', 'spo', 'pon', 'onj', 'nja', 'onj', 'nja', 'onj', 'nja', 'onj', 'nja', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpveesp3.wav.npy =VE A LA ESPONJA ('Viterbi:', array(dtype='—S3'))

Frase:simpleEntrenamiento/tmpvena3.wav.npy =VE A ZONA NARANJA ('Viterbi:', array(['sil', 'be', 'ira', 'a', 'al', 'a', 'anz', 'nza', 'anz', 'nza', 'esp', 'nza', 'zon', 'ona', 'nar', 'ara', 'nar', 'ara', 'nar', 'ara', 'ran', 'anj', 'nja', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpvena4.wav.npy =VE A ZONA NARANJA ('Viterbi:', array(['sil', 'be', 'ira', 'a', 'zon', 'ona', 'nar', 'ara', 'nar', 'ara', 'nar', 'ara', 'ran', 'anj', 'ran', 'anj', 'nja', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpveneg2.wav.npy =VE A ZONA NEGRA ('Viterbi:', array(['sil', 'ojo', 'be', 'ber', 'al', 'a', 'zon', 'ona', 'garr', 'aga', 'arra', 'aga', 'arra', 'aga', 'arra', 'aga', 'arra', 'aga', 'arra', 'gra', 'arra', 'gra', 'neg', 'egr', 'spo', 'sal', 'gra', 'sil'], dtype='—S4'))

Frase:simpleEntrenamiento/tmpveneg3.wav.npy =VE A ZONA NEGRA ('Viterbi:', array(['sil', 'be', 'ber', 'be', 'a', 'be', 'a', 'arra', 'a', 'arra', 'a', 'azu', 'a', 'azu', 'zon', 'ona', 'ska', 'zul', 'rde', 'zul', 'rde', 'neg', 'egr', 'gra', 'sil'], dtype='—S4'))

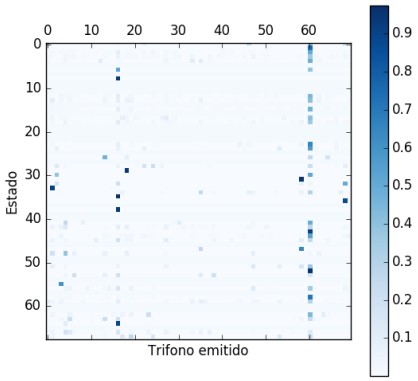
Frase:simpleEntrenamiento/tmpveneg4.wav.npy =VE A ZONA NEGRA ('Viterbi:', array(['sil', 'be', 'ira', 'be', 'ira', 'be', 'a', 'zon', 'ona', 'aga', 'nja', 'gra', 'nja', 'gra', 'nja', 'gra', 'rde', 'neg', 'egr', 'gra', 'sil'], dtype='—S3'))

Frase:simpleEntrenamiento/tmpveneg6.wav.npy =VE A ZONA NEGRA ('Viterbi:', array(['sil', 'ska', 'be', 'el', 'be', 'el', 'be', 'el', 'be', 'ede', 'sil', 'ede', 'be', 'dad', 'a', 'azu', 'zon', 'ona', 'aga', 'neg', 'egr', 'gra', 'sil'], dtype='—S3'))

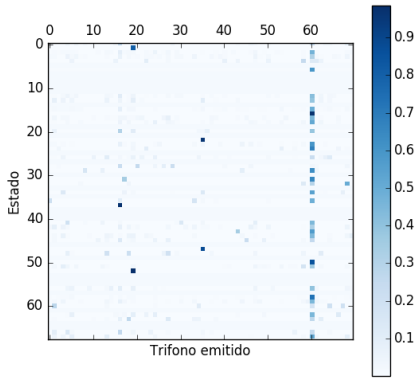
Apéndice D

Matrices de emisión

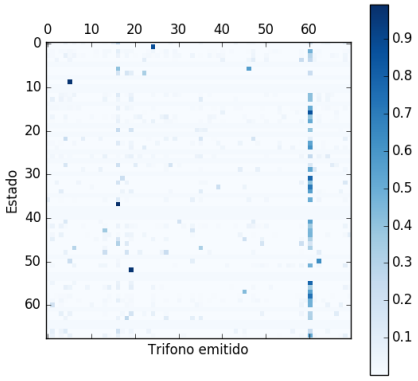
En este apartado se exponen las matrices de emisión que fueron comentadas en el capitulo 7. Las matrices fueron aprendidas usando seqlearn y la diferencia entre ellas radica en la cantidad de iteraciones que tenia la red neuronal con la cual se realizo el el ajuste del HMM (Solo se presentan las iteraciones en las cuales mejoro el rendimiento de la red).
En cada renglón de la matriz se representa la probabilidad de emitir alguno de los trifonos representados en las columnas en un estado concreto; mientras mas blanco es un color, menos probable es que ese trifono sea emitido.



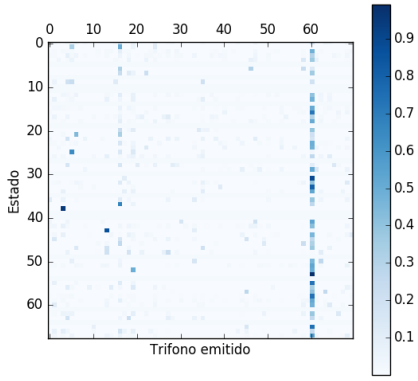
Matriz de emisión de seqlearn usando la red neuronal después de la primera iteración de entrenamiento se observa que, dado un silencio se puede emitir prácticamente cualquier trifono



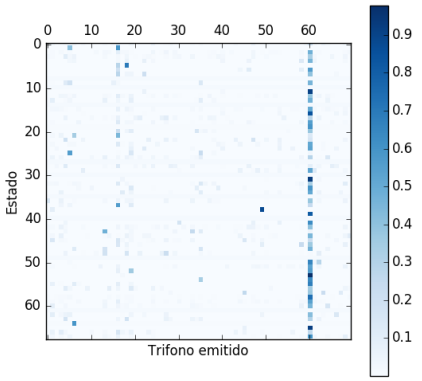
Matriz de emisión de seqlearn usando la red neuronal después de dos épocas de entrenamiento, las probabilidades de emisión del resto de los trifonos comienza a aumentar



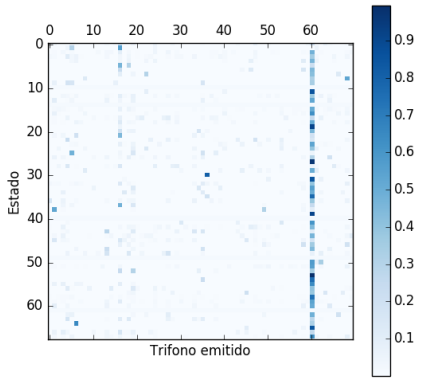
Matriz de emisión de seqlearn usando la red neuronal después de tres épocas de entrenamiento



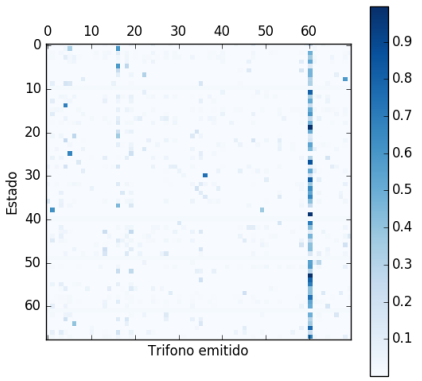
Matriz de emisión de seqlearn usando la red neuronal después de cuatro épocas de entrenamiento



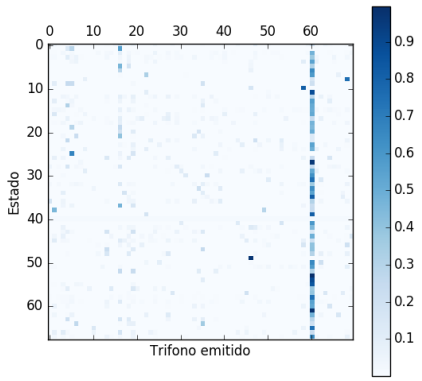
Matriz de emisión de seqlearn usando la red neuronal después de cinco épocas de entrenamiento



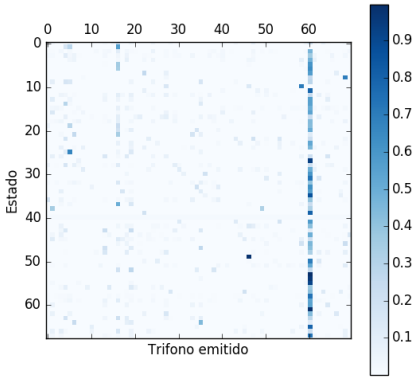
Matriz de emisión de seqlearn usando la red neuronal después de seis épocas de entrenamiento



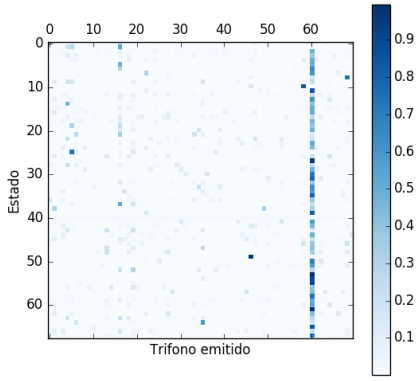
Matriz de emisión de seqlearn usando la red neuronal después de siete épocas de entrenamiento.Se comienza a observar la formación de la diagonal de la matriz



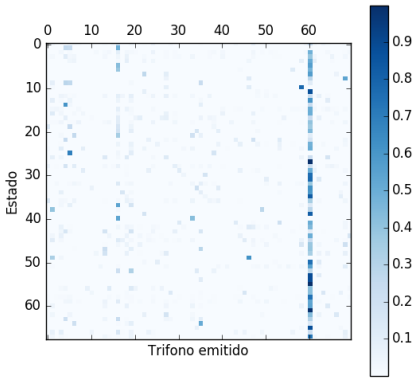
Matriz de emisión de seqlearn usando la red neuronal después de diez épocas de entrenamiento



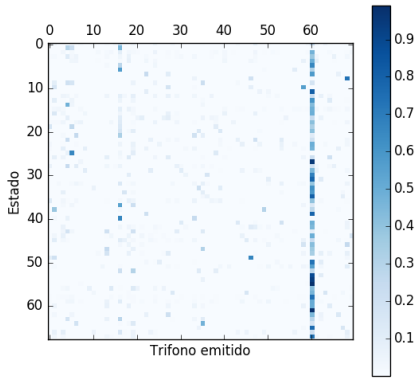
Matriz de emisión de seqlearn usando la red neuronal después de once épocas de entrenamiento



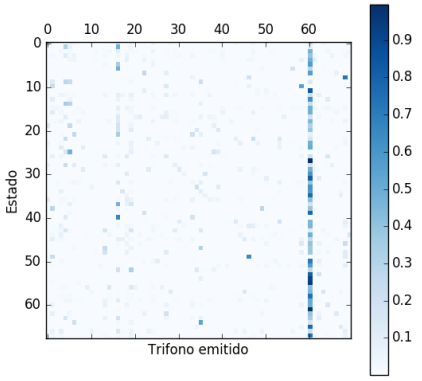
Matriz de emisión de seqlearn usando la red neuronal después de doce épocas de entrenamiento



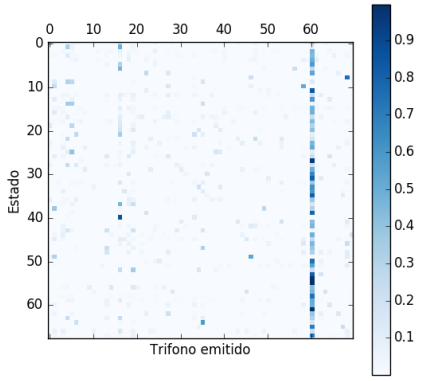
Matriz de emisión de seqlearn usando la red neuronal después de catorce épocas de entrenamiento



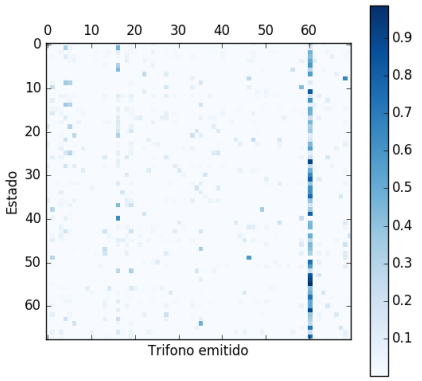
Matriz de emisión de seqlearn usando la red neuronal después de quince épocas de entrenamiento



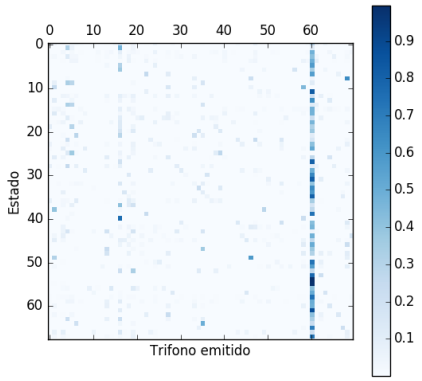
Matriz de emisión de seqlearn usando la red neuronal después de diecisiete épocas de entrenamiento



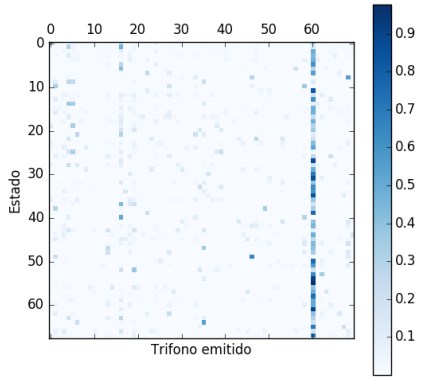
Matriz de emisión de seqlearn usando la red neuronal después de dieciocho épocas de entrenamiento



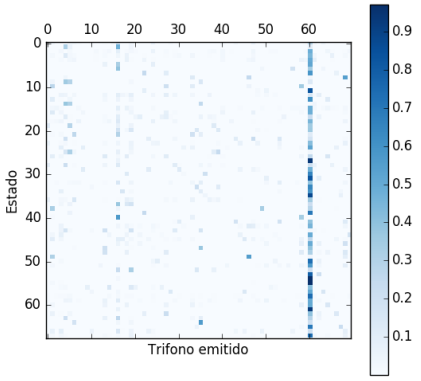
Matriz de emisión de seqlearn usando la red neuronal después de veinte épocas de entrenamiento



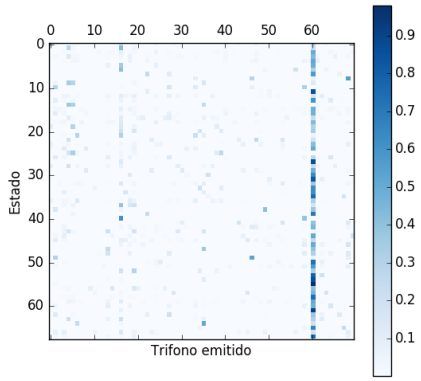
Matriz de emisión de seqlearn usando la red neuronal después de veintiún épocas de entrenamiento



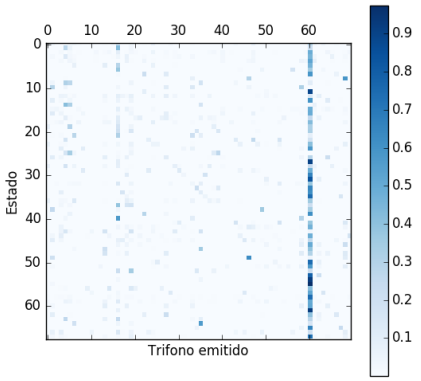
Matriz de emisión de seqlearn usando la red neuronal después de veintidós épocas de entrenamiento



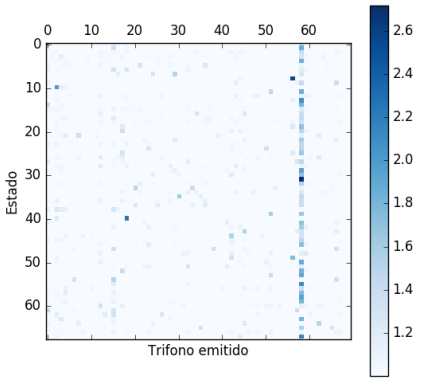
Matriz de emisión de seqlearn usando la red neuronal después de veinticuatro épocas de entrenamiento



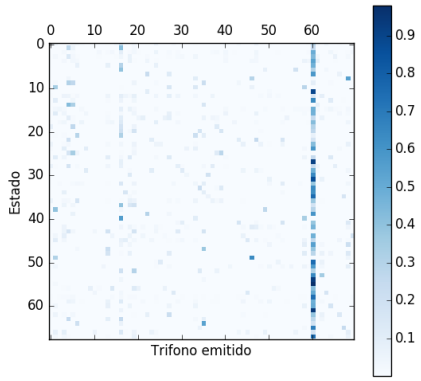
Matriz de emisión de seqlearn usando la red neuronal después de veintiséis épocas de entrenamiento



Matriz de emisión de seqlearn usando la red neuronal después de veintinueve épocas de entrenamiento



Matriz de emisión calculada usando la red neuronal después de veintinueve épocas de entrenamiento.Las probabiliades de la diagonal incrementan su intensidad con respecto al paso graficado anterior.



Matriz de emisión de seqlearn usando la red neuronal después de treinta épocas de entrenamiento

Apéndice E

Predicciones con el algoritmo propuesto

En este apéndice se muestran los resultados de la predicción en el conjunto de entrenamiento usando la red neuronal al final de su entrenamiento y haciendo uso del algoritmo propuesto en 7.2.
Después del signo igual se muestran las frases que contenía cada archivo.En el renglón siguiente se muestra la puntuación (sobre 100) que tiene la predicción mostrada a la derecha.
Frase:simpleEntrenamiento/tmpagarra41P.wav.npy =Agarra la esponja
[(99, 'Agarra la esponja'), (77, 'Ve a esponja'), (72, 'Busca la esponja')]
Frase:simpleEntrenamiento/tmpagarra41.wav.npy =Agarra la esponja
[(99, 'Agarra la esponja'), (77, 'Ve a esponja'), (72, 'Busca la esponja')]
Frase:simpleEntrenamiento/tmpagarra43P.wav.npy =Agarra el dado verde
[(99, 'Agarra el dado verde'), (80, 'Agarra el dado rojo'), (80, 'Agarra el dado azul')]
Frase:simpleEntrenamiento/tmpagarra43.wav.npy =Agarra el dado verde
[(90, 'Agarra el dado verde'), (84, 'Ve al dado verde'), (70, 'Agarra el dado rojo')]
Frase:simpleEntrenamiento/tmpagarra53.wav.npy =Agarra el dado verde
[(99, 'Agarra el dado verde'), (80, 'Agarra el dado rojo'), (80, 'Agarra el dado azul')]
Frase:simpleEntrenamiento/tmpagarra61.wav.npy =Agarra la esponja
[(99, 'Agarra la esponja'), (32, 'Saluda'), (24, 'Ve a zona amarilla')]
Frase:simpleEntrenamiento/tmpagarra62.wav.npy =Agarra el dado rojo
[(100, 'Agarra el dado rojo'), (84, 'Ve al dado rojo'), (80, 'Agarra el dado azul')]
Frase:simpleEntrenamiento/tmpagarra64.wav.npy =Agarra el dado zul
[(100, 'Agarra el dado azul'), (84, 'Ve al dado azul'), (80, 'Agarra el dado rojo')]

Frase:simpleEntrenamiento/tmpalto1.wav.npy =Alto
[(100, 'Alto'), (36, 'Ve al dado azul'), (36, 'Ve al dado rojo')]

Frase:simpleEntrenamiento/tmpalto2.wav.npy =Alto
[(100, 'Alto'), (36, 'Ve al dado azul'), (36, 'Ve al dado rojo')]

Frase:simpleEntrenamiento/tmpalto5.wav.npy =Alto
[(100, 'Alto'), (50, 'Busca el dado rojo'), (48, 'Saluda')]

Frase:simpleEntrenamiento/tmpavanza1.wav.npy =Avanza
[(100, 'Avanza'), (44, 'Retrocede'), (33, 'Ve a esponja')]

Frase:simpleEntrenamiento/tmpavanza3.wav.npy =Avanza
[(100, 'Avanza'), (36, 'Ve al dado rojo'), (32, 'Saluda')]

Frase:simpleEntrenamiento/tmpavanza6.wav.npy =Avanza
[(75, 'Avanza'), (50, 'Agarra el dado rojo'), (50, 'Agarra el dado azul')]

Frase:simpleEntrenamiento/tmpdadoa4.wav.npy =Busca el dado azul
[(100, 'Busca el dado azul'), (90, 'Busca el dado verde'), (80, 'Busca el dado rojo')]

Frase:simpleEntrenamiento/tmpdadoa5.wav.npy =Busca el dado azul
[(100, 'Busca el dado azul'), (84, 'Ve al dado azul'), (80, 'Busca el dado rojo')]

Frase:simpleEntrenamiento/tmpdador4.wav.npy =Busca el dado rojo
[(100, 'Busca el dado rojo'), (90, 'Agarra el dado rojo'), (80, 'Busca el dado azul')]

Frase:simpleEntrenamiento/tmpdador6.wav.npy =Busca el dado rojo
[(100, 'Busca el dado rojo'), (84, 'Ve al dado rojo'), (80, 'Agarra el dado rojo')]

Frase:simpleEntrenamiento/tmpdadov4.wav.npy =Busca el dado verde
[(100, 'Busca el dado verde'), (90, 'Busca el dado azul'), (80, 'Busca el dado rojo')]

Frase:simpleEntrenamiento/tmpdadov5.wav.npy =Busca el dado verde
[(90, 'Busca el dado verde'), (84, 'Ve al dado verde'), (80, 'Busca el dado azul')]

Frase:simpleEntrenamiento/tmpesponja1.wav.npy =Busca la esponja
[(99, 'Busca la esponja'), (88, 'Ve a esponja'), (77, 'Agarra la esponja')]

Frase:simpleEntrenamiento/tmpesponja4P.wav.npy =Busca la esponja
[(90, 'Busca la esponja'), (88, 'Ve a esponja'), (66, 'Agarra la esponja')]

Frase:simpleEntrenamiento/tmpesponja5.wav.npy =Busca la esponja
[(99, 'Busca la esponja'), (77, 'Ve a esponja'), (66, 'Agarra la esponja')]

Frase:simpleEntrenamiento/tmpgder2.wav.npy =Gira ala derecha
[(90, 'Gira a la derecha'), (64, 'Saluda'), (50, 'Alto')]

Frase:simpleEntrenamiento/tmpgder4P.wav.npy =Gira a la derecha
[(90, 'Gira a la derecha'), (48, 'Gira a la izquierda'), (32, 'Saluda')]

Frase:simpleEntrenamiento/tmpgder4.wav.npy =Gira a la derecha
[(100, 'Gira a la derecha'), (48, 'Gira a la izquierda'), (45, 'Ve a zona naranja')]

Frase:simpleEntrenamiento/tmpgder6.wav.npy =Gira a la derecha
[(100, 'Gira a la derecha'), (48, 'Gira a la izquierda'), (44, 'Ve a zona negra')]

Frase:simpleEntrenamiento/tmpgizq1.wav.npy =Gira a la izquierda

[(80, 'Gira a la izquierda'), (50, 'Alto'), (50, 'Gira a la derecha')]
Frase:simpleEntrenamiento/tmpgizq4P.wav.npy =Gira a la izquierda
[(96, 'Gira a la izquierda'), (72, 'Ve al dado rojo'), (60, 'Gira a la derecha')]
Frase:simpleEntrenamiento/tmpgizq5.wav.npy =Gira a la izquierda
[(96, 'Gira a la izquierda'), (60, 'Gira a la derecha'), (50, 'Alto')]
Frase:simpleEntrenamiento/tmpretrocede1.wav.npy =Retrocede
[(99, 'Retrocede'), (48, 'Saluda'), (48, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpretrocede2.wav.npy =Retrocede
[(99, 'Retrocede'), (32, 'Saluda'), (24, 'Ve a zona amarilla')]
Frase:simpleEntrenamiento/tmpretrocede6.wav.npy =Retrocede
[(99, 'Retrocede'), (32, 'Saluda'), (30, 'Agarra el dado rojo')]
Frase:simpleEntrenamiento/tmpsaluda2.wav.npy =Saluda
[(96, 'Saluda'), (33, 'Ve a esponja'), (33, 'Ve a zona negra')]
Frase:simpleEntrenamiento/tmpsaluda3.wav.npy =Saluda
[(96, 'Saluda'), (24, 'Ve a zona amarilla'), (24, 'Ve al dado azul')]
Frase:simpleEntrenamiento/tmpsaluda5.wav.npy =Saluda
[(96, 'Saluda'), (24, 'Gira a la izquierda'), (24, 'Ve a zona amarilla')]
Frase:simpleEntrenamiento/tmpveam1.wav.npy =Ve a zona amarilla
[(96, 'Ve a zona amarilla'), (66, 'Ve a esponja'), (66, 'Ve a zona negra')]
Frase:simpleEntrenamiento/tmpveam2.wav.npy =Ve a zona amarilla
[(80, 'Ve a zona amarilla'), (66, 'Ve a zona negra'), (63, 'Ve a zona naranja')]
Frase:simpleEntrenamiento/tmpveam4P.wav.npy =Ve a zona amarilla
[(81, 'Ve a zona naranja'), (77, 'Ve a zona negra'), (72, 'Ve a zona amarilla')]
Frase:simpleEntrenamiento/tmpveam5.wav.npy =Ve a zona amarilla
[(72, 'Ve a zona amarilla'), (66, 'Ve a zona negra'), (60, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpvedada3.wav.npy =Ve a dado azul
[(96, 'Ve al dado azul'), (84, 'Ve al dado rojo'), (72, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpvedada4P.wav.npy =Ve a dado azul
[(96, 'Ve al dado azul'), (72, 'Ve al dado rojo'), (72, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpvedada4.wav.npy =Ve a dado azul
[(96, 'Ve al dado azul'), (90, 'Busca el dado azul'), (72, 'Ve al dado rojo')]
Frase:simpleEntrenamiento/tmpvedada6.wav.npy =Ve a dado azul
[(96, 'Ve al dado azul'), (72, 'Ve al dado rojo'), (72, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpvedadr1.wav.npy =Ve al dado rojo
[(96, 'Ve al dado rojo'), (84, 'Ve al dado azul'), (72, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpvedadr3.wav.npy =Ve al dado rojo
[(96, 'Ve al dado rojo'), (84, 'Ve al dado azul'), (84, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpvedadr5.wav.npy =Ve al dado rojo
[(96, 'Ve al dado rojo'), (72, 'Ve al dado azul'), (72, 'Ve al dado verde')]
Frase:simpleEntrenamiento/tmpvedav1.wav.npy =Ve al dado rojo
[(96, 'Ve al dado verde'), (72, 'Ve al dado azul'), (72, 'Ve al dado rojo')]

Frase:simpleEntrenamiento/tmpvedav2.wav.npy =Ve al dado verde
[(96, 'Ve al dado verde'), (80, 'Busca el dado verde'), (72, 'Ve al dado azul')]

Frase:simpleEntrenamiento/tmpveesp2.wav.npy =Ve al dado verde
[(99, 'Ve a esponja'), (66, 'Agarra la esponja'), (63, 'Busca la esponja')]

Frase:simpleEntrenamiento/tmpveesp3.wav.npy =Ve a la esponja
[(99, 'Ve a esponja'), (63, 'Busca la esponja'), (60, 'Ve al dado azul')]

Frase:simpleEntrenamiento/tmpvena3.wav.npy =Ve a la esponja
[(99, 'Ve a zona naranja'), (66, 'Ve a esponja'), (66, 'Ve a zona negra')]

Frase:simpleEntrenamiento/tmpvena4.wav.npy =Ve a la esponja
[(99, 'Ve a zona naranja'), (66, 'Ve a zona negra'), (56, 'Ve a zona amarilla')]

Frase:simpleEntrenamiento/tmpveneg2.wav.npy =Ve a la zona naranja
[(99, 'Ve a zona negra'), (60, 'Ve al dado verde'), (56, 'Ve a zona amarilla')]

Frase:simpleEntrenamiento/tmpveneg3.wav.npy =Ve a la zona naranja
[(99, 'Ve a zona negra'), (56, 'Ve a zona amarilla'), (54, 'Ve a zona naranja')]

Frase:simpleEntrenamiento/tmpveneg4.wav.npy =Ve a zona negra
[(99, 'Ve a zona negra'), (72, 'Ve a zona naranja'), (56, 'Ve a zona amarilla')]

Frase:simpleEntrenamiento/tmpveneg6.wav.npy =Ve a zona negra
[(99, 'Ve a zona negra'), (60, 'Busca el dado azul'), (60, 'Ve al dado azul')]

Bibliografía

[1] Li Deng and Dong Yu. Deep learning: Methods and applications. Technical report, Microsoft Research, May 2014. URL https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/.

[2] Roger Parloff. Why deep learning is suddenly changing your life. http://fortune.com/ai-artificial-intelligence-deep-machine-learning/, Septiembre 2016. En linea;consultado el 20 de agosto del 2017.

[3] Lawrence Rabiner. Speech recognition in machines. http://ato.ms/MITECS/Entry/rabiner.html. En linea; Accedido el 20 de agosto del 2017.

[4] Amandeep Ummat Jaspreet Kaur Navjot Kaur Iqbaldeep Kaur, Navneet Kaur. Automatic speech recognition: A review. *International Journal of Computer Science And Technology*, 7:43–49, 2016. ISSN ISSN : 0976-8491 (Online) , ISSN : 2229-4333 (Print). consultado en linea http://www.ijcst.com/vol74/1/10-iqbaldeep-kaur.pdf.

[5] Chris Adams. Types of voice recognition. https://www.thoughtco.com/types-of-voice-recognition-1205856, June 2015.

[6] Recognize-speech. http://recognize-speech.com/. En linea; consultado el 26 de agosto del 2017.

[7] Michael Lutter. Feature extraction. http://recognize-speech.com/feature-extraction, Enero 2015. En linea; consultado el 26 de agosto del 2017.

[8] Hynek Hermansky. Perceptual linear predictive (PLP) analysis of speech. http://seed.ucsd.edu/mediawiki/images/5/5c/PLP.pdf, 1989. En linea;Accedido 4 de abril del 2017.

[9] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK book*. 2006.

[10] Mark Gales and Steve Young. The application of hidden markov models in speech recognition. *Foundations and trends in signal processing*, 1(3): 195–304, 2008.

[11] Sonido. https://es.wikipedia.org/wiki/Sonido. En linea; accedido el 20 de agosto del 2017.

[12] Fono. https://es.wikipedia.org/wiki/Fono, 2017. En linea;Accedido 4 de abril del 2017.

[13] R Thangarajan, AM Natarajan, and M Selvam. Word and triphone based approaches in continuous speech recognition for tamil language. *WSEAS transactions on signal processing*, 4(3):76–86, 2008.

[14] Tomáš Pavelka and Kamil Ekštein. A comparison of acoustic models based on neural networks and gaussian mixtures. *Text, Speech & Dialogue (9783642042072)*, page 291, 2009. ISSN 9783642042072. URL http://pbidi.unam.mx:8080/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=76739836&lang=es&site=eds-live.

[15] Linear prediction. https://en.wikipedia.org/wiki/Linear_prediction, 2017. En linea;Accedido 4 de abril del 2017.

[16] Transformada de Fourier. https://es.wikipedia.org/wiki/Transformada_de_Fourier. En linea;Accedido el 23 de julio del 2017.

[17] Steven's power law. https://en.wikipedia.org/wiki/Stevens%27s_power_law, 2017. En linea;Accedido 4 de abril del 2017.

[18] B. B. Vachhani and H. A. Patil. Use of plp cepstral features for phonetic segmentation. In *2013 International Conference on Asian Language Processing*, pages 143–146, Aug 2013. doi: 10.1109/IALP.2013.47.

[19] Mfcc. https://es.wikipedia.org/wiki/MFCC, . En linea; Consultado el 4 de septiembre del 2017.

[20] S.S. Haykin. *Neural Networks and Learning Machines*. Neural networks and learning machines. Prentice Hall, 2009. ISBN 9780131471399. URL https://books.google.com.mx/books?id=K7P36lKzI_QC.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[22] Kaiming He Xiaoou Tang Chao Dong, Chen Change Loy. Image super-resolution using deep convolutional networks. *Proceedings of European Conference on Computer Vision (ECCV)*, 2014.

[23] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc., 2006. ISBN 0387310738. URL http://www.ebook.de/de/product/5324937/christopher_m_bishop_pattern_recognition_and_machine_learning.html.

[24] What is the intuition behind softmax function? https://www.quora.com/What-is-the-intuition-behind-SoftMax-function. En linea;accedido el 18 de agosto del 2017.

[25] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *ICASSP*, pages 8609–8613, 2013.

[26] Stuart Russell, Peter Norvig, and Artificial Intelligence. *Artificial Intelligence a modern approach*. Pearson, 1995.

[27] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

[28] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[29] Measuring performance. https://www.youtube.com/watch?v=--E5qo_XnXo. En linea, consultado 17 de agosto del 2017. Forma parte de un curso en linea que puede consultarse en https://www.udacity.com/course/ud730.

[30] Nikhil Buduma. Data science 101: Preventing overfitting in neural networks. http://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html/2. En linea; accedido el 17 de Agosto del 2017.

[31] Categorical variable. https://en.wikipedia.org/wiki/Categorical_variable. En linea; consultado el 18 de agosto del 2017.

[32] Jason Brownlee. Why one-hot encode data in machine learning? http://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/. En linea; accedido el 19 de agosto del 2017.

[33] Propiedad de Márkov. https://es.wikipedia.org/wiki/Propiedad_de_M%C3%A1rkov, . En linea;Accedido 4 de abril del 2017.

[34] Proceso de Márkov. https://es.wikipedia.org/wiki/Proceso_de_M%C3%A1rkov, . En linea;Accedido 4 de abril del 2017.

[35] Markov chain. https://en.wikipedia.org/wiki/Markov_chain. En linea;Accedido 4 de abril del 2017.

[36] Doroteo Torre Toledano y Joaquín González Rodríguez. Notas sobre HMMs. http://arantxa.ii.uam.es/~jortega/HMMs_ASAL.pdf, 2017. En linea;Accedido 15 de Junio del 2017.

[37] Algoritmo de Baum-Welch. https://es.wikipedia.org/wiki/Algoritmo_de_Baum-Welch, 2017. En linea;Accedido el 23 de julio del 2017.

[38] Michael Tros. Context-dependent deep neural networks: Breakthrough of neural networks for speech recognition. http://recognize-speech.com/acoustic-model/knn/comparing-different-architectures/context-dependent-deep-neural-networks, 2015.

[39] Researchgate. https://www.researchgate.net/post/Is_there_an_ideal_ratio_between_a_training_set_and_validation_set_Which_trade-off_would_you_suggest, 2017. En linea; Accedido 14 de Agosto del 2017.

[40] Computer Science Department Stanford University. Autoencoders. http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/, . En linea; Consultado el 17 de agosto del 2017.

[41] Autoencoder. https://en.wikipedia.org/wiki/Autoencoder, . En linea; Accedido el 17 de agosto del 2017.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[43] Computer Science Department Stanford University. Stacked autoencoders. http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders, . En linea; Consultado 17 de agosto del 2017.

[44] Francois Chollet. Building autoencoders in keras. https://blog.keras.io/building-autoencoders-in-keras.html, May 2016.

[45] Kai Ming Ting. *Confusion Matrix*, pages 209–209. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_157. URL https://doi.org/10.1007/978-0-387-30164-8_157.

[46] Kai Ming Ting. *Precision and Recall*, pages 781–781. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_652. URL https://doi.org/10.1007/978-0-387-30164-8_652.

[47] Computing precision and recall for multi-class classification problems. http://text-analytics101.rxnlp.com/2014/10/computing-precision-and-recall-for.html, Octubre 2014.

[48] Tensorflow installation. https://www.tensorflow.org/install/. En linea;Accedido 4 de abril del 2017.