UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Predicción de propiedades termodinámicas en fluidos utilizando técnicas de aprendizaje de máquinas

TESIS
QUE PARA OPTAR POR EL GRADO DE
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
Jorge Arturo Rodríguez Horcasitas

TUTOR O TUTORES PRINCIPALES
Dr. Gibran Fuentes Pineda IIMAS
Dr. Martín Salinas Vázquez INSTITUTO DE INGENIERÍA

CIUDAD UNIVERSITARIA, CIUDAD DE MÉXICO, ENERO 2019.

Resumen

La dinámica de fluidos computacionales o DFC es un área que se encuentra en la intersección de: las matemáticas aplicadas, las ciencias de la computación, la mecánica de fluidos y la ingeniería aeronáutica¡ se encarga de analizar y resolver problemas sobre el comportamiento de los fluidos. A pesar de los grandes avances tecnológicos y una mayor capacidad de cómputo derivada de éstos, aún resulta computacionalmente extenuante generar simulaciones con un alto grado de exactitud, por lo que es común el surgimiento de nuevos e innovadores algoritmos para combatir este problema.

Por otra parte, en el área de ciencias de la computación, el modelo de redes neuronales ha demostrado ser apropiado para una gran y diversa cantidad de tareas en diferentes dominios debido, entre otras cosas, a su propiedad como aproximador universal de funciones [34, 66] y a su intrínseca compatibilidad con tecnologías de cómputo en paralelo.

En esta tesis se hace una exploración y un análisis de diversos algoritmos basados en redes neuronales en tareas relacionadas a la DFC. De manera más específica, se examina el desempeño que éstos tienen en la simulación de fluidos compresibles e incompresibles en donde la tarea puntual yace en la predicción de propiedades termodinámicas que definen el estado actual del sistema.

Índice general

Índice de figuras

Índice de tablas

Lista de Abreviaturas

AL Álgebra Lineal.

AM Aprendizaje de Máquinas.

AP Aprendizaje Profundo.

DFC Dinámica de Fluidos Computacionales.

FS Fluido Supercrítico.

IA Inteligencia Artificial.

RN Redes Neuronales.

RNC Redes Neuronales Convolucionales.

RNM Red Neuronal Multicapa.

Notación

Números y Arreglos

a	Un escalar
$\boldsymbol{a}$	Un vector
$\boldsymbol{A}$	Una matriz
$\mathbf{A}$	Un tensor

Conjuntos

$\mathbb{R}$	El conjunto de los números reales
$\mathbb{C}$	El conjunto de los números complejos
$\{0, 1\}$	El conjunto que contiene a 0 y 1
$\{0, 1, ..., n\}$	El conjunto de todos los enteros entre 0 y n

Índices

a_i	Elemento i del vector $\boldsymbol{a}$
$A_{i,j}$	Elemento i, j de la matriz $\boldsymbol{A}$
A_i	Fila i de la matriz $\boldsymbol{A}$

Álgebra Lineal

$\boldsymbol{AB}$	Multiplicación de la matriz $\boldsymbol{A}$ por la matriz $\boldsymbol{B}$
$\boldsymbol{A^T}$	Matriz transpuesta de $\boldsymbol{A}$

Cálculo

dx	Diferencial de x

$\frac{dy}{dx}$	Derivada de y con respecto a x
$\frac{\partial y}{\partial x}$	Derivada parcial de y con respecto a x
∇y	Gradiente de y
$\nabla \cdot y$	Divergencia de y
$\nabla^2 y$	Laplaciano de y

Capítulo 1

Introducción

1.1. Perspectiva histórica de la DFC

La búsqueda de un modelo que represente fielmente la dinámica de fluidos tiene una larga historia. En 1822, Claude-Louis Navier y en 1845 George Stokes formularon, de manera independiente y por vez primera, las ecuaciones que describen la dinámica de los fluidos [51], llamadas actualmente ecuaciones de Navier-Stokes. Éstas fueron derivadas de tres principios de conservación en la física. El primero de éstos dicta que la tasa de cambio de momento en una partícula de fluido es igual a la suma de las fuerzas que actúan sobre ésta [73] i.e. La Segunda Ley de Newton. El segundo principio se expresa por la ecuación de continuidad y describe la conservación de masa. Finalmente, el último principio es derivado de la primera ley de la termodinámica y describe que la tasa de cambio de energía es igual a la suma de la tasa de adición de calor y de la tasa de trabajo hecha en una partícula de fluido [73], a ésta última comúnmente se le conoce como ecuación de estado.

Durante el siglo XIX, las aplicaciones derivadas de las ecuaciones de Navier-Stokes fueron restringidas a problemas simplificados y de topología sencilla, en aras de poder obtener soluciones analíticas; esto debido a que el esfuerzo que conlleva la aplicación de métodos numéricos en forma manual es intratable incluso en problemas bidimensionales.

Una de la primeras contribuciones al área de la DFC data de 1910, en donde L.F. Richardson [58] introduce el método de diferencias finitas para la aproximación de soluciones aritméticas. Su trabajo descansa sobre computadoras humanas a una velocidad de 2,000 operaciones por semana durante tres años [79]. Progresivamente, la simulación de fluidos por computadora se comenzó a desarrollar durante las

primeras décadas del siglo XX; sin embargo, debido a la baja capacidad computacional de aquellos años, ésta se concentró en aplicaciones bidimensionales.

Los grandes avances en la DFC fueron relegados hasta el advenimiento de los semiconductores en la década de los 40 gracias a un dramático incremento en el poder de cómputo [79]; que entre otras cosas, dio como resultado el nacimiento de las primeras supercomputadoras. Una de éstas, la CDC 6,600 fungió como buque insignia de esta nueva generación de ordenadores y podía realizar hasta 3 millones de operaciones de punto flotante por segundo, lo que la mantuvo como la computadora más rápida de 1964 a 1969; como consecuencia, surgieron las primeras simulaciones de fluidos tridimensionales. Aunado a esto, se establecieron firmemente algunos principios y algoritmos de la DFC (flujo potencial, ecuaciones de Euler, RANS) [36].

De manera histórica, el desarrollo de la DFC en la década de 1970 fue ocasionado por las necesidades de la comunidad aeroespacial [3], en donde el crecimiento fue nuevamente impulsado por un aumento en la capacidad computacional. Durante estos años, surgió el método de volúmenes [33]; adicionalmente, apareció el algoritmo SIMPLE que proporciona una solución iterativa a las ecuaciones de Navier-Stokes y el modelo $k - \epsilon$ para el tratamiento de fluidos con turbulencia. Ambos son aún usados hoy en día por programas comerciales para la simulación de fluidos [79].

La transición del nivel teórico al práctico ocurrió durante la década de los ochentas y noventas, gracias al surgimiento de códigos comerciales flexibles. Se eliminó en cierta medida la necesidad de desarrollar código privado en las empresas y, por consiguiente, aumentó el número de firmas que se valían de la DFC para el desarrollo de proyectos [79].

Desde su nacimiento, la DFC ha planteado grandes retos en cuanto a tratabilidad computacional debido a la enorme cantidad de operaciones necesarias para la simulación de un fluido [29]; poniéndolo en perspectiva, algunas simulaciones pueden llegar a requerir varios meses para alcanzar el estado estacionario. El problema se vuelve notorio cuando entendemos la naturaleza iterativa que ocurre en la industria durante el desarrollo de un producto; más aún, encontramos aplicaciones que requieren de una simulación de fluidos en tiempo real, como es el caso de los entornos virtuales [76], el de los videojuegos y en la supervisión y monitoreo de sistemas dinámicos. En los últimos años se han propuesto nuevas metodologías y algoritmos en pos de resolver este problema [77] [69].

1.2. Conceptos básicos de fluidos

Los fluidos se clasifican en compresibles e incompresibles; los primeros son aquellos cuyo volumen puede variar mientras que los segundos en los que permanece constante [8]. En la realidad, todos los fluidos cambian en cierto grado su volumen, pero el cambio de éste es tan pequeño que para una gran cantidad de aplicaciones (e.g. animación por computadora) se puede hacer una buena aproximación tratándolos como incompresibles. Por otra parte, no es recomendable hacer esta aproximación si se trabaja con escenarios en donde el volumen del fluido varía considerablemente, como en el caso de un estampido sónico o de ondas de choque [8].

En la simulación de fluidos existen dos enfoques principales para especificar el campo de flujo, estos son, el marco de referencia Euleriano y el Lagrangiano [51].

- En la perspectiva Euleriana, la especificación de un campo de flujo es representada como función de la posición $\boldsymbol{x}$ y del tiempo t; e.g. la velocidad de flujo puede ser escrita como $\boldsymbol{u}(\boldsymbol{x}, t)$; de manera abstracta, podemos decir que es un manera de ver el movimiento del fluido que se enfoca en posiciones específicas del espacio en las que el fluido se desplaza con el pasar del tiempo [5].
- Por otra parte, en la especificación Lagrangiana se sigue a parcelas individuales de fluido a través del tiempo. Para identificar parcelas específicas se utiliza un campo vectorial $\boldsymbol{a}$ independiente del tiempo t, el cual es normalmente definido en $t = 0$ [57]. De esta manera podemos escribir la posición del flujo como $\boldsymbol{x}(\boldsymbol{a}, t)$.

Finalmente, cabe mencionar que existen ciertas aplicaciones en donde las condiciones extremas de presión y de temperatura llevan a trabajar con fluidos supercríticos o FS. Éstos se caracterizan por no tener una separación de rasgos determinante entre la fase líquida y la sólida, i.e. pueden disolver materiales como los líquidos y efusionar como un gas. Una sustancia es catalogada como FS una vez que sus condiciones de presión y temperatura son superiores a su punto crítico (ver figura 1.1 [6]).

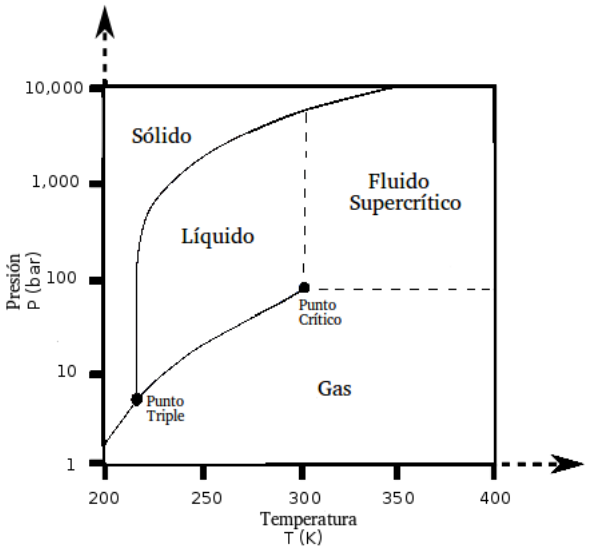

Figura 1.1: Diagrama de fases de presión y temperatura mostrando el punto crítico del dióxido de carbono

1.3. Conceptos básicos de inteligencia artificial

Las Ciencias de la Computación son el campo que se dedica al estudio sistemático de algoritmos y estructuras de datos [19]. Dentro de ésta, se le conoce comúnmente como inteligencia artificial o IA al subcampo encargado de resolver tareas que cotidianamente realizan los humanos, e.g. comprensión, movimiento, reconocimiento de imágenes, capacidad de comunicación, traducción, trabajo creativo, negociación, etc [60].

Dentro de la IA está el área de aprendizaje de máquinas o AM, el cual se encarga del estudio de algoritmos que aprenden de la experiencia o de los datos [11], dicho de otro modo, se busca que a partir de un conjunto de datos y un problema a resolver, se encuentre de manera autónoma (sin participación del programador) una solución a éste. Finalmente, el aprendizaje profundo o AP es una sección del AM dedicada al estudio e implementación de algoritmos modulares, i.e. métodos capaces de estructurarse en cascadas o capas en donde cada una de éstas representa una abstracción del problema inicial [13, 72]. La figura 1.2 muestra un esquema de las áreas anteriormente mencionadas.

"De manera formal, se dice que una máquina aprende con respecto a una tarea T, una métrica de desempeño D y una experiencia E si el sistema de manera

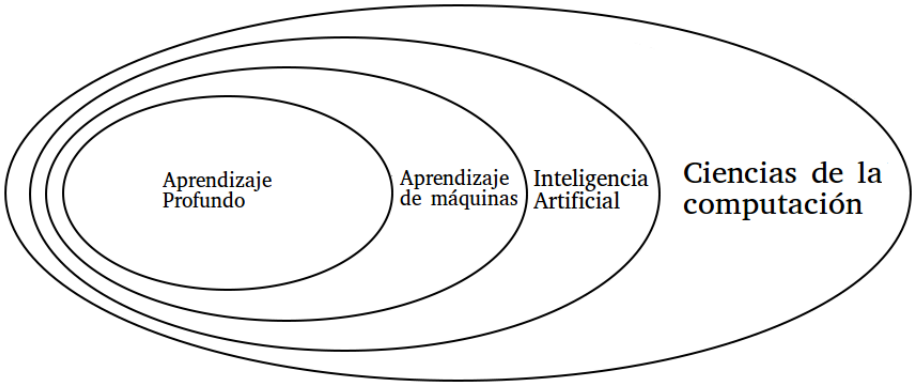

Figura 1.2: Diagrama con relación jerárquica de disciplinas en inteligencia artificial.

contundente mejora su desempeño D en la tarea T después de una experiencia E"[48]. Además, podemos tener dos tipos principales de aprendizaje: supervisado y no supervisado.

En el aprendizaje supervisado, el objetivo es aprender una función de mapeo que convierta un vector de entradas $\boldsymbol{x_i}$ a uno de salidas $\boldsymbol{y_i}$ dado un conjunto de datos $D = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{i=1}^{N}$. En el aprendizaje no supervisado, la entrada del algoritmo únicamente es el conjunto de datos $D = \{\boldsymbol{x_i}\}_{i=1}^{N}$, y el objetivo es encontrar patrones que sean considerados más que ruido estructurado [52].

Durante el proceso de entrenamiento, una práctica común e importante es generar una partición de D en dos subconjuntos denominados entrenamiento y prueba. El primero de éstos se utiliza para entrenar el modelo y el segundo para validar los resultados con miras a demostrar que el algoritmo es capaz de generalizar el conocimiento aprendido y procesar entradas nunca antes vistas.

En esta tesis, se hace uso de redes neuronales o RN y de redes neuronales convolucionales o RNC, ambas forman parte del campo de aprendizaje de máquinas y normalmente son entrenadas a través del algoritmo del descenso del gradiente. La forma de medir el desempeño de estos métodos es a través de una función de costo que evalúa la distancia que existe entre el valor actual y el deseado.

1.4. Justificación

Gracias a los recientes avances tecnológicos y a una mayor capacidad de cómputo, la DFC es usada rutinariamente en una gran cantidad de áreas y disciplinas [12], algunas de las más relevantes son: ingeniería aeroespacial, diseño de edificios, industria química, industria de alimentos, medicina, ingeniería nuclear, ingeniería petroquímica, ingeniería naval, ingeniería automotriz, sistemas de aguas residuales y plantas de energía. De manera general, las principales razones para utilizar la DFC en la industria son:

- Reducción en el precio del producto derivado de una disminución en el costo de la etapa de desarrollo.
- Generación y solución de modelos de optimización.
- Reducción de la necesidad de experimentos físicos.
- Tiempo de lanzamiento al mercado disminuido.
- Mejora en diseño del producto.

En pos de llegar a una apreciación de la utilidad de la DFC en la industria, se analizarán algunos de los casos más importantes.

En la industria aeronáutica, el uso de la DFC en el diseño de aeronaves comerciales está bien documentado [65]. La fortaleza de la DFC yace en su habilidad para generar datos de vuelo a través de simulaciones por computadora a un bajo costo en comparación con pruebas en túneles de viento y de vuelos en tiempo real; además, es común su aplicación en problemas de minimización e.g. encontrar la geometría ideal de un perfil alar para reducir el arrastre o maximizar la sustentación [30].

La aplicación de la DFC es relativamente reciente en el diseño de edificios. El objetivo es crear ambientes térmicamente confortables y saludables haciendo uso óptimo de la energía. Además, existen importantes intereses económicos y ambientales; por ejemplo, en Estados Unidos los edificios son responsables de un tercio de la energía primaria utilizada y dos tercios de toda la energía consumida [1].

Existen varias áreas de oportunidad en el interior de un edificio. Los sistemas de aire acondicionado, por ejemplo, pueden ser optimizados para maximizar el confort de sus ocupantes. Para los sistemas de control y dispersión de contaminantes

se pueden encontrar las particiones de cuartos, diseño de pasillos y localización de muebles que favorezcan a sus ocupantes; finalmente, se puede hacer un estudio de la ventilación natural del edificio [78].

La DFC históricamente ha otorgado gran valor a la industria automotriz; por ejemplo, ha sido utilizada como herramienta auxiliar durante el diseño de automóviles y ha permitido una reducción en los costos de producción [14]. De manera concreta, algunas de las aplicaciones más importantes de la DFC son: estudio de la dinámica exterior de la carrocería, simulación del proceso de combustión, diseño del control de clima, análisis del enfriamiento del motor, sistemas de escape, maquinaria rotacional (ventiladores, convertidores de torque, frenos de disco), enfriamiento y protección del capó, entre otras. Existen dos retos principales en esta industria: la simulación precisa de fluidos y la obtención rápida de resultados.

Menos intuitivo pero igualmente relevante, es la aplicación de la DFC en la industria de la comida, en ésta, las áreas más relevantes de aplicación son: procesos de mezclas, deshidratación, cocinado, esterilización y almacenamiento en frío. El objetivo principal es la mejora de la calidad, la seguridad y el tiempo de vida de la comida en almacén [74].

La medicina puede utilizar la DFC para la prevención y el tratamiento de enfermedades; por ejemplo, es posible definir las condiciones de estrés y fuerza suficientes para la ruptura de una vena o arteria, lo cual puede resultar en un aneurisma. También puede utilizarla como una herramienta para la exploración rápida de hipótesis, lo cual resulta en una investigación dinámica [59].

Las anteriores aplicaciones no son sino un pequeño subconjunto del total; más aún, todas están relacionadas con importantes actividades o con el bienestar humano. Por esto, es indudable que el entendimiento y simulación de fluidos ha traído grandes beneficios a la sociedad.

1.5. Planteamiento del problema

En esta tesis se busca estimar la densidad y la presión de un flujo compresible y uno incompresible respectivamente. Por lo tanto, se planteó la tarea como un problema de regresión.

Para el desarrollo del trabajo con respecto al fluido compresible se colaboró con el

alumno de doctorado de la UNAM Christian Lagarza, quien suministro el código para la generación de simulaciones utilizado en esta tesis. De manera específica, el fluido con el que se trata se encuentra en estado supercrítico y es estudiado para entender el comportamiento del proceso de combustión en cohetes de combustión líquida [61].

En éste, el cálculo más costoso durante cada ciclo de la iteración se da en el cómputo de la densidad. Para computar esta, se utiliza la ecuación de estado de Peng-Robinson [55]. De manera explícita la podemos escribir como:

$$P = \frac{RT}{v-b} - \frac{\theta\left(T\right)}{v^2 + d_1 b v + d_2 b^2} \tag{1.1}$$

Donde P es la presión, T la temperatura, R la constante universal de los gases, v el volumen molar, $\theta\left(T\right)$ y b son parámetros computados con respecto a la temperatura y presión crítica y $(d_1, d_2) = (2, -1)$. Definiendo $D(v) = v^2 + d_1 b v + d_2 b^2$ podemos reescribir (1.1) como:

$$P = \frac{RT}{v-b} - \frac{\theta\left(T\right)}{D(v)} \tag{1.2}$$

Multiplicando la ecuación (1.2) por $(v-b)$ y por $D(v)$ se genera un polinomio de tercer grado sobre el volumen molar con la siguiente forma:

$$f(x) = a_0 v^3 + a_1 v^2 + a_2 v + a_3 = \sum_{k=0}^{n=3} a_{n-k} v^k \tag{1.3}$$

En donde $v, a_i \in R$. De manera explícita, podemos escribir los coeficientes de (1.3) de la siguiente forma:

$$a_0 = P \tag{1.4}$$
$$a_1 = P d_1 b - (RT - b) \tag{1.5}$$
$$a_2 = P d_2 b^2 - (RT - b)\, d_1 b + \theta \tag{1.6}$$
$$a_3 = -b\left[\theta + (RT - b)\, d_2 b\right] \tag{1.7}$$

Podemos encontrar la densidad dividiendo la masa molar W entre el volumen molar. Esto es:

$$\rho = \frac{W}{v} \tag{1.8}$$

Si dividimos (1.3) entre a_0 obtenemos:

$$g(v) = v^3 + b_0 v^2 + b_1 v + b_2 \tag{1.9}$$

En donde:

$$b_0 = \frac{a_1}{a_0} \tag{1.10}$$

$$b_0 = \frac{a_2}{a_0} \tag{1.11}$$

$$b_0 = \frac{a_3}{a_0} \tag{1.12}$$

Debido a que las raíces de (1.3) son las mismas que las de (1.9), podemos trasladar el problema hacia la solución de esta última. Además, si definimos el vector de raíces de (1.9) como $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \lambda_3]^T$, podemos reescribir $g(v)$ como:

$$g(v) = \prod_{i=1}^{n=3} (x - \lambda_i) \tag{1.13}$$

El problema se puede resolver utilizando técnicas de AM al encontrar las raíces de (1.13) a través de una RN; más aún, podemos incorporar restricciones matemáticas sobre la RN para facilitar la tarea de aprendizaje. Éstas últimas representan información adicional del problema, y en términos generales, pueden ser descritas como relaciones o limitantes teóricas sobre los valores que puede tomar una variable.

Por otra parte, para un fluido incompresible dentro del marco de especificación Euleriano la variable que requiere mayor cómputo es la presión [69]. En la figura 1.3 se muestra una secuencia de pasos básica que normalmente se toma para la simulación de fluidos incompresibles [77]. Durante la iteración del algoritmo, el cálculo de la función Proyección() toma alrededor del 85 % del tiempo del ciclo (medición empírica sobre simulaciones personales para un fluido incompresible no viscoso utilizando el método del gradiente conjugado precondicionado [4]); en ésta, se resuelve la ecuación de Poisson y se computa la presión para posteriormente calcular la velocidad.

Tomando en cuenta esto, podemos proponer el cómputo de la presión a través de un algoritmo de AM cuyas entradas sean:

1. La topología del problema representada por un arreglo de valores binarios.
2. La velocidad de advección.
3. La presión en el instante de tiempo anterior.

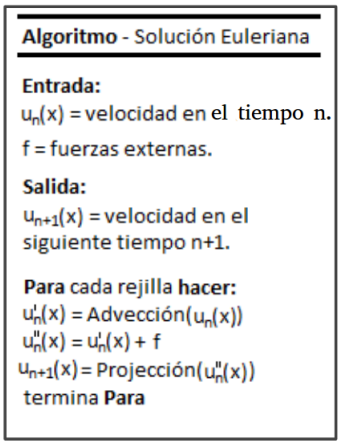

Figura 1.3: Ejemplo del algoritmo para la simulación de un fluido incompresible

Poniéndolo en perspectiva, el diagrama de la figura 1.4 detalla el procedimiento a seguir. En éste, el bloque Red Neuronal Convolucional es usado para el computo de la presión y sustituye al cómputo de la función Proyección().

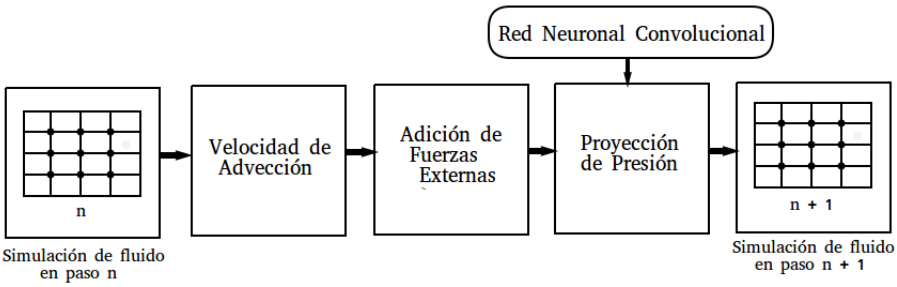

Figura 1.4: Diagrama de de flujo para la simulación de fluidos incompresible

1.6. Objetivos

Objetivo general

Implementar algoritmos de aprendizaje de máquinas en el campo de la DFC para la predicción de propiedades termodinámicas en fluidos compresibles e incompresibles, analizando el efecto de éstos en la precisión y el tiempo de cómputo demandado.

Objetivos específicos

- Entrenar una RN con restricciones capaz de predecir la densidad dentro de un espacio de simulación tridimensional para un fluido compresible.
- Generar una base de datos con simulaciones de fluidos incompresibles no viscosos de una fase en un espacio bidimensional a través de algoritmos clásicos pertenecientes al área de la DFC.
- Entrenar una RNC para realizar la predicción de la presión dentro de un espacio de simulación bidimensional para un fluido incompresible.
- Diseñar un algoritmo de aprendizaje innovador para el cálculo de la presión en un fluido incompresible tomando en cuenta que $\nabla \cdot \vec{u} = 0$.
- Comparar la eficiencia entre los métodos clásicos y los basados en AM, tomando en cuenta las condiciones específicas del flujo.
- Validar ambos modelos entrenados a través de datos obtenidos de la ejecución de simulaciones mediante métodos clásicos.

1.7. Hipótesis

Hipótesis general

Las RN y las RNC han sido usadas extensamente en visión por computadora; por ejemplo, en tareas de clasificación de imágenes debido a su gran desempeño. Sin embargo, también es posible ver este tipo de métodos en tareas de regresión como en el trabajo de C. Chen et al. [9], en donde se entrenó una RNC para inferir a través de una imagen: el ángulo del automóvil con respecto a la carretera, la distancia contra la marca de carril y el espacio entre diversos vehículos. De manera similar, Alexander Toshev y Christian Szegedy entrenaron una RNC capaz de localizar las articulaciones humanas [70], esto es, a partir de una imagen, la RNC computa las coordenadas de las articulaciones del cuerpo.

Los problemas de regresión en RN no se limitan a la estimación de un número pequeño de variables; por ejemplo, en el trabajo de K. Sirinukunwattana et al. [64] se entrena una RNC para la detección de los núcleos de células relacionadas con el cáncer de colon. Para esto, la entrada a la red es una imagen con el tejido celular a estudiar y la salida es otra imagen (de igual dimensión que la de entrada) en donde el valor de cada píxel representa la probabilidad de que éste sea el

centro de la célula en cuestión.

De esta manera, y tomando en cuenta la anterior evidencia, se postula la siguiente hipótesis:

La densidad en un fluido compresible y la presión en uno incompresible son propiedades termodinámicas computables a través de un modelo de aprendizaje de máquinas basado en redes neuronales.

Hipótesis específica

Uno de los resultados más importantes con respecto a las RN es el establecimiento teórico riguroso por parte de Kolmogorov [34] y A. Sprecher [66] acerca de las cualidades de éstas como aproximadores de funciones universales una vez que son propiamente entrenadas. Por esta razón, la selección de un algoritmo de entrenamiento es de vital importancia.

Uno de los algoritmos de aprendizaje más utilizados y eficientes es el del descenso del gradiente. Sin embargo, éste tiene importantes inconvenientes que han sido ampliamente estudiados; por ejemplo, en algunas aplicaciones puede tener una tasa de aprendizaje lenta y en general se corre el peligro de no converger a un mínimo global [42]; además, es posible enfrentar el problema de sobreajuste, esto es, presentar un buen desempeño únicamente sobre el conjunto de prueba.

Dimitris A. Karas y Stavros J. Perantonis presentaron una mejora al algoritmo del descenso del gradiente [31] que incorpora información a priori a través de restricciones utilizando el método de multiplicadores de Lagrange.

La principal razón para incorporar este tipo de restricciones es debido a que normalmente en el algoritmo del descenso del gradiente la función de costo es complicada y contiene regiones planas y depresiones profundas [27], lo que lleva en muchas ocasiones a soluciones no óptimas y a largos tiempos de entrenamiento.

Con base en estas observaciones, se postula la siguiente hipótesis:

El diseño de un algoritmo de AM basado en RN que incorpore restricciones en su algoritmo de aprendizaje puede generar mejores estimaciones (en este caso sobre las propiedades termodinámicas de un fluido) y adquirir una mayor capacidad de generalización que su homólogo no restringido.

1.8. Metodología

De manera general, podemos describir tres pasos fundamentales para la implementación de los modelos descritos en esta tesis:

1. Generar una simulación o un conjunto de simulaciones a través de algoritmos clásicos/tradicionales.
2. Utilizar los datos obtenidos en el paso 1 para el entrenamiento de los algoritmos. En este caso se usa una RN para el fluido compresible y una RNC para el incompresible.
3. Una vez entrenado el modelo, de ser necesario, se valida a través de un conjunto de datos de prueba.

1.9. Aportes

Los principales aportes de esta tesis son:

- El diseño de un algoritmo de aprendizaje innovador para la estimación de la presión de un fluido incompresible en un espacio de simulación bidimensional.
- El desarrollo de un modelo basado en RN para estimar la densidad de un fluido compresible en un espacio de simulación tridimensional.
- La creación de una base de datos de fluidos incompresibles utilizando diferentes geometrías y condiciones iniciales en un espacio de simulación bidimensional. Dicha base de datos se hará pública.
- Una referencia de diseño de algoritmos de AM para la simulación de fluidos. Actualmente existen muy pocos trabajos que funjan como puente entre estas dos áreas.

1.10. Organización de la tesis

La tesis se encuentra organizada de la siguiente manera:

- En el capítulo 2 se hace un resumen del estado del arte.
- En el capítulo 3 se explica a detalle el funcionamiento de las RN y las RNC.
- En el capítulo 4 se explica la teoría, el desarrollo y los resultados obtenidos del algoritmo de AM utilizado en la predicción de la densidad en un fluido compresible.
- En el capítulo 5 se presentan la teoría, el desarrollo y los resultados obtenidos del algoritmo de AM utilizado en la predicción de la presión en un fluido incompresible.
- En el capítulo 6 se presentan las conclusiones del trabajo así como un análisis para posibles investigaciones futuras.

Capítulo 2

Estado del arte

En este capítulo se analizan los principales y recientes trabajos relacionados con la presente tesis. En primer lugar, se muestran los avances de la DFC haciendo énfasis en los métodos impulsados por datos; es importante notar que la bibliografía sobre este tema es extensa y sólo se menciona lo que se consideró más relevante para esta tesis. También se muestran los trabajos que unen el poder generalizador de las RN con el álgebra linear y las ecuaciones diferenciales. Asimismo, se presentan los recientes esfuerzos que han surgido para incorporar las RN en las simulaciones de fluidos, esto es, trabajo relacionado con la física impulsada por datos. Finalmente se muestra el uso y resultados de las restricciones sobre el algoritmo del descenso del gradiete en las RN.

2.1. Avances en la DFC

En 1999 Stam et al. mostraron que los métodos Eulerianos pueden resolver las ecuaciones de Navier-Stokes de manera estable [67], en contraste con los Lagrangianos [51]. Años más tarde, los mismos autores sugirierón el uso del método iterativo denominado Gauss-Seidel para la solución de la ecuación de Poisson [68]. Aunque éste todavía es utilizado, está siendo remplazado por el método PCG publicado por N. Foster y R. Fedkiw en el 2001 [17]. A. McAdams et al. publicaron una técnica para acelerar el cómputo de la ecuación de Poisson en forma de un paso de pre-procesamiento paralelo [44]; sin embargo, es muy difícil de implementar y paralelizar [69].

Además de buscar una reducción en el tiempo de de cómputo de los algoritmos utilizados para resolver las ecuaciones de Navier-Stokes, se puede tratar de reducir la dimensión del problema; por ejemplo, Zhu et al. propusieron hacer una

malla de dimensiones irregulares en donde se disminuye el tamaño en los lugares de interés, mientras que se aumenta en aquellos poco relevantes [80]. De manera similar, Treuille et al. demostraron que es posible realizar una reducción del espacio de simulación, esto es, disminuir la dimensionalidad de los vectores que definen el estado actual del sistema a un espacio de representación [71]; lo que se busca es realizar operaciones sobre este vector reducido para posteriormente, a través de una transformación, regresar al espacio original. Finalmente, se han propuesto metodologías para la generación rápida pero poco precisa de simulaciones [49].

Por otra parte, recientemente se han desarrollado diferentes métodos para acelerar las simulaciones de fluidos a través de algoritmos impulsados por datos. Entre los mas importantes destaca el trabajo de M. Lentine et al. [43], en donde se generan nuevas simulaciones a través de otras previamente computadas utilizando métodos de interpolación. Otra idea importante es presentada nuevamente por A. Treuille et al. [71] en donde se utiliza la transformación de Garlekin para generar el cómputo de la dinámica del fluido a través de una combinación lineal de simulaciones pre-procesadas.

2.2. Redes neuronales en aplicaciones de álgebra lineal

El álgebra lineal o AL es una rama de las matemáticas que estudia las relaciones lineales de ecuaciones y su representación a través de matrices y espacios vectoriales; se encuentra normalmente presente en simulaciones numéricas de física, química, ingeniería, economía, finanzas, etc. Por la necesidad de computar resultados en estas áreas surgió el campo del álgebra lineal numérica, cuyo objetivo es el estudio de algoritmos para realizar cálculos de AL [2].

Desde un punto de vista teórico, varios problemas del AL están resueltos; sin embargo, el constante e impresionante progreso en el poder computacional ha transformado los problemas teóricos en prácticos. Es indispensable que los algoritmos encargados de computar operaciones de AL sean eficientes tanto en memoria como en tiempo de cómputo y estables (pequeños errores en la entrada del algoritmo generan pequeños errores en la salida de éste) [39].

Las RN han sido utilizadas de manera satisfactoria para atacar este problema. Por ejemplo, Zheru Chi et al. [26] diseñaron una red neuronal capaz de resolver

un sistema de ecuaciones lineales; Z. R. Ghassabi et al. [18] y D.S. Huang [25] diseñaron de manera independiente una RN para la inversión de matrices singulares; S. Perantonis et al. [56] crearon una red neuronal para la factorización de polinomios; R. Hormis et al. [Hormis et al.] diseñaron una RN para la separación de polinomios bidimensionales en un producto de polinomios de una sola variable.

Más relacionado a la presente tesis es el trabajo D. S. Huang et al. [25], quienes diseñaron una RN para encontrar las raíces de un polinomio arbitrario. De manera específica, se buscó encontrar las raíces de:

$$p(z) = a_0 z^n + a1 z^{n-1} + ... + a_n = \sum_{k=0}^{n} a_{n-k} z^k \tag{2.1}$$

Donde $a, z \in C$ y n es el grado del polinomio.

Además de resolver problemas similares, los trabajos [26], [24], [56], [25] tienen en común el uso de restricciones en redes neuronales; más aún, la metodología seguida por estos trabajos es similar y se puede resumir en los siguientes puntos:

- Diseño de la arquitectura de la RN.
- Inclusión de restricciones al algoritmo de aprendizaje (dadas por la naturaleza del problema).
- Derivación del nuevo algoritmo de aprendizaje a través de la función de error con la información de las restricciones y del método de multiplicadores de Lagrange.
- Obtención y análisis de resultados.

Finalmente, cabe destacar que en todos los trabajos ya mencionados, el uso de restricciones mejoró la convergencia de uno a dos órdenes de magnitud manteniendo la precisión constante.

2.3. Redes neuronales en la solución de ecuaciones diferenciales

El análisis numérico de las ecuaciones diferenciales resulta útil en áreas como la física, las matemáticas aplicadas, la ingeniería eléctrica, la bioquímica, entre otras [41], para entender fenómenos complejos que son difíciles o imposibles de tratar de forma analítica. De manera general, lo que se busca es resolver una ecuación diferencial ya sea ordinaria o parcial de la forma [37]:

$$f(\boldsymbol{x}, \nabla\psi(\boldsymbol{x}), \nabla^2\psi(\boldsymbol{x}), ...) = 0, \tag{2.2}$$

sujeta a ciertas condiciones de frontera (e.g. Dirichlet o Neumann), en donde $\boldsymbol{x} = (x_1, ..., x_n)$ y $\psi(\boldsymbol{x})$ es la solución a computar. La hipótesis para sustentar el uso de RN en este problema se debe a que éstas pueden aproximar cualquier función a una precisión arbitraria [34, 37, 66]; además de esto, es favorable su elección debido a su intrínseca compatibilidad con tecnologías de cómputo en paralelo.

Existen tres principales métodos para utilizar las RN en la solución de ecuaciones diferenciales:

1. Discretizar el dominio (e.g. diferencias finitas) y utilizar las RN para resolver las ecuaciones algebraicas resultantes.

2. Buscar los parámetros a través de una RN de un *spline* (curva diferenciable definida en porciones mediante polinomios) que represente la solución a la ecuación diferencial.

3. Mediante un proceso iterativo buscar una solución a la ecuación diferencial: $\psi(\boldsymbol{x}) = A(\boldsymbol{x}) + F(\boldsymbol{x}, N(\boldsymbol{x}, \boldsymbol{w}))$, en donde el término $A(\boldsymbol{x})$ no contiene parámetros ajustables y satisface las condiciones de frontera, $N(\cdot)$ representa la salida de la RN y $F(\cdot)$ es una función de propuesta de solución.

Los métodos (1) y (2) se basan en la aproximación de ecuaciones algebraicas. De manera específica, la primera puede ser resuelta a través de restricciones como se muestra en [24]; sin embargo, al utilizar diferencias finitas como método de discretización, se acarrea un error intrínseco y para minimizarlo es necesario disminuir

el tamaño de la malla, lo que aumenta el tiempo de cómputo [20]. La segunda propuesta tiene la desventaja de necesitar un gran número de parámetros para la RN; además, no es sencillo extender esta técnica a dominios multidimensionales [37]. Finalmente, en la propuesta (3) se tiene el problema de proponer de manera explícita y manual la función $F(\cdot)$.

2.4. Aprendizaje de máquinas en la simulación de fluidos

El AM ha demostrado ser eficiente en una gran y diversa cantidad de disciplinas [38]; entre ellas destacan biología, física, química e ingeniería eléctrica. Recientemente ha surgido el campo de la física impulsada por datos, la cual tiene como objetivo simular sistemas físicos. Uno de estos sistemas es la simulación de fluidos por computadora.

L. Ladický et al. en el 2015 publicaron un artículo en el que generan simulaciones de diversos fluidos incompresibles en un marco de referencia Lagrangiano a través del método de bosques aleatorios [7], [28]. En éste, se ataca uno de los principales problemas para generar simulaciones con alta precisión: las restricciones que existen sobre el paso del tiempo para asegurar la estabilidad numérica. Las dos principales contribuciones de Ladický et al. son:

1. Posibilidad de generar simulaciones con un Δt mayor preservando la precisión, en comparación con los algoritmos clásicos (no impulsados por datos).

2. Diseño de un vector de características que representa de manera completa el estado de una partícula, el cual es lo suficientemente rico para codificar toda la información necesaria en aras de predecir el siguiente estado del sistema.

Por otra parte, Yang et al. [77] abordaron el problema de la simulación de fluidos desde una perspectiva Euleriana. Su trabajo se concentra en reemplazar el algoritmo iterativo clásico para la solución de la presión en fluidos incompresibles por una RN que tiene como entradas el estado de la presión en el estado anterior, el vector de velocidad y la topología del problema. A pesar de que en los resultados se logró un aumento en la velocidad de cómputo, se vio degradada la precisión de la solución. Esto es debido a que en la RN no se tomó en cuenta la condición de incompresibilidad de manera explícita; además, no se muestra la capacidad de

generalización de la red, ya que se trabaja con una cantidad reducida de simulaciones.

Para resolver los problemas presentados por Yang et al. [77], J. Tompson et al. [69] diseñaron una RNC para resolver el mismo problema, esto es, computar la presión. Además, incluyeron la condición de divergencia que mantienen los fluidos incompresibles: $\nabla \vec{u} = 0$. Para esto, conmutarón la propuesta de un sistema de aprendizaje supervisado por uno no supervisado. Dicho de otra manera, la función de error de la red no representa la distancia entre la presión predicha y la deseada, sino la magnitud de la divergencia del vector de velocidad obtenida a través de la salida de la red. No obstante, son dos las principales desventajas de este método:

- No se incluye a la presión en el estado anterior como entrada de la red, la cual ha demostrado ser valiosa en otras arquitecturas [77].
- La red presenta largos tiempos de entrenamiento, lo cual resulta en una desventaja cuando se busca obtener una gran capacidad de generalización utilizando una base de datos grande.

2.5. Restricciones en redes neuronales

Como se mencionó en el capítulo 1, las RN han sido sujeto de estudio en años recientes debido a sus interesantes cualidades de generalización y a su flexibilidad para ser usadas en problemas tanto de clasificación como de regresión [31]. Para su entrenamiento se utiliza el algoritmo del descenso del gradiente, el cual busca adaptar de manera iterativa los parámetros de la RN hasta obtener una función de salida conforme a los datos de entrenamiento.

Este algoritmo fue uno de los primeros en usarse para entrenar RN y probablemente sea el más popular [31]; sin embargo, se le ha criticado por ser lento para converger a una solución[15], ineficiente para alcanzar un alto grado de generalidad [50] y biológicamente improbable (las neuronas humanas 'aprenden' de otra manera) [10]. Por estas razones, entre otras, es importante enriquecer el algoritmo de aprendizaje en las RN teniendo como prioridad el aumento de la velocidad de convergencia, la conservación de la precisión y la mejora en la capacidad de generalización.

Para encontrar una solución al problema, varios trabajos han propuesto modificaciones al algoritmo del descenso del gradiente. Por ejemplo, Y. LeCun propuso restringir la arquitectura y los pesos de la red [40], Patrice Simard et al. diseñaron

una nueva métrica de error que puede servir para el entrenamiento de redes neuronales [63], T. Kathirvalavakumar y S. Jeyaseeli propusieron entrenar las capas de la RN con diferentes funciones de error [32] y Fei Han et al. propusieron un algoritmo de aprendizaje que toma en cuenta derivadas de orden superior [21]. A pesar de que todos estos trabajos mejoraron en alguna medida el desempeño del algoritmo de aprendizaje, ninguno incorpora información adicional del problema a éste.

A. Karras y J. Perantonis propusieron incorporar información adicional del problema al algoritmo del descenso del gradiente en forma de restricciones sobre la función de error [31]; para esto, utilizan el método de los multiplicadores de Lagrange deduciendo así un nuevo algoritmo para la actualización de pesos en cada iteración. El nuevo algoritmo llamado ALECO ha resultado en una disminución en el tiempo de convergencia y en una mayor capacidad de generalización de la red [24, 25, 26, 31].

Capítulo 3

Introducción a las redes neuronales

El desarrollo de las redes neuronales tiene ya más de 60 años y fue producto del deseo de generar un modelo matemático y computacional que intentara emular algunas de las fortalezas de las neuronas biológicas [16]. En 1943 McCulloch y Pitts introdujeron un modelo simplificado de neurona artificial [45], el cual podía realizar tareas computacionales similares a las de una compuerta lógica.

Posteriormente, el científico Frank Rosenblatt desarrollaría en la década de los 50 el perceptrón, el cual es un tipo especial de neurona artificial [53] que de manera general se puede interpretar como una función de mapeo $F : (X, \Theta) \rightarrow \{0, 1\}$ en donde X es la entrada del perceptrón y Θ es el conjunto de parámetros que lo caracterizan. Este desarrollo generó un entusiasmo en la comunidad científica y dio como resultado el diseño de diversas neuronas artificiales con sus respectivas reglas y algoritmos de entrenamiento para definir sus parámetros Θ. Entre las neuronas artificiales más importantes, destaca ADALINE ('Adaptative Linear Combiner') con la subsecuente regla delta para la selección de parámetros; ésta última sería precursora del algoritmo de retropropagación.

En 1969 Minsky y Papert publicaron su libro titulado "Perceptrones"[47] en el cual explican las limitaciones de las neuronas artificiales, en específico, mencionan los tipos de funciones y aproximaciones que pueden hacer. Debido en gran parte a esta publicación, el momento que tenía la investigación en neuronas artificiales se desvaneció durante la siguiente década. Sólo algunos investigadores continuaron, entre ellos los más destacados son: Teuvo Kohonen, Stephen Grossberg, James Anderson y Kunihiko Fukushima [35].

Al inicio de la década de los 80 todavía se encontraba abierto el problema de entrenamiento para las RN, esto es, cómo encontrar de manera eficiente los parámetros Θ de una serie de neuronas artificiales interconectadas. El primer precursor del

algoritmo que resolvería este problema fue publicado en 1974 por P.J. Werbos [75] en donde se establecen los principios para propagar información de la capa de salida hacia las capas ocultas. De manera independiente, Parker en 1985 [54] y LeCun en 1986 [16] llegaron al mismo método. Sin embargo, no fue hasta que Geoffrey Hinton et al. definieron e hicieron conocido el ARP como método para encontrar los parámetros de una red neuronal multicapa [62].

El éxito de las RN en recientes años es debido (además del algoritmo de retropropagación) a la gran cantidad de información disponible para el entrenamiento de modelos y a una capacidad de cómputo en constante crecimiento que ha permitido el diseño de redes cada vez más profundas. Existe una gran variedad de redes neuronales en la actualidad, en las siguientes secciones se introduce el perceptrón, la neurona artificial, las redes neuronales multicapa y las redes neuronales convolucionales, las cuales serán utilizadas para la generación de modelos en los capítulos 4 y 5.

3.1. El perceptrón

El perceptrón como se mencionó anteriormente, es una unidad de cómputo que toma un vector de entrada $\boldsymbol{x} = [x_1, x_2, \cdots x_n]^T$, un vector de pesos $\boldsymbol{w} = [w_1, w_2, \cdot\cdot \cdot w_n]^T$, un valor de umbral u y calcula una función de salida $O(\boldsymbol{x}, \boldsymbol{w}, u)$ en donde $O \in \{0, 1\}$ y $w_i, u \in \mathbb{R}$. La salida está definida de la siguiente manera [53]:

$$O(\boldsymbol{x}, \boldsymbol{w}, u) = \begin{cases} 0 & si \quad \sum_j w_j x_j \leq u \\ 1 & si \quad \sum_j w_j x_j > u \end{cases} \tag{3.1}$$

En la figura 3.1 se muestra un diagrama general del perceptrón. Aunque no es parte de la convención, aquí se introduce la nomenclatura de capa de entrada y capa de salida en referencia al vector de entrada $\boldsymbol{x}$ y a la salida O. Pese a que en este ejemplo la capa de salida sólo contiene una neurona, esta no es la norma, y en general, se puede tener un número arbitrario de éstas.

Es importante no confundir los círculos que rodean a las entradas $x_1, x_2, \cdots, x_n$ con neuronas, éstos son utilizados por convención en la comunidad científica y no simbolizan nada en específico. Otra cosa importante a destacar es que el sentido de las conexiones solamente va de izquierda a derecha, esto es, de la capa de entrada a la capa de salida, lo que implica que el perceptrón se puede representar

como una gráfica sin ciclos. A este tipo de redes se les conoce como redes neuronales prealimentadas.

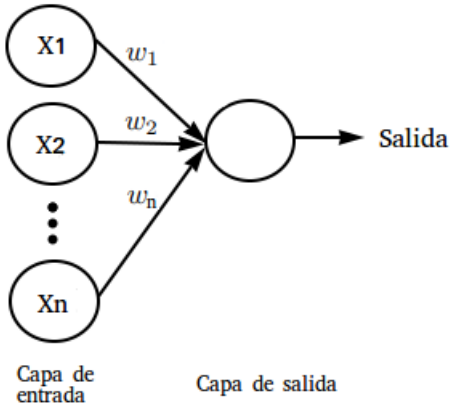

Figura 3.1: Diagrama general del perceptrón

Una vez establecidos los principios de operación del perceptrón, es sencillo ver cómo se puede utilizar esta unidad de cómputo para el cálculo de funciones lógicas. Por ejemplo, supongamos que queremos diseñar dos perceptrones P_1 y P_2 para el cómputo de la conjunción y disyunción lógica respectivamente (cada una de estas funciones para dos entradas binarias). Con este fin, definimos $\boldsymbol{x} = [x_1, x_2]^T$, $\boldsymbol{w}_1 = \boldsymbol{w}_2 = [1, 1]^T$, $u_1 = 1$ y $u_2 = 0$. En esta notación, los subíndices sobre $\boldsymbol{w}$ y u hacen referencia al perceptrón al que pertenecen.

Evaluando la salida del primer perceptrón para cualquier posible entrada $\boldsymbol{x}$ podemos corroborar que se computa la conjunción lógica. $O_1([0, 0]^T, [1, 1]^T, 1) = 0$, $O_1([0, 1]^T, [1, 1]^T, 1) = 0$, $O_1([1, 0]^T, [1, 1]^T, 1) = 0$ y $O_1([1, 1]^T, [1, 1]^T, 1) = 1$. Podemos verificar de similar manera los resultados del segundo perceptrón y comprobar que calcula la disyunción lógica. $O_2([0, 0]^T, [1, 1]^T, 0) = 0$, $O_2([0, 1]^T, [1, 1]^T, 0) = 1$, $O_2([1, 0]^T, [1, 1]^T, 0) = 1$ y $O_2([1, 1]^T, [1, 1]^T, 0) = 1$.

X_1	X_2	$\wedge$
0	0	0
0	1	0
1	0	0
1	1	1

X_1	X_2	$\vee$
0	0	0
0	1	1
1	0	1
1	1	1

Figura 3.2: Conjunción lógica (izquierda) y disyunción lógica (derecha)

3.2. La neurona artificial

El perceptrón es un tipo especial de neurona artificial que computa una función de umbral sobre la suma de cada una de las entradas multiplicada por su peso correspondiente. Si pasamos el valor del umbral al lado izquierdo de la desigualdad podemos reescribir (3.1) de la siguiente manera (por convención la variable b se ha utilizado para indicar el negativo del umbral de la neurona i.e. $b = -u$, y se le conoce como sesgo):

$$O(\boldsymbol{x}, \boldsymbol{w}, b) = \begin{cases} 0 & si \quad \sum_j w_j x_j + b \leq 0 \\ 1 & si \quad \sum_j w_j x_j + b > 0 \end{cases} \tag{3.2}$$

De manera general, definimos a una neurona artificial como una unidad de procesamiento de información que está basada y comparte características con la neurona biológica humana. Ésta se ve caracterizada por un vector de pesos $\boldsymbol{w} = [w_1, w_2, \cdots, w_n]^T$, un valor de sesgo b y una función de activación $f(\cdot)$. La salida de la neurona se obtiene al aplicar la función de activación a la suma de cada una de las entradas multiplicadas por sus respectivos pesos más el sesgo, esto es, $O = f\left(\sum_j w_j x_j + b\right)$, tomando en cuenta que $\sum_j w_j x_j$ es igual a $\boldsymbol{w}^T \boldsymbol{x}$ podemos reescribir la salida como $O = f\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$. Al contrario que en el perceptrón, la variable O no tiene que ser binaria y puede tomar valores reales o complejos.

La figura 3.3 representa un diagrama general de una neurona artificial, en ésta, la neurona ha sido dividida en dos partes para hacer énfasis en el orden de cómputo, esto es, primero se computa $\boldsymbol{w}^T \boldsymbol{x} + b$ y después la función de activación.

En el caso del perceptrón, se definió el valor de los pesos w_i en forma manual; sin embargo, esto resulta impractico en arquitecturas y en problemas más complejos. La solución es diseñar un algoritmo capaz de encontrar este conjunto de pesos de manera automática.

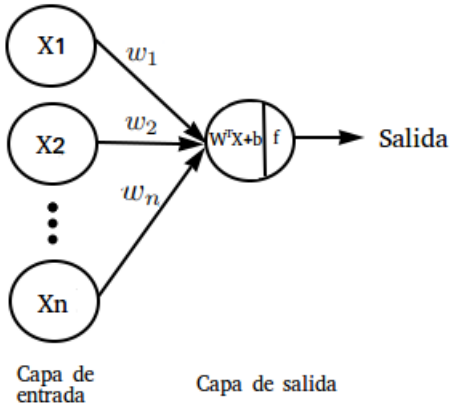

Figura 3.3: Diagrama general de una neurona artificial

3.3. Redes neuronales multicapa

Cuando hablamos de redes neuronales multicapa o RNM nos referimos a una interconexión de neuronas artificiales con el objetivo de generar una mejor función de mapeo entre las entradas y la salida deseada en comparación con la neurona artificial. La figura 3.4 muestra un diagrama general que describe a las RNM que poseen una neurona en la capa de salida. La notación capa oculta hace referencia a las neuronas que no son ni parte de la capa de entrada ni de la de salida, los términos capa 1, capa 2, ..., capa M han sido utilizados para agrupar a las neuronas por bloques de cómputo. Finalmente, se utilizará el término capa de entrada y capa 0 de manera intercambiable.

Para denotar un peso específico de la red utilizaremos la notación $w_{i,j}^{[l]}$ en donde i denota el índice de la neurona donde inicia la conexión en la capa $l-1$ y j la del fin de ésta en la capa l e.g. $w_{1,2}^{[2]}$ representa el peso que existe entre la primera neurona de la capa 1 y la segunda neurona de la capa 2. También denotamos como $\boldsymbol{W}^{[l]}$ a la matriz de pesos de la capa l. De manera explícita la escribimos como:

$$\boldsymbol{W}^{[l]} = \begin{pmatrix} w_{1,1}^{[l]} & w_{2,1}^{[l]} & w_{3,1}^{[l]} & \cdots & w_{k_{l-1},1}^{[l]} \\ w_{1,2}^{[l]} & w_{2,2}^{[l]} & w_{3,2}^{[l]} & \cdots & w_{k_{l-1},2}^{[l]} \\ w_{1,3}^{[l]} & w_{2,3}^{[l]} & w_{3,3}^{[l]} & \cdots & w_{k_{l-1},3}^{[l]} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ w_{1,k_l}^{[l]} & w_{2,k_l}^{[l]} & w_{3,k_l}^{[l]} & \cdots & w_{k_{l-1},k_l}^{[l]} \end{pmatrix} \tag{3.3}$$

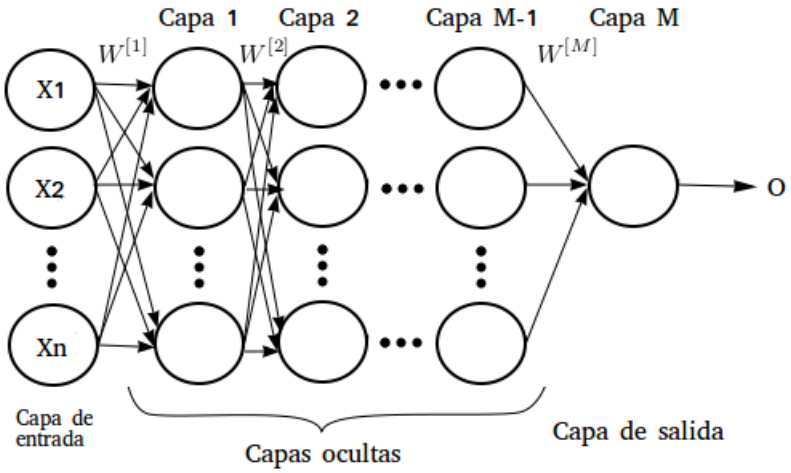

Figura 3.4: Diagrama general de una red neuronal multicapa.

En la ecuación (3.3) la variable k_l representa el número de neuronas en la capa l. Denotamos el sesgo de cada neurona como $b_i^{[l]}$ en donde i representa el índice de la neurona en la capa l a la cual nos referimos, e.g. $b_3^{[2]}$ representa el sesgo de la tercera neurona de la segunda capa. Asimismo, denotaremos a $b^{[l]}$ como el vector de sesgos de la capa l. De manera explícita lo escribimos como:

$$\boldsymbol{b}^{[l]} = \begin{bmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \cdot \\ \cdot \\ b_{k_l}^{[l]} \end{bmatrix} \tag{3.4}$$

Recordando el funcionamiento de la neurona artificial, vemos que cada neurona realiza dos operaciones de manera secuencial, primero calcula la suma del producto de las entradas con sus respectivos pesos agregándole a ésta el sesgo de la neurona, después evalúa este resultado a través de una función de activación f(·). Durante las siguientes páginas, al resultado de la primera parte se le denotará con la letra Z y al de la segunda con la A. Para una neurona artificial podemos escribir:

$$\boldsymbol{Z} = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b} \tag{3.5}$$

$$\boldsymbol{A} = f\left(\boldsymbol{Z}\right) \tag{3.6}$$

Para las RNM introducimos de manera similar la variable $\boldsymbol{Z^{[l]}} = [z_1^{[l]}, z_2^{[l]}, \cdots z_{k_l}^{[l]}]^T$ en donde $z_i^{[l]}$ representa la salida de la neurona i de la capa l previo a la evaluación de la función de activación. Asimismo, definimos la variable $\boldsymbol{A^{[l]}} = [a_1^{[l]}, a_2^{[l]}, \cdots a_{k_l}^{[l]}]^T$ en donde $a_i^{[l]} = f(z_i^{[l]})$.

A manera de ilustración, en lo que resta de la sección se obtendrá la salida de la RNM que se muestra en la figura 3.5. Vemos que $X = [x_1, x_2]^T$ (sólo es una entrada o patrón de entrenamiento) y que tenemos dos matrices de pesos $W^{[1]}$ y $W^{[2]}$; además, pese a que en la figura no se muestran de manera explicita los sesgos de las neuronas (por cuestiones estéticas y de convención), en este ejercicio sí los tomaremos en cuenta.

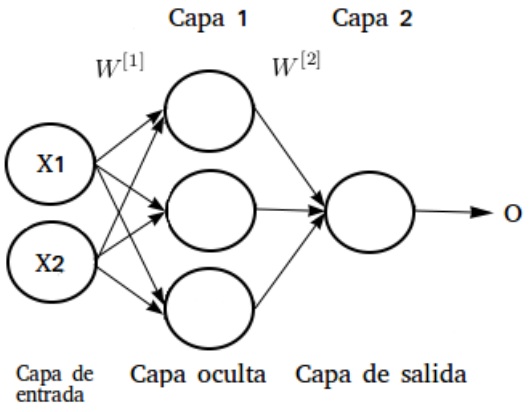

Figura 3.5: Ejemplo red neuronal multicapa.

Siguiendo la notación previamente mostrada, definimos las matrices $W^{[1]}$ y $W^{[2]}$ como:

$$\boldsymbol{W^{[1]}} = \begin{bmatrix} w_{1,1}^{[1]} & w_{2,1}^{[1]} \\ w_{1,2}^{[1]} & w_{2,2}^{[1]} \\ w_{1,3}^{[1]} & w_{2,3}^{[1]} \end{bmatrix} \tag{3.7}$$

$$\boldsymbol{W}^{[2]} = \left[w_{1,1}^{[2]}, w_{2,1}^{[2]}, w_{3,1}^{[2]} \right] \tag{3.8}$$

Una vez en este formato, podemos encontrar $\boldsymbol{Z}^{[1]}$ al multiplicar $\boldsymbol{W}^{[1]}$ por $\boldsymbol{x}$ y posteriormente sumando el sesgo, esto es, $\boldsymbol{Z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}$. Para obtener la matriz $\boldsymbol{A}^{[1]}$ aplicamos la función de activación f a cada elemento de la matriz $\boldsymbol{Z}^{[1]}$, dicho de otro modo, $\boldsymbol{A}^{[1]} = f(\boldsymbol{Z}^{[1]})$. La salida de la red neuronal (antes de la aplicación de la función de activación) se denota como $\boldsymbol{Z}^{[2]}$ y se calcula a través de la multiplicación de los pesos de la segunda capa con la salida de las neuronas de la primera capa más el sesgo, poniéndolo en ecuación: $\boldsymbol{Z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{A}^{[1]} + \boldsymbol{b}^{[2]}$. Finalmente, la salida de la red neuronal se obtiene al aplicar la función de activación a cada elemento de $\boldsymbol{Z}^{[2]}$, esto es, $\boldsymbol{O} = \boldsymbol{A}^{[2]} = f(\boldsymbol{Z}^{[2]})$.

Si analizamos el tamaño de la entrada en el ejemplo anterior, vemos que $\boldsymbol{x}$ es un vector de dos filas por una columna, de manera mas compacta podemos escribir esto como: $S(\boldsymbol{x}) = (2, 1)$. Haciendo un proceso similar para las matrices de pesos tenemos que $S(\boldsymbol{W}^{[1]}) = (3, 2)$ y $S(\boldsymbol{W}^{[2]}) = (1, 3)$. A partir de estas tres tuplas, podemos deducir el tamaño del resto de las matrices. Por ejemplo, $S(\boldsymbol{Z}^{[1]}) = S(\boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}) = (3, 1)$ y $S(\boldsymbol{Z}^{[2]}) = S(\boldsymbol{W}^{[2]}\boldsymbol{A}^{[1]} + \boldsymbol{b}^{[2]}) = (1, 1)$. Las matrices $\boldsymbol{A}^{[1]}$ y $\boldsymbol{A}^{[2]}$ tienen el mismo tamaño que $\boldsymbol{Z}^{[1]}$ y $\boldsymbol{Z}^{[2]}$ respectivamente. Por lo tanto, la salida de la red neuronal es un escalar.

A manera de resumen, las ecuaciones que describen el procesamiento de la RNM son:

$$\boldsymbol{Z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]} \tag{3.9}$$

$$\boldsymbol{A}^{[1]} = f(\boldsymbol{Z}^{[1]}) \tag{3.10}$$

$$\boldsymbol{Z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{A}^{[1]} + \boldsymbol{b}^{[2]} \tag{3.11}$$

$$\boldsymbol{O} = \boldsymbol{A}^{[2]} = f(\boldsymbol{Z}^{[2]}) \tag{3.12}$$

Hasta este punto se ha planteado la entrada como un solo vector; sin embargo, es muy común que se requiera el procesamiento de múltiples entradas de manera simultánea, para esto, redefinimos a la entrada como una matriz en donde cada columna es una instancia o patrón a computar. Esto lo escribimos como: $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_p}]$, en donde $\boldsymbol{x_i} = [x_{i,1}, x_{i,2}, \cdots]^T$. En este escenario, es fácil demostrar que las ecuaciones (3.9 - 3.12) siguen siendo válidas; sin embargo, los tamaños de las matrices varían.

Analizando de nuevo los tamaños de las matrices, pero tomando en cuenta que X es una matriz que consta de p vectores de entrenamiento, vemos que $S(\boldsymbol{Z^{[1]}}) = S(\boldsymbol{W^{[1]}}\boldsymbol{X} + \boldsymbol{b^{[1]}}) = (3, p)$, $S(\boldsymbol{Z^{[2]}}) = S(\boldsymbol{W^{[2]}}\boldsymbol{A^{[1]}} + \boldsymbol{b^{[2]}}) = (1, p)$, $S(\boldsymbol{A^{[1]}}) = (3, p)$ y $S(\boldsymbol{A^{[2]}}) = (1, p)$. Como podemos apreciar, en este caso la salida de la RNM es un vector con p elementos en donde el i componente representa la salida de la red cuando se ingresa a ésta el vector X_i.

3.4. Funciones de costo

Se mencionó anteriormente que es posible encontrar a través del algoritmo del descenso del gradiente el conjunto de pesos que acerque la salida de la red neuronal al valor deseado. Esto se logra a través de un proceso iterativo en donde en cada ciclo se intenta mejorar el desempeño de la red. Para lograr esto, es necesario evaluar qué tan cerca o lejos se está del objetivo en cada iteración; las funciones de costo cumplen esta meta.

Existen diversas funciones de costo y la selección de alguna dependerá de la naturaleza del problema a tratar. En específico, de si se trata de una tarea de regresión o clasificación. Además, las funciones de costo tienen que poseer dos propiedades (las cuales se aclararán en la sección de retropropagación).

1. La función de costo total debe de poder ser escrita como la suma del costo de cada instancia o patrón de entrenamiento de $\boldsymbol{X}$.

2. La función de costo debe de poder evaluarse a partir de la salida de la última capa de la RN.

Para problemas de regresión univariable, una de las funciones de costo más utilizadas es la del error cuadrático medio, la cual podemos escribir de la siguiente

manera: $ECM = \frac{1}{2P}\sum_{p=1}^{P}(T_i - O_i)^2$, en donde el termino O_i representa la salida de la red neuronal para la instancia $\boldsymbol{x_i}$ y el termino T_i representa la salida deseada o real del sistema para la misma entrada. La suma es sobre el número de patrones de entrenamiento.

Para problemas de clasificación binaria, una de las funciones de costo más utilizadas es la de entropía cruzada, la cual se define como: $C = -\frac{1}{P}\sum_{p=1}^{P} T_i ln(O_i) + (1 - T_i)ln(1 - O_i)$. Durante el resto de la tesis nos interesaremos más por problemas de regresión y no se mencionara más a las funciones de costo específicas para problemas de clasificación.

3.5. Funciones de activación

Hasta este punto, no se ha definido ninguna función de activación específica; como veremos, éstas se seleccionan dependiendo de la naturaleza del problema y sólo cuentan con una restricción: no pueden ser lineales en las capas ocultas. La razón de esto es que si se utiliza una función de activación lineal en las capas ocultas, la red neuronal no es capaz de aprender patrones no lineales. Para ver esto, retomemos las ecuaciones (3.9 - 3.12) con un pequeño cambio, redefinamos $A^{[1]}$ y $A^{[2]}$ a partir del uso de la función identidad, esto es:

$$\boldsymbol{A^{[1]}} = \boldsymbol{Z^{[1]}} = \boldsymbol{W^{[1]}X} + \boldsymbol{b^{[1]}} \tag{3.13}$$

$$\boldsymbol{A^{[2]}} = \boldsymbol{Z^{[2]}} = \boldsymbol{W^{[2]}A^{[1]}} + \boldsymbol{b^{[2]}} \tag{3.14}$$

Sustituyendo (3.13) en (3.14) tenemos:

$$\boldsymbol{A^{[2]}} = \boldsymbol{W^{[2]}}(\boldsymbol{W^{[1]}X} + \boldsymbol{b^{[1]}}) + \boldsymbol{b^{[2]}} \tag{3.15}$$

O lo que es lo mismo:

$$\boldsymbol{A^{[2]}} = (\boldsymbol{W^{[2]}W^{[1]}})\boldsymbol{X} + (\boldsymbol{W^{[2]}b^{[1]}} + \boldsymbol{b^{[2]}}) \tag{3.16}$$

Realizando las siguientes definiciones:

$$\boldsymbol{W'} \equiv \boldsymbol{W^{[2]}W^{[1]}} \tag{3.17}$$

$$\boldsymbol{b'} \equiv \boldsymbol{W^{[2]}b^{[1]}} + \boldsymbol{b^{[2]}} \tag{3.18}$$

Es posible reescribir (3.16) como:

$$\boldsymbol{A}^{[2]} = \boldsymbol{W}'\boldsymbol{X} + \boldsymbol{b}' \tag{3.19}$$

Podemos darnos cuenta que la función que se está aproximando resulta una combinación lineal de las entradas, los pesos y los sesgos. Por esta razón, es común encontrar en la literatura funciones de activación no lineales, entre estas, destacan tres por ser las más utilizadas y por ser la base de otras, estas son: la función sigmoide, la tangente hiperbólica y la ReLU. La figura 3.6 muestra la gráfica de éstas. A pesar de su gran importancia en la literatura, no se detallarán más éstas ya que en la presente tesis no se utilizan (con excepción de la ReLU la cual se detallará cuando se utilice); sin embargo, se hará uso de una función de activación logarítmica, la cual se explicará en el capítulo 4.

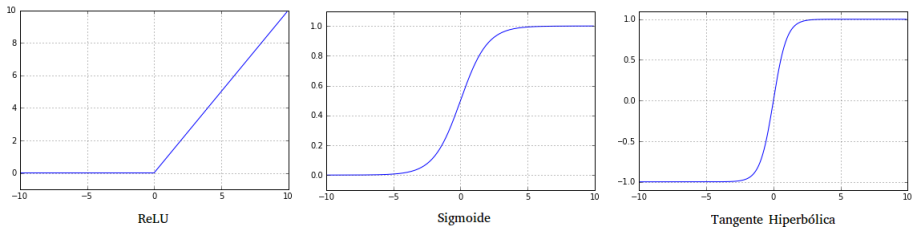

Figura 3.6: Funciones de activación comunes

3.6. El algoritmo del descenso del gradiente

Vimos que la función de costo es una métrica para evaluar lo cercano o lejano que se está de la función a representar por la red neuronal. Dependiendo de la función de costo, se busca obtener el mínimo o el máximo de ésta. En el caso específico de la función del ECM se busca minimizar el valor. Para encontrar éste, es común hacer uso del algoritmo del descenso del gradiente.

Denotaremos a la función de costo con la letra mayúscula C y vemos que está en función de los pesos y del sesgo de la red. Ahora, supongamos que estamos modelando una RN sin sesgo, por lo tanto, $C(\boldsymbol{W})$; más aún, nuestra RN sólo tiene un peso, y lo denotamos como w_1. La figura 3.7 muestra una gráfica ejemplo que podría representar a nuestra función de costo.

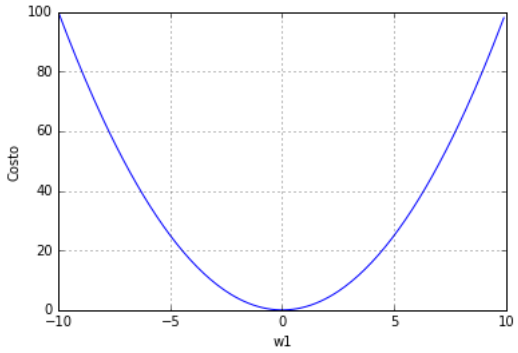

Figura 3.7: Ejemplo de función de costo

Gráficamente, podemos ver que obtenemos un mínimo global al establecer $w_1 = 0$; sin embargo, esto es rara vez posible debido al número de parámetros de la red, así que lo que se busca es un algoritmo para encontrar este mínimo de manera automática. Supongamos que actualmente nuestra red se encuentra caracterizada con $w_1 = 5$; si calculamos la pendiente en este punto, obtenemos la tasa de crecimiento de w con respecto al costo, si ésta es positiva implica que $C(w_1 + \delta w_1) > C(w_1)$ y $C(w_1 - \delta w_1) < C(w_1)$. En otras palabras, cuando la tasa de crecimiento es positiva, una reducción en el parámetro w_1 de magnitud δw_1 disminuye la función de costo, por otra parte, cuando la tasa de crecimiento es negativa, un aumento en el parámetro w_1 de magnitud δw_1 disminuye la función de costo. La figura 3.8 contiene la pendiente en el punto a tratar.

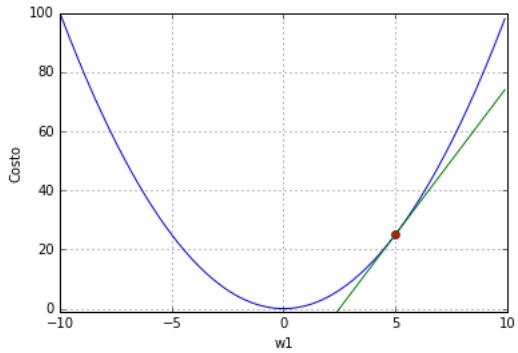

Figura 3.8: Pendiente de la función costo en $w_1 = 5$

Habiendo establecido lo anterior, podemos deducir un algoritmo iterativo para disminuir la función de costo en cada ciclo. Definimos la regla de actualización de parámetros de la siguiente manera:

$$w_i \leftarrow w_i - \left(TA * \frac{dC(w_1)}{dw_1} \right) \tag{3.20}$$

La variable TA es una constante y es el acrónimo de tasa de aprendizaje. Si la derivada de la función de costo con respecto a w_1 nos da la dirección en la que se debe de cambiar w_1 para reducir la función de costo, la TA nos da la magnitud de este cambio. En la práctica, la TA se selecciona de manera empírica.

Cuando tratamos con funciones de costo multivariables podemos aplicar el mismo principio, pero en lugar de utilizar la derivada de un solo parámetro computamos el gradiente de la matriz de pesos. Notamos que $S(\nabla \boldsymbol{W}) = S(\boldsymbol{W})$ y que $\nabla w_{i,j}$ es la derivada parcial de la función de costo con respecto al componente $w_{i,j}$. De manera concreta podemos escribir:

$$\boldsymbol{W}^{[i]} \leftarrow \boldsymbol{W}^{[i]} - \left(TA * \nabla \boldsymbol{W}^{[i]} \right) \tag{3.21}$$

3.7. El algoritmo de retropropagación

Como hemos visto, nuestro objetivo es encontrar el conjunto de pesos que disminuyan la función de costo de la red a través del algoritmo del descenso del gradiente. El ARP nos permite encontrar el gradiente de cada uno de los parámetros; para lograrlo, éste propaga la tasa de cambio del error a través de las capas de la red hasta alcanzar al parámetro en cuestión haciendo uso de la regla de la cadena para derivadas parciales. Para entender el procedimiento, veamos un ejemplo. La figura 3.9 muestra la RN a estudiar.

En este caso, tenemos una RN de tres capas. Supongamos que estamos interesados en conocer el gradiente de la matriz de pesos $\boldsymbol{W}^{[2]}$ para un solo patrón de entrenamiento $\boldsymbol{x}$. Dicho de otra manera, se requiere computar:

$$\nabla \boldsymbol{W}^{[2]} = \left[\begin{array}{cccc} \frac{\partial E}{\partial w^{[2]}_{1,1}}, & \frac{\partial E}{\partial w^{[2]}_{2,1}}, & \frac{\partial E}{\partial w^{[2]}_{3,1}}, & \frac{\partial E}{\partial w^{[2]}_{4,1}} \end{array} \right] \tag{3.22}$$

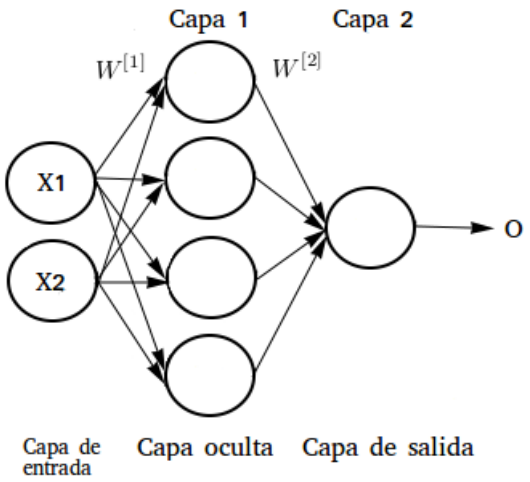

Figura 3.9: Red neuronal ejemplo a estudiar.

Si el problema es de clasificación, utilizamos la función de entropía cruzada como nuestro costo. Denotamos ésta con la letra mayúscula E. De las secciones anteriores vemos que $E = -(T \cdot ln(O) + (1 - T) \cdot ln(1 - O)$ (en donde O es la salida de la red y T es el valor deseado). Empezaremos calculando el primer elemento del gradiente en la ecuación (3.22). Utilizando la regla de la cadena podemos obtenerlo de la siguiente manera:

$$\frac{\partial E}{\partial w_{1,1}^{[2]}} = \frac{\partial E}{\partial O} \cdot \frac{\partial O}{\partial \boldsymbol{Z}^{[2]}} \cdot \frac{\partial \boldsymbol{Z}^{[2]}}{\partial w_{1,1}^{[2]}} \tag{3.23}$$

De izquierda a derecha, el primer término de la expresión (3.23) se calcula de la siguiente forma:

$$\frac{\partial E}{\partial O} = -\left(\frac{T}{O} - \frac{1-T}{1-O}\right) = \frac{O-T}{O(1-O)} \tag{3.24}$$

Por otra parte, podemos obtener la derivada parcial de la salida de la red con respecto a $\boldsymbol{Z}^{[2]}$ de la siguiente manera:

$$\frac{\partial O}{\partial \boldsymbol{Z}^{[\mathbf{2}]}} = \frac{\partial\, f(\boldsymbol{Z}^{[\mathbf{2}]})}{\partial \boldsymbol{Z}^{[\mathbf{2}]}} = f^{'}(Z^{[2]}) \tag{3.25}$$

Finalmente, podemos calcular el tercer elemento como:

$$\frac{\partial \boldsymbol{Z}^{[\mathbf{2}]}}{\partial w_{1,1}^{[2]}} = \frac{\partial\, \left(\boldsymbol{W}^{[\mathbf{2}]}\boldsymbol{A}^{[\mathbf{1}]} + \boldsymbol{b}^{[\mathbf{2}]}\right)}{\partial w_{1,1}^{[2]}} \tag{3.26}$$

$$\frac{\partial \boldsymbol{Z}^{[\mathbf{2}]}}{\partial w_{1,1}^{[2]}} = \frac{\partial\, \left(w_{1,1}^{[2]}a_1^{[1]} + w_{2,1}^{[2]}a_1 2[1] + w_{3,1}^{[2]}a_3^{[1]} + w_{4,1}^{[2]}a_4^{[1]} + b_1^{[2]}\right)}{\partial w_{1,1}^{[2]}} \tag{3.27}$$

$$\frac{\partial \boldsymbol{Z}^{[\mathbf{2}]}}{\partial w_{1,1}^{[2]}} = a_1^{[1]} \tag{3.28}$$

Sustituyendo (3.24, 3.25 y 3.28) en (3.23):

$$\frac{\partial E}{\partial w_{1,1}^{[2]}} = \frac{O-T}{O(1-O)} \cdot f^{'}(\boldsymbol{Z}^{[\mathbf{2}]}) \cdot a_1^{[1]} \tag{3.29}$$

De manera similar podemos obtener los valores de cada uno de los elementos restantes del vector $\nabla \boldsymbol{W}^{[\mathbf{2}]}$. A manera de resumen escribimos:

$$\nabla \boldsymbol{W}^{[\mathbf{2}]^{\boldsymbol{T}}} = \begin{bmatrix} \frac{\partial E}{\partial w_{1,1}^{[2]}} \\ \frac{\partial E}{\partial w_{2,1}^{[2]}} \\ \frac{\partial E}{\partial w_{3,1}^{[2]}} \\ \frac{\partial E}{\partial w_{4,1}^{[2]}} \end{bmatrix} = \begin{bmatrix} \frac{(O-T)}{O(1-O)} \cdot f^{'}(\boldsymbol{Z}^{[\mathbf{2}]}) \cdot a_1^{[1]} \\ \frac{(O-T)}{O(1-O)} \cdot f^{'}(\boldsymbol{Z}^{[\mathbf{2}]}) \cdot a_2^{[1]} \\ \frac{(O-T)}{O(1-O)} \cdot f^{'}(\boldsymbol{Z}^{[\mathbf{2}]}) \cdot a_3^{[1]} \\ \frac{(O-T)}{O(1-O)} \cdot f^{'}(\boldsymbol{Z}^{[\mathbf{2}]}) \cdot a_4^{[1]} \end{bmatrix} \tag{3.30}$$

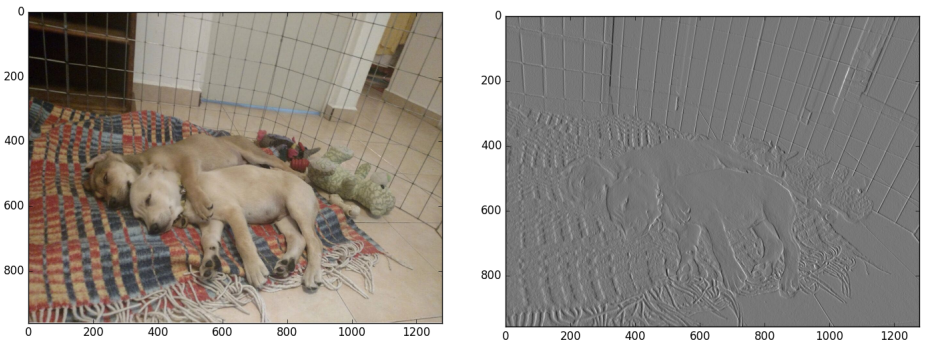

Figura 3.10: A la izquierda imagen original, a la derecha imagen con resultado

3.8. Redes neuronales convolucionales

En la literatura de visión por computadora, es común el uso de filtros (denotados con la letra $\boldsymbol{F}$) para obtener patrones o características de una imagen I. Un filtro es una matriz cuadrada de tamaño f. Para hacer uso de éste y obtener las propiedades deseadas de la imagen, se hace uso de la operación de convolución, la cual se denota con el símbolo *.

En la figura 3.10 se muestra un ejemplo de convolución para la obtención de bordes verticales. A la izquierda está la imagen original con dimensiones (800, 600, 3) y a la derecha la imagen resultado de dimensiones (800, 600). Para llegar a ésta última, primero se convierte la original a blanco y negro, posteriormente se le aplica la operación de convolución con un filtro de detección de bordes verticales. El filtro F utilizado fue:

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{3.31}$$

Anteriormente, una práctica común era proponer los valores de la matriz F de manera manual y analizarlos resultados empíricamente. Por supuesto, el proceso era lento y hasta cierto punto subjetivo. Con las RNC se resolvió en gran parte este problema, ya que la red permite aprender los valores del filtro de manera automática; más aún, nos permite concatenar filtros en cascada para capturar resultados mas complejos, tarea que resultaría prácticamente imposible de lograr

en forma manual.

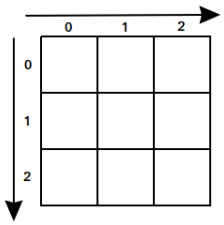

Figura 3.11: Dirección de convolución

3.9. Convolución

En esta sección se presenta la operación de convolución. Supongamos que tenemos una matriz bidimensional de entrada $\boldsymbol{I}$ de dimensiones (n_1, n_2), un filtro $\boldsymbol{F}$ de dimensiones (f_1, f_2) y queremos obtener el resultado de la convolución de la primera con la segunda, esto es, $\boldsymbol{I} * \boldsymbol{F}$. El tamaño de la matriz de salida $\boldsymbol{O}$ será $((n_1 - f_1 + 1),(n_2 - f_2 + 1))$. Los índices para referirnos a elementos de ambas matrices comienzan en 0 y su sentido es de arriba hacia abajo para filas y de izquierda a derecha para columnas, como se muestra en la figura 3.11.

Expresamos el resultado de la convolución para cada elemento de la matriz de salida $\boldsymbol{O}$ de la siguiente manera:

$$O_{i,j} = \sum_{m=0}^{f_1-1} \sum_{n=0}^{f_2-1} I_{i+m,j+n} F_{m,n} \tag{3.32}$$

A manera de ejemplo, la figura 3.12 muestra el cálculo de la matriz $\boldsymbol{O}$ a partir de dos matrices seleccionadas aleatoriamente (una representa la entrada $\boldsymbol{I}$ y la otra el filtro $\boldsymbol{F}$).

Para calcular el elemento $O_{0,0}$ de ésta, escribimos:

$$O_{0,0} = \sum_{m=0}^{2} \sum_{n=0}^{2} I_{m,n} F_{m,n} \tag{3.33}$$

$$O_{0,0} = I_{0,0}F_{0,0} + I_{0,1}F_{0,1} + \cdots + I_{2,2}F_{2,2} \tag{3.34}$$

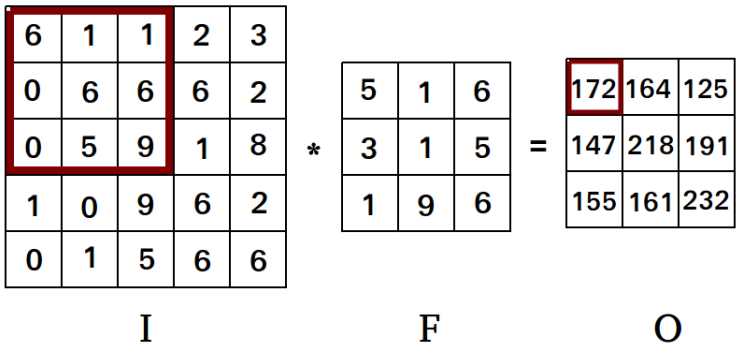

Figura 3.12: Ejemplo de operación de convolución

$$O_{0,0} = 30 + 1 + 6 + 0 + 6 + 30 + 0 + 45 + 54 = 172 \tag{3.35}$$

En la figura 3.12 se remarcaron 9 elementos de la matriz $\boldsymbol{I}$ y 1 de la matriz $\boldsymbol{O}$ con rojo para indicar que los primeros son los únicos que intervienen en el computo del segundo. Podemos visualizar el proceso de convolución de manera gráfica al imaginar una ventana sobre la matriz $\boldsymbol{I}$ de igual tamaño al filtro que se desliza a paso de una unidad de izquierda a derecha y de arriba a abajo en donde se realiza una multiplicación elemento por elemento entre los componentes que contiene esta ventana y el filtro, dando como resultado cada uno de los elementos de la matriz de salida.

Si la matriz de entrada $\boldsymbol{I}$ tiene tres dimensiones (n_1, n_2, n_3), el filtro $\boldsymbol{F}$ tiene que corresponder a ésta en profundidad y será de dimensiones (f_1, f_2, n_3). La salida $\boldsymbol{O}$ de la convolución entre $\boldsymbol{I}$ y $\boldsymbol{F}$ es bidimensional y tiene dimensiones $((n_1 - f_1 + 1),(n_2 - f_2 + 1))$. De similar manera podemos definir el resultado de la convolución para cada elemento en la matriz $\boldsymbol{O}$ de la siguiente forma:

$$O_{i,j} = \sum_{m=0}^{f_1-1} \sum_{n=0}^{f_2-1} \sum_{l=0}^{f_3-1} I_{i+m,j+n,k+l} F_{m,n,l} \tag{3.36}$$

En secciones anteriores, mostramos la importancia de utilizar funciones de activación no lineales; en las RNC se utiliza esta misma idea. Una capa convolucional se obtiene a partir del cálculo de 4 partes.

1. La convolución entre entre la matriz de entrada y el filtro, i.e. $(\boldsymbol{I} * \boldsymbol{F})$
2. Adición del sesgo a cada elemento de la matriz resultado de la convolución.
3. Aplicación una función de activación no lineal en cada elemento.
4. Implementación de una capa de muestreo (opcional y se detallará más adelante).

De manera concreta, definimos los elementos del proceso de convolución de una capa convolucional para entradas tridimensionales (antes de aplicar la función de activación) de la siguiente manera:

$$\boldsymbol{Z_{i,j}} = \sum_{m=0}^{f_1-1} \sum_{n=0}^{f_2-1} \sum_{l=0}^{f_3-1} I_{i+m,j+n,k+l} F_{m,n,l} + b \tag{3.37}$$

Aplicando la función de activación tenemos:

$$\boldsymbol{A_{i,j}} = f\left(\sum_{m=0}^{f_1-1} \sum_{n=0}^{f_2-1} \sum_{l=0}^{f_3-1} I_{i+m,j+n,k+l} F_{m,n,l} + b \right) \tag{3.38}$$

3.10. Convolución simétrica

Viendo los resultados de la sección anterior, notamos que la salida del proceso de convolución es de menor tamaño a la imagen original. En muchos casos, este fenómeno no es deseado y se busca que la matriz de salida tenga las mismas dimensiones que la de entrada. Para lograrlo, se amplía ésta a través de un proceso llamado rellenado, en el cual se agregan p columnas y filas saturadas de ceros en los limites de la matriz de entrada. En la figura 3.13 se hace un rellenado con $p = 1$ a la matriz de la izquierda. Si la imagen de entrada $\boldsymbol{I^{(1)}}$ tiene dimensiones (n_1, n_2), la imagen de salida $\boldsymbol{I^{(2)}}$ tendrá dimensiones $(n_1 + 2p, n_2 + 2p)$.

Previo a la operación de convolución es posible rellenar la matriz de entrada con un valor específico p para lograr que la matriz de salida tenga las mismas dimensiones que ésta. Cuando esto ocurre, hablamos de una convolución simétrica. Finalmente, mencionar que no es una práctica común el utilizar este método para el canal de profundidad (en caso de que se trate con una imagen tridimensional).

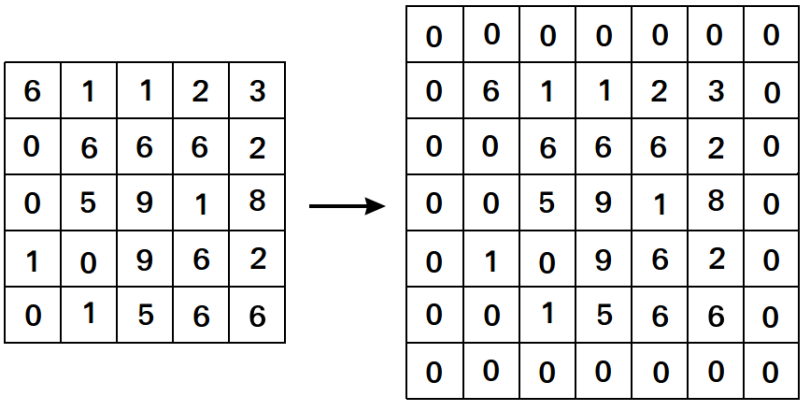

Figura 3.13: Ejemplo de rellenado con p = 1

3.11. Capa de muestreo

En algunas ocasiones, después de la convolución se utiliza un proceso de muestreo. En la gran mayoría de casos éste es de dos tipos: de máximo muestreo y de muestreo promedio. En ambas se define un tamaño de filtro f y un paso s.

Supongamos que tenemos una matriz de entrada $\boldsymbol{I}$ de dimensiones (n_1, n_2) a la cual deseamos aplicar una capa de muestreo, para esto, nos centramos en una subregión de $\boldsymbol{I}$ de dimensiones (f,f) abarcando desde la primera hasta la fila f-1. Desplazamos la región S elementos de izquierda a derecha y de arriba a abajo. El resultado de la capa de muestreo es el valor máximo de ésta región en cada desplazamiento en caso de utilizar una capa de máximo muestreo y el valor promedio en caso de hacer uso de una capa de muestreo promedio.

En la figura 3.14 se muestra un ejemplo de la aplicación de una capa de máximo muestreo. En ésta, la matriz de entrada es de dimensiones (4,4) y $S = f = 2$. La primera región es marcada con color rojo; en ésta, el valor máximo que encontramos es 6, por lo que éste pasa como resultado a la matriz de salida. Posteriormente, deslizamos la región dos unidades a la derecha y vemos que nuevamente el 6 es el valor máximo. Como no se puede deslizar más la región hacia la derecha, la movemos hacia abajo y empezamos nuevamente los desplazamientos de izquierda a derecha. Las dimensiones de la matriz de salida son $(n_1 - f + 1, n_2 - f + 1)$.

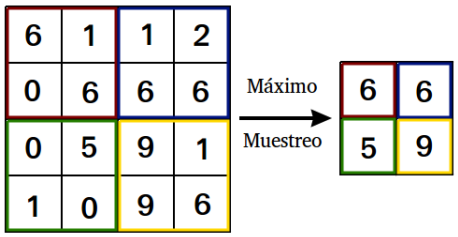

Figura 3.14: Ejemplo de capa de máximo muestreo.

3.12. Dispersión de conexiones y división de parámetros

Las RNC tienen dos propiedades importantes: la dispersión de conexiones y la distribución de parámetros. Para entenderlas, resulta útil transformar el proceso de convolución en una RN tradicional. En la figura 3.15 se muestra en la izquierda una matriz de entrada con 9 valores, un filtro de 4 elementos y una matriz resultado con 4 componentes; a la derecha encontramos la RN tradicional que modela esta operación. Para facilitar la visualización sólo se muestra el peso w_1.

Como podemos ver en la imagen, la red no está completamente conectada. Cada salida depende únicamente de 4 conexiones; además, a pesar de que existen 16 conexiones, únicamente se necesitan 4 pesos. Esta dispersión de conexiones y compartimiento de pesos es una de las razones por la que las RNC se han destacado, ya que nos permiten modelar y procesar entradas de gran tamaño con un número relativamente pequeño de parámetros.

3.13. Ejemplo de red neuronal convolucional y notación

En esta sección detallamos una pequeña RNC y definimos la notación que se usará en el resto de la tesis. En la figura 3.16 tenemos una red de dos capas, en la primera únicamente se realiza la operación de convolución y en la segunda, además del proceso de convolución, se incluye una capa de máximo muestreo. Notamos nuevamente que por convención a la capa de entrada se le denota también como capa 0.

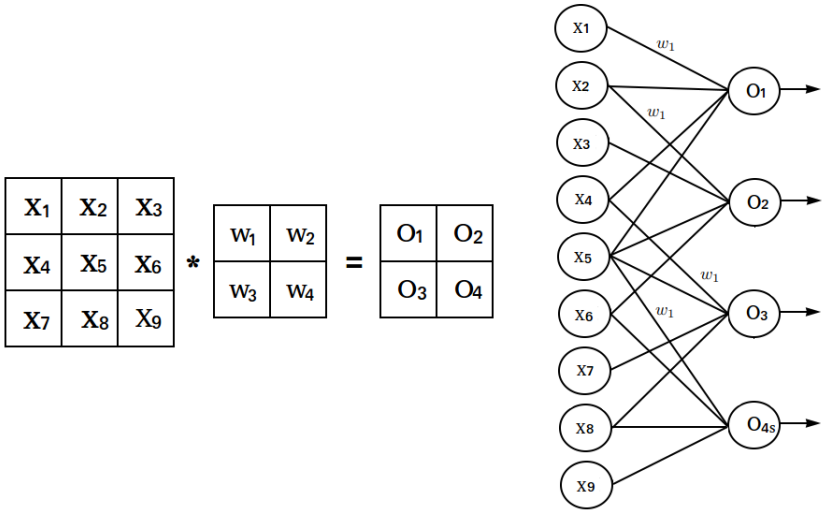

Figura 3.15: Propiedades de las redes neuronales convolucionales.

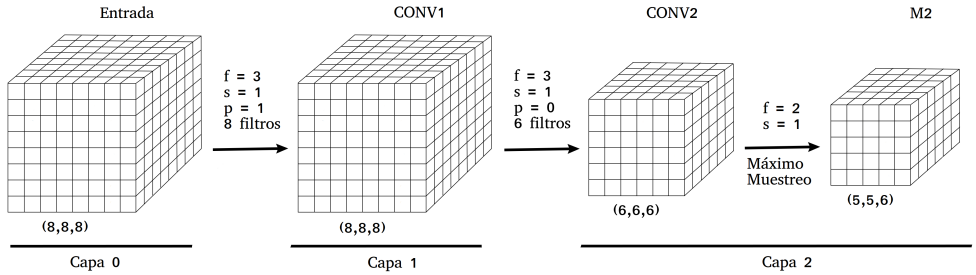

Figura 3.16: Ejemplo de red neuronal convolucional

Se utilizará la notación $\boldsymbol{A}^{[l]}$ pero ahora para referirse al resultado del proceso de convolución después de añadir el sesgo y aplicar la función de activación, por ejemplo, en la figura 3.16 $\boldsymbol{A}^{[1]}$ hace referencia al tensor de dimensiones (8,8,8) situada abajo del indicador CONV1. $\boldsymbol{Z}^{[l]}$ hará referencia a la matriz o tensor resultado del proceso de convolución previo a la aplicación de la función de activación. Importante notar que $\boldsymbol{Z}^{[l]}$ no se muestra gráficamente en la figura 3.16.

Las flechas entre matrices indican el flujo y procesamiento de datos, los parámetros que se muestran arriba y abajo de éstas resumen y nos indican qué tipo de operación se llevó a cabo y cómo se implementó. Recordamos que los parámetros p, S, y f son el rellenado, el paso y el tamaño del filtro.

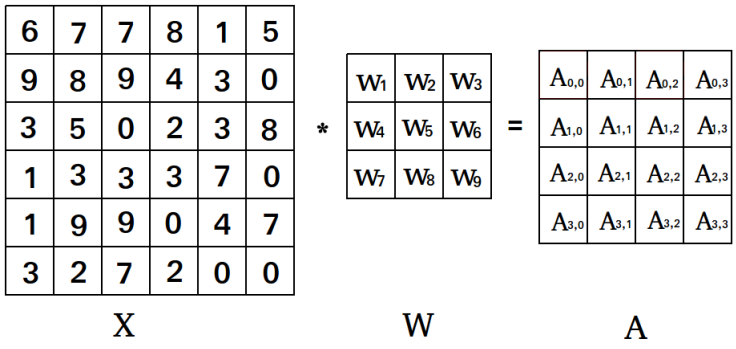

Figura 3.17: Ejemplo de RNC.

Finalmente, notamos dos cosas: el tipo de convolución para la primera capa de la figura 3.16 es simétrica, ya que mantiene las dimensiones de la entrada. En segundo lugar, por convención se agrupa al proceso de convolución con el de muestreo en una misma capa, lo que facilita y reduce el número de índices y capas descritas.

3.14. Algoritmo de retropropagación

Al igual que en la implementación de la RNM, el ARP hace uso de la regla de la cadena en cálculo multivariable para obtener el cambio de los parámetros. En la figura 3.17 se muestra una pequeña RNC, la cual va a fungir como ejemplo de implementación.

Como preámbulo, vemos que en la imagen se muestra explícitamente el filtro, lo cual no es común; sin embargo, en este ejemplo nos servirá para detallar el proceso de retropropagación. Como podemos apreciar, la RNC consta de una sola capa, la cual utiliza la función de activación ReLU; además, a pesar de que no se muestra en la imagen, se considerará el sesgo. No se aplica el proceso de rellenado y el paso es de 1 elemento por movimiento. Finalmente, mencionamos que por el momento sólo se introduce una patrón de entrenamiento.

Definimos (i, j), (m, n) y (α, β) como los índices de las matrices $\boldsymbol{X}$, $\boldsymbol{W}$ y $\boldsymbol{A}$ respectivamente. Por cuestiones prácticas, denotamos a $\boldsymbol{A}^{[1]}$, $\boldsymbol{Z}^{[1]}$, $b^{[1]}$ y $\boldsymbol{W}^{[1]}$ como $\boldsymbol{A}$, $\boldsymbol{Z}$, b y $\boldsymbol{W}$ respectivamente. Supongamos que deseamos que la suma sobre los elementos de la matriz $\boldsymbol{A}$ sea igual a una cantidad predeterminada T, en este caso 250. Podemos definir nuestra función de costo como:

$$E = \frac{1}{2}(T - Y)^2 \tag{3.39}$$

En donde:

$$Y = \sum_{\alpha,\beta} A_{\alpha,\beta} \tag{3.40}$$

El siguiente paso es calcular el gradiente de la matriz $\boldsymbol{W}$ y del sesgo b. Aplicando la regla de la cadena podemos escribir:

$$\frac{\partial E}{\partial w_{m',n'}} = \frac{\partial E}{\partial Y} \cdot \frac{\partial y}{\partial \boldsymbol{A}} \cdot \frac{\partial \boldsymbol{A}}{\partial \boldsymbol{Z}} \cdot \frac{\partial \boldsymbol{Z}}{\partial w_{m',n'}} \tag{3.41}$$

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial y} \cdot \frac{\partial Y}{\partial \boldsymbol{A}} \cdot \frac{\partial \boldsymbol{A}}{\partial \boldsymbol{Z}} \cdot \frac{\partial \boldsymbol{Z}}{\partial b} \tag{3.42}$$

Desarrollando las ecuaciones (3.41) y (3.42) llegamos a las siguientes expresiones:

$$\frac{\partial E}{\partial w_{m',n'}} = -(T - Y) \sum_{\alpha} \sum_{\beta} ReLU^{'}(Z_{\alpha,\beta}) X_{\alpha+m',\beta+n'} \tag{3.43}$$

$$\frac{\partial E}{\partial b} = -(T - Y) \sum_{\alpha} \sum_{\beta} ReLU^{'}(Z_{\alpha,\beta}) \tag{3.44}$$

Habiendo calculado las dos expresiones anteriores, podemos hacer uso del algoritmo del descenso del gradiente para actualizar los parámetros de nuestra red. Para lograr esto, utilizamos las siguientes dos ecuaciones:

$$w_{m',n'} \leftarrow w_{m',n'} - \left(TA * \frac{\partial E}{w_{m',n'}} \right) \tag{3.45}$$

$$b \leftarrow b - \left(TA * \frac{\partial W}{\partial b} \right) \tag{3.46}$$

Teniendo nuestras ecuaciones para actualizar los parámetros de la red en cada iteración, podemos entrenar el modelo. En la figura 3.18 se muestra la gráfica de costo contra el número de iteración. En este caso, el costo se reduce de manera drástica en los primeros ciclos y se acerca a 0. En la figura 3.19 se muestra la evolución del costo en las primeras diez iteraciones, la matriz de pesos W y el sesgo b del último ciclo.

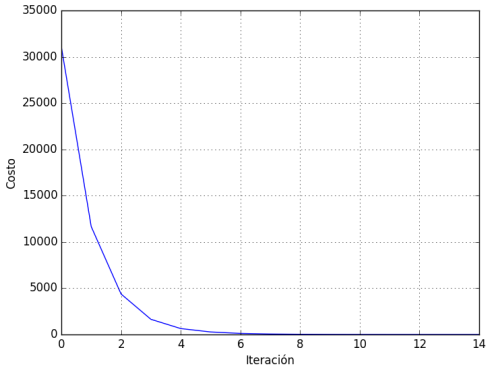

Figura 3.18: Gráfica de costo contra iteración.

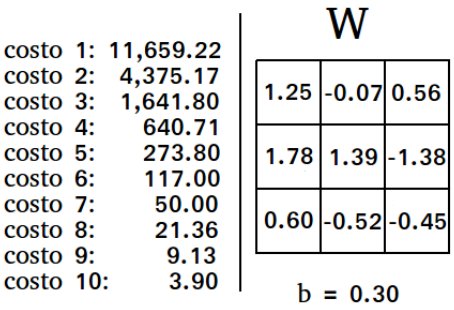

Figura 3.19: Resultados de la RNC ejemplo

Capítulo 4

Predicción de la densidad en un fluido compresible

Como preámbulo, hago mención y agradecimiento al estudiante de doctorado Christian Lagarza, quien es el autor del código para la simulación del fluido compresible que será utilizada durante el resto del capítulo.

En esta sección se detallará la teoría de redes neuronales en la solución de raíces de polinomios y se mostrará la implementación de ésta en un caso de aplicación para la simulación de un fluido compresible. El fluido a tratar es nitrógeno en condiciones supercríticas en un espacio de simulación de dimensiones (128, 128, 350), dando un total de 5,734,400 partículas.

Retomando lo que se mencionó en el capítulo 1, de manera general podemos expresar un polinomio como:

$$f(x) = a_0 x^n + a_1 x_{n-1} + a_2 x_{n-1} + \cdots + a_{n-1} x + a_n = \sum_{k=0}^{n} a_{n-k} x^k \tag{4.1}$$

En donde $x, a_i \in \mathbb{C}$. Dividiendo (4.1) entre a_0 obtenemos:

$$p(x) = x^n + b_0 x^{n-1} + b_1 x^{n-2} + \cdots + b_{n-1} x + b_n \tag{4.2}$$

Debido a que las raíces de (4.1) son las mismas que las de (4.2), podemos trasladar el problema hacia la solución de (4.2). Además, si definimos un vector de raíces $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \lambda_3, ...]^T \in \mathbb{C}$, podemos reescribir (4.2) como:

$$p(x) = \prod_{i=1}^{n} (x - \lambda_i) \tag{4.3}$$

El objetivo entonces es buscar una solución para las raíces de (4.3). En este caso se utilizaran RN y se propondrán dos métodos de solución. El primero planteado for Huang et al. [25] hace uso de restricciones sobre el algoritmo de entrenamiento, el segundo, desarrollado personalmente en esta tesis, sugiere el uso de restricciones sobre los pesos de la red.

4.1. Método 1: Restricciones sobre el algoritmo de entrenamiento

Como se mencionó en el capítulo 1, es posible incorporar restricciones sobre el algoritmo de aprendizaje para mejorar el tiempo de convergencia y la generalidad de la red. Huang et al. [25] propusieron la incorporación de las identidades de Newton [46] al algoritmo de aprendizaje, las cuales podemos escribir para un polinomio arbitrario como:

$$\begin{aligned}
\Phi_1 &= S_1 + a_1 = 0 \\
\Phi_2 &= S_2 + a_1 S_1 + 2a_2 = 0 \\
&\cdot \\
&\cdot \\
\Phi_m &= S_m + a_1 S_{m-1} + \cdots + a_n S_{m-n} = 0
\end{aligned} \tag{4.4}$$

En donde S_m es el m momento de raíz. Y se define como:

$$S_m = \lambda_1^m + \lambda_2^m + \cdots + \lambda_n^m = \sum_{i=1}^{n} \lambda_i^m \tag{4.5}$$

Además, $S_0 = n$ y vemos que $\left(\frac{dS_m}{d\lambda_i} = m\lambda_i^{m-1}\right)$. El símbolo Φ_m hace referencia a la m restricción, en donde $m \in \{0 \leq \mathbb{Z} \leq n\}$. Aterrizando las anteriores ecuaciones en un ejemplo, supongamos que tenemos un polinomio de quinto grado en la forma:

$$p(x) = x^5 + a_0 x^4 + a_1 x^3 + a_2 x^2 + a_3 x + a_4 \tag{4.6}$$

Para este polinomio, podemos calcular las identidades de Newton como:

$$\begin{bmatrix}\Phi_1\\ \Phi_2\\ \Phi_3\\ \Phi_4\\ \Phi_5\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0 & 0\\ S_1 & 2 & 0 & 0 & 0\\ S_2 & S_1 & 3 & 0 & 0\\ S_3 & S_2 & S_1 & 4 & 0\\ S_4 & S_3 & S_2 & S_1 & 5\end{bmatrix}\begin{bmatrix}a_1\\ a_2\\ a_3\\ a_4\\ a_5\end{bmatrix} + \begin{bmatrix}S_1\\ S_2\\ S_3\\ S_4\\ S_5\end{bmatrix} \tag{4.7}$$

Una vez establecidas las restricciones que se usarán en este método, podemos pasar al diseño de la arquitectura de la RN, el cual puede ser de dos tipos: Sigma y Pi. En las siguientes dos secciones se detallan cada una de éstas; sin embargo, para contrastar el efecto de las restricciones, primero se implementan estas dos arquitecturas sin ellas.

4.1.1. Arquitectura Pi sin restricciones

Esta primera red consta de tres capas, la primera de éstas contiene dos neuronas, la segunda n (el grado del polinomio) y la tercera solamente una. Un patrón de entrada $\boldsymbol{x}$ para esta red es un vector de la forma: $\boldsymbol{x} = [-1, x]^T$, en donde $x \in \mathbb{C}$. Definimos la matriz de pesos $\boldsymbol{W^{[1]}}$ de la siguiente forma:

$$\boldsymbol{W^{[1]}} = \begin{bmatrix} w_{1,1}^{[1]} & w_{2,1}^{[1]} \\ w_{1,2}^{[1]} & w_{2,1}^{[1]} \\ \cdot & \cdot \\ \cdot & \cdot \\ w_{1,n}^{[1]} & w_{2,n}^{[1]} \end{bmatrix} = \begin{bmatrix} 1 & w_{2,1}^{[1]} \\ 1 & w_{2,1}^{[1]} \\ \cdot & \cdot \\ \cdot & \cdot \\ 1 & w_{2,n}^{[1]} \end{bmatrix} \tag{4.8}$$

Como podemos ver, los pesos entre la entrada x y la capa oculta se definen con la constante 1; lo que en otras palabras significa que solamente los pesos que se encuentran entre la entrada marcada con -1 y la capa oculta serán entrenados. Por otra parte, definimos la matriz de pesos $\boldsymbol{W^{[2]}}$ como:

$$\boldsymbol{W^{[2]}} = \begin{bmatrix} w_{1,1}^{[2]} \\ w_{2,1}^{[2]} \\ \cdot \\ \cdot \\ w_{n,1}^{[2]} \end{bmatrix}^T = \begin{bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ 1 \end{bmatrix}^T \tag{4.9}$$

Habiendo establecido los parámetros de la red, mencionamos que no existe función de activación ni sesgo para ninguna de sus capas, más adelante se mostrará porqué se toma esta decisión.

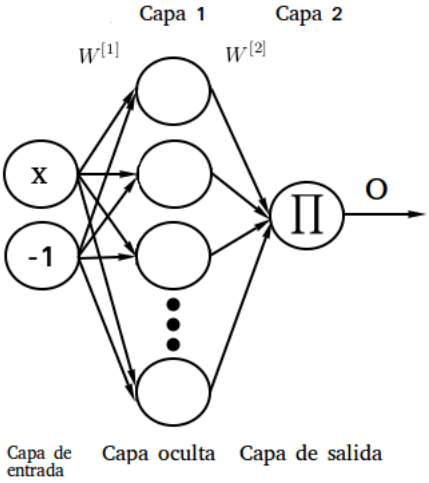

Figura 4.1: Modelo Pi - Huang.

Para entender el razonamiento detrás del diseño de la arquitectura, veamos lo que se computa en la capa oculta. Recordamos que la salida de ésta se denota con $\boldsymbol{Z^{[1]}}$ y es un vector que contiene las salidas de cada una de las neuronas, i.e. $\boldsymbol{Z^{[1]}} = [z_1^{[1]}\ z_2^{[1]}\ \cdots\ z_n^{[1]}]^T$; notamos que en esta ocasión $\boldsymbol{A^{[1]}} = \boldsymbol{Z^{[1]}}$ y finalmente precisamos que:

$$\boldsymbol{A^{[1]}} = \boldsymbol{Z^{[1]}} = \boldsymbol{W^{[1]}}\boldsymbol{A^{[0]}} \tag{4.10}$$

En donde $\boldsymbol{A^{[0]}}$ es la matriz que contiene las entradas y es de dimensiones $(2, P)$. Podemos escribir la salida de cada una de las neuronas de la capa oculta como:

$$a_i^{[1]} = z_i^{[1]} = (x - w_{2,i}^{[1]}) \tag{4.11}$$

Pasando a la capa de salida, notamos que nuevamente no existe función de activación, por lo que al igual que en la capa 1, $\boldsymbol{A^{[2]}} = \boldsymbol{Z^{[2]}}$. Para calcular la salida definimos a la neurona como una unidad de multiplicación sobre sus entradas, o lo que es lo mismo, sobre la salida de las neuronas de la capa oculta. De manera concreta, podemos escribir la salida de la segunda capa como:

$$\boldsymbol{A^{[2]}} = \boldsymbol{Z^{[2]}} = \prod_{i=1}^{n} a_i^{[1]} \tag{4.12}$$

Sustituyendo (4.11) en (4.12):

$$\boldsymbol{O} = \boldsymbol{A^{[2]}} = \prod_{i=1}^{n} (z - w_{2,i}^{[1]}) \tag{4.13}$$

Si comparamos las ecuaciones (4.13) y (4.3) vemos que la única diferencia es el sustraendo dentro de la multiplicación, en otras palabras, si los pesos de la red se eligen con igual valor a las raíces del polinomio, la salida de la red representará a éste. En capítulos anteriores se mostró cómo una RN puede actualizar sus parámetros hasta alcanzar una salida deseada, en este caso, los parámetros por aprender representan las raíces del polinomio y el valor deseado es la evaluación de éste.

4.1.1.1. Algoritmo de aprendizaje

Para la arquitectura pi definimos la función de costo como:

$$E = \frac{1}{2P}\sum_{p=1}^{P}|T_p - O_p|^2 \tag{4.14}$$

Podemos utilizar el algoritmo de retropropagación para encontrar la tasa de cambio del costo con respecto a los pesos de la red de la siguiente forma:

$$\frac{\partial E}{\partial w_{2,i}^{[1]}} = \frac{\partial E}{\partial O_p}\cdot\frac{\partial O_p}{\partial w_{2,i}^{[1]}} \tag{4.15}$$

Empezando por el primer término del lado derecho de la igualdad y recordando que $\frac{d|x|}{dx} = \frac{x}{|x|}$:

$$\frac{\partial E}{\partial O_p} = \frac{1}{P}\sum_{p=1}^{P}|T_p - O_p|\cdot\frac{\partial|T_p - O_p|}{\partial O_p} \tag{4.16}$$

$$= -\frac{1}{P}\sum_{p=1}^{P}|T_p - O_p|\cdot\frac{T_p - O_p}{|T_p - O_p|} \tag{4.17}$$

$$= -\frac{1}{P}\sum_{p=1}^{P}(T_p - O_p) \tag{4.18}$$

Por otra parte, podemos obtener el segundo término de la siguiente manera:

$$\frac{\partial O_p}{\partial w_{2,i}^{[1]}} = \frac{\partial}{\partial w_{2,i}^{[1]}} \prod_{i=1}^{n} \left(x - w_{2,i}^{[1]}\right) \tag{4.19}$$

$$= \frac{\partial}{\partial w_{2,i}^{[1]}} \left(x - w_{2,1}^{[1]}\right)\left(x - w_{2,2}^{[1]}\right) \cdots \left(x - w_{2,n}^{[1]}\right) \tag{4.20}$$

$$= -\prod_{i \in R} \left(x - w_{2,i}^{[1]}\right) \tag{4.21}$$

En esta última ecuación, $R = \{j : j \in \mathbb{Z}, 1 \leq j \leq n, j \neq i\}$. Sustituyendo (4.18) y (4.21) en (4.15):

$$\frac{\partial E}{\partial w_{2,i}^{[1]}} = \frac{1}{P} \sum_{p=1}^{P} \left((T_p - O_p) \cdot \prod_{i \in R} \left(x - w_{2,i}^{[1]}\right) \right) \tag{4.22}$$

Finalmente, podemos utilizar el algoritmo del descenso del gradiente para actualizar los pesos de la red en cada iteración de la siguiente forma:

$$w_{2,i}^{[1]} \leftarrow w_{2,i}^{[1]} - \left(TA * \frac{\partial E}{\partial w_{2,i}^{[1]}} \right) \tag{4.23}$$

4.1.1.2. Resultados

Debido a que la definición de la última capa en la red funge como un operador de multiplicación, se ha demostrado empíricamente [25] que la superficie de la función de costo es irregular y no convexa, por lo que encontrar un mínimo global resulta difícil para el algoritmo del descenso del gradiente. Esta arquitectura no va a ser utilizada en el resto de la tesis por su pobre desempeño; sin embargo, es importante mostrar un ejemplo de su implementación para contrastar resultados con el resto de las arquitecturas.

Supongamos que queremos encontrar las raíces del polinomio:

$$p(x) = x^5 - 4x^4 + 4.43x^3 - 0.164x^2 - 1.464x + 0.144 \tag{4.24}$$

En este caso, nuestra red tendrá 5 neuronas en la capa oculta. Además; definimos los hiperparámetros de la red como: $P = 50$, $TA = 0.00005$, $iteraciones = 2,000,000$. En la figura 4.2 se muestra el valor de la función de costo con respecto al número de iteración.

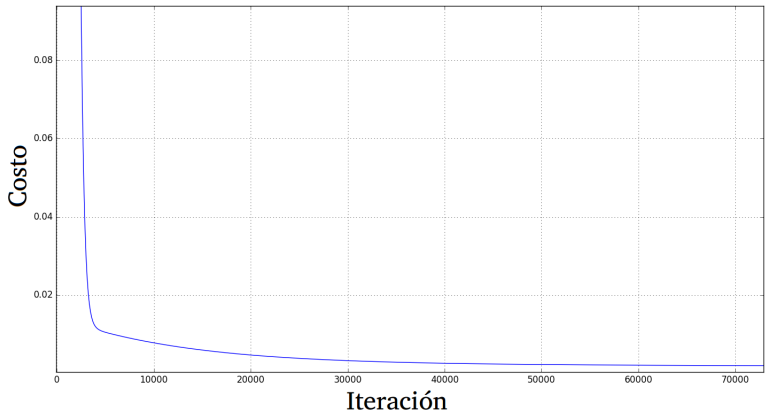

Figura 4.2: Costo contra iteración de modelo Pi.

Como podemos ver, la función de costo se reduce en las primeras 40,000 iteraciones (aproximadamente) y después continua un camino asintótico hacia cero. Puede parecer que la aproximación de las raíces es buena ya que el costo se aproxima a cero; sin embargo, para una gran cantidad de aplicaciones en física se requiere de una alta precisión. Veamos ahora las aproximaciones de las raíces. La figura 4.3 muestra las 5 aproximaciones de éstas por la red en negro y en rojo el valor real.

Es importante destacar el lento aprendizaje de la red, a ésta le toma millones de iteraciones alcanzar aproximaciones aceptables en algunas raíces, por lo que no podrá ser utilizada en aplicaciones donde la velocidad de cómputo es un factor importante. En cuanto a resultados se refiere, sólo la primera, la tercera y la quinta raíz logran alcanzar valores aceptables, el resto de las raíces de la red tienen un mal desempeño.

El vector de raíces del polinomio es:

$$\boldsymbol{\lambda} = [1.2, 2, 0.1, -0.5, 1.2] \tag{4.25}$$

Por otro lado, las raíces resultado de la red son:

$$\boldsymbol{w} = [-0.4994, 1.2847, 0.1007, 1.1164, 1.9982] \tag{4.26}$$

Podemos definir la siguiente métrica para medir el desempeño de la red:

$$d = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{\lambda_i - w_i}{\lambda_i} \right| \tag{4.27}$$

En donde d representa el error relativo promedio. Utilizando los valores en (4.25) y en (4.26) podemos computar (4.27). En este caso: $d = 0.02987$.

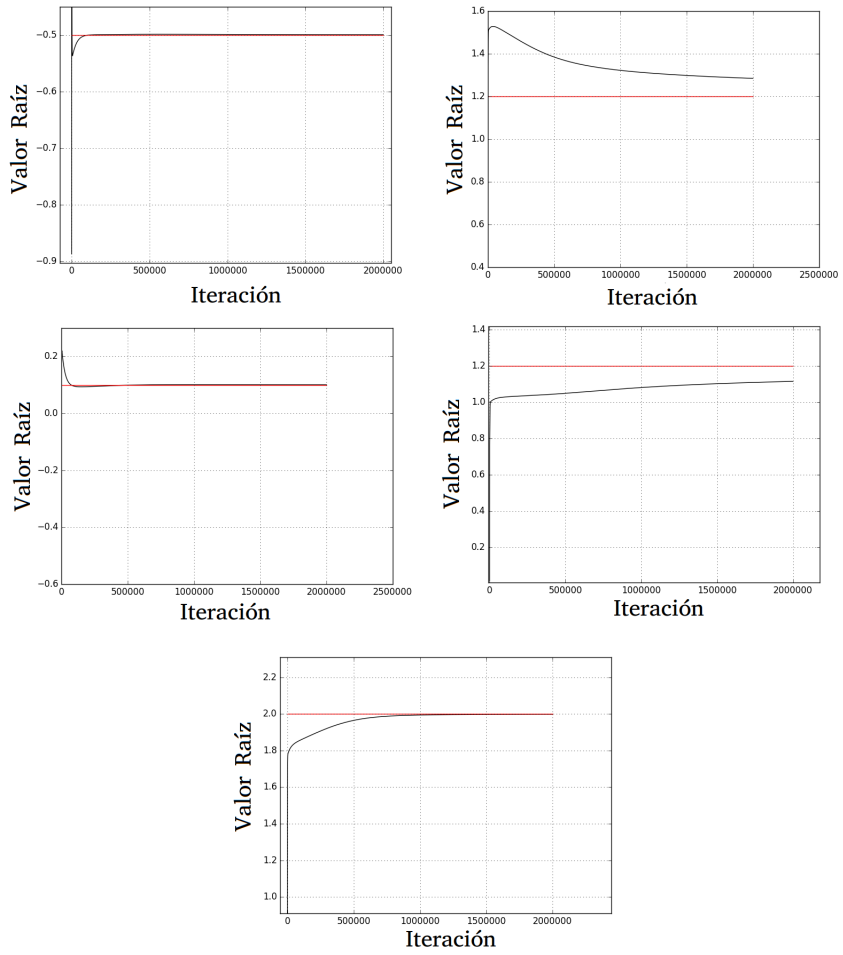

Figura 4.3: Costo contra iteración para las raíces del modelo Pi.

4.1.2. Arquitectura Sigma sin restricciones

El modelo Sigma propuesto por Huang et al. es muy similar al modelo Pi. Sin embargo, existen diferencias específicas que producen cambios en el comportamiento, velocidad y consistencia de la red. En primer lugar, destacamos que las matrices de pesos $\boldsymbol{W}^{[1]}$ y $\boldsymbol{W}^{[2]}$ y las entradas de la red no presentan cambios; por otra parte, se agrega una función de activación sobre la capa oculta, la cual definimos como:

$$f(\cdot) = ln\left(|\cdot|\right) \tag{4.28}$$

Otro cambio importante ocurre en la capa de salida, en la anterior arquitectura la única neurona de esta capa funge como operador de multiplicación, en contraste, en la red Sigma, la neurona de la capa de salida produce la suma de los resultados de la capa oculta. En la figura 4.4 se muestra la arquitectura de este modelo.

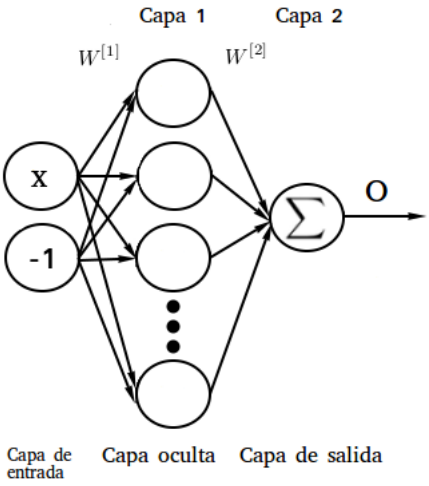

Figura 4.4: Modelo Sigma - Huang.

Las ecuaciones que caracterizan a esta red son:

$$\boldsymbol{Z^{[1]}} = \boldsymbol{W^{[1]}}\boldsymbol{A^{[0]}} \tag{4.29}$$

$$\boldsymbol{A^{[1]}} = ln\left|\boldsymbol{Z^{[1]}}\right| = ln\left|\boldsymbol{W^{[1]}}\boldsymbol{A^{[0]}}\right| \tag{4.30}$$

$$\boldsymbol{A^{[2]}} = \boldsymbol{W^{[2]}}\boldsymbol{A^{[1]}} \tag{4.31}$$

Para esclarecer el proceso interno de la red, podemos escribir la salida de cada una de las neuronas pertenecientes a la capa oculta de la siguiente manera (para un patrón de entrenamiento):

$$z_i^{[1]} = \left(x - w_{2,i}^{[1]}\right) \tag{4.32}$$

$$a_i^{[1]} = ln|z_i^{[1]}| = ln\left|x - w_{2,i}^{[1]}\right| \tag{4.33}$$

Además, podemos escribir la salida de la red para un patrón de entrenamiento como:

$$O = A^{[2]} = ln\left|x - w_{2,1}^{[1]}\right| + ln\left|x - w_{2,2}^{[1]}\right| + \cdots + ln\left|x - w_{2,n}^{[1]}\right| \tag{4.34}$$

Por otra parte, si aplicamos nuestra función de activación en la ecuación (4.3) obtenemos:

$$ln\left|\prod_{i=1}^{n}(x - \lambda_i)\right| = ln\left|(x - \lambda_1)(x - \lambda_2)\cdots(x - \lambda_n)\right| \tag{4.35}$$

$$= ln\left|x - \lambda_1\right| + ln\left|x - \lambda_2\right| + \cdots + ln\left|x - \lambda_n\right| \tag{4.36}$$

Si comparamos las ecuaciones (4.34) y (4.36) vemos que la única diferencia se encuentra en los sustraendos de cada expresión. En otras palabras, nuestra red equivale al logaritmo natural del valor absoluto del polinomio a tratar, si y sólo si, los pesos de la red $w_{2,i}^{[1]}$ son iguales a las raíces del polinomio.

4.1.2.1. Algoritmo de aprendizaje

Nuevamente utilizaremos la misma función de costo:

$$E = \frac{1}{2P}\sum_{i=1}^{n}|T_p - O_p|^2 \tag{4.37}$$

Para realizar el proceso de aprendizaje utilizando el algoritmo del descenso del gradiente necesitamos computar la derivada del error con respecto a los pesos $w_{2,i}^{[1]}$. Para esto, podemos escribir:

$$\frac{\partial E}{\partial w_{2,i}^{[1]}} = \frac{\partial E}{\partial O_p}\cdot\frac{\partial O_p}{\partial w_{2,i}^{[1]}} \tag{4.38}$$

El primer término del lado derecho de la igualdad lo podemos calcular de la siguiente forma:

$$\frac{\partial E}{\partial O_p} = \frac{1}{P}\sum_{p=1}^{P}|T_p - O_p|\cdot\frac{\partial|T_p - O_p|}{\partial O_p} \tag{4.39}$$

$$= -\frac{1}{P}\sum_{p=1}^{P}|T_p - O_p|\cdot\frac{T_p - O_p}{|T_p - O_p|} \tag{4.40}$$

$$= -\frac{1}{P}\sum_{p=1}^{P}(T_p - O_p) \tag{4.41}$$

Para calcular el segundo término recordamos que $\frac{dln(x)}{dx} = \frac{1}{x}$. De esta manera:

$$\frac{\partial O_p}{\partial w_{2,i}^{[1]}} = \frac{\partial}{\partial w_{2,i}^{[1]}} \left\{ ln \left| x - w_{2,1}^{[1]} \right| + ln \left| x - w_{2,2}^{[1]} \right| + \cdots + ln \left| x - w_{2,n}^{[1]} \right| + \right\} \tag{4.42}$$

$$= \frac{\partial}{\partial w_{2,i}^{[1]}} ln \left| x - w_{2,i}^{[1]} \right| = - \frac{x - w_{2,i}^{[1]}}{\left| x - w_{2,i}^{[1]} \right|^2} \tag{4.43}$$

Sustituyendo las ecuaciones (4.41) y (4.43) en (4.38):

$$\frac{\partial E}{\partial w_{2,i}^{[1]}} = \frac{1}{P} \sum_{p=1}^{P} (T_p - O_p) \cdot \frac{x - w_{2,i}^{[1]}}{\left| x - w_{2,i}^{[1]} \right|^2} \tag{4.44}$$

Finalmente, podemos definir el algoritmo de aprendizaje a través de la siguiente regla de actualización de pesos:

$$w_{2,i}^{[1]} \leftarrow w_{2,i}^{[1]} - \left(TA * \frac{\partial E}{\partial w_{2,i}^{[1]}} \right) \tag{4.45}$$

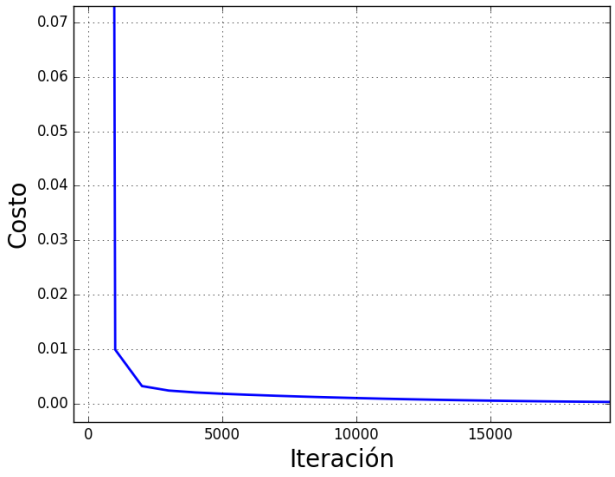

Figura 4.5: Costo contra iteración modelo Sigma.

4.1.2.2. Resultados

Lo que se espera de esta arquitectura es una mejor aproximación de las raíces y una reducción del tiempo de cómputo. Probemos nuevamente con el polinomio

descrito en (4.24). En la figura 4.5 se muestra la función de costo contra el número de iteración.

Nuevamente vemos que es difícil interpretar el desempeño de la red a partir de la gráfica de costo. Para resolver esto, en la figura 4.6 se muestra en negro la aproximación de la red y en rojo el valor objetivo.

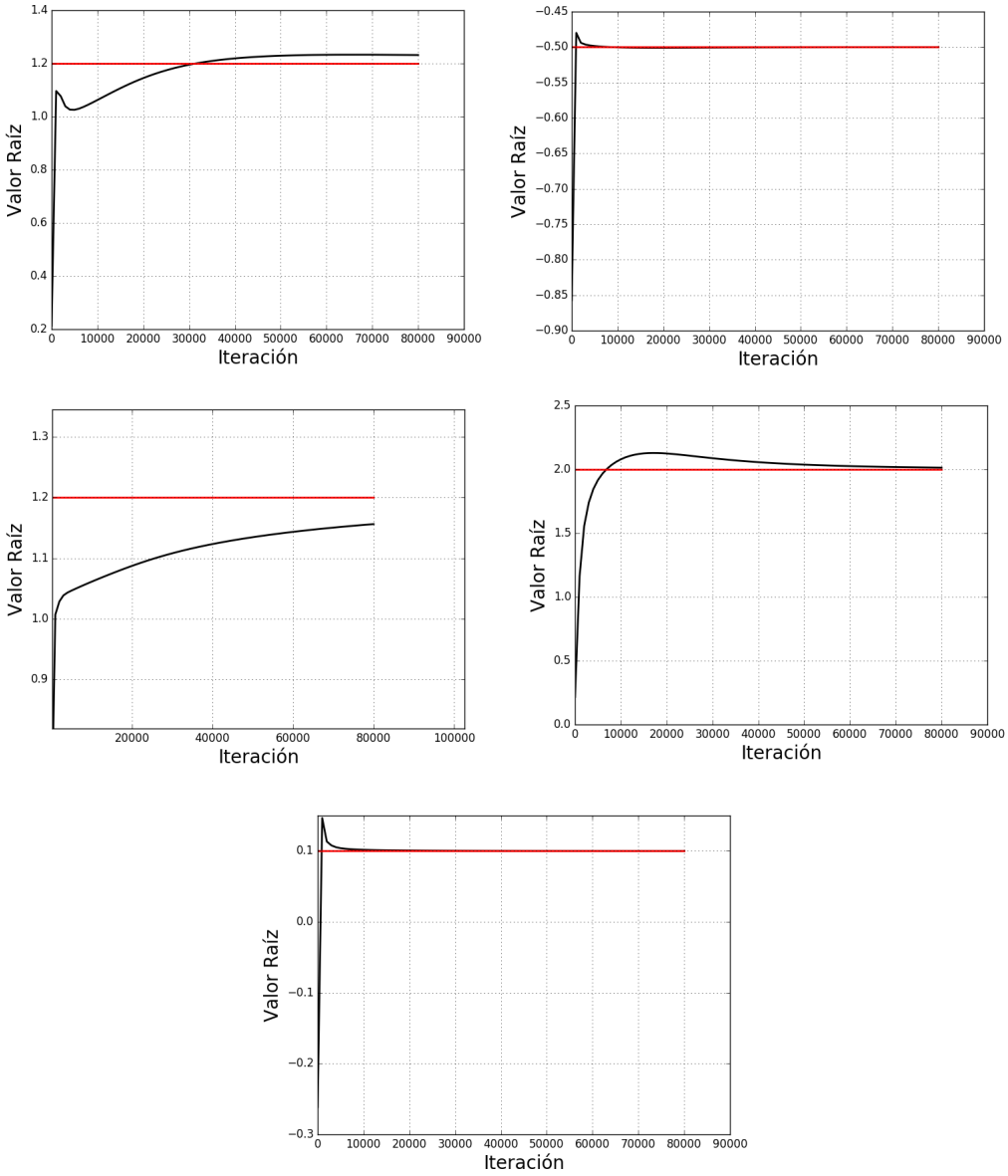

Figura 4.6: Aproximación de pesos contra iteración.

El vector de raíces resultado de la red es:

$$\boldsymbol{w} = [1.2317, -0.5001, 1.1560, 2.0138, 0.1000] \tag{4.46}$$

Recordando que las raíces reales del polinomio son: $\boldsymbol{\lambda} = [1.2, -0.5, 1.2, 2, 0.1]$, nuestra métrica de distancia es: $d = 0.0140$. Claramente podemos ver que el desempeño de la red mejoró, no sólo aumentó la precisión de las raíces, sino que disminuyó el número de iteraciones necesarias drásticamente; sin embargo, a la red aún le toma un gran número de ciclos obtener una buena aproximación, por lo que no es factible utilizarla como método alternativo en la gran mayoría de aplicaciones.

4.1.3. Arquitectura Sigma con restricciones

En esta sección se detallará e implementará la red neuronal Sigma incluyendo las restricciones mencionadas al inicio del capítulo 4. Se denotará con la letra mayúscula Phi al vector de restricciones del polinomio, esto es, $\boldsymbol{\Phi} = [\Phi_1, \Phi_2, \cdots, \Phi_3]^T$.

Anteriormente definimos nuestra regla de actualización de pesos como:

$$w_{2,i}^{[1]} \leftarrow w_{2,i}^{[1]} - \left(TA \cdot \frac{\partial E}{\partial w_{2,i}^{[1]}} \right) \tag{4.47}$$

Si introducimos la variable t para representar el número de iteración, podemos expresar (4.47) como:

$$w_{2,i}^{[1]}(t+1) = w_{2,i}^{[1]}(t) - \left(TA \cdot \frac{\partial E}{\partial w_{2,i}^{[1]}} \right) \tag{4.48}$$

Entonces:

$$\Delta w_{2,i}^{[1]} = - \left(TA \cdot \frac{\partial E}{\partial w_{2,i}^{[1]}} \right) \tag{4.49}$$

Si las variaciones en los pesos son lo suficientemente pequeños, podemos aproximar el cambio de éstos a través de su primer diferencial, i.e. $\Delta w_{2,i}^{[1]} \approx dw_{2,i}^{[1]}$, de manera similar, podemos aproximar el cambio en la función de error como:

$$dE = \frac{\partial E}{\partial w_{2,1}^{[1]}} dw_{2,1}^{[1]} + \frac{\partial E}{\partial w_{2,2}^{[1]}} dw_{2,2}^{[1]} + \cdots + \frac{\partial E}{\partial w_{2,n}^{[1]}} dw_{2,n}^{[1]} \tag{4.50}$$

$$= \sum_{i=1}^{n} J_i dw_{2,i}^{[1]} \tag{4.51}$$

En donde $J_i = \frac{\partial E}{w_{2,i}^{[1]}}$. Aunado a esto, vemos que inicialmente $\mathbf{\Phi} \neq 0$, por lo que definimos un vector $\boldsymbol{\delta Q}$ de igual dimensiones a $\mathbf{\Phi}$ cuyo propósito es acercar ésta a su objetivo (cero) en cada ciclo. Por otra parte, podemos escribir el diferencial de $\mathbf{\Phi}$ como:

$$d\mathbf{\Phi} = [d\Phi_1, d\Phi_1, \cdots, d\Phi_n,]^T \tag{4.52}$$

$$d\mathbf{\Phi} = \frac{\partial \mathbf{\Phi}}{\partial w_{2,1}^{[1]}} \cdot dw_{2,1}^{[1]} + \cdots + \frac{\partial \Phi}{\partial w_{2,n}^{[1]}} \cdot dw_{2,n}^{[1]} \tag{4.53}$$

$$= \sum_{i=1}^{n} F_i \cdot dw_{2,i}^{[1]} \tag{4.54}$$

En donde:

$$\boldsymbol{F_i} = \left[\frac{\partial \Phi_1}{\partial w_{2,i}^{[1]}}, \frac{\partial \Phi_2}{\partial w_{2,i}^{[1]}}, \cdots, \frac{\partial \Phi_n}{\partial w_{2,i}^{[1]}} \right]^T \tag{4.55}$$

Por otra parte, incorporamos una segunda restricción sobre el algoritmo de aprendizaje, en términos generales ésta nos sirve para restringir la magnitud de cambio de los pesos de la red, lo que nos permite aproximar $dw_{2,i}^{[1]}$ a $\Delta w_{2,i}^{[1]}$ de manera arbitraria. Escribimos esta segunda restricción como:

$$\sum_{i=1}^{n} \left(dw_{2,i}^{[1]} \right)^2 = (\delta P)^2 \tag{4.56}$$

En donde δP es una constante elegida en forma manual. El objetivo del nuevo algoritmo de aprendizaje es tener el mayor cambio posible de la función de costo respetando las dos restricciones presentadas. Este problema se puede resolver a través del método de los multiplicadores de Lagrange de la siguiente forma:

$$L = dE + \left(\boldsymbol{\delta Q^T} - \boldsymbol{d\Phi^T} \right) \cdot \boldsymbol{V} + \left((\delta P)^2 - \sum_{i=0}^{n} \left(dw_{2,i}^{[1]} \right)^2 \right) \cdot \mu \tag{4.57}$$

En donde μ es un escalar y $\boldsymbol{V} = [V_1, V_2, \cdots, V_n]^T$. Ambos son coeficientes de Lagrange. Incorporando (4.55) y (4.51) en (4.57):

$$L = \sum_{i=1}^{n} J_i dw_{2,i}^{[1]} + \left(\boldsymbol{\delta Q^T} - \sum_{i=1}^{n} F_i^T dw_{2,i}^{[1]} \right) \boldsymbol{V} + \left((\delta P)^2 - \sum_{i=0}^{n} \left(dw_{2,i}^{[1]} \right)^2 \right) \mu \tag{4.58}$$

Para maximizar el diferencial de la función de costo tenemos que obtener el gradiente del Lagrangiano e igualarlo a 0. Esto es:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_{2,1}^{[1]}} \\ \frac{\partial L}{\partial w_{2,2}^{[1]}} \\ . \\ . \\ \frac{\partial L}{\partial w_{2,n}^{[1]}} \end{bmatrix} = 0 \tag{4.59}$$

Cada uno de los elementos de este vector lo podemos obtener de la siguiente manera:

$$\frac{\partial L}{\partial w_{2,i}^{[1]}} = J_i - \boldsymbol{F_i^T V} - 2\mu dw_{2,i}^{[1]} = 0 \tag{4.60}$$

Podemos reescribir y resolver para el diferencial de pesos:

$$J_i - \boldsymbol{F_i^T V} = 2\mu dw_{2,i}^{[1]} \tag{4.61}$$

$$dw_{2,i}^{[1]} = \frac{J_1}{2\mu} - \frac{\boldsymbol{F_i^T V}}{2\mu} \tag{4.62}$$

La ecuación (4.62) representa nuestro nuevo algoritmo de actualización de pesos. Huang et al. demostraron en [26] como obtener los valores de μ y $\boldsymbol{V}$. En el capítulo 5 se deriva un algoritmo similar para la RNC en donde se detalla este procedimiento.

4.1.3.1. Resultados

Inicialmente podemos probar el polinomio examinado con las anteriores arquitecturas y contrastar resultados. En primer lugar veremos el polinomio:

$$p(x) = x^5 - 4x^4 + 4.43x^3 - 0.164x^2 - 1.464x + 0.144 \tag{4.63}$$

Recordamos que para éste, $\boldsymbol{\lambda} = [1.2, -0.5, 2, 1.2, 0.1]$ y utilizando el modelo Sigma con restricciones obtenemos la gráfica de la figura 4.7.

Claramente, el tiempo de cómputo disminuyó drásticamente en comparación con las dos anteriores arquitecturas, esto debido a que en la implementación de restricciones se busca alcanzar un máximo del diferencial de la función de error en cada iteración (respetando las restricciones impuestas). En la figura 4.8 se muestran las raíces estimadas por la red en negro y en rojo la raíz real.

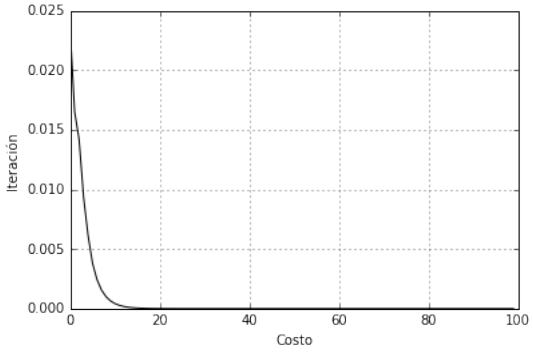

Figura 4.7: Costo contra iteración para modelo Sigma con restricciones.

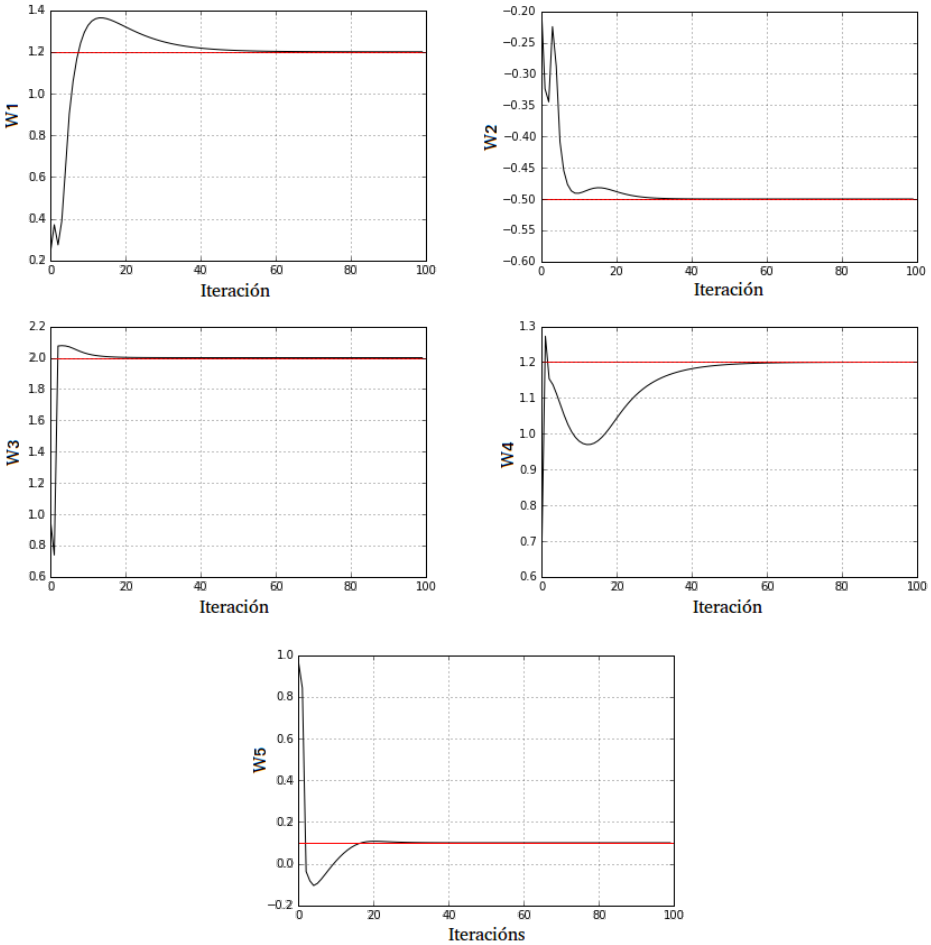

Figura 4.8: Raíces estimadas por la red.

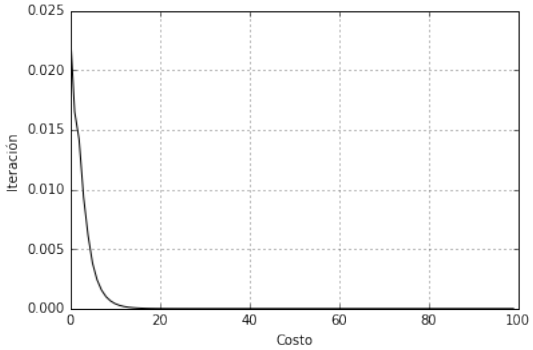

Figura 4.9: Costo contra iteración.

Como era de esperarse, las raíces aproximadas por la red siguieron el mismo comportamiento que la función de costo, y en tan solo 60 iteraciones obtenemos un valor de distancia con $d = 0.00009656$. Las raíces aproximadas por la red durante la iteración número 60 fueron:

$$\boldsymbol{w} = \begin{bmatrix} 1.20000101 - 2.89592317 \cdot 10^{-04} i \\ -0.5 + 4.37883007 \cdot 10^{-10} i \\ 1.99999991 + 7.50878973 \cdot 10^{-11} i \\ 1.199999 + 2.89593640 \cdot 10^{-04} i \\ 0.09999999 - 1.77985832 \cdot 10^{-09} i \end{bmatrix} \tag{4.64}$$

Por otra parte, podemos probar la arquitectura en polinomios con raíces complejas. Por ejemplo, supongamos que estamos interesados en conocer las raíces del siguiente polinomio:

$$p(x) = x^6 + (3+i)x^5 + (2+2i)x^4 - x^2 + (-3-i)x + (-2-2i) \tag{4.65}$$

El vector de raíces de éste último es: $\boldsymbol{\lambda} = [-i, -1, -2, -1-i, i, 1]$. En la figura 4.9 se muestra la gráfica de costo contra iteración al implementar la arquitectura Sigma con restricciones para este polinomio.

También podemos ver el resultado de las raíces. En la figura 4.10 se muestran en negro la parte real de las raíces aproximadas por la red y en rojo el valor objetivo. En la figura 4.11 se visualiza la correspondiente parte imaginaria.

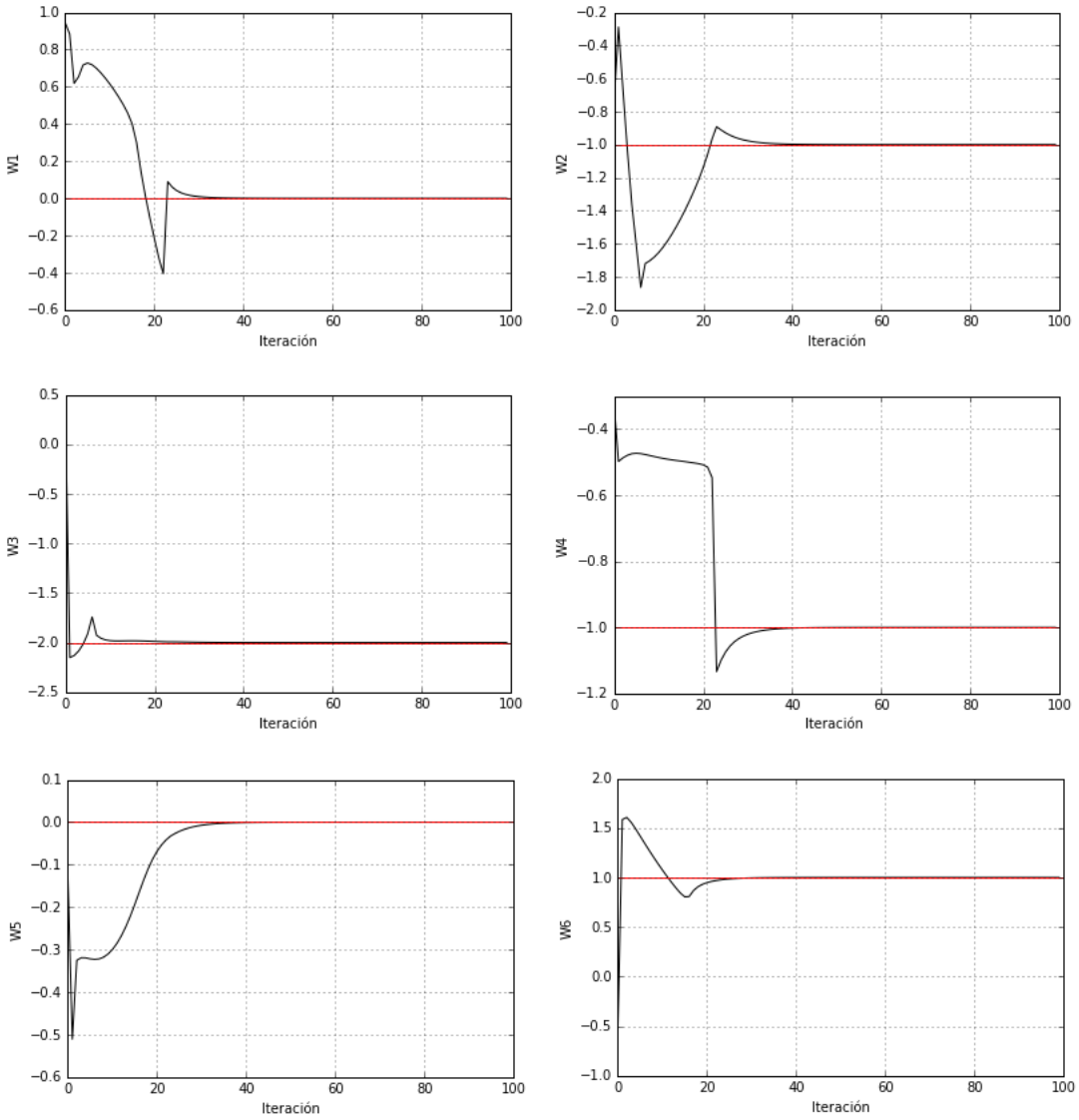

Figura 4.10: Raíces reales aproximadas por la red.

Al cabo de 60 iteraciones, las raíces aproximadas por la red fueron:

$$\boldsymbol{w} = \begin{bmatrix} 1.19964954 \cdot 10^{-09} - 9.99999994 \cdot 10^{-01} i \\ -9.99999996 \cdot 10^{-01} - 8.77014956 \cdot 10^{-09} i \\ -2.00000000 + 5.55700803 \cdot 10^{-10} i \\ -1.00000000 - 9.99999996 \cdot 10^{-01} i \\ -1.14158480 \cdot 10^{-09} + 9.99999999 \cdot 10^{-01} i \\ 9.99999999 \cdot 10^{-01} - 4.96326824 \cdot 10^{-10} i \end{bmatrix} \tag{4.66}$$

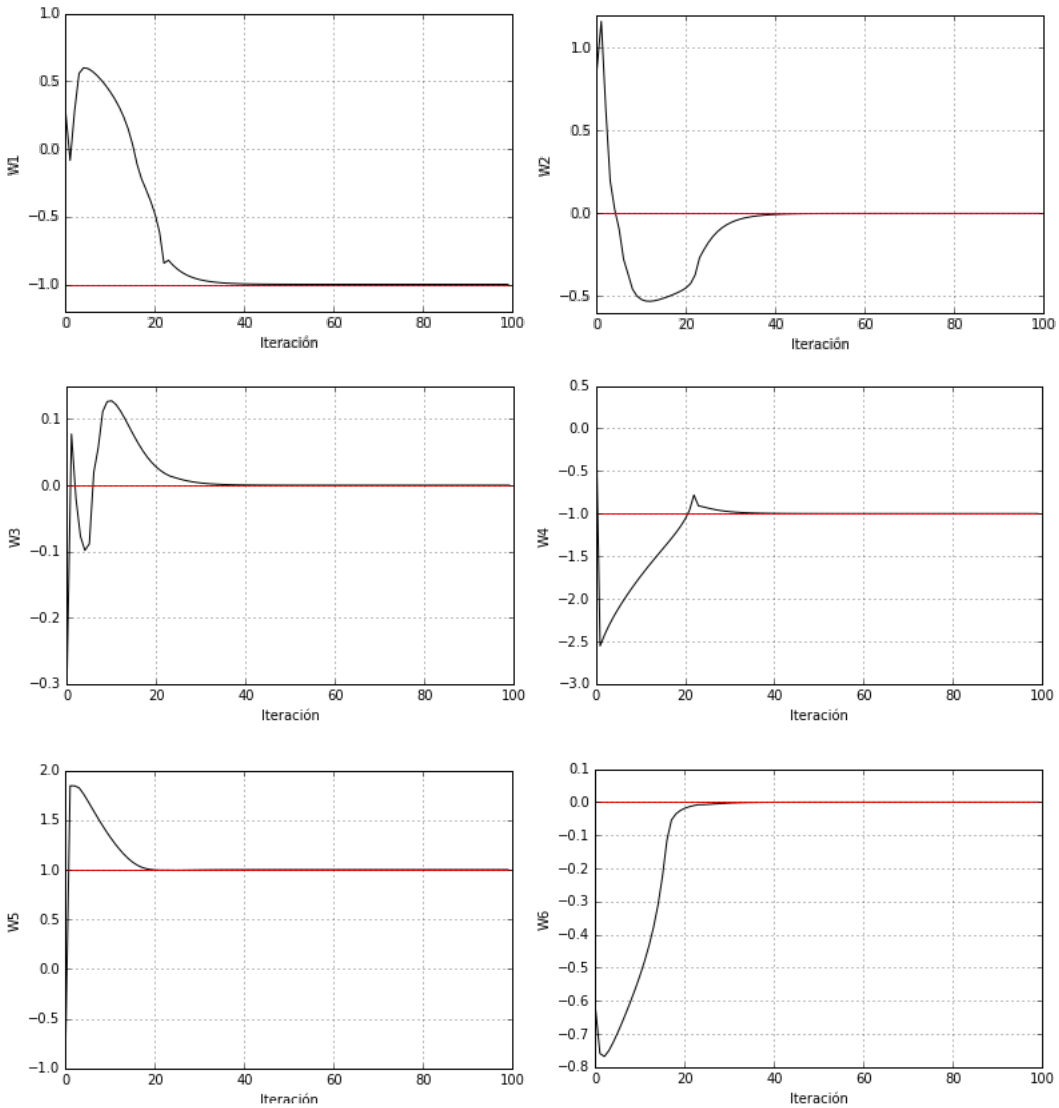

Figura 4.11: Raíces imaginarias aproximadas por la red.

La distancia entre las raíces reales y las aproximadas por la red es: $d = 1.11640 \cdot 10^{-9}$. Claramente las restricciones le permitieron a la arquitectura aprender de manera más rápida y llegar a un menor error en un reducido número de iteraciones.

En la sección pasada se habló acerca del concepto de consistencia; sin embargo, no se definió. El objetivo de esta medición es estimar el número de iteraciones promedio necesarias para encontrar las raíces de un polinomio, para esto se creó

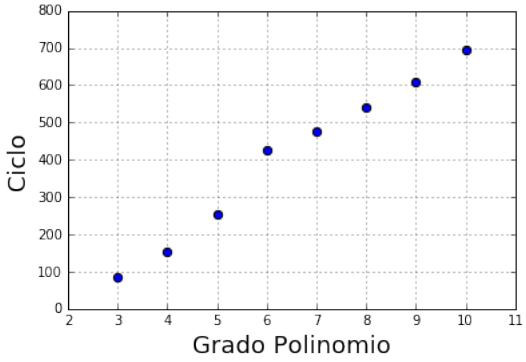

Figura 4.12: Número de iteraciones contra grado del polinomio.

una base de datos con polinomios de grado 3 a grado 10 con un total de 7,000 instancias (1,000 para cada grado). En la figura 4.12 se muestran los resultados de éste análisis.

Como se ha mostrado, la RN puede aproximar con un grado de error arbitrario las raíces de los polinomios. Los resultados que se muestran en la figura 4.12 son producto de una aproximación con un error menor o igual a 1^{-10}.

4.2. Método 2: Restricciones sobre los pesos de la red

En las secciones pasadas, se mostró cómo incorporar las identidades de Newton sobre el algoritmo de aprendizaje, lo que permitió una rápida convergencia y mayor precisión; sin embargo, en muchas aplicaciones no es necesario computar todas las raíces en paralelo. En la presentación de este segundo método para la incorporación de restricciones se deducirá un modelo para el cómputo de raíces en serie, lo que otorgará una ganancia en la velocidad de cómputo.

En primer lugar, presentamos las restricciones por incorporar a la RN. Las fórmulas de Vieta son un conjunto de ecuaciones que relacionan a los coeficientes de un polinomio con sus raíces; supongamos que tenemos el siguiente polinomio (en donde $a_0 = 1$):

$$p(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_{n-1} x + a_n \tag{4.67}$$

Las correspondientes fórmulas de Vieta son:

$$\begin{gathered} w_1 + w_2 + \cdots + w_n = -a_0 \\ (w_1 w_2 + w_1 w_3 + \cdots) + (w_2 w_3 + \cdots) + \cdots = a_1 \\ \cdot \\ \cdot \\ w_1 w_2 \cdots w_n = (-1)^n a_n \end{gathered} \tag{4.68}$$

Podemos agrupar cada una de las anteriores restricciones en un vector denotado con la letra $\mathbf{\Phi}$. Éste tiene n elementos (1 por restricción), en otras palabras: $\mathbf{\Phi} = [\Phi_1, \Phi_2, \cdots, \Phi_n]^T$. De manera general, podemos escribir el k componente del vector $\mathbf{\Phi}$ como:

$$\Phi_k = \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq n} w_{i_1} w_{i_2} \cdots w_{i_k} = (-1)^k a_k \tag{4.69}$$

En donde $i_1, i_2, \cdots, i_k \in \mathbb{N}$. En la siguiente sección se mostrará cómo incorporar estas restricciones a los pesos de la red para encontrar raíces de manera secuencial.

4.2.1. Arquitectura Pi y Sigma

Supongamos que deseamos encontrar una raíz del polinomio mostrado en (4.67), para esto, lo reescribimos en función de sus raíces:

$$p(x) = (x - \lambda_1)(x - \lambda_2) \cdots (x - w_n) \tag{4.70}$$

Si definimos la función g(x) como:

$$g(x) = (x - \lambda_2)(x - \lambda_3) \cdots (x - \lambda_n) \tag{4.71}$$

Podemos reescribir(4.70) de la siguiente manera:

$$p(x) = (x - \lambda_1)\, g(x) \tag{4.72}$$

Por otra parte, podemos expandir la ecuación (4.71) como:

$$g(x) = b_0 x^{n-1} + b_1 x^{n-2} + b_2 x^{n-3} + \cdots + b_{n-1} x + b_n \tag{4.73}$$

Notamos que $b_0 = 1$ y que $b_i(\lambda_1, \lambda_2, \cdots, \lambda_n)$ para i $\neq 0$. Podemos construir una arquitectura Sigma y una Pi para modelar p(x) tomando en cuenta g(x). La figura 4.13 y 4.14 muestran ambas redes.

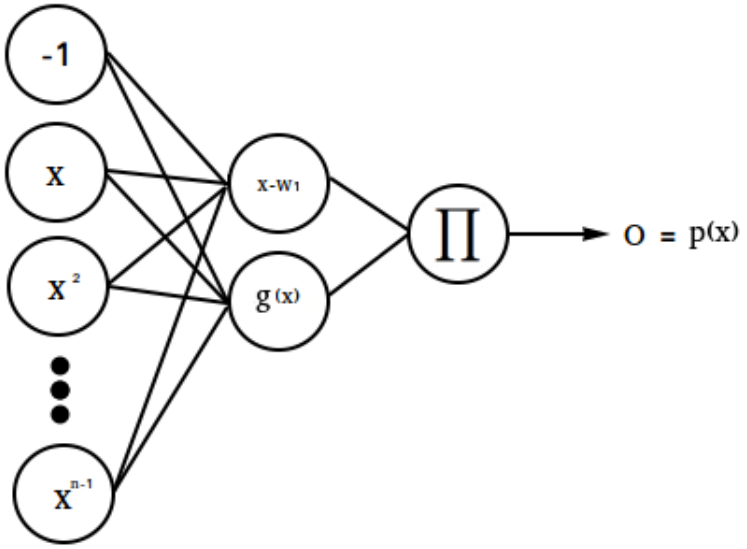

Figura 4.13: Arquitectura Pi.

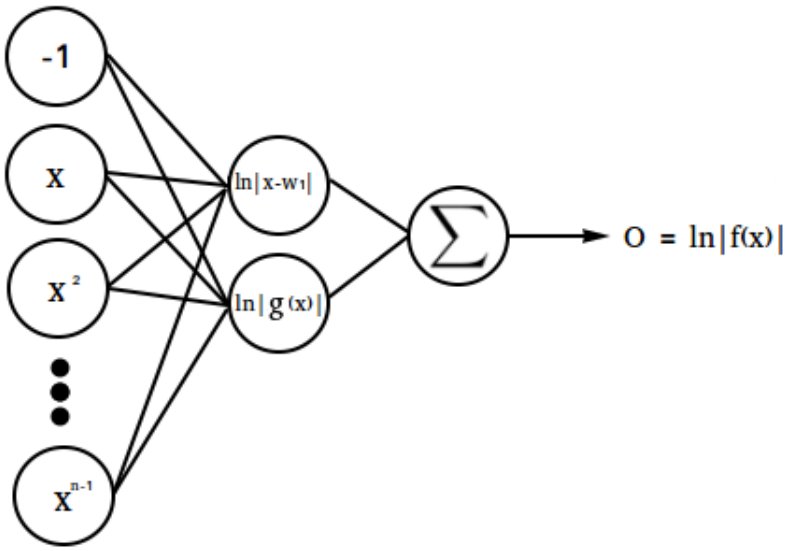

Figura 4.14: Arquitectura Sigma.

Podemos describir la salida de la red de la arquitectura Pi con las siguientes ecuaciones:

$$\boldsymbol{Z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{A}^{[0]} \tag{4.74}$$
$$\boldsymbol{A}^{[1]} = \boldsymbol{Z}^{[1]} \tag{4.75}$$
$$\boldsymbol{Z}^{[2]} = \prod_{i=1}^{2} a_i^{[1]} \tag{4.76}$$
$$\boldsymbol{O} = \boldsymbol{Z}^{[2]} \tag{4.77}$$

Por su parte, las ecuaciones que describen la salida de la red Sigma son:

$$\boldsymbol{Z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{A}^{[0]} \tag{4.78}$$
$$\boldsymbol{A}^{[1]} = ln\left|\boldsymbol{Z}^{[1]}\right| \tag{4.79}$$
$$\boldsymbol{Z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{A}^{[1]} \tag{4.80}$$
$$\boldsymbol{O} = \boldsymbol{Z}^{[2]} \tag{4.81}$$

En ambas arquitecturas definimos $\boldsymbol{W}^{[2]} = [1,1]^T$. Como podemos ver $(x - w_1)$ y g(x) o $a_1^{[1]}$ y $a_2^{[1]}$ se obtienen a partir de $\boldsymbol{W}^{[1]}$ y $\boldsymbol{A}^{[0]}$. De éstas últimas dos, solamente $\boldsymbol{W}^{[1]}$ contiene las aproximaciones de las raíces; por lo tanto, para encontrar raíces de manera secuencial tenemos que reescribir $\boldsymbol{W}^{[1]}$ en función de una sola raíz e.g., w_1. Inicialmente podemos escribir de manera explícita $\boldsymbol{W}^{[1]}$ como:

$$\boldsymbol{W}^{[1]} = \begin{bmatrix} w_1 & 1 & 0 & 0 & \cdots & 0 \\ -b_{n-1} & b_{n-2} & b_{n-3} & b_{n-4} & \cdots & b_0 \end{bmatrix} \tag{4.82}$$

Por consiguiente, el problema se reduce a reescribir los coeficientes de g(x) en términos de w_1. Para esto, podemos reescribir la ecuación (4.69) de la siguiente manera:

$$\Phi_k = \sum_{1=i_1<i_2<\cdots<i_k\leq n} w_{i_1}w_{i_2}\cdots w_{i_k} + \sum_{1<i_1<i_2<\cdots<i_k\leq n} w_{i_1}w_{i_2}\cdots w_{i_k} \tag{4.83}$$
$$= w_1 \sum_{2\leq i_2<\cdots<i_k\leq n} w_{i_2}w_{i_3}\cdots w_{i_k} + \sum_{2\leq i_1<i_2<\cdots<i_k\leq n} w_{i_1}w_{i_2}\cdots w_{i_k} \tag{4.84}$$
$$= (-1)^{k-1} w_1 b_{k-1} + (-1)^k b_k \tag{4.85}$$

Sustituyendo este último resultado en el lado izquierdo de la ecuación (4.69), tenemos que:

$$(-1)^{k-1} w_1 b_{k-1} + (-1)^k b_k = (-1)^k a_k \tag{4.86}$$

$$(-1)^k b_k = (-1)^k a_k - (-1)^{k-1} w_1 b_{k-1} \tag{4.87}$$

$$b_k = a_k + w_1 b_{k-1} \tag{4.88}$$

En este punto hemos mostrado cómo escribir el k coeficiente del polinomio g(x) en términos del coeficiente k-1 de éste mismo. Finalmente, remarcamos que $b_0 = 1$ y por consiguiente $b_1 = a_1 + w_1$, por lo que podemos notar que es posible escribir los coeficientes de g(x) en términos de la primera raíz de p(x) y por lo tanto reescribir $\boldsymbol{W}^{[1]}$ en función de ésta.

4.2.2. Algoritmo de aprendizaje

Al igual que en las secciones pasadas, escribimos nuestra regla de actualización de pesos como:

$$w_1 \leftarrow w_1 - TA * \frac{\partial E}{\partial w_1} \tag{4.89}$$

En esta ocasión no se utilizó la notación tradicional para referirnos a un peso específico de la red e.g. $w_{i,j}^{[l]}$; en su lugar se utiliza w_1 para describir el peso $w_{0,0}^{[1]}$ ya que como podemos ver en la ecuación (4.82), éste es el único que aísla a w_1.

En primer lugar derivaremos el algoritmo de aprendizaje para la arquitectura Pi. Para esto necesitamos obtener el término $\frac{\partial E}{\partial w_1}$. Definimos nuestra función de error como:

$$E = \frac{1}{2P} \sum_{p=1}^{P} (T_p - O_p)^2 \tag{4.90}$$

Podemos calcular la derivada parcial de la función de error con respecto a w_1 de la siguiente forma:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial O_p} \cdot \frac{\partial O_p}{\partial w_1} \tag{4.91}$$

El primer término del lado derecho de la ecuación anterior lo podemos escribir como:

$$\frac{\partial E}{\partial O_p} = -\frac{1}{P}\sum_{p=1}^{P}(T_p - O_p) \tag{4.92}$$

Por otra parte, para encontrar el término $\frac{\partial O_p}{\partial w_1}$ vemos que:

$$O_p = (x_p - w_1)\, g\,(x_p, w_1, A) \tag{4.93}$$

En donde A es el vector de coeficientes de p(x). Habiendo descrito esto, podemos computar la derivada parcial de O_p con respecto a w_1 de la siguiente forma:

$$\frac{\partial O_p}{\partial w_1} = \frac{\partial (x_p - w_1)}{\partial w_1} g\,(x_p, w_1, A) + \frac{\partial g\,(x_p, w_1, A)}{\partial w_1}(x_p - w_1) \tag{4.94}$$

Para calcular los términos de la anterior ecuación, vemos que:

$$\frac{\partial (x_p - w_1)}{\partial w_1} = -1 \tag{4.95}$$

$$g^{'}(x_p, w_1, A) \equiv \frac{\partial g(x_p, w_1, A)}{\partial w_1} \tag{4.96}$$

Sustituyendo (4.95) y (4.96) en (4.94):

$$\frac{\partial O_p}{\partial w_1} = -g\,(x_p, w_1, A) + g^{'}\,(x_p, w_1, A)\,(x_p - w_1) \tag{4.97}$$

Sustituyendo (4.97) y (4.92) en (4.91):

$$\frac{\partial E}{\partial w_1} = \frac{1}{P}\sum_{p=1}^{P}(T_p - O_p)\left\{g\,(x_p, w_1, A) - g^{'}\,(x_p, w_1, A)\,(x_p - w_1)\right\} \tag{4.98}$$

Con esto, se define completamente el modelo de aprendizaje para la arquitectura Pi. Para detallar este procedimiento para la red Sigma, podemos volver a utilizar la regla de actualización de pesos definida en (4.89); además, la función de error y su derivada con respecto a O_p permanecen intactas. Nuevamente necesitamos encontrar la derivada de la función de error con respecto a w_1, para esto escribimos:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial O_p} \cdot \frac{\partial O_p}{\partial w_1} \tag{4.99}$$

Podemos escribir O_p como:

$$O_p = ln\,|x_p - w_1| + ln\,|g(x_p)| \tag{4.100}$$

Y su derivada con respecto a w_1 de la siguiente manera:

$$\frac{\partial O_p}{\partial w_1} = \frac{\partial\, ln\,|x_p - w_1|}{\partial w_1} + \frac{\partial\, ln\,|g(x_p)|}{\partial w_1} \tag{4.101}$$

Es fácil demostrar que:

$$\frac{\partial\, ln\,|x_p - w_1|}{\partial w_1} = -\frac{x - w_1}{|x - w_1|^2} \tag{4.102}$$

$$\frac{\partial\, ln\,|g(x_p, w_1, A)|}{\partial w_1} = \frac{g(x_p, w_1, A)g^{'}(x_p, w_1, A)}{|g(x_p, w_1, A)|^2} \tag{4.103}$$

Sustituyendo las ecuaciones (4.102) y (4.103) en (4.101) y ésta última en (4.99) tenemos:

$$\frac{\partial E}{\partial w_1} = \frac{1}{P}\sum_{p=1}^{P}(T_p - O_p)\left\{\frac{x_p - w_1}{|x - w_1|^2} - \frac{g(x_p, w_1, A)g^{'}(x_p, w_1, A)}{|g(x_p, w_1, A)|^2}\right\} \tag{4.104}$$

4.2.3. Resultados de la arquitectura Sigma

En esta sección solamente se muestran los resultados obtenidos a través de la arquitectura Sigma por ser la más relevante. Supongamos que estamos interesados en conocer la raíz positiva del polinomio:

$$f(x) = x^3 + x^2 - x - 1 \tag{4.105}$$

Para éste, el vector de raíces es: $\boldsymbol{\lambda} = [-1, -1, 1]^T$. Implementando la arquitectura Sigma con restricciones sobre los pesos, obtenemos la gráfica de costo contra iteración de la figura 4.15.

Como podemos ver, el costo disminuye más rápido en comparación con arquitecturas anteriores; sin embargo, notamos la existencia de picos y cambios abruptos, lo cual nos indica que la función de costo posiblemente posea estas propiedades. En la figura 4.16 se muestra el comportamiento de la aproximación de la raíz contra el número de iteración.

De manera similar a escenarios anteriores, en azul se muestra el valor aproximado por la arquitectura y en rojo el valor real de la raíz. Ahora supongamos que estamos interesados en conocer la raíz positiva del siguiente polinomio:

$$f(x) = x^4 + 1.7x^3 - 1.2x^2 - 1.3x - 0.2 \tag{4.106}$$

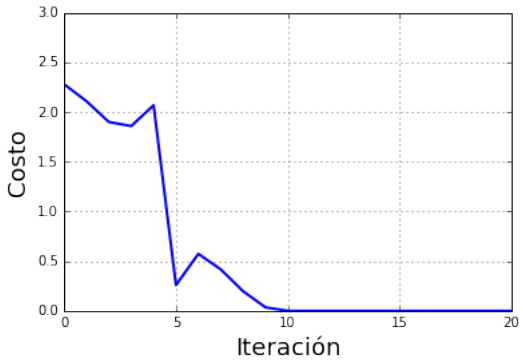

Figura 4.15: Costo contra iteración arquitectura Sigma.

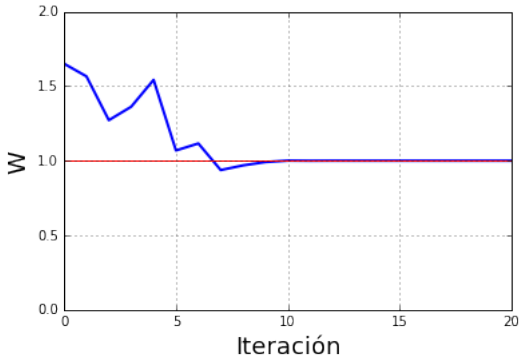

Figura 4.16: w_1 contra iteración de la arquitectura Sigma.

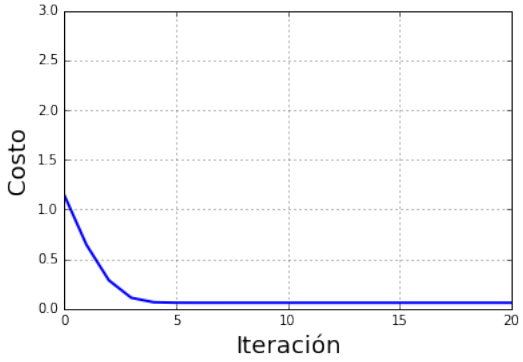

Figura 4.17: - Costo contra iteración arquitectura Sigma segundo polinomio.

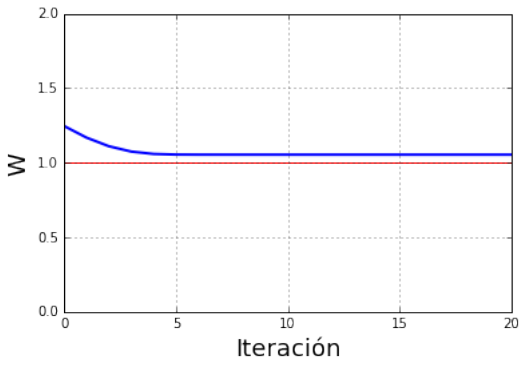

Figura 4.18: w_1 contra iteración arquitectura Sigma segundo polinomio.

Para este polinomio tenemos que $\boldsymbol{\lambda} = [-2, -0.5, -0.2, 1]^T$. La figura 4.17 muestra el costo contra el número de iteración.

Al igual que con el polinomio anterior, podemos graficar la aproximación de la raíz contra el número de iteración. La figura 4.18 muestra este resultado.

Como podemos ver en este caso, a pesar de que la curva de costo es suave, no se llega al valor deseado (marcado con rojo). Además, ya que podemos escribir $W^{[1]}$ en función de w_1, es posible ver la función de costo en términos de éste. La figura 4.19 muestra este resultado para el primer polinomio.

Como esperábamos, la superficie de la función de error presenta picos y valles. Vemos que existe un mínimo (aunque no global) cerca de la raíz $w_1 = 1$; sin embargo, alrededor de ésta encontramos que la pendiente es cercana a 0, lo que dificulta su aproximación. Analizando estos resultados, vemos que es posible encontrar

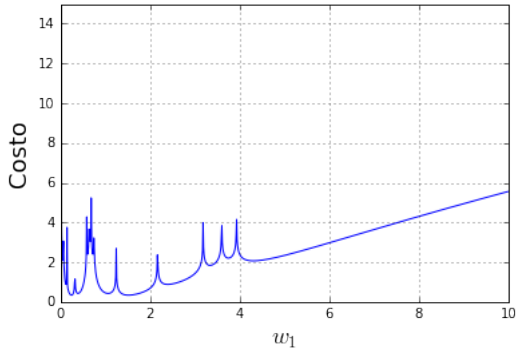

Figura 4.19: Superficie de la función de costo.

raíces con este modelo; sin embargo, no es consistente y debido a la naturaleza de la función de costo y a los hiperparámetros de la red, se corre el riesgo de no aproximar correctamente las raíces.

4.3. Redes neuronales para el cómputo de densidad en un fluido compresible

Retomando lo mencionado al inicio del capítulo 4, es posible utilizar las arquitecturas previamente mostradas para el cálculo de la densidad. El fluido en cuestión es nitrógeno en estado supercrítico con un total de 5,734,400 partículas, el algoritmo toma como entrada la presión, la temperatura y un cierto número de variables características del fluido para formar un polinomio de tercer grado cuya raíz real positiva representa el inverso de la densidad. En la figura 4.20 se muestra un diagrama de flujo con los pasos seguidos durante esta implementación.

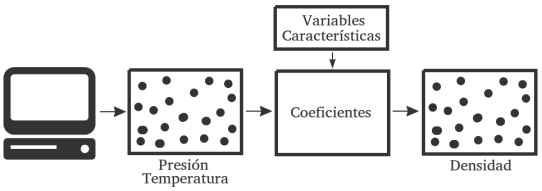

Figura 4.20: Diagrama de flujo de implementación para la estimación de la densidad.

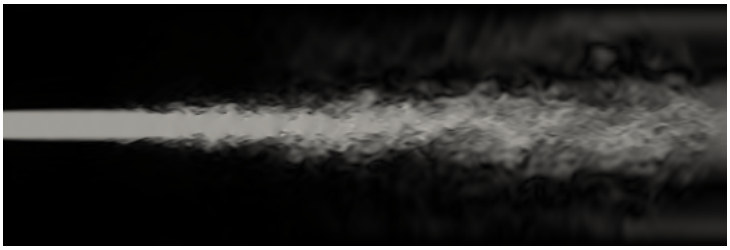

Figura 4.21: Corte bidimensional del campo de velocidades.

En primer lugar, se obtienen los campos de presión y temperatura a través de algoritmos tradicionales. Éstos se incorporan en conjunto con las variables características del fluido para formar los coeficientes de los polinomios a resolver (uno por cada nodo). Finalmente, se obtiene la raíz real positiva del polinomio formado y se invierte para obtener la densidad en ese punto.

En aras de entender más el tipo de fluido a tratar, en la figura 4.21 se muestra un corte bidimensional del campo de velocidades.

La intensidad de la imagen indica la magnitud de la velocidad. Como podemos ver, el fluido empieza con una gran velocidad cerca de la fuente y conforme avanza desaselera. Para este mismo instante de tiempo, podemos obtener en la misma perspectiva el campo de temperatura y de presión. En la figura 4.22 se muestran ambos.

La temperatura varía de 126 a 298 grados Celsius, y como podemos ver, el fluido está siendo inyectado a un espacio con una alta temperatura; por otra parte, el campo de presión es prácticamente homogéneo. El rango de ésta va de $3.25 \cdot 10^6$ a $4.5 \cdot 10^6$ pascales. Dadas estas circunstancias, el nitrógeno se encuentra en condiciones supercríticas. Una vez en este punto y como se mostro en el capítulo 1, podemos generar los coeficientes del polinomio a resolver para cada uno de los nodos de la siguiente forma:

$$a_0 = P \tag{4.107}$$
$$a_1 = Pd_1b - (RT - b) \tag{4.108}$$
$$a_2 = Pd_2b^2 - (RT - b)\, d_1b + \theta \tag{4.109}$$
$$a_3 = -b\left[\theta + (RT - b)\, d_2b\right] \tag{4.110}$$

Una vez calculados los coeficientes, podemos implementar nuestra RN, para esto, se eligió el modelo Sigma de Huang et al. con restricciones. A pesar de que el

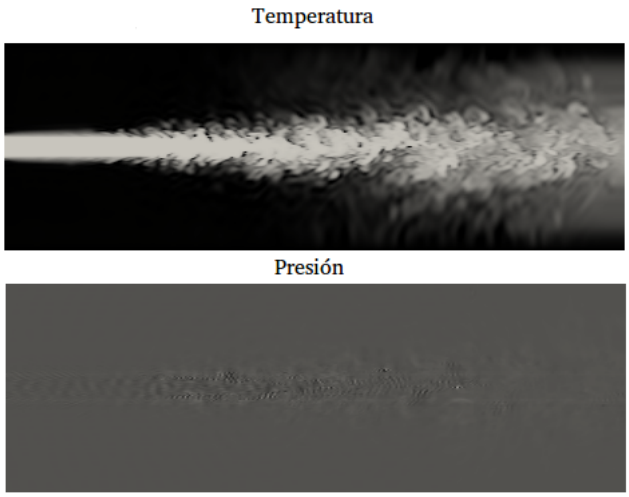

Figura 4.22: Corte bidimensional mostrando el cmapo de temperatura (arriba) y de presión (abajo).

modelo de restricciones sobre los pesos de red es más rápido, su inestabilidad no lo hace el mejor candidato para la tarea. En la figura 4.23 se muestra el campo de densidad computado con un método tradicional (Cardano) y con la RN.
Para el entrenamiento de cada red se utilizó una inicialización aleatoria de los pesos; además, para cada nodo se delimito el número de iteraciones máximas a 500. Esta última condición es la causante de que el resultado producido por la RN en la figura 4.23 posea puntos donde no se converge al resultado (pequeñas manchas blancas). En la figura 4.24 se muestra el histograma de iteraciones producido durante el computo del campo de la densidad.

Con respecto a la forma de entrenar las RN, podríamos inicializar los pesos de manera aleatoria (como se hizo en el resultado anterior); lo que sería el mejor método si el valor de la densidad en cada punto de la malla fuera independiente del resto; sin embargo, claramente ese no es el caso para esta aplicación. Las restricciones impuestas sobre el modelo matemático del fluido limitan el número de valores que la densidad puede tomar, por lo cual es importante definir un método que tome en consideración la distribución de está.

Para resolver esto, podemos inicializar los parámetros de la RN a través del resultado de las raíces de algún nodo adyacente. Para entender esto, imaginemos que estamos simulando el mismo problema pero en un espacio de simulación bidimensional, la figura 4.25 muestra dos maneras de lograrlo; las flechas indican los caminos de cómputo a seguir.

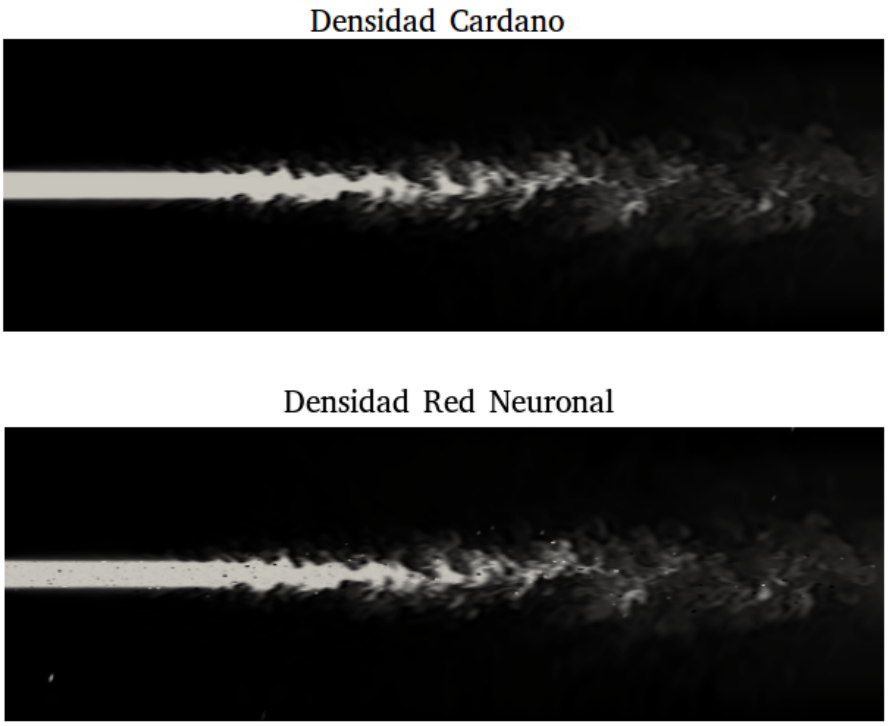

Figura 4.23: Comparativa del campo de densidad.

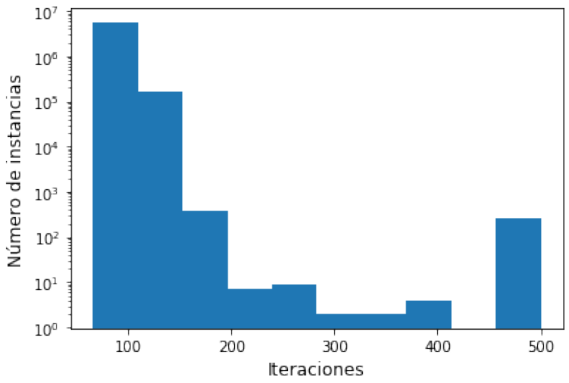

Figura 4.24: Histograma del número de iteraciones.

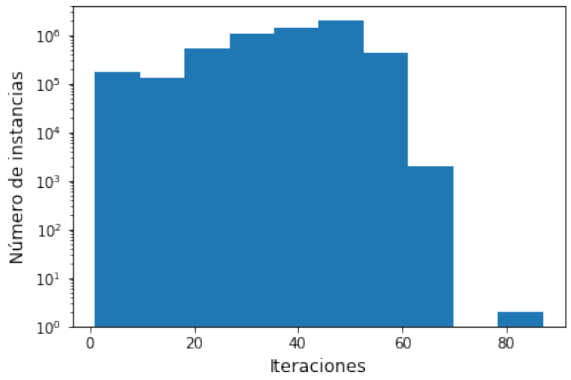

Figura 4.26: Histograma del número de iteraciones con camino de cómputo.

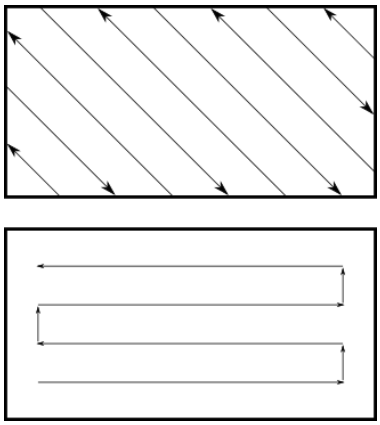

Figura 4.25: Posibles caminos de cómputo para un fluido en un espacio bidimensional.

Por camino de cómputo me refiero al orden secuencial en el que se computa la densidad, en donde se inicializan los pesos de cada RN con los resultados obtenidos en el procesamiento del nodo anterior.

Incorporando el concepto de camino de cómputo, obtenemos el histograma de iteraciones de la figura 4.26.

Por otra parte, también es posible utilizar esta forma de entrenamiento incorporando cómputo en paralelo. Para esto, se puede atacar el problema de dos maneras. En primer lugar, simplemente se pueden convertir los caminos de cómputo en

planos de cómputo, i.e., es posible inicializar los pesos que se encuentren dentro de un plano en el espacio de simulación a partir del resultado de las raíces de las neuronas de un plano adyacente.

Si lo que se busca es implementar paralelización sobre todo el espacio de simulación, se tienen que hacer unos ajustes sobre la función de costo. La metodología a seguir en este caso es la siguiente:

- Generar una base de datos de entrenamiento con los resultados de las raíces con N polinomios.
- Entrenar una RN que aproxime de la mejor manera al conjunto de polinomios de la base de datos.
- Inicializar los pesos de futuras RN a partir de los resultados obtenidos.

Para lograr el segundo punto, como se mencionó, es necesario modificar la función de costo para incorporar información acerca de todos los polinomios en lugar de uno solo. Esto se puede conseguir de la siguiente forma :

$$E = \frac{1}{2PN} \sum_{n=1}^{N} \sum_{p=1}^{P} |T_i - O_i|^2 \tag{4.111}$$

La red entrenada con la función de costo presentada en (4.111) aproximará al polinomio que mejor representa al conjunto de N polinomios en la base de datos.

Por otra parte, también es posible inicializar los pesos de cada red en la posición α en el espacio de simulación a través de las raíces computadas por la neurona en la posición α de la iteración anterior. Por supuesto, este método conlleva como desventaja el uso de memoria adicional, el cuál escala de manera lineal con el número de nodos en el espacio de simulación. Implementado este concepto, obtenemos el histograma de la figura 4.27.

Para comparar el método aquí presentado, podemos obtener el campo de densidad a través de otros algoritmos. Con vistas a esto, y con ayuda de la librería NumPy (librearía de Python enfocada al procesamiento y soporte de matrices), la cual incorpora un algoritmo para la obtención de raíces basado en el cómputo de los vectores propios de la matriz compañera del polinomio a tratar [23], es posible obtener los resultados de la figura 4.28, en ésta, se muestra el campo de densidad obtenido con ayuda de la librería NumPy (arriba) y a través de la RN con inicialización de pesos con el estado anterior.

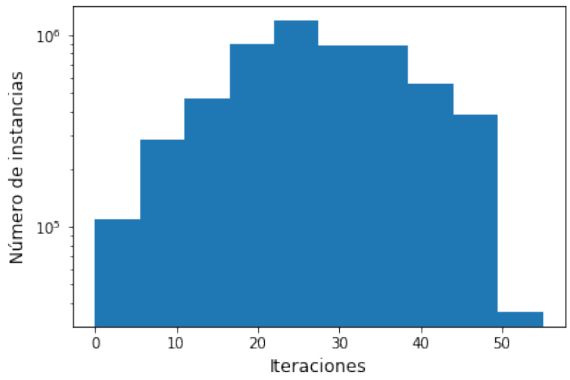

Figura 4.27: Histograma del número de iteraciones para el método de inicialización de pesos a través del estado anterior.

Para la comparación, se utilizará la medida de error propuesta en la ecuación (4.111), a la que se le denominara como el promedio del error cuadrático medio o PECM, de manera explícita:

$$PECM = \frac{1}{2PN} \sum_{n=1}^{N} \sum_{p=1}^{P} |T_i - O_i|^2 \tag{4.112}$$

Para obtener el campo de densidad mostrado en la figura 4.28 a partir de la RN, se estableció como condición de paro al llegar a un error por nodo menor o igual a $1X10^{-20}$; por lo tanto, este también es el valor del PECM. Por otra parte, utilizando la librería NumPy, se obtuvo un PECM de $1X10^{-30}$. Por otra parte, si comparamos la velocidad promedio de cómputo, veremos que el algoritmo que utiliza redes neuronales es 16 veces más lento. Por lo tanto, a pesar de que este método se puede utilizar para computar el campo de densidad con un grado de precisión arbitrario; para este caso particular de aplicación, no es el más eficiente.

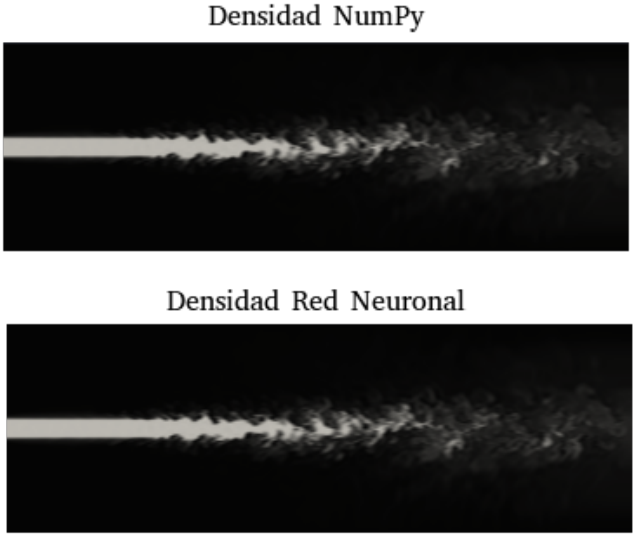

Figura 4.28: Comparativa del campo de densidad.

Capítulo 5

Predicción de la presión en un fluido incompresible

Como se mencionó anteriormente, la simulación de fluidos por computadora requiere de grandes cantidades de recursos computacionales; esto no es excepción para los fluidos incompresibles, en donde se requiere resolver la ecuación de Poisson (para los métodos eulerianos). Para esto, usualmente se utilizan métodos numéricos iterativos, lo que vuelve a éste procedimiento uno de los más costosos.

Una de las posibles soluciones a éste problema es intentar predecir el valor de la presión a través de un algoritmo de aprendizaje de máquinas eliminando la necesidad de utilizar un algoritmo iterativo. Un posible diagrama general de cómputo para lograr esto se muestra en la figura 5.1.

Como podemos ver, el primer paso consiste en computar la velocidad ignorando el término de la presión, posteriormente se le agregan fuerzas externas para después fungir como entrada al cálculo de la presión. Para este último punto, vemos

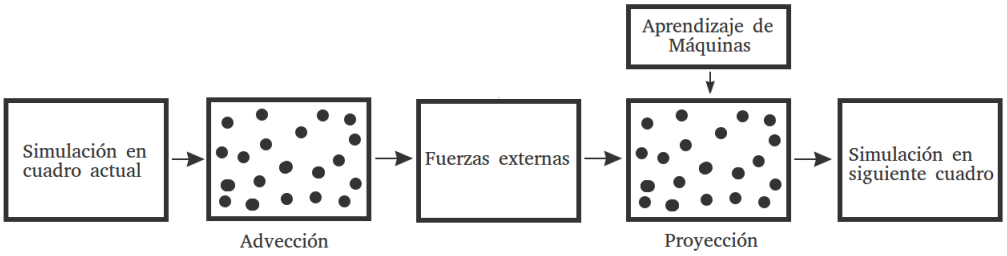

Figura 5.1: Diagrama general a implementar.

en el diagrama de la figura 5.1 un recuadro con las palabras <Aprendizaje de Máquinas> para simbolizar que el algoritmo de inteligencia artificial reemplaza al tradicional.

Una vez establecido esto, tenemos que seleccionar un algoritmo de aprendizaje de máquinas adecuado para esta situación y definir las entradas y salidas de éste; además, es necesario generar una base de datos para asegurarnos de que nuestro algoritmo tenga una gran capacidad de generalización.

En formulaciones tradicionales, la presión es obtenida al resolver la ecuación de Poisson [69]:

$$\nabla^2 p_t = \frac{1}{\Delta t} \nabla \cdot \boldsymbol{u_t^*} \tag{5.1}$$

En donde p_t es la presión en el tiempo t y $\boldsymbol{u_t^*}$ es la velocidad computada sin tener en consideración la presión pero incluyendo las fuerzas externas que actúan sobre el sistema en el tiempo t; en lo que resta del capítulo se referirá a u_t^* como velocidad de advección.

Teniendo en cuenta esto, podemos establecer a la velocidad de advección en el tiempo $t-1$ como entrada del algoritmo; además, sabemos que la salida es la presión. Para facilitar el aprendizaje del algoritmo, podemos incorporar dos entradas adicionales, la topología y la presión en el tiempo $t-1$. La primera se agrega debido a que la geometría del espacio de simulación tiene un impacto directo sobre los valores que la presión puede tomar; la segunda se anexa para incorporar información histórica pasada del sistema.

En las siguientes secciones se detallarán las entradas y las salidas del algoritmo; sin embargo, por ahora es importante notar que cada una de las entradas es una matriz de igual tamaño que el espacio de simulación. Para el procesamiento de éste tipo de datos es común y ha resultado eficiente el uso de redes neuronales convolucionales debido a las conexiones dispersas y a los pesos compartidos (propiedades explicadas en el capítulo 3). Por esta razón, en esta tesis se optó por desarrollar el algoritmo utilizando redes neuronales convolucionales

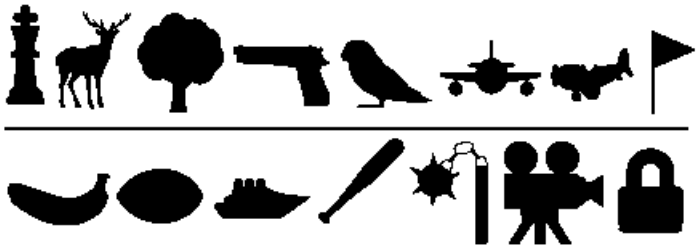

Figura 5.2: Muestra de obstáculos utilizados.

5.1. Base de datos

Para generar la base de datos se utilizó el software Mantaflow creado por Tobias Pfaff y Nils Thuerey. Mantaflow es un programa de código abierto enfocado a la investigación de la simulación de fluidos. Está programado en c++; sin embargo, posee una interfaz en Python para lograr un fácil y rápido desarrollo de simulaciones; además, la interfaz con Python permite el uso de librerías especializadas en inteligencia artificial y álgebra lineal como Numpy y PyTorch.

Debido al tiempo que toma hacer una base de datos de simulaciones y entrenar la RNC, se optó por generar simulaciones en un espacio bidimensional. De manera específica, cada simulación tiene una resolución de 128 X 128 y el número total de simulaciones es de 2,574 (cada una con 140 pasos de tiempo), lo que en total nos da 360,360 escenas o patrones de entrenamiento.

Para realizar las simulaciones, se recolectó un conjunto de 143 imágenes de libre acceso y uso de internet en dos resoluciones: 32x32 y 64x64, dando un total de 286 imágenes. Éstas fungen como obstáculos para el fluido y aseguran una diversidad de estados en la base de datos. En la figura 5.2 se muestra un subconjunto de los obstáculos seleccionados (preprocesados para únicamente tener dos colores: blanco y negro). Finalmente, para cada uno de estos obstáculos se generaron 9 simulaciones (3 variaciones en la posición y 3 del tamaño de la fuente) lo que da el total de 2,574 simulaciones. Una muestra de las simulaciones realizadas se presenta en la figura 5.3.

Como se había mencionado, se requiere de 3 entradas: la velocidad, la topología y la presión en el tiempo $t-1$. Para formar la base de datos, éstas tres fueron tomadas de Mantaflow y convertidas en arreglos de la librería NumPy para finalmente ser guardadas en memoria en formato binario. La tres entradas y la salida del algoritmo poseen la misma dimensión de 128 X 128.

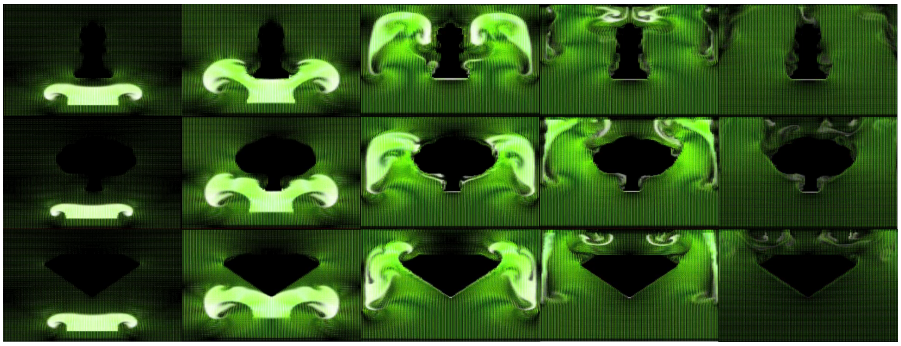

Figura 5.3: Muestra de simulaciones realizadas.

A la matriz de topología se le denotará con la letra minúscula o, y a un valor predeterminado de ésta como $o(x_{i,j})$. Esta matriz se define como:

$$o(x_{i,j}) = \begin{cases} 1 & si\ obstáculo\ en\ x_{i,j} \\ 0 & de\ otra\ manera \end{cases} \tag{5.2}$$

5.2. Método 1: Red neuronal convolucional sin restricciones sobre el algoritmo de entrenamiento

La primera arquitectura utilizada se muestra en la figura 5.4, su lectura es de arriba a abajo, en donde la entrada es un tensor de dimensiones ($n_1^{[0]}$, $n_2^{[0]}$, $n_3^{[0]}$), $n_1^{[0]} = 4$, $n_2^{[0]} = 128$, $n_3^{[0]} = 128$ y la salida por su parte es de dimensiones ($n_1^{[7]}$, $n_2^{[7]}$, $n_3^{[7]}$) en donde $n_1^{[6]} = 1$, $n_2^{[6]} = 128$ y $n_3^{[6]} = 128$.

En la presente arquitectura existen cuatro procesos distintos de convolución, los cuales serán denominados como: Conv 1, Conv 2, Conv 3 y Conv 4. También se tiene un procedimiento de submuestreo y otro de remuestreo para reducir e incrementar respectivamente la dimensión de las matrices.

Conv 1 toma como entrada un volumen de dimensiones $(P, 4, 128, 128)$ en donde P representa el número de instancias o patrones de entrenamiento a procesar de la base de datos. La salida es un volumen de dimensiones $(P, 8, 128, 128)$. Esto

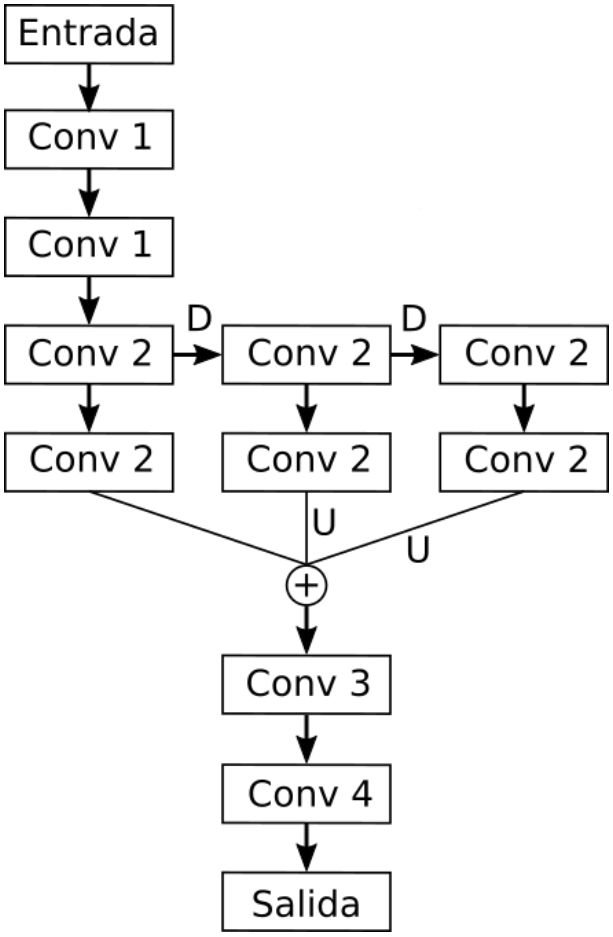

Figura 5.4: Arquitectura de red neuronal convolucional.

implica que esta capa utiliza 8 filtros y que la convolución es simétrica; dado que los filtros son de dimensiones $(3, 3)$ se utilizó un rellenado y un desplazamiento de una unidad.

Conv 2 toma como entrada un volumen de dimensiones $(P, 8, 128, 128)$ y entrega como salida otro de dimensiones $(P, 8, 128, 128)$. Las dimensiones de los filtros son iguales a las de Conv 1, esto es, $(3, 3)$. La convolución nuevamente es simétrica, por lo que se vuelve a utilizar un rellenado y un desplazamiento de una unidad.

Conv 3 toma como entrada un tensor de dimensiones $(P, 24, 128, 128)$ y entrega otro de dimensiones $(P, 24, 128, 128)$. Además, utiliza 24 filtros de dimensiones (1, 1), un desplazamiento de una unidad y no utiliza rellenado; dadas estas características, la convolución es simétrica.

Conv 4 toma como entrada un volumen de dimensiones $(P, 24, 128, 128)$ y entrega otro de dimensiones $(P, 1, 128, 128)$ el cual es el resultado de la predicción de la presión. Los filtros son de dimensiones $(1, 1)$, el desplazamiento es de una unidad y no se utiliza rellenado. Al igual que en las anteriores convoluciones, esta operación es simétrica.

El proceso de submuestreo se simboliza con la letra mayúscula D en la figura 5.4; para realizar éste, se utilizó el procedimiento de máximo muestreo descrito en el capítulo 3 con un filtro de dimensiones $(2, 2)$ y un desplazamiento de dos unidades. Como resultado de ésto, el volumen de entrada es reducido de (P, n_1, n_2, n_3) a (P, n_1, n_2, n_3). Por otra parte, para el remuestreo se utilizó un método de interpolación bilinear con el objetivo de convertir un volumen de dimensiones $(P, n_1, \frac{n_2}{2}, \frac{n_3}{2})$ a otro de tamaño $(P, n_1, 2n_2, 2n_3)$.

Previo a el entrenamiento de la RNC, se dividió la base de datos en dos conjuntos, uno de entrenamiento y otro de prueba. El primero contiene el 80 % de las instancias, lo que equivale a 247,200 ejemplos; el segundo contiene el resto de las instancias, lo que equivale a 61,800 ejemplos.

Se utilizaron dos tarjetas de video Nvidia GeForce GTX 1080 de 8 GB de memoria para el entrenamiento de la red. Debido a la capacidad limitada de memoria y al tamaño de la base de datos, es necesario dividir ésta última en subsecciones o baches; al número de elementos en estas regiones se le denominara tamaño de bache. Definiendo la taza de aprendizaje con la constante 0.0005, el tamaño de bache con 500 y utilizando la función de costo del error cuadrático medio obtenemos la imagen de la figura 5.5.

Cada vez que el algoritmo recorre completamente la base de datos, diremos que se ha logrado una época. Podemos medir el ECM para el subconjunto de entrenamiento y de prueba al término de cada una de éstas. Las figuras 5.6 y 5.7 muestran estos resultados, y como podemos ver, los valores son similares para ambos conjuntos; no obstante, durante cada época el error del conjunto de entrenamiento es menor al de prueba.

En la siguiente tabla se resumen algunos de los resultados obtenidos por la red.

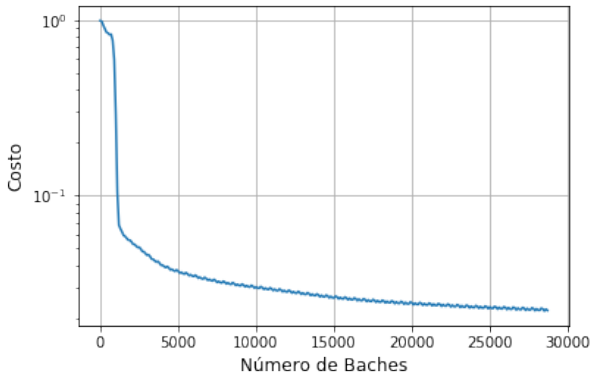

Figura 5.5: ECM para cada bache de la base de datos

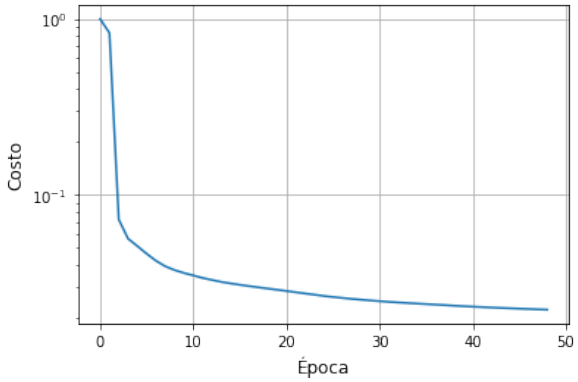

Figura 5.6: Costo contra época en el conjunto de entrenamiento.

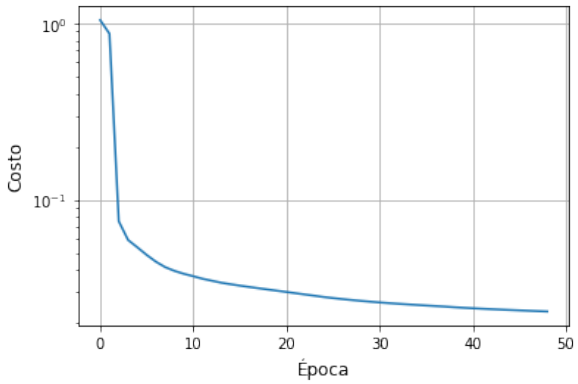

Figura 5.7: Costo contra época en el conjunto de prueba.

Figura 5.8: Simulación ejemplo utilizando RNC.

Época	ECM entrenamiento	ECM prueba
0	1.00012251	1.04269791
1	0.8333058	0.87364825
2	0.07239872	0.07560173
5	0.04622882	0.04872293
10	0.03483063	0.03266048
20	0.02839928	0.03007941
30	0.02485611	0.02624044
40	0.02313225	0.02431603
48	0.02223536	0.02331508

Una vez entrenada la red, es posible incluirla al ciclo de generación de cada simulación para ver su desempeño. Siguiendo esta lógica, obtenemos la imagen de la figura 5.8. Como podemos ver, el fluido en general se muestra de manera inestable y muy poco preciso.

5.3. Método 2: Red neuronal convolucional con restricciones sobre el algoritmo de entrenamiento

Una de las mayores debilidades del método propuesto anteriormente es la falta de restricciones impuestas sobre el valor que la presión puede tomar, ya que en ningún punto se incluye información acerca de la divergencia del vector de velocidad, el cual para un fluido incompresible debe ser igual a cero, esto es:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{5.3}$$

Inspirado en los modelos de restricciones presentados en el capítulo 4 y desarrollados por D. Karras y S. Perantonis [31] es posible crear un algoritmo de

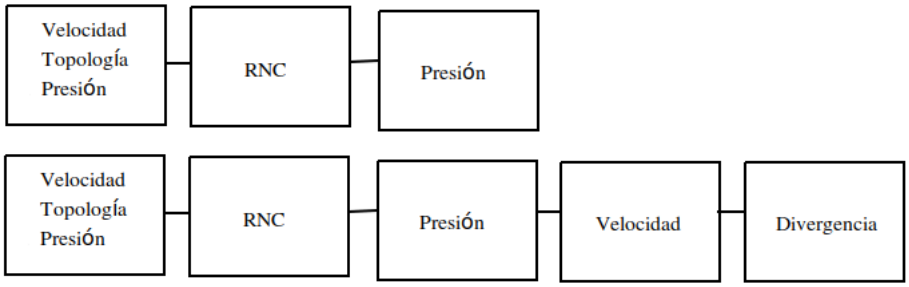

Figura 5.9: Diagrama de cómputo previo (arriba), diagrama de cómputo actual (abajo).

aprendizaje que incorpore información sobre la divergencia del vector de velocidad en la RNC. El primer paso para lograr esto es extender nuestro diagrama de cómputo y poner a la divergencia en función de la salida de la red, y por ende, de los pesos de ésta. La figura 5.9 muestra el previo y el nuevo esquema de cómputo utilizado.

Como se puede ver, el nuevo esquema de cómputo incluye dos pasos adicionales, en primer lugar se utiliza la presión calculada por la red para corregir el valor de la velocidad de advección, para esto, simplemente se sustrae el gradiente de la presión de la velocidad de advección; en segundo lugar, se calcula la divergencia de este nuevo campo de velocidad.

Lo que se busca con el modelo de restricciones es reducir el valor de la función de costo en cada iteración tomando en cuenta ciertas limitaciones sobre los valores que ésta puede tomar. Para esto, notamos que si los cambios sobre los pesos de la red son lo suficientemente pequeños, éstos pueden ser aproximados por su primer diferencial, en otras palabras:

$$\Delta w_i \approx dw_i \tag{5.4}$$

De manera similar, es posible aproximar los cambios sobre la función de costo y sobre la divergencia de la velocidad inducidos por los cambios en los pesos de la red a través de sus primeros diferenciales. De manera concreta:

$$\Delta E \approx dE \tag{5.5}$$

$$\Delta \left(\nabla \cdot u \right) \approx d\nabla \cdot \boldsymbol{u} \tag{5.6}$$

Estrictamente, la divergencia de la velocidad presentada en la ecuación (5.6) es una matriz de igual dimensiones al espacio de simulación; sin embargo, en esta ocasión, y durante el resto de la sección, utilizaremos la notación $\nabla \cdot \boldsymbol{u}$ para referirnos al valor promedio de la divergencia de la velocidad, y por lo tanto, será un escalar. Habiendo establecido esto, buscamos maximizar el diferencial de la función de costo sometido a las siguientes condiciones:

$$\sum_{i=1}^{n}(dw_i)^2 = (\delta P)^2 \tag{5.7}$$

$$d\nabla \cdot \boldsymbol{u} = \delta Q \tag{5.8}$$

En la condición de la ecuación (5.7) exigimos que la suma de los cambios en los pesos se mantenga acorde con la constante δP. El objetivo es tener un hiperparámetro que nos permita especificar la precisión que queremos que se mantenga en las ecuaciones (5.4), (5.5) y (5.6).

Por otra parte, la condición de la ecuación (5.8) nos permite establecer δQ como el diferencial de la divergencia en cada iteración. Dicho de otra forma, durante cada ciclo del entrenamiento, el valor de la divergencia es diferente de cero; para resolver esto, definimos un valor para el diferencial de la divergencia con el objetivo de acercar ésta a su objetivo (cero) y de disminuir el valor de la función de error.

Podemos expandir el valor del diferencial de la función de error de la siguiente forma:

$$dE = \frac{\partial E}{\partial w_1}dw_1 + \frac{\partial E}{\partial w_2}dw_2 + \cdots + \frac{\partial E}{\partial w_n}dw_n \tag{5.9}$$

$$dE = \sum_{i=1}^{n}\frac{\partial E}{\partial w_i}dw_i \tag{5.10}$$

$$dE = \sum_{i=1}^{n} J_i dw_i \tag{5.11}$$

Por otra parte:

$$d\nabla \cdot \boldsymbol{u} = \frac{\partial \nabla \cdot \boldsymbol{u}}{\partial w_1}dw_1 + \frac{\partial \nabla \cdot \boldsymbol{u}}{\partial w_2}dw_2 + \cdots + \frac{\partial \nabla \cdot \boldsymbol{u}}{\partial w_n}dw_n \tag{5.12}$$

$$d\nabla \cdot \boldsymbol{u} = \sum_{i=1}^{n}\frac{\partial \nabla \cdot \boldsymbol{u}}{\partial w_i}dw_i \tag{5.13}$$

$$d\nabla \cdot \boldsymbol{u} = \sum_{i=1}^{n} F_i dw_i \tag{5.14}$$

En donde:

$$J_i = \frac{\partial E}{\partial w_i} \tag{5.15}$$

$$F_i = \frac{\partial \nabla \cdot u}{\partial w_i} \tag{5.16}$$

Utilizando la metodología del método de multiplicadores de Lagrange establecemos que se busca maximizar el diferencial de la función de costo sujeto a (5.7) y (5.8). Para esto definimos las funciones R_1 y R_2 de la siguiente forma:

$$R_1(dw_1, dw_2, \cdots, dw_n) = \sum_{i=1}^{n} (dw_i)^2 \tag{5.17}$$

$$R_2(dw_1, dw_2, \cdots, dw_n) = \sum_{i=1}^{n} F_i dw_i \tag{5.18}$$

De esta manera, exigimos que:

$$\nabla dE = \lambda_1 \nabla R_1 + \lambda_2 \nabla R_2 \tag{5.19}$$

Por otra parte, vemos que:

$$\nabla dE = [J_1, J_2, \cdots, J_n]^T \tag{5.20}$$

$$\nabla R_1 = [dw_1, dw_2, \cdots, dw_n]^T \tag{5.21}$$

$$\nabla R_2 = [F_1, F_2, \cdots, F_n]^T \tag{5.22}$$

Incorporando (5.20), (5.21) y (5.22) en (5.19):

$$J_i = 2\lambda_1 dw_i + \lambda_2 F_i \tag{5.23}$$

Resolviendo para el diferencial del peso w_i:

$$dw_i = \frac{J_i}{2\lambda_1} - \frac{\lambda_2 F_i}{2\lambda_1} \tag{5.24}$$

Tomando en cuenta las ecuaciones (5.7), (5.8) y (5.24), tenemos un sistema de 3 ecuaciones y 3 incógnitas $(dw_i, \lambda_1, \lambda_2)$. Sustituyendo (5.24) en (5.8):

$$\sum_{i=1}^{n} F_i \left(\frac{J_i - \lambda_2 F_i}{2\lambda_1} \right) = \delta Q \tag{5.25}$$

$$\frac{1}{2\lambda_1} \sum_{i=1}^{n} \left(F_i J_i - \lambda_2 F_i^2 \right) = \delta Q \tag{5.26}$$

$$\frac{1}{2\lambda_1} \left(\sum_{i=1}^{n} F_i J_i - \lambda_2 \sum_{i=1}^{n} F_i^2 \right) = \delta Q \tag{5.27}$$

Introduciendo las variables auxiliares A y B definidas como: $A = \sum_{i=1}^{n} F_i J_i$, $B = \sum_{i=1}^{n} F_i^2$. Podemos reescribir (5.27) de la siguiente forma:

$$\frac{1}{2\lambda_1} \left(A - \lambda_2 B \right) = \delta Q \tag{5.28}$$

$$\lambda_2 = \frac{A - 2\lambda_1 \delta Q}{B} \tag{5.29}$$

Por otra parte, sustituyendo (5.24) en (5.7):

$$\sum_{i=1}^{n} \left(\frac{J_i - \lambda_2 F_i}{2\lambda_1} \right)^2 = (\delta P)^2 \tag{5.30}$$

$$\sum_{i=1}^{n} \left(J_i^2 - 2\lambda_2 J_i F_i + \lambda_2^2 F_i^2 \right) = 4\lambda_1^2 (\delta P)^2 \tag{5.31}$$

Introduciendo la variable auxiliar $C = \sum_{i=1}^{n} J_i$ podemos reescribir (5.31) como:

$$C - 2A\lambda_2 + B\lambda_2^2 = 4\lambda_1^2 (\delta P)^2 \tag{5.32}$$

Sustituyendo (5.29) en (5.32):

$$C - 2A \left(\frac{A - 2\lambda_1 \delta Q}{B} \right) + B \left(\frac{A - 2\lambda_1 \delta Q}{B} \right)^2 = 4\lambda_1^2 (\delta P)^2 \tag{5.33}$$

Simplificando y resolviendo para λ_1, es posible llegar a la siguiente expresión:

$$\lambda_1 = \frac{1}{2} \sqrt{\frac{C - \frac{A^2}{B}}{(\delta P)^2 - \frac{(\delta Q)^2}{B^2}}} \tag{5.34}$$

Llegado a este resultado, utilizamos (5.24) en conjunto con (5.29) y (5.38) como nuestra regla para la actualización de pesos, esto es:

$$w_i \leftarrow w_i + \frac{J_i}{2\lambda_1} - \frac{\lambda_2 F_i}{2\lambda_1} \tag{5.35}$$

Por otra parte, para poder utilizar (5.35) es necesario definir δP y δQ. Como se mencionó, el objetivo de δQ es acercar a la divergencia de la velocidad en cada iteración a su objetivo (cero), por lo que la podemos definir como: $\delta Q = -k\nabla \cdot \boldsymbol{u}$, el signo negativo en la anterior expresión asegura la convergencia de la divergencia al valor deseado (y por ende, la reducción de la función de costo). La constante de proporcionalidad k es un hiperparámetro elegido por el usuario, a la cual se le demanda que: $k \in \{\mathbb{R} > 0\}$.

La constante δP representa la precisión con la que queremos aproximar a Δw_i, ΔE y $\Delta\left(\nabla \cdot \boldsymbol{u}\right)$ y puede ser simplemente definida por un número pequeño (e.g. $\delta P = 0.001$); sin embargo, esto puede llevar a tiempos más largos de entrenamiento. Una solución a esto fue propuesta por Huang et. al. [26] en donde se comienza con un valor grande y se disminuye con el paso de las iteraciones, de manera explícita:

$$\delta P(0) = \delta P_0 \tag{5.36}$$

$$\delta P(t) = \delta P_0 \left(1 - e^{-\frac{\Theta}{t}}\right) \tag{5.37}$$

En donde δP_0 y Θ son hiperparámetros definidos previo al entrenamiento por el usuario, con la única restriccion: $\Theta \in \{\mathbb{R} > 0\}$. Finalmente, notamos que la ecuación (5.38) puede tomar dos valores (uno para el valor positivo y otro para el negativo de la raíz cuadrada), es posible demostrar que para alcanzar un máximo sobre el diferencial de la función de costo es necesario tomar en cuenta únicamente el valor negativo [31], [25]. De esta manera, tenemos que:

$$\lambda_1 = -\frac{1}{2}\left|\sqrt{\frac{C - \frac{A^2}{B}}{(\delta P)^2 - \frac{(\delta Q)^2}{B^2}}}\right| \tag{5.38}$$

Con esto, queda completamente definido el algoritmo de entrenamiento de la RNC; sin embargo, la implementación del modelo se deja como tarea para una futura investigación.

Capítulo 6

Conclusiones

En el capítulo 4 se mostraron varios métodos para la solución de las raíces de polinomios, incluyendo las ventajas y desventajas de cada uno de estos; también se analizó y visualizó el beneficio que existe al incorporar restricciones sobre el algoritmo de aprendizaje y los pesos de la red. De manera concreta, el primero de estos paradigmas sirve para el cómputo de raíces en paralelo y el segundo para su procesamiento de manera secuencial.

Como se mostró en la segunda parte del capítulo 4, estos métodos pueden ser utilizados en la simulación de fluidos. En este caso, por cuestiones de consistencia se eligió el método de restricciones sobre el algoritmo de entrenamiento. Gracias a la naturaleza de las redes neuronales y a la forma en que se planteó el problema, es posible disminuir el error sobre la función de costo de manera arbitraría; sin embargo, se tiene que considerar que a mayor exigencia de precisión, mayor será tiempo de cómputo.

Por otra parte, se mostró como utilizar la distribución del campo de densidad para inicializar los pesos de las redes neuronales, lo que resultó en una disminución sustancial en el tiempo de cómputo. A pesar de las medidas implementadas, el algoritmo basado en redes neuronales no pudo superar en rendimiento al tradicional; sin embargo, dado que el primero de estos escala de manera lineal con el grado del polinomio, resulta imposible descartar su utilidad en otras aplicaciones; por ejemplo, ya que la precisión de la red es un hiperparámetro impuesto por el usuario, es posible obtener aproximaciones de las raíces en cortos plazos de tiempo.

Es importante destacar que la implementación de todos los métodos para la solución de raíces fueron programados en Python, por lo que es posible obtener un incremento en el desempeño de éstos utilizando lenguajes compilados y de bajo

nivel como c++. Por otra parte, queda como incógnita saber si es posible la implementación de un modelo que incorpore tanto restricciones sobre el algoritmo de aprendizaje como sobre los pesos de la red.

En el capítulo 5, por otra parte, se abordo el problema utilizando el marco teórico de los fluidos incompresibles. Se genero una base de datos con un total de 309,000 escenas en un espacio de simulación bidimensional, y se fraseo el problema de predecir el valor de la presión como una tarea de aprendizaje supervisado.

En la implementación, se utilizo una RNC para obtener el campo de presiones; sin embargo, los resultados de ésta no fueron alentadores. Una de las razones de esto posiblemente sea la complejidad que conlleva aproximar a través de un proceso de regresión 16, 384 presentes en la capa de salida. Para resolver esto, en la segunda parte del capítulo 5 se propone y deduce un algoritmo de aprendizaje innovador para la RNC, el cual incluye información a priori del problema a tratar, en este caso, sobre la divergencia del vector de velocidad. La implementación y análisis de este último modelo queda propuesto como trabajo futuro.

Bibliografía

[1] (1995). United states energy information. *State Energy Data Report*. 6

[2] Allaire, G. and Kaber, S. M. (2008). *Numerical linear algebra*, volume 55. Springer. 16

[3] Anderson, J. D. and Wendt, J. (1995). *Computational fluid dynamics*, volume 206. Springer. 2

[4] Barrett, R., Berry, M. W., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and Van der Vorst, H. (1994). *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43. Siam. 9

[5] Batchelor, G. K. (2000). *An introduction to fluid dynamics*. Cambridge university press. 3

[6] Ben Finney, M. J. (2013). Carbon dioxide, solid, liquid and gas. `http://www.shashikallada.com/carbon-dioxide-solid-liquid-and-gas/`. 3

[7] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32. 19

[8] Bridson, R. and Müller-Fischer, M. (2007). Fluid simulation: Siggraph 2007 course notes video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 courses*, pages 1–81. ACM. 3

[9] Chen, C., Seff, A., Kornhauser, A., and Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 2722–2730. IEEE. 11

[10] Crick, F. (1989). The recent excitement about neural networks. *Nature*, 337(6203):129–132. 20

[11] Daumé III, H. (2012). A course in machine learning. *Publisher, ciml. info*, pages 5–73. 4

[12] Davidson, D. (2003). The role of computational fluid dynamics in process industries. In *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2002 NAE Symposium on Frontiers of Engineering*, page 21. National Academies Press. 6

[13] Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387. 4

[14] Dhaubhadel, M. (1996). Cfd applications in the automotive industry. *Journal of fluids engineering*, 118(4):647–653. 7

[15] Fahlman, S. E. (1988). Faster-learning variations on back-propagation: An empirical study. In *Proc. 1988 Connectionist Models Summer School*. Morgan Kaufman. 20

[16] Fausett, L. (1994). *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc. 22, 23

[17] Foster, N. and Fedkiw, R. (2001). Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30. ACM. 15

[18] Ghassabi, Z., Moaveni, B., and Khaki-Sedigh, A. (2006). Solving systems of linear equations and finding the inversion of a matrix by neural network using genetic algorithms (nn using ga). In *5th WSEAS Int. Conf. on Computational Intelligence, Man-Machine Systems and Cybernetics*, pages 129–132. 17

[19] Gibbs, N. E. and Tucker, A. B. (1986). A model curriculum for a liberal arts degree in computer science. *Communications of the ACM*, 29(3):202–210. 4

[20] Gobovic, D. and Zaghloul, M. E. (1994). Analog cellular neural network with application to partial differential equations with variable mesh-size. In *Circuits and Systems, 1994. ISCAS'94., 1994 IEEE International Symposium on*, volume 6, pages 359–362. IEEE. 19

[21] Han, F., Li, X.-Q., Lyu, M. R., and Lok, T.-M. (2006). A modified learning algorithm incorporating additional functional constraints into neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 20(02):129–142. 21

[Hormis et al.] Hormis, R., Antoniou, G., and Mentzelopoulou, S. Separation of two-dimensional polynomials via a sigma-pi neural net. In *Proc. Int. Conf. on Modelling and Simulation*, pages 304–306. 17

[23] Horn, R. A. (1985). Cr johnson matrix analysis. 80

[24] Huang, D. (1996). One layer linear perceptron for the inversion of nonsingular matrix. In *IC On ROVPIA'96*. 17, 18, 21

[25] Huang, D.-S. (2004). A constructive approach for finding arbitrary roots of polynomials by neural networks. *IEEE Transactions on Neural Networks*, 15(2):477–491. 17, 21, 48, 52, 95

[26] Huang, D.-S. and Chi, Z. (2001). Solving linear simultaneous equations by constraining learning neural networks. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, page A31. IEEE. 16, 17, 21, 61, 95

[27] Hush, D. R., Horne, B., and Salas, J. M. (1992). Error surfaces for multilayer perceptrons. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1152–1161. 12

[28] Jeong, S., Solenthaler, B., Pollefeys, M., Gross, M., et al. (2015). Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6):199. 19

[29] John, D. and Anderson, J. (1995). Computational fluid dynamics: the basics with applications. *P. Perback, International ed., Published*. 2

[30] Johnson, F. T., Tinoco, E. N., and Yu, N. J. (2005). Thirty years of development and application of cfd at boeing commercial airplanes, seattle. *Computers & Fluids*, 34(10):1115–1151. 6

[31] Karras, D. A. and Perantonis, S. J. (1995). An efficient constrained training algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 6(6):1420–1434. 12, 20, 21, 90, 95

[32] Kathirvalavakumar, T. and Subavathi, S. J. (2011). Modified backpropagation training algorithm with functional constraints. *International Journal of Neural Networks and Applications*, 4:47–54. 21

[33] Khalil, E. (2012). Cfd history and applications. *CFD Letters*, 4(2):43–46. 2

[34] Kolmogorov (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Dokl. Akad. Nauk. SSSR*, volume 114, pages 953–956. II, 12, 18

[35] Kröse, B., Krose, B., van der Smagt, P., and Smagt, P. (1993). An introduction to neural networks. 22

[36] Kutler, P. (1988). Computational fluid dynamics: Past, present, and future. 2

[37] Lagaris, I. E., Likas, A., and Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000. 18, 19

[38] Langley, P. and Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the ACM*, 38(11):54–64. 19

[39] Layton, W. and Sussman, M. (2014). *Numerical Linear Algebra*. Lulu. com. 16

[40] LeCun, Y. et al. (1989). Generalization and network design strategies. *Connectionism in perspective*, pages 143–155. 20

[41] Lee, H. and Kang, I. S. (1990). Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131. 18

[42] Lee, Y., Oh, S.-H., and Kim, M. W. (1993). An analysis of premature saturation in back propagation learning. *Neural networks*, 6(5):719–728. 12

[43] Lentine, M., Zheng, W., and Fedkiw, R. (2010). A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Transactions on Graphics (TOG)*, 29(4):114. 16

[44] McAdams, A., Sifakis, E., and Teran, J. (2010). A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74. Eurographics Association. 15

[45] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133. 22

[46] Mead, D. (1992). Newton's identities. *The American mathematical monthly*, 99(8):749–751. 48

[47] Minsky, M. and Papert, S. (1990). Perceptrons. 1969. *Cited on*, page 1. 22

[48] Mitchell, T. M. (2009). The discipline of machine learning. july 2006. Technical report, CMU-ML-06-108. 5

[49] Molemaker, J., Cohen, J. M., Patel, S., and Noh, J. (2008). Low viscosity flow simulations for animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 9–18. Eurographics Association. 16

[50] Mori, S., Suen, C. Y., and Yamamoto, K. (1992). Historical review of ocr research and development. *Proceedings of the IEEE*, 80(7):1029–1058. 20

[51] Müller, M., Charypar, D., and Gross, M. (2003). Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association. 1, 3, 15

[52] Murphy, K. P. (2012). Machine learning: A probabilistic perspective. adaptive computation and machine learning. 5

[53] Nielsen, M. A. and Chuang, I. L. (2010). *Neural Networks and Deep Learning*. 22, 23

[54] Parker, D. B. (1985). Learning logic. 23

[55] Peng, D.-Y. and Robinson, D. B. (1976). A new two-constant equation of state. *Industrial & Engineering Chemistry Fundamentals*, 15(1):59–64. 8

[56] Perantonis, S., Ampazis, N., Varoufakis, S., and Antoniou, G. (1998). Constrained learning in neural networks: Application to stable factorization of 2-d polynomials. *Neural Processing Letters*, 7(1). 17

[57] Price, J. F. (2006). *Lagrangian and eulerian representations of fluid flow: Kinematics and the equations of motion*. MIT OpenCourseWare. 3

[58] Richardson, L. F. (1911). Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Phil. Trans. R. Soc. Lond. A*, 210(459-470):307–357. 1

[59] Robertson, A. and Watton, P. (2012). Computational fluid dynamics in aneurysm research: critical reflections, future directions. 7

[60] Rowe, N. C. (1988). *Artificial Intelligence through Prolog by Neil C. Rowe*. Prentice-Hall. 4

[61] Ruiz, A. (2012). *Unsteady numerical simulations of transcritical turbulent combustion in liquid rocket engines*. PhD thesis. 8

[62] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533. 23

[63] Simard, P., LeCun, Y., and Denker, J. S. (1993). Efficient pattern recognition using a new transformation distance. In *Advances in neural information processing systems*, pages 50–58. 21

[64] Sirinukunwattana, K., Raza, S. E. A., Tsang, Y.-W., Snead, D. R., Cree, I. A., and Rajpoot, N. M. (2016). Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. *IEEE transactions on medical imaging*, 35(5):1196–1206. 11

[65] Spalart, P. and Venkatakrishnan, V. (2016). On the role and challenges of cfd in the aerospace industry. *The Aeronautical Journal*, 120(1223):209–232. 6

[66] Sprecher, D. A. (1965). On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, 115:340–355. II, 12, 18

[67] Stam, J. (1999). Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co. 15

[68] Stam, J. (2003). Real-time fluid dynamics for games. In *Proceedings of the game developer conference*, volume 18, page 25. 15

[69] Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. (2016). Accelerating eulerian fluid simulation with convolutional networks. *arXiv preprint arXiv:1607.03597*. 2, 9, 15, 20, 84

[70] Toshev, A. and Szegedy, C. (2014). Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1653–1660. 11

[71] Treuille, A., Lewis, A., and Popović, Z. (2006). Model reduction for real-time fluids. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 826–834. ACM. 16

[72] Tutorial, D. L. (2014). Lisa lab. *University of Montreal*. 4

[73] Versteeg, H. K. and Malalasekera, W. (2007). *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education. 1

[74] Wang, L. and Sun, D.-W. (2003). Recent developments in numerical modelling of heating and cooling processes in the food industry—a review. *Trends in Food Science & Technology*, 14(10):408–423. 7

[75] Werbos, P. J. (1974). *Beyond regression*. 23

[76] Wu, E., Zhu, J., and Chang, Y. (2009). Particle importance based fluid simulation. 2

[77] Yang, C., Yang, X., and Xiao, X. (2016). Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424. 2, 9, 19, 20

[78] Zhai, Z. (2006). Application of computational fluid dynamics in building design: aspects and trends. *Indoor and built environment*, 15(4):305–313. 7

[79] Zhang, D. (2014). Optimize sedimentation tank and lab flocculation unit by cfd. Master's thesis, Norwegian University of Life Sciences. 1, 2

[80] Zhu, B., Lu, W., Cong, M., Kim, B., and Fedkiw, R. (2013). A new grid structure for domain extension. *ACM Transactions on Graphics (TOG)*, 32(4):63. 16