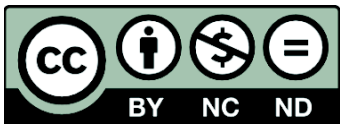
Exploración de métodos en aprendizaje automatizado y su uso en Física de altas energías

Hugo de Jesús Valenzuela Chaparro

Universidad de Sonora
Departamento de Física
Hermosillo, Sonora
2020

Universidad de Sonora

Repositorio Institucional UNISON



Excepto si se señala otra cosa, la licencia del ítem se describe como openAccess

Exploración de métodos en aprendizaje automatizado y su uso en Física de altas energías

Hugo de Jesús Valenzuela Chaparro

Tesis presentada como requisito parcial para optar al título de:
Licenciado en Física

Director:
Dr. Alfredo Martín Castañeda Hernández

Línea de Investigación:
Altas Energias Experimental
Grupo de Investigación:
Grupo de Altas Energías

Universidad de Sonora
Departamento de Física
Hermosillo, Sonora
2020

Dedicatoria

A mi mamá y a mi papá, sin el amor y apoyo incondicional de ella y él no hubiera llegado a esta etapa. A mi hermano por su apoyo e inspiración a dar lo mejor de mí en mi formación de físico.

Lista de traducciones

La gran mayoría de referencias usadas para este trabajo están escritas en inglés, por tal motivo se deja esta lista de palabras clave de cómo se están tomando las traducciones al español.

- Machine learning $\longrightarrow$ aprendizaje automatizado.
- Deep learning $\longrightarrow$ aprendizaje profundo.
- Feedforward neural network $\longrightarrow$ red neuronal prealimentada.
- Activation function $\longrightarrow$ función de activación
- Bias $\longrightarrow$ parámetro de sesgo.
- Weights $\longrightarrow$ parámetros/pesos.
- Cost function $\longrightarrow$ función de costo.
- Loss function $\longrightarrow$ función de pérdida.
- Gradient descent $\longrightarrow$ descenso por gradiente.
- Backpropagation $\longrightarrow$ propagación hacia atrás.
- Learning rate $\longrightarrow$ tasa de aprendizaje.
- Dataset $\longrightarrow$ conjunto de datos.
- Multilayer perceptron $\longrightarrow$ perceptrón multicapa.
- Epoch $\longrightarrow$ época.

Contenido

Capítulo 1

Introducción

El presente estudio se divide en cuatro partes, en la primera se da una introducción que da contexto a la tesis, en la segunda se da una descripción detallada de los elementos de una red neuronal multicapa, incluyendo la arquitectura de la misma. En la tercera parte se explica el origen de los datos mediante simulación estocástica, los que serán posteriormente procesados por el modelo computacional que se propondrá y las variables relevantes en el proceso de aprendizaje. En una cuarta parte se presenta un caso de estudio del uso de redes neuronales para la clasificación de dos procesos en física de partículas, finalmente se discuten los resultados del aprendizaje, las propiedades del modelo y su validez.

1.0.1. Inteligencia artificial, aprendizaje automatizado y aprendizaje profundo

Primeramente debe hacerse una distinción entre las tres áreas mencionadas en el titulo de esta subsección. La inteligencia artificial (IA) surge con la visión de que una computadora sea capaz de aprender por sí misma, tal como los humanos y muchos seres vivos podemos hacerlo. Entonces la IA es un conjunto amplio de algoritmos y métodos para lograr tal meta, por ejemplo, un algoritmo que permita a un robot caminar por un laberinto sin chocar con las paredes. Ahora, dentro de la IA existe un área de métodos conocida como aprendizaje automatizado, que más específicamente trata de que una computadora pueda realizar un proceso sin tener que ser programada explícitamente para ello, esto mediante matemáticas en general, estadística y por medio de datos. Por último, el aprendizaje profundo es a su vez un subconjunto del aprendizaje automatizado, debe a su nombre a que sus algoritmos pueden llegar a tener una secuencia muy grande de composición de cálculos, como se explicará más adelante en este texto. Podemos ver la contención de estas áreas de manera muy clara en la figura **1-1**.

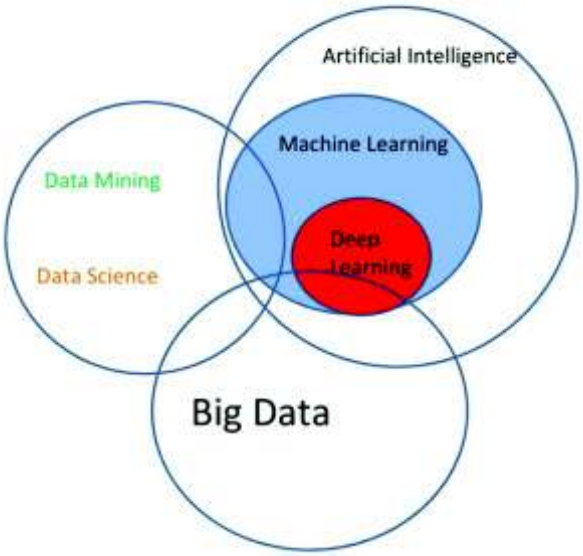


Figura 1-1: Representación de la relación entre experimentos de Big data e inteligencia artificial. A su vez, aprendizaje profundo (deep learning) es un subárea del aprendizaje automatizado (Machine Learning).

1.0.2. Aprendizaje profundo en el contexto de física de partículas

El área de la inteligencia artificial y muy en particular la aplicación de algoritmos con redes neuronales artificiales (aprendizaje profundo) han tomado un gran auge en los últimos tiempos, esto se debe a que usualmente estos algoritmos consumen muchos recursos computacionales y suelen funcionar mejor entre más datos se les proporcionen. Aunque las redes neuronales han sido estudiadas en forma teórica desde hace bastante tiempo, la aplicación de las mismas en el análisis de datos ha evolucionado de forma exponencial con la introducción de nuevas arquitecturas de redes neuronales artificiales, sobre todo con la incorporación de las llamadas capas ocultas (*hidden layers*) las cuales permiten la clasificación de datos con dimensionalidad de variables alta y relaciones no lineales entre las mismas. Fue específicamente en 2012 cuando el aprendizaje profundo comenzó a emerger de manera más notoria, cuando estos algoritmos dieron mejores resultados que el estado del arte en ese entonces.
Los experimentos que recolectan una gran cantidad de datos (o también conocidos como de "big data") pueden beneficiarse del poder predictivo y de clasificación que proporcionan dichos algoritmos. En particular los experimentos en física de partículas, como aquellos asociados al Gran Colisionador de Hadrones del CERN (LHC por sus siglas en ingles) han empezado a incorporar dichos algoritmos inteligentes para optimizar varios de sus procesos. El programa de física del LHC tiene el potencial de contestar algunas de las preguntas fundamentales en la física moderna, como lo son la naturaleza de la masa, la dimensionalidad del espacio, la unificación de las fuerzas fundamentales, el origen de la materia oscura y la complementación del modelo estandar [1].

1.0.3. Procesos de señal y ruido

En los experimentos de física de partículas usualmente se busca un proceso de interés, al cual se le llama señal y que usualmente se encuentra mezclado con otros procesos los cuales dificultan su identificación,

a los cuales se les conoce como ruido. Para distinguir entre los dos procesos se desarrollan métodos que en su mayoría se basan en seleccionar eventos a partir de variables físicas que discriminen entre la señal y el ruido, ejemplos de estas variables pueden ser la energía de las partículas, el momento asociado o parámetros de la trayectoria. Cada una de estas variables y dependiendo de la naturaleza del proceso proporcionan cierta separación entre la señal y el ruido, sin embargo la combinación de estas puede brindar un mayor poder de discriminación ya sea con los métodos multivariables o como en este estudio con la incorporación de redes neuronales. Las variables antes mencionadas pueden ser alimentadas a una red neuronal para mediante un proceso de aprendizaje, por medio de muestras simuladas, lo anterior para la construcción de un modelo que pueda predecir a qué categoría pertenece cada evento. Dicho procedimiento se ha empezado a utilizar en varios análisis en física de partículas, muestra de ellos son los estudios publicados recientemente donde se combina la física de partículas y algoritmos de aprendizaje automatizado [2] [3] [4].

1.0.4. Simulación en física de partículas

Una pieza fundamental para el entrenamiento de las redes neuronales son las muestras de datos, para el caso muy particular del área de altas energías estas muestras pueden ser producidas por medio de paquetes de simulación los cuales usan el método de Monte Carlo para describir los procesos aleatorios. Actualmente existen entornos de simulación que permiten desarrollar muestras de determinados procesos los cuales contienen la información fundamental del proceso físico de estudio, es decir, desde los diagramas de feynman los cuales describen las interacciones fundamentales, las probabilidades de producción de partículas, los diferentes tipos de decaimientos, etc. Adicionalmente dichos paquetes de simulación incluyen una detallada estimación de la respuesta del detector al paso de las partículas, debido a este proceso se pueden estudiar variables reconstruidas como lo son la energía, el momento, parámetros de trayectoria, entre otras. Estas variables pueden ser usadas como datos de entrada a una red neuronal para la creación de un modelo predictivo que pueda diferenciar entre dos procesos.

Capítulo 2

Aprendizaje Automatizado

El aprendizaje automatizado es un campo de estudio de las ciencias computacionales, forma parte del área de la inteligencia artificial y se enfoca principalmente en el desarrollo de algoritmos y técnicas estadísticas con el objetivo principal de lograr que los sistemas computacionales adquieran la habilidad de aprender y solucionar tareas complejas sin haber sido explicitamente programados para ello. Este campo de estudio ha sido desarrollo desde hace ya varias decadas, donde se puede mencionar logros importantes como lo son el desarrollo del *perceptron* en 1958 por Frank Rosenblatt [5], el perceptron fue la primera red neuronal artificial con un algoritmo de ajuste de parámetros. En 1997 la computadora "Deep Blue"desarrollada por la compania IBM logró vencer en una partida de ajedrez al entonces campeon del mundo Garry Kasparov [6]. Del año 2000 en adelante el uso del aprendizaje automatizado ha venido en aumento debido al desarrollo de aplicaciones por grande compañías como lo son Google, Facebook, Uber, entre otras.

2.1. Redes neuronales biológicas

En términos muy generales, las neuronas están constituidas por 3 componentes más importantes. El *cuerpo* de la neurona, llamado soma, que contiene al núcleo y del cual se desprenden ramificaciones llamadas dendritas; el *axón* que es un filamento que se extiende del cuerpo hacia la *sinápsis*, siendo ésta última el espacio que interconecta con otras neuronas.
Básicamente, el proceso consiste en que pueden tenerse señales excitatorias que generan un potencial de acción el cual provoca la transmisión de señales electroquímicas hacia las otras neuronas, o por otro lado, señales inhibitorias que hacen que sea menos probable tener el potencial de acción y por lo tanto no enviando la señal en la sinápsis.
Es la vasta cantidad de neuronas y su compleja conexión que hace que el cerebro humano sea capaz de realizar las operaciones de aprendizaje, además de la plasticidad sináptica que permite la modificación de los pesos de las señales excitatorias o inhibitorias en el transcurso del tiempo.

2.2. Redes neuronales artificiales

Las redes neuronales artificiales (RNA) son un concepto computacional el cual está fuertemente inspirado en el proceso de aprendizaje que realiza el cerebro humano. Esto lo hace por medio de capas de nodos interconectados que simulan las conexiones neuronales, se busca que el sistema computacional pueda resolver y generalizar problemas complejos sin haber sido programado explícitamente para ello, de igual manera que el cerebro humano adquiere conocimiento por medio del aprendizaje, donde la toma de decisiones se realiza por medio de estímulos externos recibidos. Generalmente las conexiones

de las RNA pueden ser muy complejas, de ahí que se le llame aprendizaje profundo al campo del aprendizaje automatizado que las estudia y aplica.
El proceso de aprendizaje en el aprendizaje automatizado (valga la redundancia) se logra mediante el uso de muestras de entrenamiento (training samples, en inglés), de las cuales se adquiere el conocimiento requerido para construir un modelo predictivo que pueda ser usado para la resolución de cierto problema de clasificación o regresión. Dicho proceso es amplio y por consiguiente se divide en varias categorías como se describe a continuación [7].

- **Aprendizaje supervisado** Los datos usados para el entrenamiento ya traen las soluciones deseadas, llamadas etiquetas, y se hace a partir de ciertas variables conocidas como características. Cuando las etiquetas son valores discretos se le llama clasificación, mientras que para valores continuos se trata de regresión. Algunos de los algoritmos más usados de este tipo son K-nearest Neighbors, Regresión Lineal ó logística, Support Vector Machines (SVMs).
- **Aprendizaje no-supervisado** Los datos de entrenamiento no poseen etiquetas, es decir, el algoritmo debe encontrar la manera de buscar patrones o clasificaciones por sí mismo. Queda a criterio de quien investiga interpretar los resultados. Una técnica muy usada es el Clustering.
- **Semi-supervisado** Se manejan datos parcialmente etiquetados, normalmente muchos con etiquetas y pocos sin ellas. El usuario traba en conjunto con el algoritmo para entrenarlo. Un ejemplo es Google Photos, el código reconoce personas, pero quien lo usa debe de dar los nombres y confirmarlo.
- **Por Refuerzo** Este aprendizaje se da a base de ensayo-error con recompensas o castigos. Aquí el Agente aprende y toma decisiones en un Ambiente de acuerdo a una determinada Acción que realiza. Se debe encontrar la mejor estrategia, llamada política, en el mejor timepo. Es un algoritmo muy popular en robótica.

El presente estudio se enfoca exclusivamente en el uso del aprendizaje supervisado, en las siguientes secciones se describen los diferentes tipos de arquitectura de las redes neuronales artificiales.

2.3. Redes neuronales prealimentadas

A continuación se describirá la estructura general de una red neuronal prealimentada, así como también el caso particular del perceptrón y el perceptrón multicapa. Al describir los componentes de dichas redes, se requiere el uso de funciones matemáticas en este contexto, por tal motivo se empieza dando una definición antes de proceder.

2.3.1. Definición de una función matemática

Sean A y B dos conjuntos. Entonces una función de A a B es un conjunto f de pares ordenados en $A \times B$ tal que para cada $a \in A$ existe un único $b \in B$ con $(a, b) \in f$. En otras palabras, una función f de un conjunto A a otro conjunto B es una regla de correspondencia que asigna a cada elemento de x en A un único elemento $f(x)$ en B [8].

2.3.2. Red neuronal prealimentada

Sean $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, $\mathbf{y} = \{y_1, y_2, ..., y_l\}$ vectores de valores numéricos de entrada y salida, respectivamente, que fueron generados por una función $f^*(\mathbf{x}) = \mathbf{y}$. Tengamos $\boldsymbol{\omega} = \{\omega_1, \omega_2, ..., \omega_k\}$ un conjunto de parámetros. Una red neuronal prealimentada define un mapeo $f(\mathbf{x}; \boldsymbol{\omega}) = \mathbf{y}$, tal que $f : \mathbb{R}^n \longrightarrow \mathbb{R}^l$, mediante el ajuste del valor de $\boldsymbol{\omega}$ que resulta en la mejor aproximación de f^* [9].

En términos del aprendizaje automatizado a los parámetros $\boldsymbol{\omega}$ se les denomina pesos, mientras que a las salidas $\mathbf{y}$, etiquetas; a las componentes de $\mathbf{x}$ se les conoce como características. Además, se tendrán m muestras de vectores $\mathbf{x}$, por lo que contaremos con un conjunto de datos con elementos $\mathbf{x}^{(i)}$, donde $i = 1, 2, ..., m$.
Es importante notar que los datos $\boldsymbol{y}$ que tengamos de la funcion f^* generalmente vienen acompañados con un cierto ruido, tal que tendremos mediciones $\mathbf{y} \approx f^*(\boldsymbol{x})$; o bien puede ser una función estocástica tal que $\mathbf{y} = f*(\boldsymbol{x})$ con $\mathbf{y} \sim P(\mathbf{y}|\boldsymbol{x})$. Además, es importante saber que se está asumiendo que la función f^* existe, es decir, que la función puede aproximarse de manera paramétrica.
La estructura de esta red se representa típicamente como una cadena de composición de funciones $f_1, f_2, f_3, ..., f_n$, de tal manera que se tiene $f(\mathbf{x}) = f_n(...f_3(f_2(f_1(\mathbf{x}))))$ [9]. A f_1 se le demomina capa de entrada, f_2 segunda capa, f_3 tercera capa, y así sucesivamente hasta llegar a la capa de salida f_n. A esta composición le podemos asociar una gráfica como en la figura **2-1**.

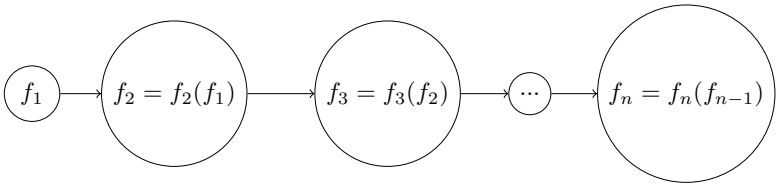


Figura 2-1: Estructura de gráfica de las composiciones de funciones que se realizan en una red neuronal prealimentada hasta llegar a la salida f_n.

Hay que mencionar que la capa f_1 solamente se encarga de transmitir el vector de entrada a la red, no realiza ninguna operación matemática, por lo que en ocasiones en la literatura no se cuenta como capa; no obstante en este texto sí se contabilizará como una capa.
Puede notarse que las capas intermedias $f_2, f_3, ..., f_{n-1}$ no procesan directamente a los datos de entrada ni dan el resultado final, por tal motivo son conocidas como capas ocultas.
La definición de composición anterior es una manera macroscópica de ver las redes neuronales prealimentadas, pero en realidad las funciones son valuadas vectorialmente. Sin embargo, es preferible interpretarlas como unidades en paralelo que transforman un vector a un escalar, en lugar de componentes vectoriales como tal, esto puede verse en la figura **2-2**, donde cada unidad de las capas (funciones) envían su salida a todas las unidades de la capa siguiente. A continuación en la sección 2.3.4 se explica la estructura de dichas unidades para completar la definición.

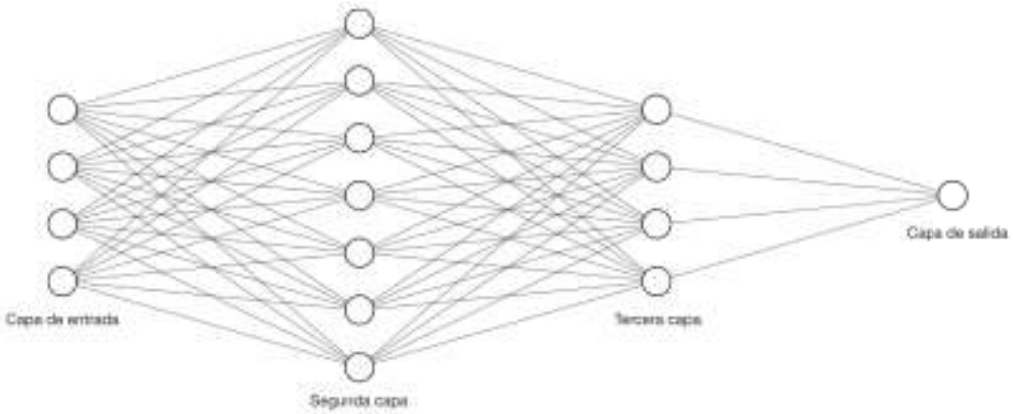


Figura 2-2: Ejemplo de una red neuronal prealimentada con una capa de entrada, dos ocultas y una de salida. Nótese que todas las unidades de una capa están conectadas a todas las de la capa siguiente, esto hace que también se les llame redes completamente conectadas (fully connected, en inglés).

2.3.3. Función de activación

Una función de activación φ es una función univaluada y con rango real, es decir $\varphi : \mathbb{R} \longrightarrow \mathbb{R}$. Su papel es transmitir el valor de salida de cada unidad, que será a su vez entrada de las unidades de las capas posteriores, o bien, la salida si se encuentra en la última capa.
Existen dos tipos de función de activación: lineales y no lineales [10].
De la parte lineal, la función más básica es la identidad definida como

$$\varphi(x) = x \tag{2-1}$$

su uso más notable es en la capa final cuando las salidas están en un rango real, es decir, cuando se trata de una regresión [11]. También se tiene la función de activación lineal

$$\varphi(x) = kx;\ k \in \mathbb{R} \tag{2-2}$$

Por otro lado, en las funciones de activación no lineales se tiene algunas como

$$\varphi(x) = sign(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases} \tag{2-3}$$

$$\varphi(x) = max\{x, 0\} \tag{2-4}$$

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{2-5}$$

$$\varphi(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2-6}$$

Donde la ecuación (2-3) es la función signo que pueda ser usada para salidas binarias. La función (2-4) se conoce como rectificadora (conocida también como ReLU, por abreviatura en inglés), muy utilizada actualmente por la facilidad del entrenamiento de redes neuronales artificiales con esta función de activación. Por otra parte, (2-5) es la función sigmoide, la cual tiene un rango en $[0, 1]$ que es útil cuando

se busca interpretar la salida como una probabilidad. Por último la función tangente hiperbólica (2-6) es similar a la sigmoide, pero ajusta su rango en $[-1, 1]$, usada cuando las salidas pueden ser tanto negativas como positivas [11], aunque ya no pudiendo ser interpretada como probabilidad.
Cuando se tienen solamente funciones de activación lineales en una RNA prealimentada, por más capas ocultas que tenga el modelo seguirá siendo lineal, pues la combinación lineal de tales funciones sigue siendo lineal. Por otro lado, si la combinación fuera de funciones no lineales, se daría más robustidad a la red de tal manera que podría captar mejor las complejidades de la función que se desea generalizar. Para aplicar un tipo y otro depende de la necesidad del problema.

2.3.4. Unidades

Cada nodo de una red neuronal se conoce como unidad. Primeramente recibe los valores generados por todas las unidades anteriores (o los datos de entrada si se encuentra en la primera capa) y luego se evaluan en una función de activación φ. En la figura **2-3** puede verse la representación esquemática. Nótese que se tienen n argumentos como entrada para la unidad, no obstante, es preferible usar funciones evaluadas con un solo argumento. Para esto se divide a cada unidad en dos partes, una función integradora g que reduzca las n entradas a un solo valor que es evaluado en una función de activación φ, ésta última se encarga de dar la salida del respectivo nodo [12]. El valor que regresa la función integradora se conoce como *valor de pre-activación* [11] y normalmente es la suma ponderada de las entradas multiplicadas con los parámetros:

$$z = g(\boldsymbol{x}; \boldsymbol{\omega}) = \sum_{i=1}^{n} \omega_i x_i \tag{2-7}$$

donde por conveniencia se denota con la letra z. Entonces, en cada unidad la función de activación tendrá la salida

$$a = \varphi(z) \tag{2-8}$$

Las unidades que estén en la última capa darán las predicciones de la RNA, denotadas por $\hat{y}$, así

$$\hat{y} = \varphi(z) \tag{2-9}$$

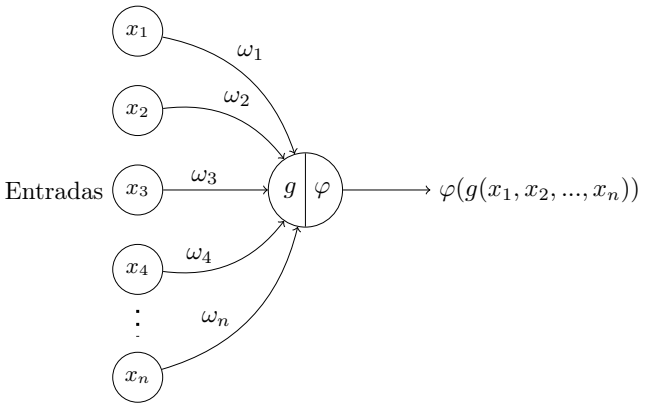


Figura 2-3: Representación de una unidad. La función integradora g convierte las entradas x_i a un escalar que se evalúa en la función de activación φ para dar la salida.

2.3.5. Parámetro de sesgo

Para que una RNA prealimentada aprenda el mapeo de un conjunto de datos, es necesario encontrar la manera de ajustar los parámetros $\boldsymbol{\omega}$ que mejor generalicen. Cada capa de la red tiene un cierto número de capas con funciones de activación, y el valor de preactivación que reciben viene dado por la ecuación (2-7).
Ahora, la ecuación de un modelo lineal, con parámetros $\boldsymbol{\omega} = \{\omega_0, \omega_1, \omega_2, ..., \omega_k\}$ y que mapea un conjunto de entradas $\boldsymbol{x} = \{x_1, x_2, ..., x_n\}$ a la salida Y, es [13]

$$Y = \omega_0 + \sum_{i=1}^{n} \omega_i x_i \tag{2-10}$$

Comparando la ecuación anterior con la función integradora (2-7), podemos notar que hace falta el término ω_0, el cual se conoce como *parámetro de sesgo* en el campo del aprendizaje automatizado.
Nótese que ω_0 es la intersección, es decir, el valor que toma Y cuando $\boldsymbol{x} = 0$. Debido a que Y está sesgada a ser ω_0 en ausencia de entradas $\boldsymbol{x}$, se le llama parámetro de sesgo [9]. Cabe aclarar que no debe confundirse con el sesgo en términos de estadística, el cual indica que el valor esperado de un estimador no es igual al valor verdadero que se pretende estimar. Además, normalmente el parámetro de sesgo suele llamarse simplemente bias en la literatura, por su equivalente en el idioma inglés.
Por otro lado, siempre se añade el parámetro de sesgo a los valores de preactivación cuando se manipula con RNA, de aquí en adelante se estará considerando en este trabajo. Se acostumbra añadir una unidad adicional, que tenga como salida el número 1, en cada capa. Esto con la intención de que en la capa posterior se tengan entradas $\boldsymbol{x} = \{1, x_1, x_2, ..., x_n\}$ y parámetros $\boldsymbol{\omega} = \{\omega_0, \omega_1, \omega_2, ..., \omega_k\}$, donde ω_0 es el parámetro de sesgo que también debe ajustarse a los datos; entonces los valores de preactivación simplemente quedarían

$$z = g(\boldsymbol{x}; \boldsymbol{\omega}) = \sum_{i=0}^{n} \omega_i x_i \tag{2-11}$$

en este texto se considerará esta convención para escribir la función integradora como la ecuación (2-11). Por lo tanto es importante considerar siempre el parámetro de sesgo cuando evaluemos las funciones de activación, justo como en (2-8) y (2-9).
Para ver por qué es importante el sesgo, veamos un ejemplo concreto en el plano, o sea solo con una variable independiente x que explica a la variable dependiente Y. Supongamos que se tiene un diagrama de dispersión y el conjunto de puntos está muy alejado del eje x, tal como la figura **2-4**. La línea color turquesa sería un ajuste a los puntos sin poder mover el parámetro de sesgo, es decir, con intersección en el origen, se ve que tiene mucho error y no generaliza la tendencia de los datos. Por otro lado, la recta roja sí tiene parámetro de sesgo y puede moverse a lo largo del eje y, pudiendo ajustarse mejor así a los datos; dicha recta fue ajustada con el método de mínimos cuadrados. Esto puede extenderse a cualquier dimensión, pues se tendría un hiperplano el cual podría ser movido del origen con el parámetro.

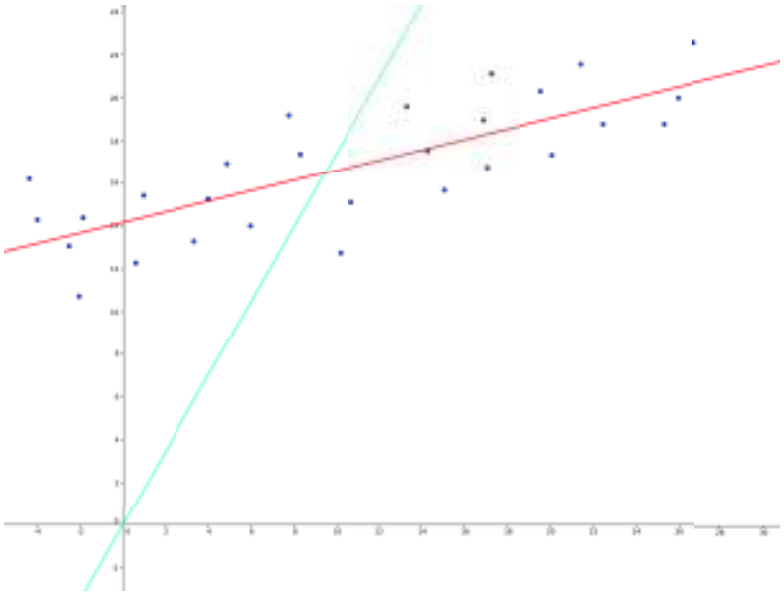


Figura 2-4: La recta roja es libre de mover su intersección en el eje y, pudiendo generalizar mejor los puntos que la línea turquesa fija en el origen.

Vemos que se pierde una importante capacidad de generalizar si no se considera al parámetro de sesgo, incluso aunque sea en la función integradora. Esto porque los valores de preactivación serán el dominio para las funciones de activación, por lo que si se tienen problemas para modelar desde ahí, la red podría no dar el mejor resultado deseado.
Otra forma de ver la importancia del parámetro de sesgo, es fijándonos en la función de activación directamente. Tomemos de ejemplo la función sigmoide $f(x) = 1/(1 + e^{-x})$. Supongamos que es evaluada por un valor de preactivación x, sin parámetro de sesgo, como $f(x)$. Véase la figura **2-7**. Si ahora se tuviera un sesgo, por ejemplo de 5, se evaluaría como $f(x+5)$, la curva recorrería su valor en el origen a 5 unidades a la derecha, tal como la figura **2-6**. Por último, similarmente, si tuviéramos un sesgo de -5 se recorrería 5 unidades a la izquierda. tal como la figura **2-5**.

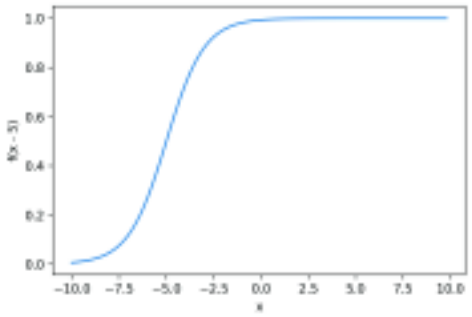


Figura 2-5: Gráfica de la función sigmoide $f(x-5)$.

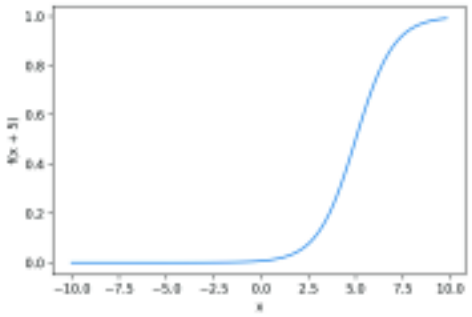


Figura 2-6: Gráfica de la función sigmoide $f(x+5)$.

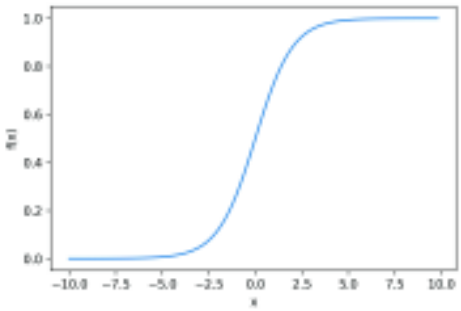


Figura 2-7: Gráfica de la función sigmoide $f(x)$.

En conclusión, las funciones de activación pueden recorrer los intervalos en los que toman los valores más importantes y el parámetro de sesgo permite que la RNA sea más robusta a la hora de mapear los datos de entrenamiento a las etiquetas.

2.3.6. Aproximadora universal

Una red neuronal prealimentada con al menos una capa oculta con función de activación no lineal, es capaz de aproximar cualquier función desde un dominio de entrada hasta otro dominio de salida, esto está demostrado en [14]. Por tal motivo a estas redes se les conoce como Aproximadoras Universales. Cabe destacar que es una prueba no constructivista, es decir, no te proporciona la estructura que debe tener la red para que converga, solamente se asume que hay suficientes unidades en la capa oculta.
Como se verá más adelante, en este estudio se ajustó una red neuronal prealimentada con una capa oculta con función de activación no lineal, teniendo entonces una red aproximadora universal, en el supuesto de que existe una función que mapea nuestros datos de entrada a los de salida.

2.4. Cuantificación del error y estimación de los parámetros

Para ajustar los parámetros $\boldsymbol{\omega} = \{\omega_0, \omega_1, \omega_2, ..., \omega_k\}$ debe construirse una cierta cuantificación del error y una medida del desempeño en las predicciones de la RNA. Puede definirse una función objetivo a optimizar y con los parámetros como argumentos. Una alternativa puede ser una función de aptitud que mida qué tan acertado es el modelo o una función de costo que cuantifica el error del modelo [7]. La primera debe maximizarse mientras que la segunda minimizarse con respecto a los parámetros. En el aŕea del aprendizaje profundo suele trabajarse con la función de costo.

2.4.1. Función de costo

Para cuantificar el error se toma en cuenta la predicción del modelo $(\hat{y})$ y el valor verdadero (y) en un solo dato mediante una función de pérdida $L = L(\boldsymbol{\omega})$. Por ejemplo, el error cuadrático se define como

$$L(\boldsymbol{\omega}) = (\hat{y} - y)^2 \tag{2-12}$$

donde, por la ecuación (2-9) y (2-7) se tiene a $\hat{y}$ en función de los parámetros porque son los que tratamos de ajustar y que hacen varíar a $\hat{y}$. Es la razón de que la pérdida esté en función de los parámetros.
Para poder optimizar a lo largo de todo el conjunto de datos, se plantea una función de costo $J = J(\boldsymbol{\omega})$ como el promedio de las funciones de pérdida respecto al total de datos, teniendo

$$J(\boldsymbol{\omega}) = \frac{1}{m}\sum_{i=1}^{m} L_i(\boldsymbol{\omega}) \tag{2-13}$$

con m la cardinalidad del conjunto de datos.
Siguiendo con el ejemplo del error cuadrático, si planteamos la función de costo se llega al conocido Error Cuadrático Medio (ECM) definido como

$$J(\boldsymbol{\omega}) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \tag{2-14}$$

En realidad, existen más funciones de costo y puede ser incluso arbitrario el proponerla dependiendo del problema. No obstante, lo anterior es válido para regresión, es decir, para predicciones con valores numéricos en donde es práctico usar $(\hat{y} - y)^2$.
En este proyecto se trabajó con clasificación, binaria específicamente, para tal caso debe usarse una función de costo adecuada. Cuando se utiliza la función de activación sigmoide para clasificar, los resultados son probabilidades y evidenemente se tendrán valores entre 0 y 1, así que se usa la función de costo

$$J(\boldsymbol{\omega}) = -\frac{1}{m}\sum_{i=1}^{m}(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)) \tag{2-15}$$

la cual es conocida como entropía cruzada (cross-entropy) [15]. Será la función de costo a minimizar en la RNA que se propondrá más adelante.

2.4.2. Descenso por gradiente

Una función $f : \mathbb{R} \longrightarrow \mathbb{R}$ es derivable en x si

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \text{ existe} \tag{2-16}$$

En tal caso, el límite se designa por f' y recibe el nombre de derivada de f en x [16]. El valor $f'(x)$ es la pendiente de la recta tangente en el punto x. Esa pendiente puede interpretarse como la razón de cambio de la función en ese punto, y el signo indica el sentido hacia donde está cambiando.
Lo anterior puede generalizarse para cualquier dimension. Sea la función $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, fijemos el vector $x \in \mathbb{R}$, y considérese la expresión

$$\lim_{\alpha \to 0} \frac{f(x + \alpha \mathrm{e}_i) - f(x)}{\alpha} \text{ existe} \tag{2-17}$$

donde e_i es el i-ésimo vector unitario. Si el limite anterior existe es llamado la derivada parcial i-ésima de f y se denota como $\partial f(x)/\partial x_i$. Cada derivada parcial da la razón de cambio en la componentes x_i del vector x. Asumiendo que todas las derivadas parciales existen, el gradiente de f en x se define como el vector columna [17]

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} \tag{2-18}$$

de manera análoga a la derivada, el gradiente es un vector e indica la dirección hacía donde la función f tiene el mayor incremento, su magnitud es la razón de cambio.
Para optimizar, específicamente para minimizar la función de costo, se hace uso del gradiente de dicha función. Tengamos el vector de parámetros $\boldsymbol{\omega} = \{\omega_0, \omega_1, ..., \omega_n\}$, el error en cada predicción en

función de tales parámetros $L = L(\boldsymbol{\omega})$. Ahora, como estamos trabajando con un conjunto datos con m vectores, la función de costo se define como la suma de los respectivos errores en cada vector:

$$J(\boldsymbol{\omega}) = \sum_{i=1}^{m} L_i(\boldsymbol{\omega}) \tag{2-19}$$

Si J es diferenciable, comenzamos con valores arbitrarios y aleatorios de los parámetros y actualizamos su valor moviéndonos una pequeña distancia, en el espacio de $\boldsymbol{\omega}$, en la dirección en que J decrece más rápidamente. Es decir, en la dirección de $-\nabla_{\boldsymbol{\omega}} J$ [18]. De esta manera, tenemos el algoritmo iterativo

$$\omega_{kj}^{(\tau+1)} = \omega_{kj}^{(\tau)} - \eta \frac{\partial}{\partial \omega_{kj}} \sum_{i=1}^{m} L_i(\boldsymbol{\omega}) \tag{2-20}$$

donde τ es un entero que indica la iteración en la que estamos, mientras que η es un número real, positivo y pequeño que controla los incrementos, es conocido como tasa de aprendizaje. Por otro lado, los subíndices indican que el peso actualizado está conectado de una unidad j a otra unidad k en la capa posterior.

La ecuación (2-20) es la actualización de un sólo parámetro y en cada paso se necesita hacer una sumatoria sobre todos los datos, lo que puede ser computacionalmente muy costoso porque en aprendizaje automático, específicamente en el aprendizaje profundo, suelen manejarse una gran cantidad de datos. Para arreglar esto se trabaja con un solo vector en cada iteración, elegido aleatoriamente y sin remplazo, por lo que tenemos la ecuación

$$\omega_{kj}^{(\tau+1)} = \omega_{kj}^{(\tau)} - \eta \frac{\partial L_m}{\partial \omega_{kj}} \tag{2-21}$$

la cual se conoce como Descenso por Gradiente Estocástico. El cálculo del gradiente de esta forma tiene más ruido, debido a la omisión de los demás datos en la suma, sin embargo puede ser una ventaja porque tiende a estancarse menos en mínimos locales. Por otro lado L_m indica que el recalibramiento de los parámetros está siendo respecto al vector m-ésimo del conjunto de datos que tengamos y además está reducida a la función de pérdida por la misma razón.

Hay que tener en cuenta que la tasa de aprendizaje debe elegirse con cuidado, pues si es muy grande puede que haya oscilaciones alrededor del mínimo que se pretende alcanzar, o bien si es muy pequeña puede tardar mucho en converger. La meta del algoritmo es iterar hasta que se llegue a un minimo de la función $J(\boldsymbol{\omega})$, lo cual es complicado porque se corre el riesgo de llegar a un mínimo local y estancarse ahí. Más adelante se verá como poder resolver ese problema.

El método para hacer las derivadas respecto a los pesos que se usan en este algoritmo, se conoce como propagación hacia atrás (backpropagation). En el apéndice B se puede encontrar su formulación.

2.4.3. Puntos críticos de la función de costo

Sea $f : U \subset \mathbb{R}^n \longrightarrow \mathbb{R}$ una función escalar, un punto $\boldsymbol{x_0} \in U$ se llama mínimo local de f si existe una vecindad V de $\boldsymbol{x_0}$ tal que para todos los puntos $\boldsymbol{x}$ en V, se tiene $f(\boldsymbol{x}) \geq f(\boldsymbol{x_0})$. Similarmente, $\boldsymbol{x_0} \in U$ es un máximo local si existe una vecindad V de $\boldsymbol{x_0}$ tal que $f(\boldsymbol{x}) \leq f(\boldsymbol{x_0})$ para todo $\boldsymbol{x} \in V$. Un punto $\boldsymbol{x_0}$ es un punto crítico de f si $\nabla f(\boldsymbol{x_0}) = 0$. Un punto crítico que no es un mínimo local, ni máximo local se denomina punto silla [19]. Hablamos de un máximo global $\boldsymbol{x_0}$ si $f(\boldsymbol{x_0})$ toma el valor absoluto máximo de f, mientras que hablamos de mínimo global si toma el mínimo absoluto de f. Un ejemplo en el plano puede verse en la figura **2-8**.

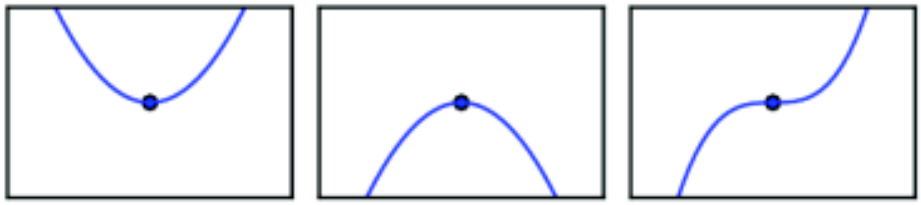


Figura 2-8: **Puntos críticos de una función**. Mínimo (izquierda), máximo (centro), punto silla (derecha). Imagen tomada de [9].

Esto quiere decir que en los puntos críticos de la función de costo, $\nabla J(\boldsymbol{\omega}) = 0$, el gradiente no puede distinguir hacia dónde la función incrementa más rápidamente, por lo que de llegar ahí, el algoritmo descenso por gradiente se detendría. En el contexto del aprendizaje profundo, puede llegarse a trabajar con funciones de costo que dependen de muchos parámetros, incluso del orden de cientos de miles, por lo que dichas funciones pueden tener muchos mínimos locales que no son óptimos, además de muchos puntos silla rodeados de superficies planas. Por lo tanto, el objetivo será encontrar un valor de la función de costo $J(\boldsymbol{\omega})$ que sea lo suficientemente pequeño, pero no necesariamente óptimo en un sentido formal [9].
El problema anterior también se debe a que jamás obtenemos una forma explícita de $J(\boldsymbol{\omega})$, ni siquiera una porción considerable de sus valores, simplemente vamos haciendo movimientos de acuerdo al descenso por gradiente sin conocer bien el espacio y estructura de la función de costo.

2.4.4. Conjuntos de datos de entrenamiento, de validación y de prueba

Para el entrenamiento de un modelo en aprendizaje automatizado, se trabaja con tres conjuntos de datos, y el aprendizaje profundo no es la excepción. Entonces tenemos el conjunto de entrenamiento en el que se aplica el descenso por gradiente para ajustar los parámetros, el conjunto de validación para corroborar qué tan bien van las medidas de desempeño del modelo después de cada iteración, y por último un conjunto de prueba para dar los resultados finales [20]. En otras palabras el conjunto de validación es para ver si el modelo generaliza bien con datos que no ha visto en su entrenamiento, mientras que el conjunto de prueba es una última prueba que da una evaluación independiente e insesgada del modelo.
Generalmente cuando se hace un proyecto, se nos da un solo conjunto de datos, entonces de ahí deben obtenerse los tres. Usualmente la proporción exacta de los tres conjuntos se decide arbitrariamente, pero normalmente se usa proporción entrenamiento-validación-prueba de 60 %/20 %/20 % si se cuenta con pocos datos (poco en contexto del aprendizaje profundo) del orden de 100, 1000 o 10000, mientras que para big data suelen usarse proporciones como 98 %/1 %/1 %. Además es muy importante que dicha división del conjunto de datos se haga de manera aleatoria uniforme, esto para mantener la misma distribución de los datos en los tres conjuntos y no crear un sesgo.

2.4.5. Medidas de desempeño de la red neuronal artificial

Después de cuantificar el error y empezar a estimar los parámetros minizando la función de costo, es necesario proponer medidas que indiquen qué tan acertado es el modelo. En el ejemplo concreto de clasificación, se tiene la exactitud definida como

$$AC = \frac{\#VP + \#VN}{m} \tag{2-22}$$

donde $\#VP$ son los verdaderos positivos y $\#VN$ los verdaderos negativos de las predicciones en total, divididos por la cardinalidad (m) del conjunto de datos. La exactitud es la proporción de predicciones correctas, por lo tanto $0 \leq AC \leq 1$.

2.5. Modelo de McCulloch-Pitts

McCulloch y Pitts desarrollaron en 1943 un modelo matemático neuronal basado en el conocimiento sobre la estructura neuronal que se tenía en esa fecha. Como la mayoría de modelos, no trata de recrear exactamente lo que pasa en las neuronas, sino que toma el concepto general para obtener un resultado práctico.
Para hacer su desarrollo matemático, consideraron "El sistema nervioso es un conjunto de neuronas, cada una teniendo un soma y un axón. Sus uniones, o sinapsis, siempre están entre el axón de una neurona y la soma de otra. En cada instante una neurona tiene un cierto umbral, el cual debe ser excedido por una señal excitatoria para mandar un impulso" [21].
Lo último se refiere a la Ley del Todo o Nada (all-or-none law), la cual dice que una neurona necesita obtener un estimulo que supere un umbral para mandar una señal. A partir de esto, la meta de McCulloch y Pitts fue representar las relaciones funcionales entre las neuronas en términos de Lógica Booleana [22].
Consecuentemente, la red neuronal de McCulloch-Pitts utiliza solamente entradas $\boldsymbol{x} = \{x_1, x_2, x_3, ..., x_n\}$ que son señales binarias, unos y ceros, las unidades producen solamente resultados binarios con su función integradora. Los nodos transmite una salida y , exclusivamente unos o ceros [12], usando como función de activación la función escalón de Heaviside, con umbral θ:

$$\varphi(x) = \begin{cases} 1 & x \geq \theta \\ 0 & \textit{en otro caso} \end{cases} \tag{2-23}$$

la estructura puede verse en la figura **2-9**.

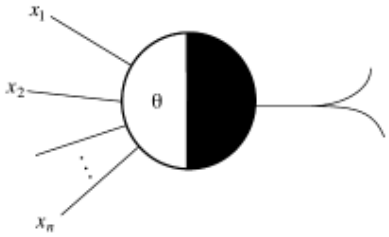


Figura 2-9: Estructura de la neurona de McCulloch-Pitts, extraída de Neural Networks: A Systematic Introduction de Raúl Rojas [12].

McCulloch-Pitts tomaron como consideración física que la estructura de la red neuronal biológica no cambia, por lo que las neuronas de su modelo siempre tienen parámetros fijos que no cambian en el tiempo y por lo tanto no consideraron un algoritmo iterativo para ajustarlos; eso tenía que hacerse manualmente por una persona. Claramente, en la realidad puede variar por el aprendizaje mismo o por factores como la edad, para tratar con esto McCulloch-Pitts consideraron que cuando ocurra ese cambio biológico, es posible remplazar la red del modelo por otra red equivalente al cambio biológico, sin embargo dejaron muy claro que esto no es una explicación factual de la realidad [21].
Por último, si tan solo una señal de las entradas es inhibitoria, la unidad siempre dará como resultado inactivación, simbolizada por el cero. El modelo puede parecer muy límitado, pero cuando no

hay señales inhibotorias, puede comportarse como una compuerta lógica capaz de implementar otras funciones lógicas de n argumentos [12]. La neurona de McCuloch-Pitts es el inicio que sirvió como inspiración de RNA que le precedieron, fue un acontecimiento científico muy importante.

2.6. Perceptrón

Otro modelo muy importante es conocido como perceptrón, está basando en las neuronas de McCuloch-Pitts. Fue creado por Frank Rosenblatt y para principios de 1960 había construido hardware especializado para implementar este modelo. La principal diferencia es que el perceptrón admite entradas númericas generales, no sólo señales binarias, además de que su estructura de parámetros puede cambiar en el tiempo.
Sean $\boldsymbol{x} = \{x_0, x_1, x_2, ..., x_n\}$, n entradas númericas, con $x_0 = 1$, y sea $\boldsymbol{\omega} = \{\omega_0, \omega_1, \omega_2, ..., \omega_n\}$ el vector de pesos. La 2-dupla (x_0, ω_0) constituye el parámetro de sesgo. El perceptrón es una red neuronal artificial prealimentada, con sólo una capa de entrada $\boldsymbol{x}$ y una capa de salida. Los resultados que puede dar el perceptrón son binarios y consiste en el conjunto $y = \{-1, +1\}$ por lo que en la capa final usa como función de activación la función signo dada por la ecuación (2-3) [11]. Puede asociarse una clase C_1 a la salida $y = 1$ y una clase C_2 a $y = -1$, por lo que puede considerarse un modelo de clasificación.
Los valores de preactivación son mapeados por la función integradora convencional, dada por (2-11), por lo tanto las predicciones que hará el perceptrón son de la forma:

$$\hat{y} = sign(\sum_{i=0}^{n} \omega_i x_i) = sign(\boldsymbol{\omega}^\top \boldsymbol{x}) \tag{2-24}$$

la representación de la estructura del perceptron puede verse en la figura **2-10**.

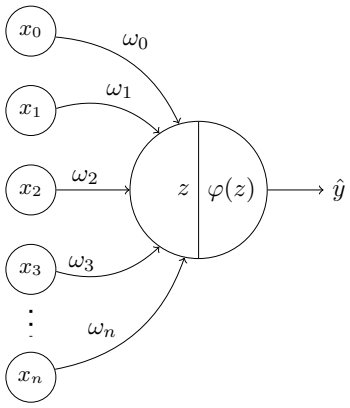


Figura 2-10: Representación gráfica de un perceptrón. Donde z representa el valor de preactivación, dado por (2-11), que posteriormente se evalúa en la función de activación signo φ dando la predicción $\hat{y}$.

2.6.1. Criterio del perceptrón

En Aprendizaje Automatizado, se denomina aprendizaje cuando un modelo encuentra los parámetros que mejor hagan la mejor generalización de los datos que se le suministren. A la acción de encontrar tales parámetros mediante algoritmos, se le conoce como entrenamiento. Normalmente, se propone una función, que representa el error en las predicciones respecto a los parámetros asignados, la cual

debe minimizarse durante el entrenamiento para que el modelo aprenda. Como ya se vio, dicha función se denomina Función de Costo

Como se mencionó anteriormente, el perceptrón tiene la propiedad de que puede cambiar sus parámetros en el tiempo, por lo que es capaz de aprender de los datos. Sin embargo, para el tiempo que Rosenblatt propuso al perceptrón, la minimización del error era hecha heurísticamente por hardware y no había una metodología formal respecto a la optimización. La meta del algoritmo era minimizar el número de clasificaciones erróneas, para lo cual no pudo darse una demostración que garantizara su convergencia.

Para explicar de una manera matemática el entrenamiento heurístico del perceptrón, después de la propuesta inicial de Rosenblatt, se ideó el concepto del criterio del perceptrón, que es básicamente una función continua definida en partes, que contabiliza las clasificaciones erróneas. Supongamos que tenemos un conjunto de datos con m vectores que se usarán para entrenar al modelo. Para cada vector $x_i = \{1, x_1, x_2, ..., x_n\}$ se tendrá asociada una salida $y_i = +1$ si pertenece a la clase C_1 o $y_i = -1$ a la clase C_2. El vector de parámetros $\omega = \{\omega_0, \omega_1, \omega_2, ..., \omega_n\}$. De la ecuación (2-24) se deduce que $\boldsymbol{\omega}^\top \boldsymbol{x_i} > 0$ para la clase C_1, mientras que $\boldsymbol{\omega}^\top \boldsymbol{x_i} < 0$ para C_2. O bien, en general podimos escribir estas condiciones como $\boldsymbol{\omega}^\top \boldsymbol{x_i} \hat{y}_i > 0$. El criterio del perceptron entonces se define por la siguiente función de costo [18]:

$$J^{perc}(\boldsymbol{\omega}) = -\sum_{x_i \in M} \boldsymbol{\omega}^\top \boldsymbol{x_i} \hat{y}_i \tag{2-25}$$

donde M es el conjunto de entradas que fueron clasificadas erróneamente por el vector de parámetros $\boldsymbol{\omega}$ actual. Nótese que $J^{perc}(\boldsymbol{\omega})$ es una sumatoria de términos positivos, de los vectores con error en su clasificación, por lo que es una medida de error e irá cambiando su contribución mientras el algoritmo vaya aprendiendo.

Se debe tener en cuenta que en el criterio del perceptrón (2-25) no se está minimizando al error o costo total, pues en cada iteración la función está cambiando al tener en cuenta solamente a los datos mal clasificados. Esta fue una función de costo ad hoc para poder aplicar el algoritmo de descenso por gradiente y que su derivada no se esté anulando siempre causando que el método no converja. Si se hubiera por ejemplo, considerado como función de costo como el total de datos mal clasificados directamente, sería una función de $\boldsymbol{\omega}$ constante definida por partes, con discontinuidades cada vez que un cambio en $\boldsymbol{\omega}$ causa que la frontera de decisión (definida por la función signo que se usa como activación) se mueva a través de un dato del conjunto total, esto hubiera causado que el gradiente fuera cero casi en todas partes [23] e imposibilitando el descenso por gradiente.

2.6.2. Entrenamiento del perceptrón

Para entrenar al perceptron, se aplica el algoritmo de descenso por gradiente estocástico (2-21) al criterio del perceptrón (2-25) obteniendo

$$\boldsymbol{\omega}_j^{(\tau+1)} = \boldsymbol{\omega}_j^{(\tau)} + \eta \hat{y}_j \boldsymbol{x_j} \tag{2-26}$$

que es el algoritmo de aprendizaje del perceptrón. El subíndice j el j-ésimo vector para el cual se está actualizando el vector de parámetros, x_i es el vector de entrada para el cual se está haciendo la predicción $\hat{y}_i$. El parámetro η sigue siendo la tasa de aprendizaje.

Ahora, el criterio del perceptrón se enfoca en la suma de las clasificaciones érroneas, por tal motivo la ecuación (2-26) solamente debe aplicarse en los vectores mal clasificados, uno cada vez hasta llegar a una convergencia, o bien a una tolerancia de error en las predicciones predefinida.

Podemos incluir a todo el conjunto de datos para usarse exlicitamente en el algoritmo, rescribiendo la ecuación a

$$\boldsymbol{\omega}_j^{(\tau+1)} = \boldsymbol{\omega}_j^{(\tau)} + \eta(y_j - \hat{y}_j)\boldsymbol{x}_j \tag{2-27}$$

donde y_j es la salida real que se quiere predecir, por lo que su diferencia con la predicción $\hat{y}_j$ dará cero y no se hará ajuste en los parámetros una vez que sea correcta.
En esta nueva expresión, habrá un factor de ± 2 multiplicando, dependiendo de la clasificación errónea que se haga. Cuando $y_j = +1$, $\hat{y}_j = -1$, será de $y_j - \hat{y}_j = 2$ por lo que se le sumará al parámetro $\boldsymbol{\omega_j}$; mientras que cuando se tenga $y_j = -1$, $\hat{y}_j = +1$ implica que $y_j^* - y_j = -2$ y se le restará al parámetro $\boldsymbol{\omega_j}$. Sin embargo, no hay que preocuparnos por esto, pues es un número muy pequeño y no afecta la convergencia.

2.6.3. Criterio de convergencia

Como una idealización, el algoritmo de descenso por gradiente debería aplicarse hasta que se redujera el error a 0 %. En la realidad esto no ocurre, por lo que no es práctico dejar el algoritmo un número indeterminado de iteraciones.
Una alternativa es dar un número entero de iteraciones. O bien un número de épocas, donde época se refiere a un ciclo donde el algoritmo de descenso por gradiente estocástico cubre todo el conjunto de datos.
No obstante, lo anterior tampoco es lo más conveniente porque si se itera demasiado, se corre el riesgo de empezar a generalizar el ruido que tienen los datos, es decir un sobreajuste (overfitting en inglés). Lo que suele hacerse es utilizar una medida de desempeño del algoritmo, como por ejemplo la exactitud (AC)

$$AC = \frac{\#VP + \#VN}{\#datos} \tag{2-28}$$

donde VP son los verdaderos positivos y VN los verdaderos negativos de las predicciones, divididos por el total de elementos del conjunto de datos.
Ahora, como se vio en la sección 2.4.4 en aprendizaje automatizado se tiene un conjunto de datos de entrenamiento y uno de validación. Así que otra opción es ir aplicando el algoritmo de gradiente descendiente en los datos de entrenamiento unas determinadas épocas, e ir calculando el costo respecto al conjunto de validación después de cada iteración. Al principio se espera que el error en el conjunto de validación (dado por la respectiva función de costo) vaya disminuyendo, pero llegará un punto en el que aumentará, justo en ese momento se detienen las iteraciones; esta técnica se conoce como detención temprana (early stopping) [20].

2.6.4. Ejemplo práctico del perceptrón

Un ejemplo sencillo de la aplicación del perceptrón, para clasificación, es la separación de puntos en un plano bidimensional de acuerdo a una recta. Para esto se hizo una simulación de 20 puntos con una distribución uniforme en la sección $[0, 10] \times [0, 10]$, y luego una etiquetación binaria según el punto se encontraba por encima (rojos) o por debajo (azules) de la recta $Y = 2X$ lo cual puede verse en la Figura **2-11**.

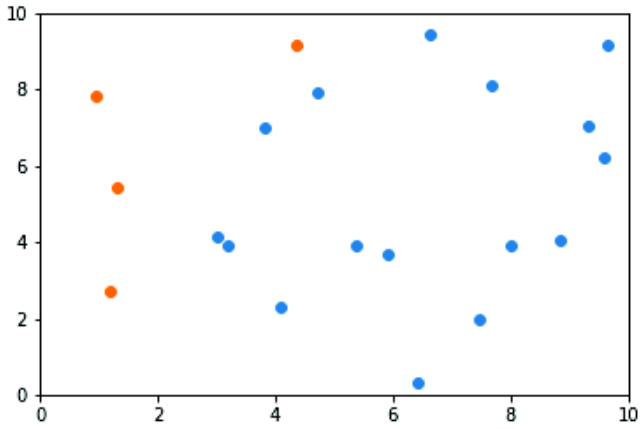


Figura 2-11: Puntos generados aleatoriamente y separados por la recta $Y = 2X$.

Claramente, estos serán los datos usados en el entrenamiento para poder hacer predicciones. Las entradas serán las coordenadas de los puntos, con los vectores $\boldsymbol{x} = (1, X, Y)$, los parámetros $\boldsymbol{\omega} = (\omega_0, \omega_1, \omega_2)$, el parámetro de sesgo dado por $(1, \omega_0)$. Por convención los puntos sobre o a la izquierda de la recta son rojos y etiquetados por $y = 1$, mientras que a la derecha son azules y etiquetados por $y = -1$. Las salidas entonces coinciden con la función signo, que es la activación del perceptrón.
Para entrenar al perceptrón se hace uso de la ecuación (2-27). Esto debe hacerse para cada entrada x_i, debido a que se está usando el algoritmo de descenso por gradiente que incluye a todas las entradas. Se esta considerando el caso en que los pesos en la primer iteración son aleatorios, para generarlos aquí se utilizó una distribución uniforme en el rango [-1,1] pues la función de activación fue la Función Signo (2-3).
Ya que se superó el umbral de exactitud dado, el perceptrón está listo para hacer predicciones con una muestra nueva; en este ejemplo la exactitud llegó a 100 % pues en cierta manera es un ejemplo idealizado. Por supuesto que en la práctica se ocuparán más datos de entrenamiento para hacer una mejor predicción, pero el concepto será el mismo. Se podrán tener las entradas que sean, y mientras el problema sea linealmente separable como se mencionó anteriormente, se convergerá a una solución, lo cual está demostrado por el Teorema de Convergencia del Perceptrón [24].

2.7. Perceptrón Multicapa

En la sección 2.3.2 se describió la estructura general de una red neuronal prealimentada y como su estructura es una composición de funciones. El nombre prealimentado viene de feed-forward en inglés, lo cual se refiere a que las múltiples composciciones van estrictamente en el orden $f_1 \to f_2 \to f_3 \to ... \to f_n$ y no hay ciclos que hagan ir de reversa en tal proceso.
Una perceptrón multicapa es una red neuronal prealimentada, con tres o más capas, es decir, con al menos una capa oculta. El término perceptrón multicapa en realidad es una denominación errónea, pues las funciones de activación que utiliza en sus múltiples capas son regresiones logísticas con no linealidades continuas, en lugar del perceptrón que utiliza no linealidades discontinuas [23] (Funcion Heaviside). Sin embargo, en este texto le llamaremos perceptrón multicapa, pues el nombre proviene de la idea de conectar múltiples perceptrones como se mostrará a continuación.

En realidad, el perceptrón multicapa tiene en cada unidad una función de activación no lineal que es diferenciable y tiene una alta conectividad [25]. Esto último se refiere a que cada unidad está interconectada, mediante los parámetros, con las unidades de la capa siguiente y así sucesivamente hasta llegar a la capa final. Con esto ya se tiene casi completa la estructura, solamente falta definir la función de perdida en la capa de salida [11]. Puede verse un ejemplo de la estructura de un perceptrón multicapa en la figura **2-12**.
En general, con el perceptron multicapa se pueden resolver problemas más complejos y no lineales escogiendo un número adecuado de capas ocultas, para saberlo no hay un criterio definido sino que debe hacerse empíricamente o con base en el comportamiento observado en los datos.

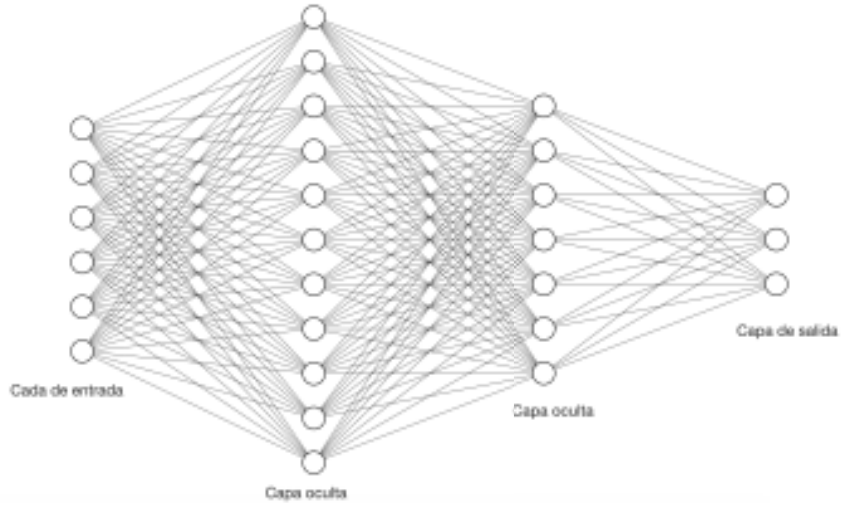


Figura 2-12: Perceptrón Multicapa

2.7.1. Ejemplo práctico del perceptrón multicapa

En la sección 2.6.4 vimos como el perceptrón puede distinguir entre dos clases binarias en el plano bidimensional que son separadas por una recta, es decir, linealmente separables. Para contrastar, ahora se verá qué pasa si consideramos la operación lógica XOR, la cual es resumida en la tabla **2-1**.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2-1: Tabla de verdad de la operación lógica XOR.

En la figura **2-13** puede verse representada la XOR en el plano, donde el color amarillo representa la clase 0 mientras que la morada la 1. El perceptrón no puede predecir esas categorías porque se ocupan

necesariamente dos rectas para discriminar. En cambio, podemos plantear un perceptrón multicapa con una capa oculta y este problema converge a una solución pues esta red neuronal artificial genera las dos rectas necesarias para lograrlo, como se observa en la misma figura.

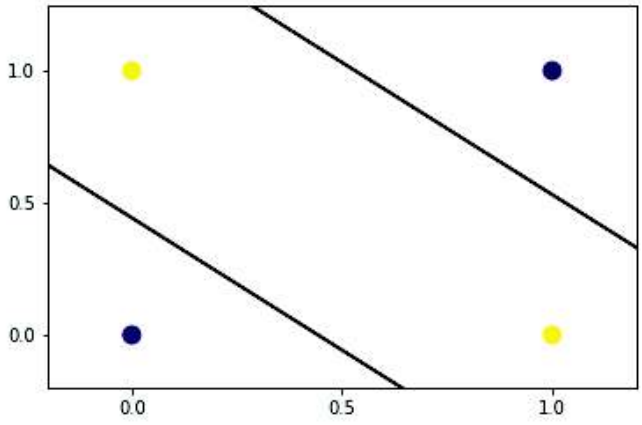


Figura 2-13: Secciones de clasificación de XOR generadas por un perceptrón multicapa con una capa oculta y función de activación sigmoide en la capa de salida.

2.8. Hiperparámetros

En este punto ya puede verse que hay muchos parámetros que ajustar para el entrenamiento de una red neuronal prealimentada. Se conocen como hiperparámetros, debido a que no dependen de los datos, si no que los ajustamos como si fueran "perillas" hasta que la red converge y cumple con una medida de desempeño dada. Se muestran unos ejemplos a continuación.

- **Tamaño de la red:** Pueden elegirse arbitrariamente el número de capas que tenga la RNA, a esto se le conoce como profundidad; mientras que también podemos elegir el numéro de unidades en las capas, lo cual se denomina anchura. Los hiperparámetros son el número de capas y unidades.
- **Función de activación:** Cada función tiene distintas propiedades que aportarán a la RNA. En la sección 2.3.3 se describieron algunas muy utilizadas.
- **Función de Costo:** Como se mencionó en 2.4.1, pueden proponerse arbitrariamente dependiendo del problema de clasificación o regresión que se trate.
- **Tasa de Aprendizaje:** Es crucial elegirlo de forma adecuada para que converga el algoritmo de Descenso por gradiente.
- **Inicialización de los pesos:** Los pesos $(\omega_0, \omega_1, ..., \omega_k)$, pueden inicializarse aleatoriamente para la primera iteración en la RNA. Pueden usarse distintas distribuciones o incluso iniciarse en 0, lo que lo convierte en un hiperparámetro.
- **Normalización del conjunto de datos:** Suelen escalarse los datos para que sean más comparables al momento de procesarse por la RNA. Esto puede ser estandarizar (media 0, varianza 1), o bien mapearlos al intervalo $[0, 1]$ lo que se conoce como normalizar.

En realidad, existen más hiperparámetros que van haciendo más complejo el proceso de aprendizaje, sin embargo, los mencionados anteriormente son suficientes para nuestros propósitos, pues hicieron que el perceptrón multicapa planteado en la sección 4.1 de resultados convergiera.

Capítulo 3

Simulación de procesos en el contexto del experimento CMS del CERN

3.1. Introducción

El CERN es la organización europea para la investigación nuclear (por sus siglas en frances), desde su fundacion se ha dedicado a la investigación de física fundamental, entre sus mas grandes descubrimientos se encuentra el bosón de Higgs (2012) [26]. Dicha partícula vino a complementar el marco teórico del modelo estándar el cual describe las interacciones entre las partículas fundamentales y los diferentes tipos de fuerzas, una representación esquemática de las interacciones se muestra en la figura **3-1**.

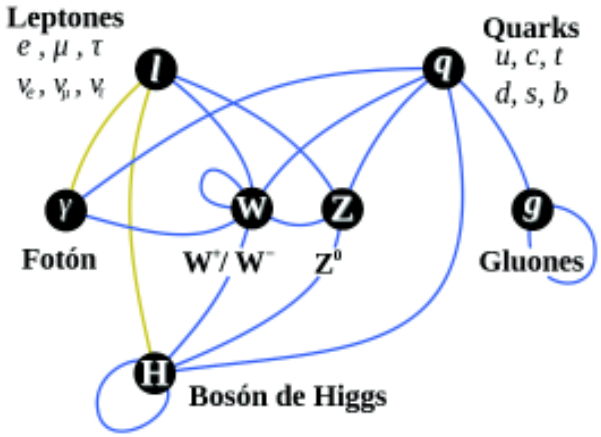


Figura 3-1: Las conexiones representan las interacciones entre las partículas del modelo estandar.

Las fuerzas fundamentales son la fuerza electromagnética, la débil, la fuerte y la gravedad, cada una de estas fuerzas está caracterizada por un rango que es la distancia en la cual se pueden sentir sus efectos y por partículas que actúan como portadoras de dicha fuerza, también conocidas como bosones. Así por ejemplo para la fuerza electromagnética la partícula portadora es el fotón; para la fuerza fuerte que es la que mantiene unidos a los quarks la partícula portadora es el gluon; para la fuerza débil son tres las partículas portadoras, el bosón W^+, W^- y el Z^0; mientras que la partícula portadora de la fuerza de gravedad seria el gravitón, aunque ésta última no ha sido observada experimentalmente y

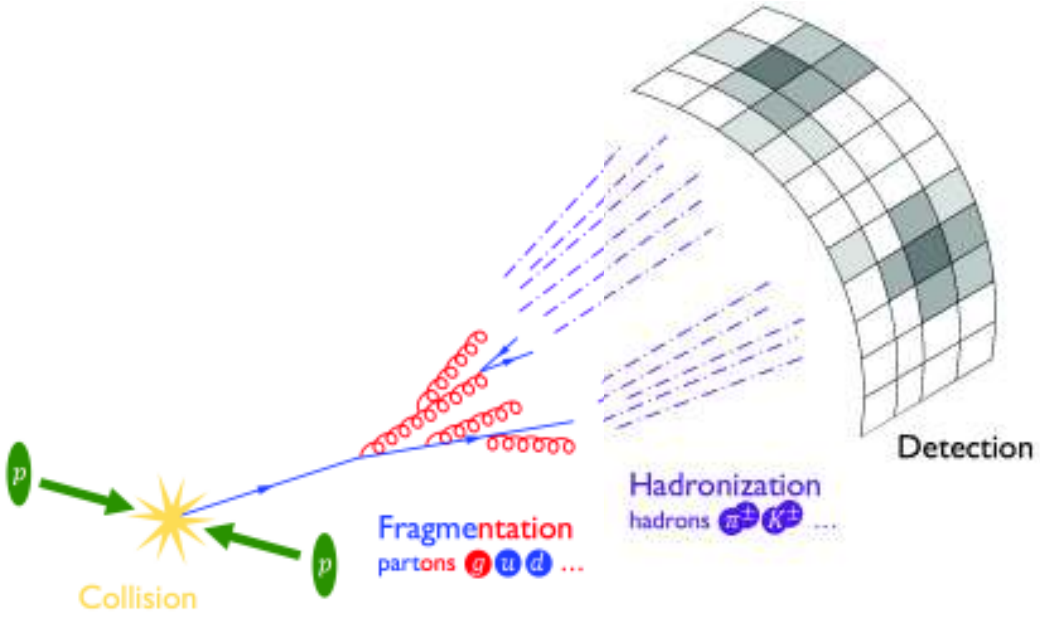


Figura 3-2: Esquema representativo de la formacion de hadrones como producto del proceso de confinamiento de los quarks.

sólo se encuentra descrita dentro de ciertos modelos teóricos. Dicha información se encuentra resumida en la tabla de la tabla **3-1**. Los bosones W^+ y W^- tienen cargas de e^+, e^- respectivamente, donde e se refiere a la unidad de carga elemental; mientras que el bosón Z es una partícula sin carga eléctrica.

Fuerza fundamental	Bosón intermediario	Masa (en $\frac{MeV}{c^2}$)	Rango esperado
Electromagnética	Fotón, γ	0 (<3x10^{-33})	Infinito
Débil	W^+, W^-, Z^0 $[W^{\pm} = \{W^+, W^-\}]$	$W^{\pm} = 80{,}6x10^3$ $Z^0 = 91{,}16x10^3$	~0.001 fm
Fuerte	Gluon, g	No observable como partícula individual, se espera 0.	~1 fm
Gravedad	¿Gravitón?	No comprobado de existir, se espera 0.	¿Infinito?

Tabla 3-1: Fuerzas fundamentales entre partículas elementales.

En particular para esta tesis se estudian partículas que son producidas por el principio de confinamiento de los quarks. Dicho principio indica que los quarks no pueden existir en estado libre sino que después de un tiempo corto se recombinan con otros quarks para formar partículas compuestas llamadas hadrones. En el caso del acelerador de hadrones del CERN se colisionan protones, estos protones están formados de quarks los cuales después de la colisión por una fracción de tiempo se encuentran libres y después se recombinan para formar una lluvia de hadrones, tal proceso de recombinación se representa en la figura **3-2**.

El gran colisionador de Hadrones, LHC (por sus siglas en inglés), es el acelerador de partículas más grande y poderoso del mundo, colisiona haces de protones logrando una energía en el punto de colisión de 13 Tera-electronvolts. Del producto de la colisión se produce una lluvia de partículas, las cuales pueden ser identificadas y sus propiedades reconstruidas (trayectoria, energía, momento, etc), por medio de esta información es posible extraer conclusiones del proceso que las originó y confrontar modelos teóricos que describen dichos procesos en el contexto del modelo estándar, o en su caso de discrepancias con el mismo que apunten a indicios de creación de nuevas partículas y fuerzas. Las

partículas, producto de las colisiones, son estudiadas por medio de grandes complejos experimentales que forman parte del Gran Colisionador de Hadrones, dos de los mas grandes e importantes son los experimentos ATLAS y CMS, dichos experimentos reportaron la detección del bosón de Higgs en el 2012. En esta tesis nos enfocamos en el detector CMS, el cual posee una serie de sistemas de detección capaces de identificar prácticamente a todas las partículas del modelo estándar. El detector CMS se encuentra representado en la Figura **3-3**. Una descripción mas completa del Gran acelerador de Hadrones puede encontrarse en el apéndice A

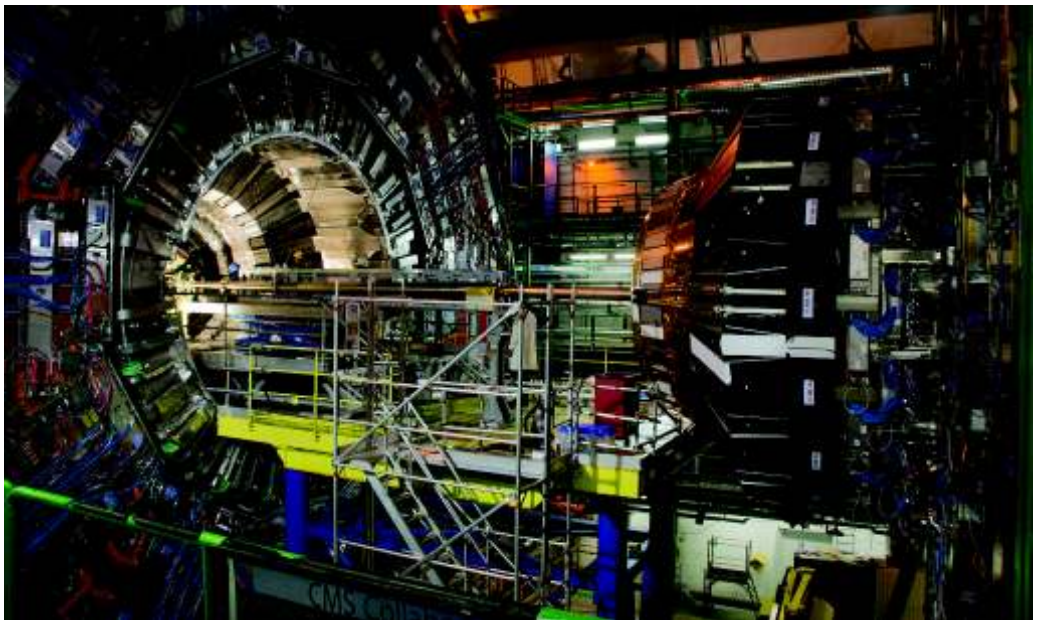


Figura 3-3: Detector CMS del CERN.

Como ya se comentó el detector CMS está acondicionado con una serie de subsistemas de detección cuya composición y posición esta diseñada para poder recolectar la señal de los diferentes tipos de partículas, pudiendo así reconstruir lo que pasó en el momento de la colisión. El detector CMS tiene una simetría cilíndrica, donde los subsistemas pueden representarse como capas en ese volumen. La posición de esas capas depende mucho de la interacción de las partículas con la materia, de manera que los subsistemas más cercanos al punto de colisión están diseñados para identificación de partículas cargadas y aquellas que responden a interacciones electromagnéticas como el caso de los electrones y fotones; mientras que los subsistemas mas alejados del punto de colisión tienen la tarea de identificar partículas que interaccionan débilmente con la materia como el caso de los muones. El esquema que representa los diferentes subsistemas en CMS y el tipo de partícula que identifica se presenta en la figura **3-4**.

3.2. Procesos de Estudio

La simulación nos permite estudiar procesos fundamentales de la física de partículas desde su creación (colisión de protones) hasta la evolución del sistema, detección de partículas y reconstrucción del evento. En este estudio los procesos usados como referencia son la cromodinamica cuántica (QCD) y el boson W prima (W'). QCD forma parte de los procesos descritos por el modelo estándar, es decir ha sido observado experimentalmente y se conocen en gran medida sus propiedades. QCD describe la

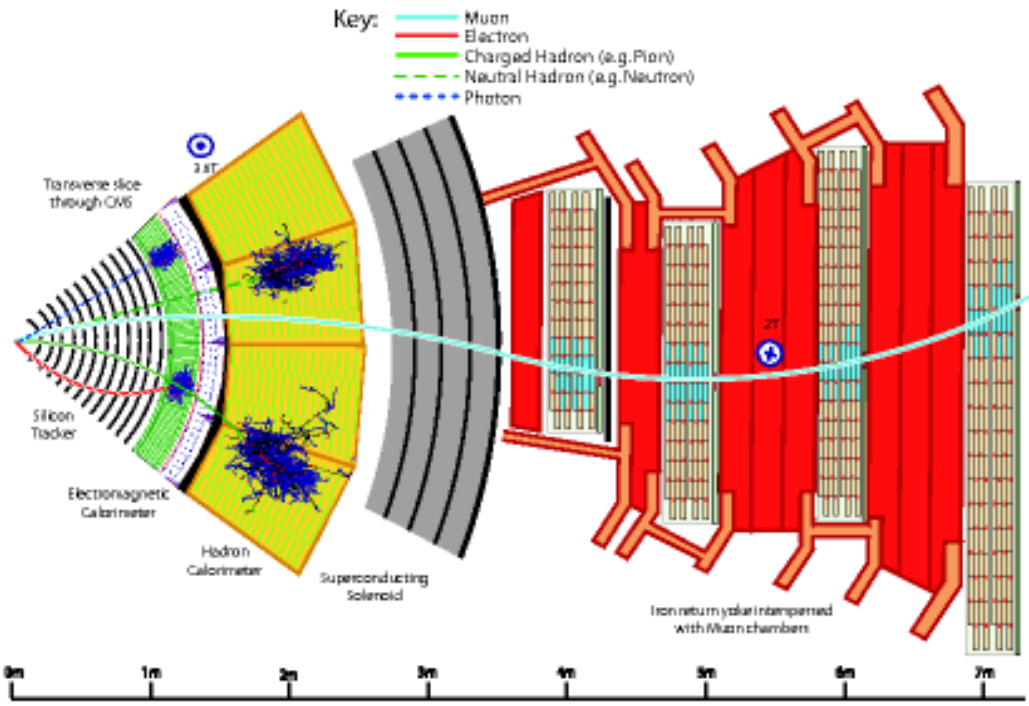


Figura 3-4: Representación del área transversal del detector CMS, que identifica los diferentes subsistemas de detección y el tipo de partículas que es capaz de identificar.

interacción entre quarks por medio del intercambio de gluones. Debido al principio de confinamiento de los quarks por lo cual no pueden existir en forma libre al momento de separarse buscan re-combinarse con otros quarks dando lugar a la creación de hadrones (partículas formadas por dos o tres quarks). Dicha recombinación forma una lluvia de partículas (en su mayoría hadrones) conocida en física de partículas como "jets". Los jets son identificados y sus propiedades son reconstruidas por medio de subsistemas especiales en CMS conocidos como calorímetros, tal como se describe en la sección sección 3.3.

Por otra parte el bosón W' es una partícula hipotética, la cual emerge por interacciones en el espectro electrodébil (por lo que está asociada a la fuerza electrodébil), con propiedades parecidas a las del bosón W del modelo estándar pero con un valor de masa mayor [27], tiene carga positiva y negativa (W'+,W'-) como su compañero del modelo estándar. El W' es el compañero pesado del bosón W del modelo estándar, surge como una extensión de la simetría electrodébil del modelo estándar, añadiendo un grupo de simetría SU(2) relativo al modelo estándar, al agregar dicho grupo quedaría SU(3)xSU(2)xU(1). Dicha extensión es interesante porque es una de las extensiones más sencillas del modelo estándar, de hecho al inicio de operaciones del LHC esta particular fue una de las primeras que se buscó con los primeros datos recolectados. Aunque no se han encontrado indicios de esta partícula bosónica la teoría dice que el valor de masa puede ser aún mayor a lo que los datos han logrado explorar.

El bosón W' se podría detectar mediante su decaimiento a un leptón y un neutrino o un quark top (t) y un quark bottom (b), los dos últimos son detectados en forma de "jets", de ahí que este proceso pueda confundirse como un proceso de QCD, la teoría señala que el valor de masa de este nuevo bosón puede

estar en el rango de Tera-electronvolts. Los procesos de QCD resultan óptimos para el entrenamiento de redes neuronales debido a que la probabilidad de producción (sección eficaz del proceso) es grande, de igual manera el uso de W' ya que es una señal simulada hipotética permite la generación de grandes cantidades de datos. Los diagramas de Feynman de ambos procesos se presentan en la Figura **3-5**.

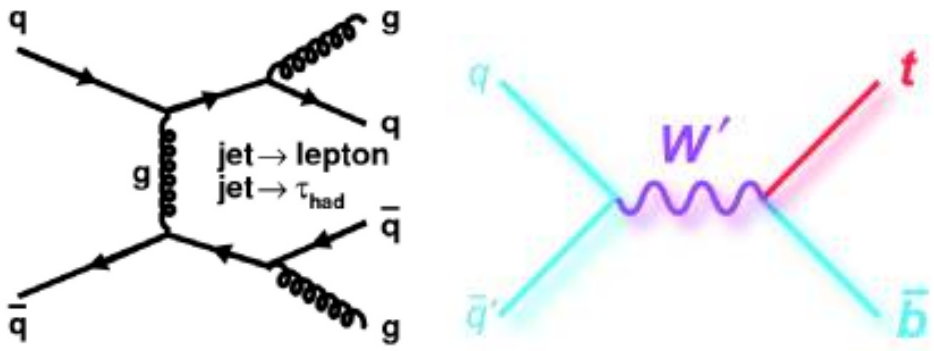


Figura 3-5: Proceso fundamentales QCD(izquierda) y W'(derecha).

3.3. Detección de Jets en CMS

Para la detección de Jets el detector CMS cuenta con sistemas especiales, entre los cuales se encuentran los Calorímetros (Electromagnético y Hadrónico). Los Jets son formados debido al principio de confinamiento de los quarks descrito en la sección 3.2, las partículas constituyentes de dichos jets interactúan con los materiales de los calorímetros y depositan su energía.
El Calorímetro Hadrónico mide la energía de los hadrones, el dispositivo está compuesto por capas de material fluorescente centellador que al interactuar con los hadrones produce un pulso de luz, dicha luz es transportada mediante fibras ópticas a un dispositivo electrónico (readout) donde fotodetectores amplifican la señal. Cuando la cantidad de luz es sumada sobre las diferentes capas que conforman el calorímetro (también llamadas torres), esta cantidad es una representación de la energía de la partícula y dependiendo del patrón se puede concluir la identificación de un jet. Una representación de los calorímetros y la visualización de la reconstrucción de jets se puede observar en la Figura **3-6**.

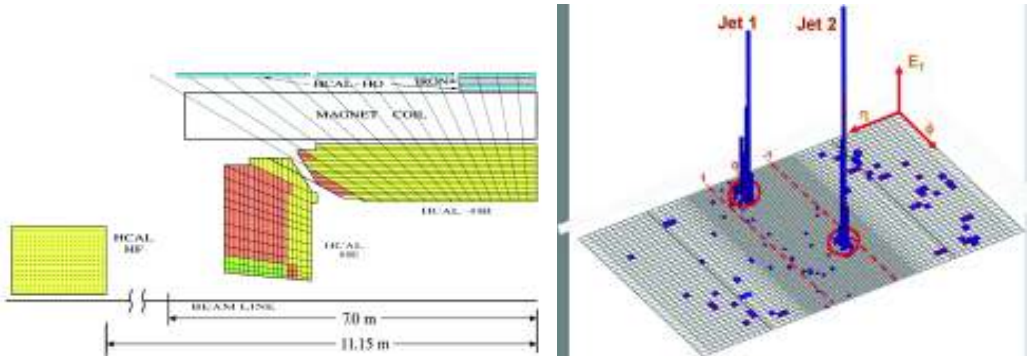


Figura 3-6: Representación de la estructura del Calorímetro de Hadrones (izquierda). Representación de la reconstrucción de jets (derecha).

3.4. Simulación

Los paquetes de simulación en el área de altas energías han evolucionado dramáticamente en los últimos años, el desarrollo de computadoras más potentes y lenguajes de programación avanzados han permitido la optimización de los procesos y la reducción del tiempo de simulación, además de hacer posible la descripción de procesos más complejos. En este estudio los procesos de simulación se realizan usando paquetes propios del área de altas energías, en una secuencia de simulación que se comienza desde la descripción y generación del proceso fundamental, el cual esta relacionado con los diagramas de Feynman, calculo de las probabilidades de decaimientos y el proceso de recombinación de quarks; hasta la simulación de la respuesta del detector y reconstrucción de las propiedades de las partículas.
La simulación incluye el uso en serie de varios programas los cuales están descritos en el diagrama de la figura **3-7**.

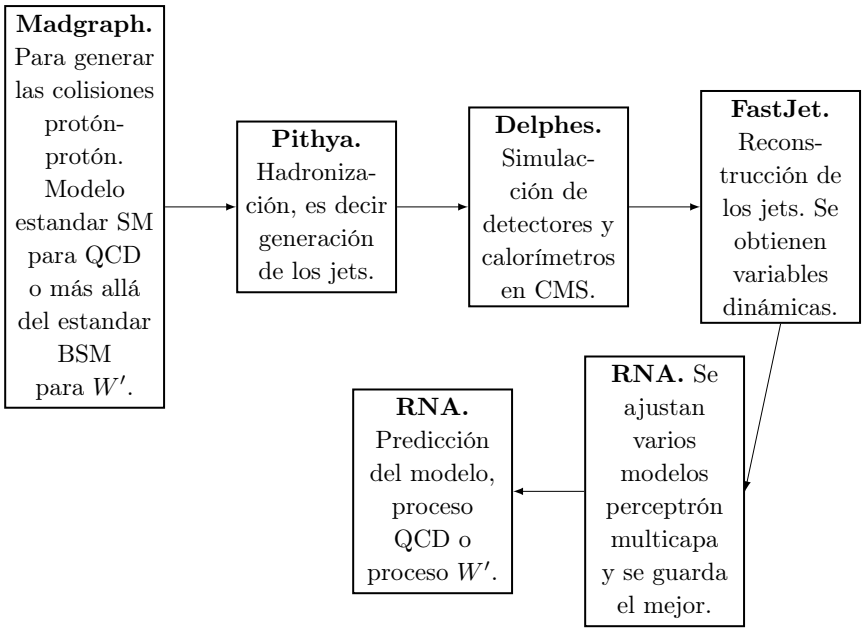


Figura 3-7: Flujo de los datos en el proceso de simulación y en la evaluación del modelo de red neuronal artificial (RNA) para clasificación.

En donde Madgraph [28] se encarga de simular los eventos de las colisiones protón-protón mediante un generador Monte Carlo, dicha simulación cuenta con la descripción más fundamental del proceso, es decir el diagrama de Feynman y amplitudes de decaimientos. En una segunda instancia se sigue con el programa Pythia8 [29] el cual se encarga de la recombinación de quarks, en un proceso que se conoce como hadronización, dicho proceso es fundamental para la evolución del sistema y la formación de los jets.
Finalmente el programa Delphes [30] se encarga de simular la respuesta del detector al paso de las partículas. En particular para la formación de los jets se usa un algoritmo conocido como FastJet, el cual reconstruye las propiedades de los jets a partir de sus constituyentes, más detalles sobre este algoritmos se describe en el apéndice D. La descripción del detector influye de manera relevante en la reconstrucción de las partículas en el caso de este estudio la geometría del detector simulado corresponde a la del CMS.

La estructura de la simulación permite considerar la respuesta de un detector multipropósito (consistente con el detector CMS), obteniendo un sistema de trazado correspondiente al del campo magnético, los calorímetros y sistemas de detección de muones. Es posible extraer variables relacionadas con la reconstrucción de jets permitiendo así un análisis más detallado.

3.5. Selección de eventos

Las muestras simuladas se encuentran organizadas por número de evento, en donde cada evento representa una colisión protón-protón, de igual manera que como se estructuran los datos extraídos del experimento.
Para cada evento se genera un número de partículas, algunas de ellas serán identificadas como "jets". Usualmente se requiere de hacer un preprocesamiento de datos en el cual se busca seleccionar eventos en los cuales los jets reconstruidos tengan una selección inicial para asegurarse su calidad, es decir que no correspondan a eventos con sólo un jet, ya que usualmente procesos de QCD y W prima vienen acompañados con una alta multiplicidad de jets ($>= 2$) además de otras propiedades.
Dicha selección de eventos se compone de los siguientes requisitos:

- Selección de eventos con al menos dos jets.
- Los jets seleccionados deben de estar dentro de una ventana espacial definida por el rango en pseudo-rapidez η de [-4.0,4.0] y ángulo azimutal ϕ de [-π,π].

En donde, la pseudorapidez está ligada a la posición angular de la partículas de interés en el plano x-z (donde z es el eje de la colisión), y el momento transveral es la proyección del momento de la partícula en el plano x-y como se representa en la figura **3-8**, donde también puede verse cómo se calcula matemáticamente. El ángulo azimutal ϕ es el ángulo que hay entre el momento transversal y el eje x. También se toma en cuenta la masa invariante M_{inv} como se indica en la sección de resultados 4.1, dicha magnitud física es la masa de las partículas que es medida desde el sistema de referencia en reposo de la partícula, y es la misma en todos los sistemas de referencia.
Los eventos que sobreviven la selección son almacenados en un archivo de datos en donde cada columna corresponda a una propiedad propiedad de los jets, dichos datos serán usados como variables de entrada para el entrenamiento de la red neuronal artificial.

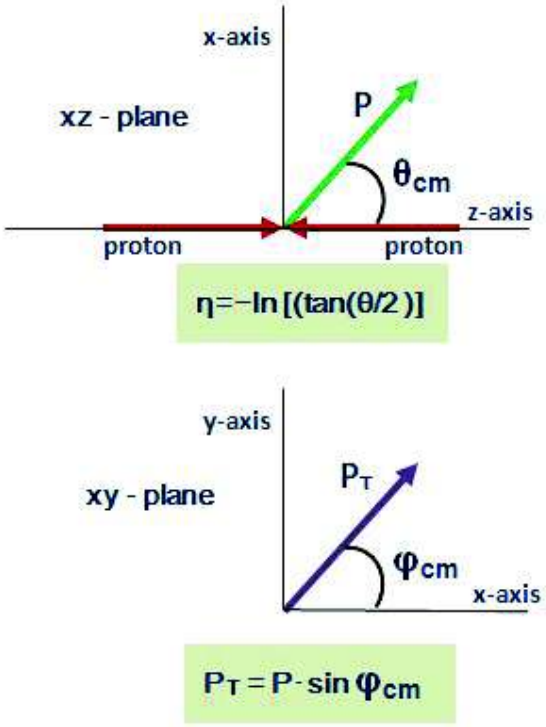


Figura 3-8: Representación gráfica de la pseudo-rapidez η y el momento transversal P_T. Nótese que es esta figura el ángulo azimutal está denotado por φ_{cm}. El ángulo θ_{cm} es el ángulo polar del momento P de la partícula, y es usado para calcular la pseudo-rapidez como puede verse.

Capítulo 4

Implementación de redes neuronales artificiales para la separación de QCD y W'

4.1. Resultados

Los datos extraídos de la simulación se almacenan en archivos en donde cada entrada o fila corresponde a un evento (colisión protón-protón) y cada columna corresponde a una propiedad reconstruida de los jets. Para cada proceso de estudio se creó un archivo el cual contiene un número similar de eventos, específicamente 777 para el proceso W'y 999 del proceso QCD, contando con un total de 1776 eventos en el conjunto de datos. La estructura del archivo puede verse en el apéndice C. Las variables usadas son representativas de la dinámica del sistema y caracterizan la identificación de los jets, esto debido a que se tiene una propiedad intrínseca como la masa invariante, una propiedad física dinámica como el momento transversal, y propiedades espaciales como la pseudo-rapidez y el ángulo azimutal. Las variables de los jets utilizadas entonces son:

- Momento transversal (Jet p_T).
- Pseudo-rapidez (Jet $|\eta|$).
- Ángulo azimutal (Jet ϕ).
- Masa invariante (Jet M_{inv}).

Dichas variables se presentan en la figura **4-1** donde se puede ver claramente la comparación entre los procesos de QCD y W', las variables que muestran una discriminación más grande entre los procesos son el momento transversal y la masa invariante, mientras que las variables de posición como la pseudo-rapidez y el ángulo azimutal reflejan poca diferencia. De cualquier manera aunque visualmente no se puedan distinguir al momento de ser procesados por la red neuronal incluso dichas variables pueden proporcionar información relevante para la clasificación de los procesos.

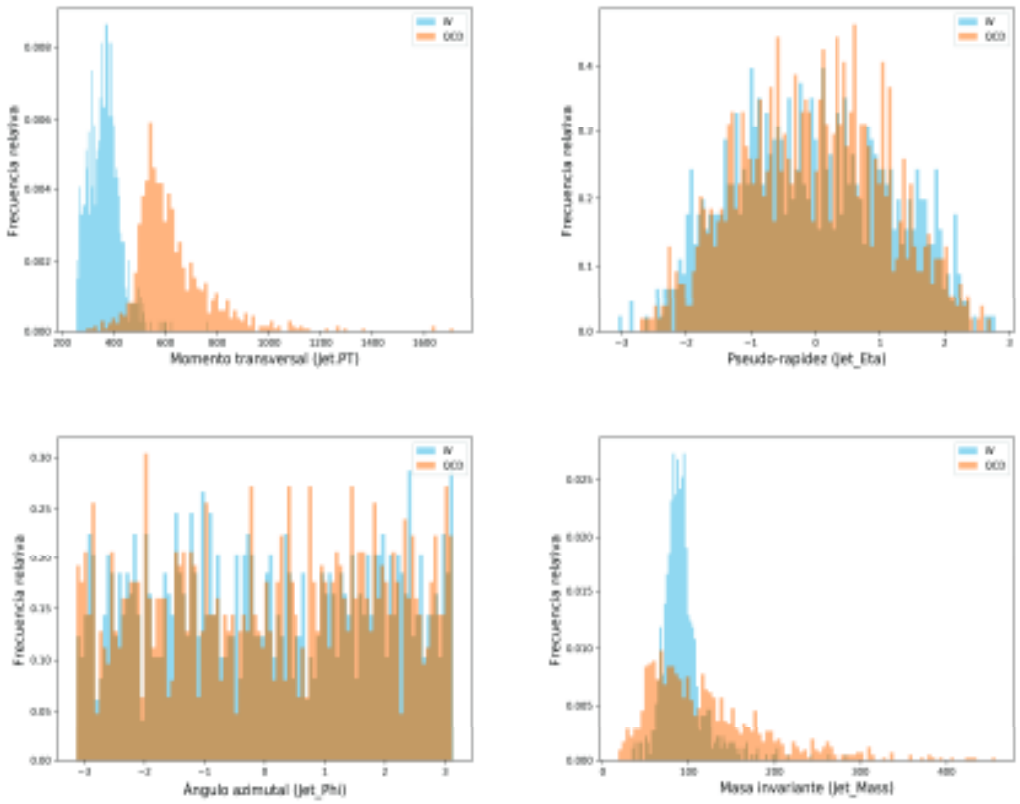


Figura 4-1: Distribución de momento transversal en GeV (arriba, izquierda), Pseudo-rapidez (arriba, derecha), Ángulo Azimutal (abajo, izquierda), Masa invariante en GeV (abajo, derecha)

La estructura de RNA que se utilizó fue un perceptrón multicapa con la capa de entrada usando las cuatro propiedades de los jets, una capa oculta constituida por 8 unidades con función de activación rectificadora (2-4), y una capa de salida con la función de activación sigmoide en dos unidades, las cuales dan la predicción $\hat{y}$. Esta predicción es binaria, y da un número 1 para el proceso W' y 0 para el proceso QCD. En la figura **4-2** puede verse la estructura de la red de manera detallada.

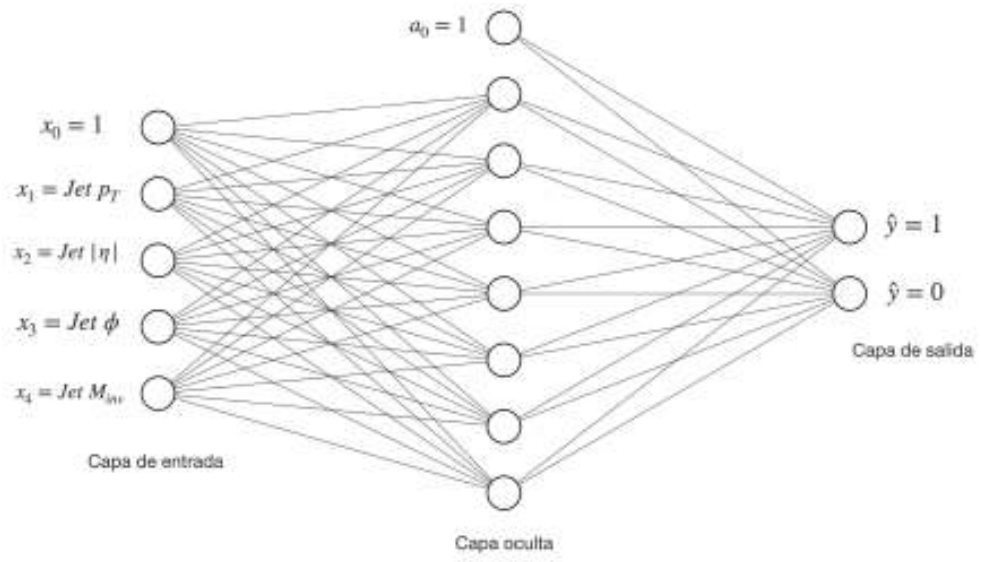


Figura 4-2: Estructura del perceptrón multicapa con una capa oculta y una de salida para clasificación binaria. Los valores x_0 y a_0 corresponden a parámetros de sesgo.

Para entrenar este perceptrón multicapa, se utilizó la función de costo de entropía cruzada para clasificación binaria, definida en la ecuación (2-15), la cual fue minimizada con el algoritmo de descenso por gradiente estocástico con una tasa de aprendizaje $\eta = 0{,}01$. El criterio para detener fue hasta que se tuviera una exactitud (AC) de al menos 0,95, es decir, una tolerancia de error del 5 %. La exactitud final que se obtuvo fue de $AC = 0{,}9522$.
La estructura de esta RNA fue encontrada a base de prueba y error. Se comenzó de lo más sencillo, primeramente se fijó una tasa de aprendizaje de $\eta = 0{,}01$ y se entrenó una red sin capas ocultas y se llegó a una exactitud de $AC = 0{,}5828$. Después se agregó una capa oculta con 2 unidades y funciones de activación rectificadoras, obteniendo $AC = 0{,}5618$, se subió a 5 unidades obteniendo $AC = 0{,}8947$, mientras que si se aumentaba a 10 unidades se tenía $AC = 0{,}9424$ menor al obtenido con 8 neuronas. El procedimiento se resume en la tabla **4-1**. Todas las funciones de activación usadas en capas ocultas fueron rectificadoras. Nótese que si se agregaba otra capa oculta al modelo que tenia una capa oculta con 8 unidades, la exactitud bajaba o quedaba igual, se opta por quedarnos con el modelo con una sola capa pues es más práctico y consume menos recursos computacionales, además de que supera el umbral de $AC > 0{,}95$. Durante todas las pruebas, no fue necesario cambiar la tasa de aprendizaje, con $\eta = 0{,}01$ se cumplió la tolerancia de error.

Estructura de la RNA	Exactitud (AC)
Sin capas ocultas	0.5829
Una capa oculta con 2 unidades	0.5618
Dos capas ocultas con 2 unidades cada una	0.5618
Una capa oculta con 5 unidades	0.8947
Una capa oculta con 8 unidades	0.9522
Una capa oculta con 10 unidades	0.9424
Una capa oculta con 12 unidades	0.9382
Una capa oculta 8 unidades, una siguiente capa oculta 8 unidades	0.9508
Una capa oculta 8 unidades, una siguiente capa oculta 10 unidades	0.9522
Una capa oculta 8 unidades, una siguiente capa oculta 12 unidades	0.9522

Tabla 4-1: Exactitud del modelo de RNA respecto a su estructura.

Todos los modelos se corrieron durante 100 épocas, es decir, 100 ciclos de los que cada uno indica que el gradiente descendiente estocástico iteró a traves de los m elementos del conjunto de datos. Se trabajó con una proporción del conjunto de datos entrenamiento-validación-prueba del 60 %/20 %/20 %, es decir 1065 eventos de entrenamiento, 355 de validación y 356 de prueba. Los resultados de la RNA final se resumen en la evolución de la función de costo de la figura **4-3** y la exactitud en la figura **4-4**. Puede observarse que en la función de costo respecto a los datos de validación queda por debajo de la que está respecto a los datos de entrenamiento, por lo que no hay sobreajuste, lo que es bueno e indica que el modelo puede generalizar a datos externos a los usados para entrenar.

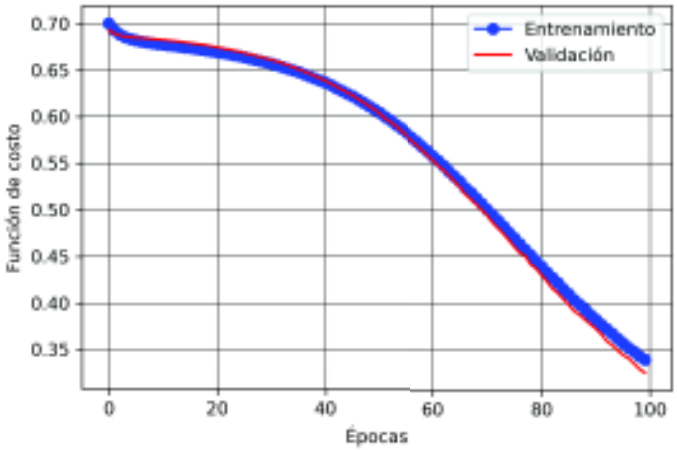


Figura 4-3: Función de costo de la RNA final a lo largo de las épocas de iteración respecto al conjunto de datos de entrenamiento y el de validación.

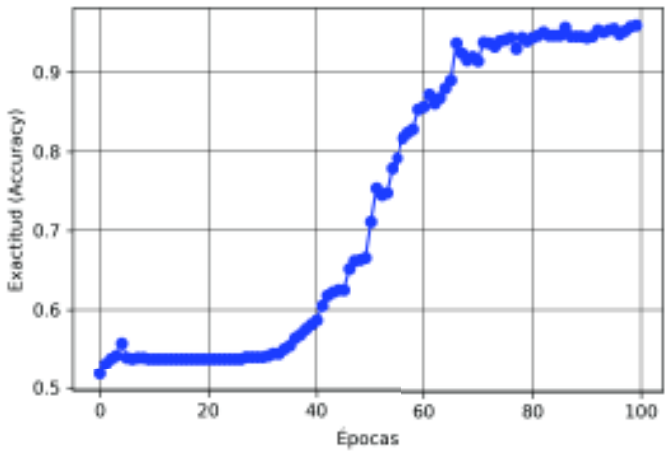


Figura 4-4: Exactitud de la RNA final a lo largo de las épocas de iteración respecto al conjunto de datos de validación. Exactitud final de 0.9522.

Se agregan también funciones de costo de los modelos probados respecto del conjunto de datos de entrenamiento y de validación, esto para observar el comportamiento que tienen, se pueden ver en las figuras **4-5** y **4-6**. Nótese que el costo en los datos de validación de las figuras (arriba, derecha), (abajo, izquierda) de **4-5** tienen un punto de inflexión en el que empiezan a incrementar superando por mucho al costo en los datos de entrenamiento, es decir, los modelos tienen sobreajuste, no generalizan bien en los datos externos al entrenamiento.

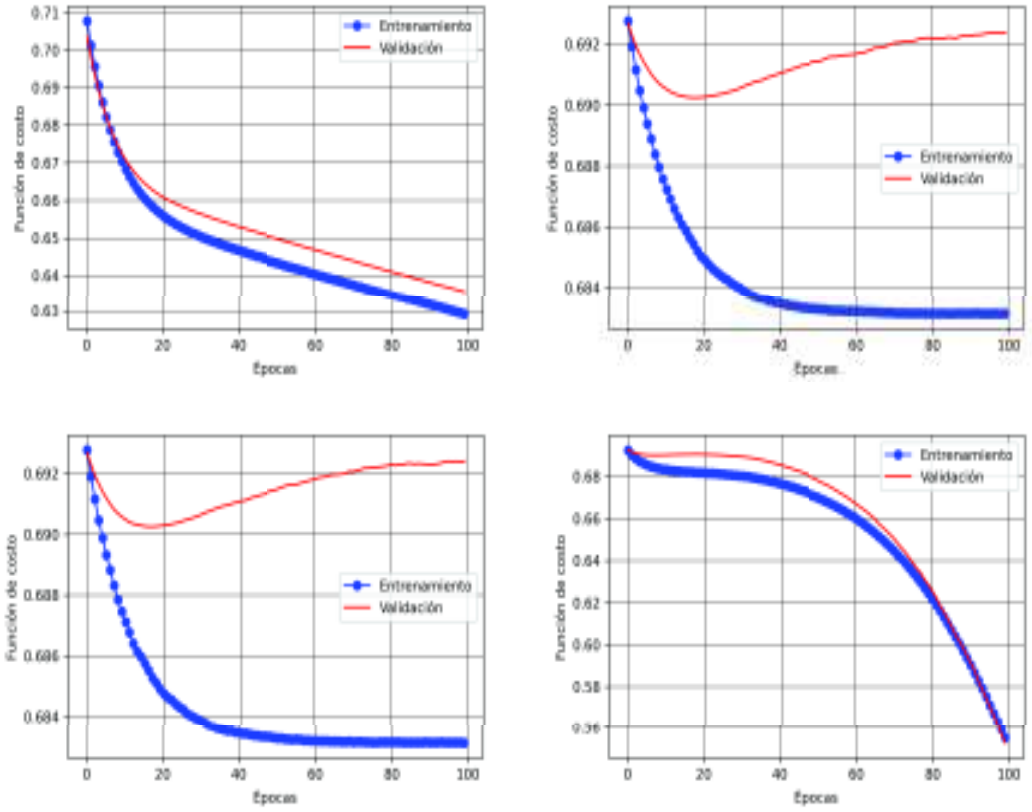


Figura 4-5: **Funciones de costo**. Sin capas ocultas (arriba, izquierda), una capa oculta dos unidades (arriba, derecha), dos capas ocultas con dos unidades cada una (abajo, izquierda), una capa oculta con 5 unidades (abajo, derecha).

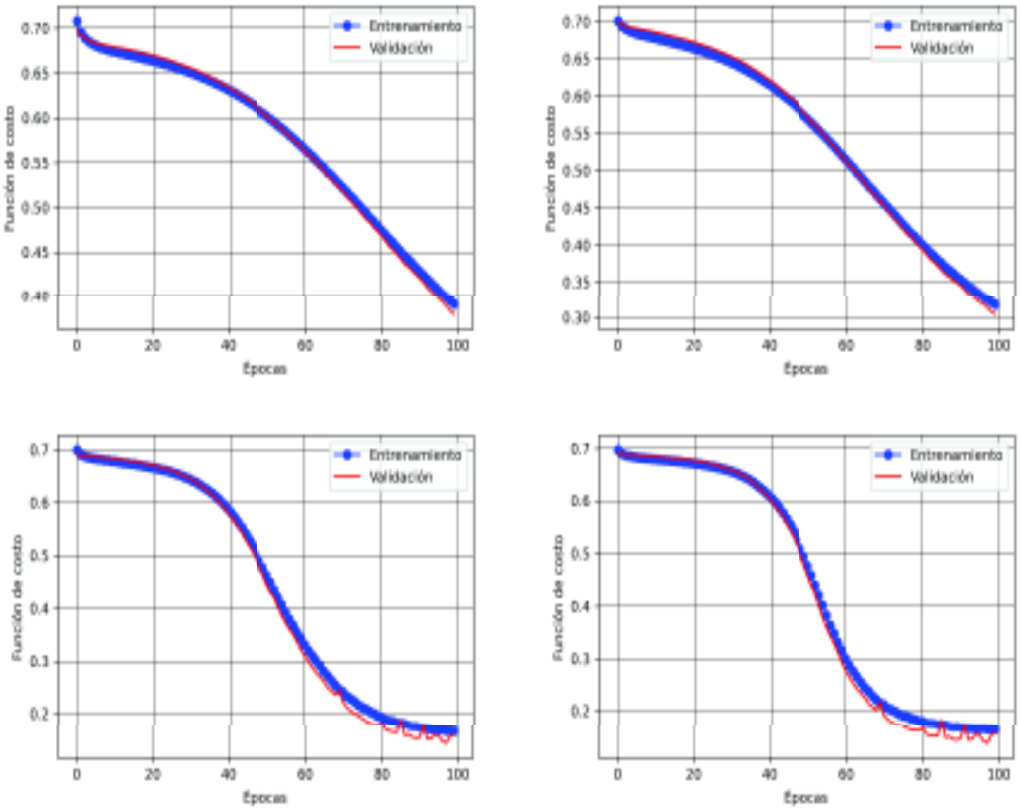


Figura 4-6: **Funciones de costo**. Una capa oculta con 10 unidades (arriba, izquierda), una capa oculta con 12 unidades (arriba, derecha), una capa oculta con 8 unidades y una siguiente capa oculta con 8 unidades (abajo, izquierda), una capa oculta con 8 unidades y una siguiente capa oculta con 12 unidades (abajo, derecha).

Por otra parte, para la ejecución computacional de lo anterior, se utilizó el lenguaje de programación Python, con la librería Keras que corre con backend de Tensorflow [31]. El código puede consultarse en el siguiente enlace https://github.com/hugojira/Tesis_licenciatura.

Capítulo 5

Conclusiones

En este trabajo se implementó un modelo de redes neuronales artificiales para lograr la clasificación de un conjunto de datos. La arquitectura de red neuronal estudiada corresponde a la que se conoce como red neuronal multicapas (perceptrón multicapa, específicamente), la cual se caracteriza por poder clasificar datos en categorías aun cuando las variables que caracterizan a esos datos tengan una correlación no lineal. El objeto de estudio para implementar dicha red neuronal fueron los datos producidos por simulación de dos procesos en altas energías, uno de ellos ya medido y que forma parte del modelo estándar de las partículas (QCD) y otro de ellos un modelo hipotético que predice la creación de una nueva partícula (W'). Ambos procesos poseen características similares ya que pueden ser identificados por la producción de particulas jets las cuales se producen por el principio de confinamiento de los quarks y son detectados por los calorímetros. La intención fue crear un filtro para aislar el proceso de interés y que sirva para que en estudios posteriores se pueden analizar sus propiedades sin la interferencia de otros procesos. Se usaron un total de cuatro variables para la discriminación de los procesos y por medio del entrenamiento de la red neuronal se logro una eficiencia de clasificación $>95\,\%$ con una reducción del error de 0.5 a 0.05, lo que garantiza el correcto entrenamiento de la red neuronal y el modelo para discriminar estos dos procesos. El presente trabajo en una muestra del trabajo interdisciplinario y la combinación de áreas como es la física fundamental de partículas y el aprendizaje automatizado.

Apéndice A

Gran Acelerador de Hadrones

Figura A-1: Vista aérea de la localización del Gran colisionador de Hadrones, el cual cubre parte de la region entre Francia y Suiza.

El Gran Colisionador de Hadrones es el acelerador de partículas más grande del mundo, con un túnel de aproximadamente 27 kilómetros de circunferencia. Se aceleran protones hasta alcanzar una velocidad cercana a la de la luz (0.999999990c). La energía de las partículas es medida en electronvolts. Un electronvolt es la energía acumulada por un electrón que es acelerado por medio de un campo eléctrico de 1 volt. En el gran colisionador de hadrones los protones adquieren unas energías de 6.5 millones de millones de electronvolts, o también expresado como 6.5 Tera-electronvolts.
El laboratorio CERN cuenta con diferentes complejos de aceleración de partículas. Estos aceleradores proveen las partículas a los diferentes tipos de experimentos o son usados como inyectores de partículas a aceleradores más grandes, es decir las partículas empiezan su proceso de aceleración en complejos como sincrotrón de protones (PS) o super sincrotón de protones (SPS) antes de ser inyectados al túnel principal, tal como se muestra en la figura **A-2**.
Para crear los protones, se inyecta gas de hidrógeno en un cilindro metálico el cual por medio de un campo eléctrico descompone el gas en sus constituyentes (protones y electrones). Dicho proceso arroja un porcentaje de 70 % de protones. Después de este proceso las partículas son aceleradas con una velocidad de 1.4 % la velocidad de la luz, son enviadas a un cuadrupolo de radio frecuencia (QRF) un componente que además de acelerar las partículas focaliza las partículas del haz. Del cuadrupolo son enviadas a un acelerador lineal (LINAC2) antes de entrar a los aceleradores circulares.

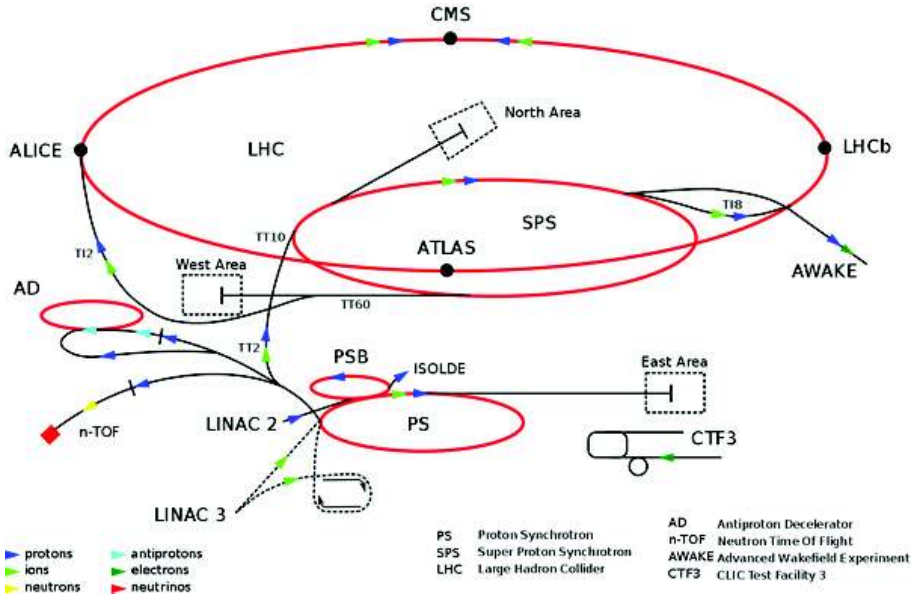


Figura A-2: Complejo de aceleradores que forman parte del Gran Acelerador de Hadrones del CERN

La cantidad de $1{,}34x10^{20}$ protones fueron acelerados en el complejo en 2016. Por exagerado que parezca la cantidad es en realidad una minúscula cantidad de materia, aproximadamente corresponde al número de protones en un grano de arena. Los protones son tan pequeños que esta cantidad es suficiente para abastecer a todos los complejos del CERN. El Gran Acelerador de Hadrones usa solo una porción de estos protones, menos del 0.1 %.

Apéndice B

Propagación Hacia Atrás (backpropagation)

Los valores de pre-activación que recibe cada unidad vienen dados por la ecuación (2-11) para transformarse en valores de activación mediante una función de activación, tal como en la (2-8). Estas operaciones pueden anidarse de una unidad i hacia una unidad j en la capa inmediatamente posterior. Teniendo así

$$z_j = \sum_i \omega_{ji} a_i \tag{B-1}$$

donde a_i indica el valor de activación que pasa una unidad que está conectada a una unidad j en una capa $l+1$. Mientras que ω_{ji} es el peso asociado a esa conexión. Nótese que la sumatoria está implicita y corre en todos los índices i dependiendo de cuántas unidades haya en esa capa.
Posteriormente, se evalua z_j en una función de activación, teniendo

$$a_j = \varphi(z_j) \tag{B-2}$$

Evaluando sucesivamente las ecuaciones (B-1) y (B-2) a través de las capas de la red neuronal prealimentada se obtiene precisamente la composición de funciones que se menciona en la sección 2.3.2. Este proceso se conoce como propagación hacia adelante (forward propagation).
Primeramente, enfoquémonos en el algoritmo del gradiente descendente estocástico, de la ecuación (2-21). Se desea calcular $\partial J_d/\partial \omega_{ji}$ con d indicando el índice de un vector dado del conjunto de datos. Ahora, como la función de pérdida J depende de ω_{ji} mediante el valor z_j, como puede verse en (B-1), podemos aplicar la regla de la cadena

$$\frac{\partial J_d}{\partial \omega_{ji}} = \frac{\partial J_d}{\partial z_j}\frac{\partial z_j}{\partial \omega_{ji}} \tag{B-3}$$

Denotemos

$$\delta_j := \frac{\partial J_d}{\partial z_j} \tag{B-4}$$

donde a los δ_j se les suele llamar errores. En lo que respecta al término $\partial z_j/\partial \omega_{ji}$ podemos obtenerlo a partir de (B-1), así

$$\frac{\partial z_j}{\partial \omega_{ji}} = a_i \tag{B-5}$$

Sustituimos (B-4) y (B-5) en (B-3) para obtener

$$\frac{\partial J_d}{\partial \omega_{ji}} = \delta_j a_i \tag{B-6}$$

La ecuación (B-6) indica que para obtener la derivada buscada de la función de pérdida, multipliquemos el valor de δ en la salida de la unidad por el valor de a la entrada de la unidad que corresponden al peso ω_{ji} en el que se esté haciendo el cálculo. De esta manera, se irán obteniendo las δ para todas las capas ocultas y la de salida de la RNA prealimentada. Para hacerlo se aplica la regla de la cadena para derivadas parciales, entonces

$$\delta_j := \frac{\partial J_d}{\partial z_j} = \sum_k \frac{\partial J_d}{\partial z_k} \frac{\partial z_k}{\partial z_j} \tag{B-7}$$

donde k corre sobre todas las unidades a las que j está conectada. Esto quiere decir que las variaciones que z_j provoca en J_d son a través de las variaciones de z_k, como una composición de funciones, es por eso que la regla de la cadena es la parte fundamental del método de propagación hacía atrás. Nótese que las unidades k pueden ser de salida u ocultas, dependiendo de dónde estemos posicionados en el cálculo.
De la ecuación (B-1) y (B-2), haciendo el ajuste de índices se tiene

$$z_k = \sum_j \omega_{kj} \varphi(z_j) \tag{B-8}$$

por lo que

$$\frac{\partial z_k}{\partial z_j} = \sum_j \omega_{kj} \frac{d}{dz_j} \varphi(z_j) = \sum_j \omega_{kj} \varphi'(z_j) \tag{B-9}$$

Sustituyendo lo anterior en (B-7), y siguiendo la notación para δ de (B-4), obtenemos la fórmula para realizar el método de propagación hacia atrás

$$\delta_j = \varphi'(z_j) \sum_k \omega_{kj} \delta_k \tag{B-10}$$

De esta manera, el valor para la δ en cualquier capa oculta puede obtenerse aplicando (B-10) de manera recursiva desde la capa de salida. Es decir, estás propagando el error hacia atras por la RNA, lo que le da el nombre al método. Posteriormente se sustituye en (B-6) para obtener la correspondiente derivada. Así, las fórmulas que rigen las derivadas de la función de costo respecto a los pesos son (B-10) y (B-6).
Debido a que se tienen que estar derivando las funciones de activación para obtener $\varphi'(z_j)$, es muy importante que sean diferenciables, de lo contrario no podríamos aplicar el algoritmo de gradiente descendiente pues los gradientes no convergerían a un valor finito. Es por esto que normalmente todas las funciones de activación son continuas y suaves. Nótese además que se está asumiendo que las unidades en una capa dada tienen las mismas funciones de activación.
Por último, lo anterior se dedujo para el método de gradiente descendente estocástico donde se va utilizando un vector del conjunto de datos a la vez para las derivadas, sin embargo, es sencillo generalizarlo para el caso de gradiente descendente. Para esto suman los errores de cada vector y se tiene

$$\frac{\partial J}{\partial \omega_{ji}} = \sum_d \frac{\partial J_d}{\partial \omega_{ji}} \tag{B-11}$$

Apéndice C

Variables usadas para el entrenamiento de la red neuronal

Como se explicó en la sección 4.1, se usaron cuatro variables y una etiqueta, que puede ser 1 si es el proceso W' y 0 para el proceso QCD, lo que puede verse en label. Es un archivo con extensión CSV y se presenta una captura de pantalla en la figura **C-1**, nótese que valores numéricos están como surgen en las simulaciones, después se normalizaron para el entrenamiento del perceptrón multicapa.

B	C	D	E	F	G
Event.Num	Jet.PT	Jet.Eta	Jet.Phi	Jet.Mass	label
487	497.02038	-1.392883	-0.298193	59.586547	1
461	526.76794	0.3908897	-1.959857	116.63297	1
700	504.5437	2.3515253	1.7303993	47.104736	1
239	348.69229	1.501484	-0.089314	97.709701	0
635	300.23922	-0.053857	-2.870188	105.97162	0
462	489.4635	-0.039591	2.200592	91.771667	0
643	285.07144	-2.828426	-2.085327	50.198436	0
617	613.85986	-1.362617	-0.02588	59.664894	1
356	306.37472	0.8649973	-1.379027	92.144447	0
315	607.10174	0.1500193	0.1951771	49.687629	1
421	588.92474	0.8541994	1.5114692	309.80841	1
289	432.53216	0.9164223	0.3028179	79.285621	0
68	806.36669	0.3463741	0.5650945	231.29548	1
158	695.86206	-2.157342	-0.109315	49.14532	1
116	388.53073	0.4670895	2.6701188	91.77494	0
247	384.69595	0.714448	2.8861906	110.71763	0
523	395.74722	-1.990384	2.2997634	105.50721	0
231	369.31143	-1.711754	0.678564	78.177749	0
588	382.70419	1.7994821	0.9594672	85.381347	0

Figura C-1: Estructura del archivo csv usado para el entrenamiento del modelo.

Apéndice D

Algoritmo FastJet para la reconstruccion de Jets

El algoritmos FastJet es el paquete encargado de reconstruccion de los jets, usa varios algortimos los cuales se encargan de agrupar (clustering) los constituyentes de los jets (hadrones) y derivar sus propiedades. El agrupamiento generalmente se hace considerando la proximidad entre los constituyentes. La lista de algoritmos que constituyen a fastjet son:

- longitudinal invariant k_t
- (inclusive) Cambridge/Aachen
- anti-k_t
- gen-k_t

Los cuales están disponibles dependiendo de la versión de FastJet. Aunque cada uno con sus peculiaridades estos algoritmos se basan en la búsqueda de cúmulos (clusters) de energía que pudieran corresponder a aquellos producidos por las partículas constituyentes de los Jets. La refinación en cuanto a los algoritmos de búsqueda y precisión puede minimizar la posible contribución de ruido de partículas que no correspondan a jets pero que hayan depositado su energía en los calorímetros.

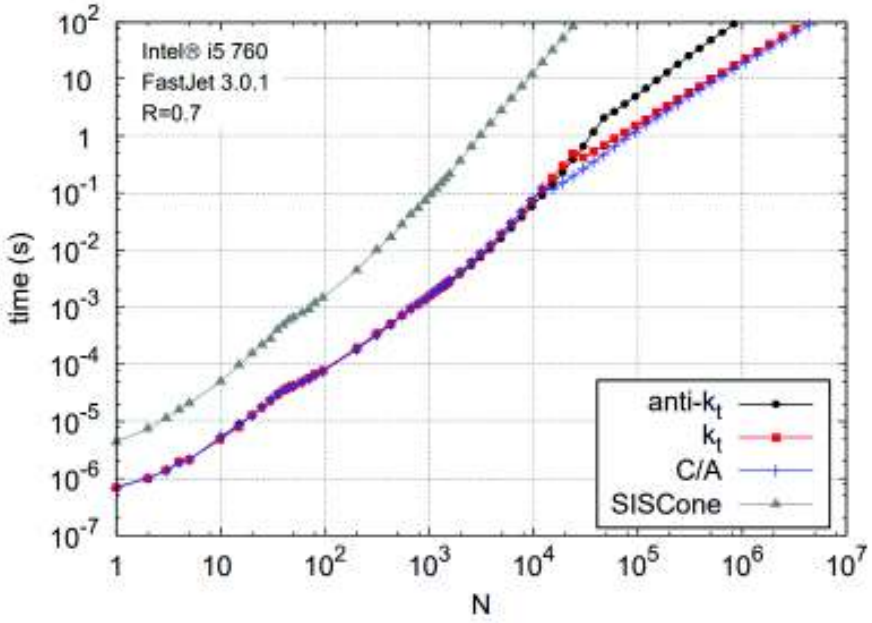


Figura D-1: Tiempo de clustering de los diferentes algoritmos usados en FastJet.

Bibliografía

[1] Dan Guest, Kyle Cranmer, and Daniel Whiteson. Deep Learning and Its Application to LHC Physics. *Annual Review of Nuclear and Particle Science*, 68(1):161–181, 2018.

[2] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1), Jul 2014.

[3] Kim Albertsson and et all. Piero Altoe. Machine learning in high energy physics community white paper, 2018.

[4] K. Lasocha, E. Richter-Was, D. Tracz, Z. Was, and P. Winkowska. Machine learning classification: Case of higgs boson cpstate in h decay at the lhc. *Physical Review D*, 100(11), Dec 2019.

[5] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[6] Feng Hsiung Hsu. IBM's Deep Blue chess grandmaster chips. *IEEE Micro*, 19(2):70–81, 1999.

[7] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc., 2019.

[8] Robert G. Bartle and Donald R. Sherbert. *Introduction to Real Analysis*. Wiley, 1999.

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

[10] Jianli Feng and Shengnan Lu. Performance Analysis of Various Activation Functions in Artificial Neural Networks. *Journal of Physics: Conference Series*, 1237(2), 2019.

[11] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer, 2018.

[12] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Springer, 1996.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. Springer, New York, NY, 2009.

[14] Stinchcombe and White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *International 1989 Joint Conference on Neural Networks*, pages 613–617 vol.1, 1989.

[15] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc., 2017.

[16] Michael Spivak. *Cálculo infinitesimal*. Reverté Ediciones, 1997.

[17] Dimitri P. Bertsekas. *Convex Optimization Algorithms*. 2015.

[18] Christopher Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1996.

[19] Jerrold Marsden and Anthony Tromba. *Cálculo vectorial*. Addison-Wesley Iberoamericana, S.A., 1991.

[20] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC, 2009.

[21] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 1943.

[22] Tara H. Abraham. Physiological circuits: The intellectual origins of the Mcculloch-Pitts neural networks. *Journal of the History of the Behavioral Sciences*, 38(1):3–25, 2002.

[23] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York.

[24] Charlie Murphy, Patrick Gray, and Gordon Stewart. Verified perceptron convergence theorem. pages 43–50, 06 2017.

[25] Simon S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 2009.

[26] Serguei Chatrchyan et al. Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC. *Phys. Lett. B*, 716:30–61, 2012.

[27] A. M. Sirunyan, A. Tumasyan, W. Adam, F. Ambrogi, E. Asilar, T. Bergauer, J. Brandstetter, E. Brondolin, M. Dragicevic, and et al. Searches for w bosons decaying to a top quark and a bottom quark in proton-proton collisions at 13 tev. *Journal of High Energy Physics*, 2017(8), Aug 2017.

[28] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.-S. Shao, T. Stelzer, P. Torrielli, and M. Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *Journal of High Energy Physics*, 2014(7), Jul 2014.

[29] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O. Rasmussen, and Peter Z. Skands. An introduction to pythia 8.2. *Computer Physics Communications*, 191:159–177, Jun 2015.

[30] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi. Delphes 3: a modular framework for fast simulation of a generic collider experiment. *Journal of High Energy Physics*, 2014(2), Feb 2014.

[31] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.