UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

Métodos de optimización de la gradiente de descenso en una red neuronal convolucional

SEMINARIO DE TESIS 1

Autor: Víctor Jesús Sotelo Chico

Asesor: Víctor Melchor Espinoza

Junio, 2018

Resumen

En las últimas décadas el campo de la inteligencia artificial se ha desarrollado rápidamente. Temas como el reconocimiento de imágenes han sido estudiado por mucho tiempo. Actualmente este campo requiere realizar un gran números de cálculos para entrenar redes neuronales que sean capaces de distinguir y clasificar distintos objetos contenidos en imágenes. Incluso este proceso puede tardar más dependiendo del tamaño del dataset. Por lo cual surge la necesidad de encontrar métodos que permitan acelerar el proceso de entrenamiento de las redes neuronales.

Por tal motivo presente seminario buscar lograr un mayor entendimiento de métodos para acelerar el proceso de entrenamiento de una red neuronal convolucional, basándonos en la teoría de redes neuronales y usando como herramienta la librería tensorflow, esta nos permite un uso controlado de las métodos de optimización.

KEYWORDS: Dataset, Métodos Adaptativos, CNN, SGD, Optimizadores.

Índice general

Índice de figuras

Índice de Acrónimos

k-nn	k- nearest neighbors
SVM	Super Vector Machine
SVC	Super Vector Regression
SVR	Super Vector Classification
SGD	Stochastic gradient descent
DNN	Deep Neural Network
CNN	Convolutional Neural Network
ETC	Etcétera

Agradecimientos

Agradezco a mis padres por todo el apoyo incondicional durante todos estos años de estudio, a mis compañeros de clase por el apoyo brindado durante el tiempo de estudio y a mi asesor por ayudarme en este seminario.

Capítulo 1

Introducción

En el campo de la inteligencia artificial las redes neuronales profundas tienen un papel muy importante, debido a que estas son el camino para que las computadoras realicen tareas que nuestros cerebros realizan de manera natural, tareas como el reconocimiento de voz, imágenes y patrones. En la actualidad empresas importantes utilizan las redes neuronales profundas, un ejemplo de esto es Google con el reconocimiento de voz e imágenes. Una característica de las redes neuronales profundas es que están compuestas por una gran cantidad de capas, lo cual dificulta el entrenamiento en computadoras que solo usan el CPU. Una manera de resolver este problema es mediante el uso de las GPU's debido que las tareas de entrenamiento son paralelizables. Además, se pueden utilizar GPU'S para acelerar el proceso de entrenamientos de nuestra red neuronal profunda. Por otro lado se necesitan métodos de optimización que junto a la fortaleza de las GPU's nos permitan obtener un mejor rendimiento.

1.1. Motivación

La inteligencia artificial constituye una base muy importante en el campo de la computación, mezcla un conjunto de disciplinas como la estadística y ciencia de la computación con el objetivo de construir modelos que puedan permitir a las computadoras realizar tareas que hace algunos años hubiesen sido considerada imposibles.

Actualmente las computadoras son capaces de reconocer objetos y clasificarlo. Esto ha permitido que la industria de la robótica se desarrolle de manera

acelerada en las últimas décadas. Además hoy en día existen muchas herramientas que nos permiten desarrollar este tema y profundizarlo, pero a medida que aumenta la complejidad del problema, el costo computacional se incrementa, lo cual se convierte en un problema importante.

Una de la soluciones que surgió fue el uso de la GPU's para acelerar procesos como el entrenamiento de una red neuronal con muchas capas ocultas, las GPU's representan una solución muy eficaz debido a que en el campo de la inteligencia artificial existen muchas tareas que son paralelizables.

Actualmente el mercado de GPU's evoluciona muy rápido debido a su gran demanda en la industria de los videojuegos, este mercado se encuentra dominado por NVIDIA y AMD, esta competencia y la alta demanda permite que las GPU's tengan mejor rendimiento lo cual puede ser usado para obtener mejores resultados en el campo del Aprendizaje Automático.

Por otro lado, la optimización no solo se basa en el uso de hardwares más potentes, sino también depende de la elección de métodos adecuados para nuestros modelos, esta elección dependerá mucho del problema a tratar. Uno de los métodos más usado en el campo del Aprendizaje Automático es la gradiente de descenso estocástica pero este métodos por sí solo no es muy óptimo.

Actualmente existe la problemática de hallar métodos más eficientes de optimización que obtengan un mejor rendimiento, el presente seminario se centra en la búsqueda y comparación de estos métodos con el fin de encontrar aquellos que sean más rápidos y eficientes. Además, adquirir el conocimiento y entender cómo es que estos funcionan.

1.2. Objetivos

El objetivo de este seminario es el de mostrar las ventajas del uso de distintos métodos de optimización para acelerar el entrenamiento de una red neuronal convolucional en una tarea de clasificación.

Específicamente, los objetivos de este trabajo con respecto al sistema son:

- Entender el funcionamiento de las redes neuronales profundas.

- Estudiar métodos de optimización en Aprendizaje Automático.
- Conocer las ventajas y desventajas de diferentes métodos de optimización.
- Mostrar los resultados de distintos métodos de optimización en el entrenamiento de una red neuronal convolucional para tareas de clasificación.

Y los objetivos con respecto a las competencias académicas desplegadas en el trabajo son:

- Desarrollar un mejor entendimiento de las redes neuronales y sus aplicaciones, para así poder lograr afrontar problemas en el campo de la inteligencia artificial.
- Obtener la capacidad de discriminar entre los distintos métodos de optimización y elegir el adecuado para un problema de aprendizaje profundo.
- Obtener un conocimiento de las herramientas y recursos que existen actualmente para abordar problemas de aprendizaje profundo, además de poder analizar que herramientas son adecuadas para algunos problemas.

1.3. Estructura del Seminario

- **Introducción:**
  En este capítulo introductorio se comenta sobre el tema a tratar, las motivaciones, intereses, objetivos con los cuales se planteo el presente seminario.
- **Estado del Arte:**
  Este capítulo muestra los trabajos e investigaciones ya realizadas, además de algunas aplicaciones que motivaron al presente seminario y además las investigaciones mostrarán el interés del problema planteado.

- **Aprendizaje automático y Redes Neuronales:**

  En este capítulo daremos una introducción general al Aprendizaje automático y distinguiremos los tipos de aprendizajes que existen, Además veremos los tipos de problema en esta área. Luego trataremos el tema de las Redes neuronales como una introducción al capítulo 4.

- **Optimizadores para la gradiente de descenso en una Red Neuronal Convolucional:**

  En este capítulo conoceremos más de un tipo específico de redes neuronal, las redes neuronales convolucionales. Detallares las principales diferencias con las redes tradicionales y describiremos sus principales hiperparámetros. Luego de eso nos enfocaremos en los optimizadores de la gradiente de descenso.

- **Resultados:**

  Se mostrarán los resultados obtenidos en las pruebas de los optimizadores además de describir los resultados.

- **Conclusiones y Trabajo Futuro:**

  En este capítulo se plantean las conclusiones y se detalla algunos inconvenientes encontrados durante el trabajo. Además que se comprueba la teoría descrita en el capítulo 4.

Capítulo 2

Estado del Arte

En este capítulo se describirán las investigaciones anteriores con relación al Aprendizaje Automático, además de sus aplicaciones. También se verán algunas investigaciones de GPUs como una manera de obtener un mejor rendimiento y nos enfocaremos principalmente en los estudios de los métodos de optimización.

Este trabajo también presentará investigaciones referentes a Aprendizaje Profundo, exclusivamente nos enfocaremos a la Redes Neuronales Convolucionales(CNN), ya que son parte del tema de estudio en la presente investigación.

2.1. GPU computing

Actualmente, el uso de GPU's permitió lograr aplicaciones que antes se podrían creer imposibles debido a su largo tiempo de ejecución. Hoy en día las GPUs son altamente usadas ya que cuentan con cientos de núcleos de procesadores en paralelo que permiten resolver rápidamente los problemas que son altamente paralelizables.

2.1.1. The GPU computing Era

El artículo se enfoca principalmente en describir la evolución que sufrieron las arquitecturas de GPUs, además de mostrar la importancia del uso de las GPUs para un mayor rendimiento y eficiencia que antes hubiesen sido

consideradas imposibles debido al alto tiempo de ejecución que requerían. Además nos muestra que la escalabilidad es la principal característica que ha permito que las GPUs aumenten su paralelismo y rendimiento.

2.2. Aprendizaje Automático

El uso del Aprendizaje Automático representa una gran ventaja para empresas que manejan gran cantidad de datos debido a que permiten descubrir patrones y analizar los datos.

2.2.1. Uso de redes neuronales para encontrar el rendimiento de una GPU

Un equipo conformado por investigadores [1]de AMD y The University of Texas at Austin, fueron quienes propusieron el uso de redes neuronales para predecir el rendimiento de una GPU. En la actualidad existen empresas dedicadas a la creación de GPUs, en el proceso una parte fundamental es la verificación del rendimiento de las GPUs. Actualmente existen simuladores conocidos como GPGPU-SIM que permiten realizar estimaciones precisas pero estos presentan algunas dificultades como el tiempo empleado en configurarlos en base al hardware real, no obstante, este proceso se encuentra propenso a errores.

2.2.2. Handshape recognition for Argentinian Sign Language using ProbSom

Investigadores de la Universidad de La Plata, en Argentina conformado por Franco Ronchetti, Facundo Quiroga, César Estrebou, y Laura Lanzarini[2], desarrollaron un sistema que permite el reconocimiento de lenguaje de señas argentino. Esta investigación fue realizada usando una técnica llamada ProbSom, esta puede ser comparada con otros métodos como las Máquinas de Soporte Vectorial, Bosques Aleatorios y Redes Neuronales.

2.3. Aprendizaje Profundo

Dentro del área de Aprendizaje Automático encontramos Deep Learning o Aprendizaje Profundo el cual consiste en un conjunto de algoritmos que modela abstracciones de alto nivel.
En esta sección hablaremos de un paper que nos sirvió de un introducción al campo del aprendizaje profundo.

2.3.1. Deep Machine Learning - A New Frontier in Artificial Intelligence

Este trabajo de investigación fue realizado por investigadores Thomas Karnowski, Derek Rose - Oak Ridge National Laboratory y Itamar Arel - University of Tennessee [3], el objetivo principal de este trabajo fue presentarnos el aprendizaje profundo como un camino para la imitación del cerebro humano y sus principales cualidades como el reconocimientos de objetos, rostros, etc.
En este paper presenta una introducción a los temas de *Convolutional Neural Network(CNN)* y *Deep Belief Network*, nos describe a las CNN como una familia de redes neuronales multicapas que fueron diseñadas para tratar datos de dimensionalidad 2 como lo son las imágenes y los videos.
Por otro lado, también nos muestra las aplicaciones del aprendizaje profundo como: análisis de documentos, detección de voz, rostro, procesamiento natural del lenguaje, etc.

La aplicación de la inteligencia artificial no solo despierto en los investigadores, también existen algunas empresas privadas que apoyan el campo del Aprendizaje Profundo con el objetivo de buscar sus aplicaciones comerciales, entre estas empresas tenemos a: Numenta y Binatix.

2.4. Métodos de optimización

El campo del Aprendizaje Automático continuamente evoluciona y con esta evolución surgen nuevas necesidades. Al trabajar con grandes conjuntos de datos se buscan cada vez obtener buenos resultados sin afectar el rendimiento. Una forma de lograr esto es mediante el uso de algoritmos de optimización.

2.4.1. Neural Network Optimization Algorithms: A comparison study based on TensorFlow

Vadim Smolyakov[4] realizo un estudio comparativo de diversos optimizadores entre los cuales se encuentran el método de gradiente de descenso estocástica, Nesterov Momentum, RMSProp y Adam. Se realizó una prueba comparativa con una arquitectura simple de CNN usando el conjunto de datos del MNIST. "Se comparó diferentes optimizadores y se obtuvo que el SGD con Nesterov y Adam producen mejores resultados en el entrenamiento de una CNN simple usando tensorflow para el dataset MNIST. "[4]

2.4.2. On Optimization Methods for Deep Learning

Un equipo de la Universidad de Standford realizó unas pruebas con el objetivo de encontrar métodos adecuados para un entrenamiento en aprendizaje profundo. El equipo se percato de lo común que resulta el uso de gradiente de descenso estocástica (SGD por sus siglas en inglés) en aprendizaje profundo . Se realizaron pruebas con otros métodos de optimización como la gradiente conjugada y Limited memmory BFGS(L-BFGS) los cuales permitieron acelerar el proceso de entrenamiento de algoritmos de Aprendizaje Profundo mostrando en su mayoría mejores resultados que el SGD. "Usando L-BFGS el modelo CNN alcanza el 0.69 % en el estándar del MNIST dataset. "[5]

2.4.3. Adam : A method for stochastic optimization

Esta investigación fue la primera en plantear el método Adam para acelerar la gradiente de descenso. Fue propuesto por Diederik P. Kingma de la Universidad de Amsterdam y Jimmy Lei Ba de la Universidad de Toronto. Ellos describen el método Adam como un método sencillo de implementar, además que este utiliza pocos requisitos de memoria. "Nuestro método esta dirigido a problemas con grandes conjunto de datos y espacio de parámetros de alta dimensión. El método combina ventajas de otros métodos de optimización, la capacidad de Adagrad para manejar gradientes dispersos y la de RMSProp para tratar con objetivos no estacionarios"[6]

2.4.4. Incorporating Nesterov Momentum into Adam

Este trabajo fue realizado por Timothy Dozat de la Universidad de Stanford, en este paper se propone una mejora al método Adam modificando su componente de momento de esta manera se obtiene una convergencia más rápida.

"Esencialmente la investigación muestra cómo combinar el momento clásico con una tasa de aprendizaje adaptativa. Este trabajo lleva un enfoque más allá de la investigación y mejora uno de los componentes principales sin aumentar la complejidad del algoritmo"[7]

2.5. Conclusiones

A medida que tratamos muchos problemas vemos la necesidad de encontrar optimizadores adecuados para los diferentes tipos de problemas. En el área de

Aprendizaje Profundo comúnmente se trabaja en el campo de reconocimiento de imágenes.

A pesar de las mejoras mediante el uso de GPUs este tipo de problemas necesitan soluciones óptimos para obtener un mejor rendimiento. Métodos como Nesterov Momemtum, RMSProp y Adam surgen como principales opciones para realizar optimizaciones de la gradiente de descenso.

Capítulo 3

Aprendizaje automático y Redes Neuronales

En este capítulo trataremos los principales conocimientos de Aprendizaje Automático como su clasificación y importancia dentro del campo de la inteligencia artificial, además exploraremos algunos modelos importantes. En este seminario se dará énfasis en los algoritmos de clasificación. Luego nos enfocaremos en las redes neuronales para tratar más a fondo los problemas de clasificación.

3.1. Aprendizaje Automático

Machine Learning o Aprendizaje Automático es una rama de la inteligencia artificial que empezó a cobrar importancia en los años 80's. En esta rama se diseñan sistemas que aprenden a identificar patrones en un conjunto de datos. A medida que se realiza este aprendizaje, la máquina podrá ser capaz de realizar una predicción o tomar decisiones sin haber estado programada explícitamente para realizar esta tarea.

El Aprendizaje Automático se puede clasificar en 3 tipos: Supervisado, No supervisado, Aprendizaje con refuerzo.[8]

3.1.1. Aprendizaje Supervizado

Este tipo de aprendizaje toma un conjunto de datos etiquetados, es decir datos cuyos resultados o clases son conocidos, estos datos serán usados como entrada al sistema. Primero se entrena el modelo con los datos de entrada y luego se trata de predecir una salida de acuerdo a las etiquetas definidas.

"El aprendizaje supervisado trata de modelar la relación entre el resultado de la predicción y características de las entradas de manera que se puede predecir nuevos valores para un nuevo conjunto de datos "[9]

Tipos de problemas

Dentro del aprendizaje supervisado podemos dividir los problemas en 2 tipos:

Problemas de Regresión Lineal

Los problemas de regresión lineal son muy conocidos en el ámbito de Aprendizaje Automático y la estadística.

Problemas de Clasificación

"En este tipo de problemas se predice una respuesta del tipo categórica de manera que se puedan separar los datos mediante clases. "[8]

"El objetivo de los problemas de clasificación es asignar resultados en categorías discretas en lugar de estimar valores continuos "[9].
Un ejemplo de su uso son las tareas de reconocimiento de imágenes.

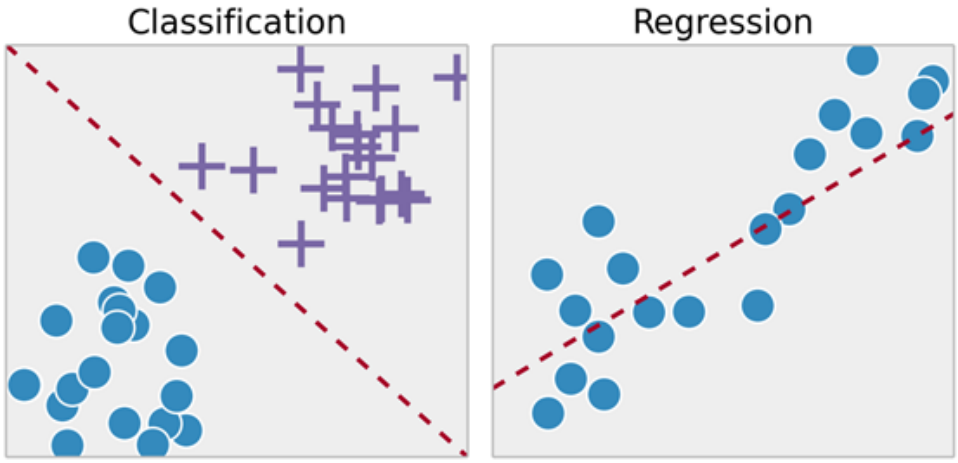


FIGURA 3.1: Regresión y clasificación
Fuente: *https://medium.com/*

Algoritmos de Aprendizaje Supervisado

Regresión Lineal

"El algoritmo de regresión lineal asume que existe una relación entre las variables de entrada $x = (x_1, ..., x_n)$ y una salida simple y. Cuando se tiene solo una variable simple x el método se conoce como *simple linear regression* y cuando se tienen múltiples entradas se le conoce como *multiple linear regression*. "[10] .Es comúnmente usado para estimar valores reales en base a variables continuas. La figura 3.2 muestra una regresión lineal simple.

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + ... + b_n * x_n \tag{3.1}$$

En esta ecuación:

- y : Variable dependiente
- x_i: Variable independiente i
- b_0: Intercepción
- b_1: Coeficiente para la primera característica
- b_n: Coeficiente para la primera característica

El objetivo del algoritmo de regresión lineal es obtener valores adecuados para los b_i de manera que se reduzca la siguiente función de costo.

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2 \tag{3.2}$$

- $pred_i$: Predicción de la i-ésima variable
- y_i : Valor real asociado a la i-ésima variable
- n : Número de datos para el entrenamiento

Un método importante es la gradiente de descenso que se usa para actualizar valores de los b_i de manera que se reduzca la función de costo J de la ecuación 3.2. La figura 3.2 Nos muestra un conjunto de datos y la aplicación de la regresión lineal en estos.

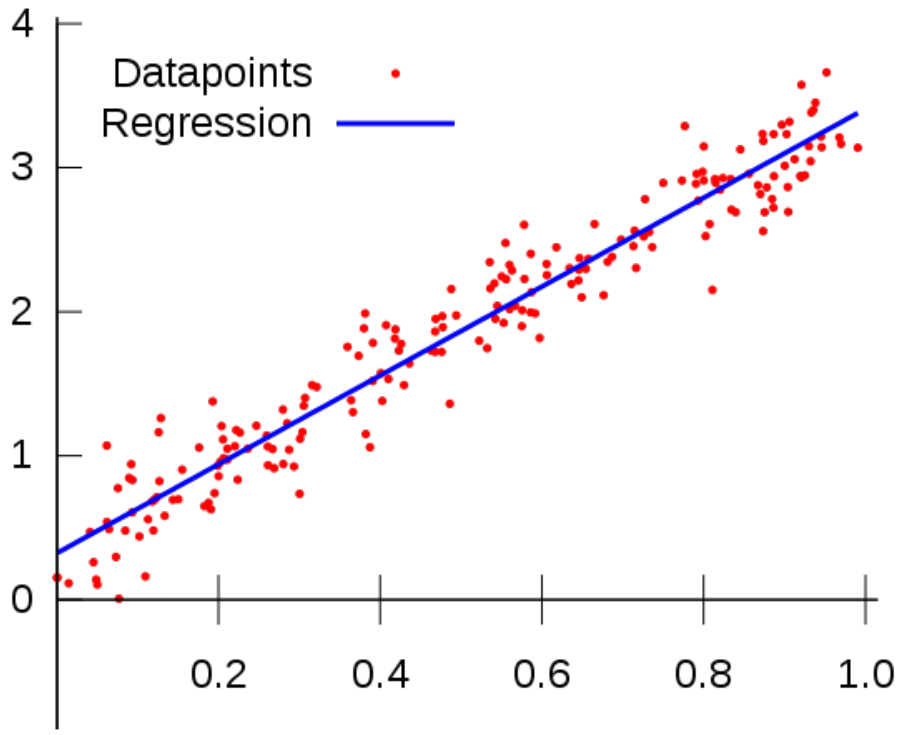


FIGURA 3.2: Regresión Lineal
Fuente: *www.forexmt4indicators.com/*

Regresión Logística

A diferencia de la Regresión Lineal, la Regresión Logística es usado para predecir el resultado de un variable de tipo categórica es decir variables que pueden ser descritas por un número finito de categorías.

La regresión Logística es usado para problemas de clasificación lo cual realiza mediante la predicción de que una salida Y dicótoma, es decir,que solo tenga 2 posibles valores. la regresión produce una curva, la cual genera valores entre 0 y 1. "Matemáticamente podemos ver como que las salidas están modeladas como una combinación de los predictores lineales."[11]

$$\begin{aligned} odds &= p/(1-p) \\ ln(odds) &= ln(p/(1-p)) \\ logit(p) &= ln(p/(1-p)) = b_0 + b_1X_1 + b_2X_2 + b_3X_3.... + b_kX_k \end{aligned} \tag{3.3}$$

- p : probabilidad de presencia de una característica de interés.
- odds: probabilidad de éxito.
- logit: función logit

Despejando p de las ecuaciones anteriores de 3.2 podemos obtener que:

$$\begin{aligned} p &= \frac{1}{1+e^{b_0+b_1X_1+b_2X_2+b_3X_3....+b_kX_k}} \\ Y_{pre} &= \frac{1}{1+e^{f(X)}} \end{aligned} \tag{3.4}$$

En la ecuación 3.3, Y define la función logística que se muestra en la figura 3.3. Esta también se puede conocer como la función sigmoidal en el perceptron. El algoritmo usa SGD para hallar los valores adecuados de b_i de manera que el $erro = Y_{pre} - Y$ sea mínimo. El valor de la predicción es 1 si $Y_{pred} > 0.5$ y 0 en caso contrario. De esta forma se determina el objeto con características X si pertenece o no a una clase.

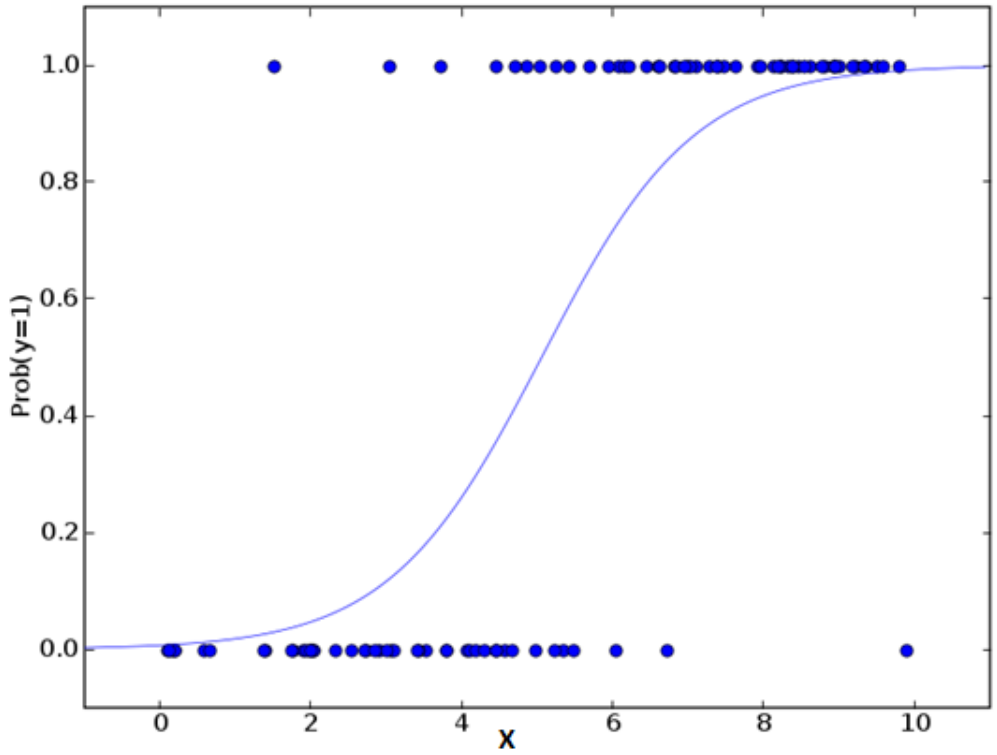


FIGURA 3.3: Regresión Logística
Fuente: *www.analyticsvidhya.com*

Nearest Neighbor

Es un algoritmo de clasificación que almacena los conjuntos de entrenamiento de manera que dado un nuevo ejemplo x lo clasifica buscando la distancia más cercana a un ejemplo de entrenamiento (x_i, y_i) de manera que identifica la clase $y = y_i$ a la que corresponde.

Comúnmente se usa el algoritmo k-nn para clasificar una entrada x en los k más cercanos conjuntos de entrenamiento y asigna el objeto a la clase de más frecuencia.

$$x^i = (x_1^i, x_2^i, ..., x_n^i)$$
$$d_E(x^i, x^j) \tag{3.5}$$

- x^i: objeto con n características.

Definimos d_E como la función distancia entre los vectores x_i y y_i .
Esta función distancia puede clasificarse en:

- Distancia Euclideana: $(\sum_{i=1}^{k}(x_i - y_i)^2)^{\frac{1}{2}}$
- Distancia Manhattan: $\sum_{i=1}^{k}|x_i - y_i)|$
- Distancia Minkowski: $(\sum_{i=1}^{k}(|x_i - y_i|)^p)^{\frac{1}{p}}$

Los 3 definiciones anteriores de distancia son usadas en variables continuas. Para el caso de las variables categóricas se deberá usar la distancia de Hamming cuya definición se muestra en la ecuación 3.6

$$\begin{aligned} D_H &= \sum_{i=1}^{k}|x_i - y_i)| \\ x = y &\Longrightarrow D = 0 \\ x \neq y &\Longrightarrow D = 1 \end{aligned} \tag{3.6}$$

"La elección de un valor óptimo k se logra mejor por la inspección de los datos. En general un valor grande de k es más preciso, ya que reduce el ruido pero no hay garantías de que sea un valor correcto, una mejor manera de calcular el valor de k es mediante el uso de la validación cruzada. "[12]

En la figurar 3.4 muestra el algoritmo de k-nn dado un nuevo ejemplo(círculo verde) este será clasificado de acuerdo al k seleccionado. Para $k = 1$ el nuevo ejemplo será clasificado en la clase 1 y $k = 3$ será clasificado en la clase 2.

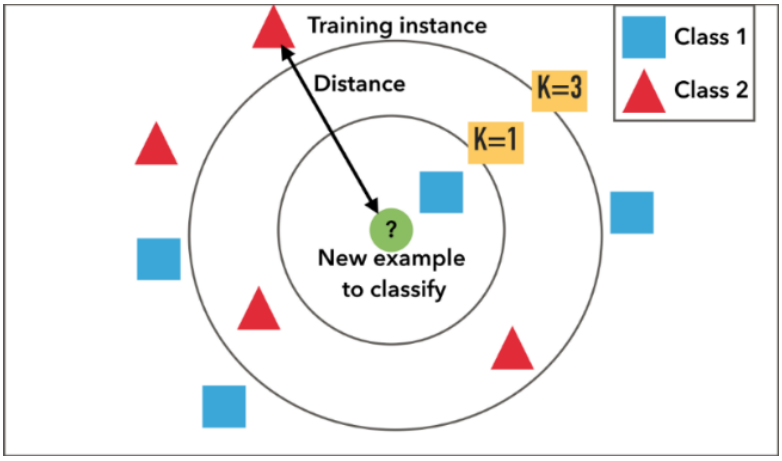


FIGURA 3.4: knn
Fuente: *www.medium.com/*

Máquinas de soporte Vectorial(SVM)

Las Maquinas de soporte vectorial fueron creadas por Vladimir Vapnik y constituyen un método para realizar tareas de clasificación y regresión.
Las SVM usan el concepto de planos de decisión. Un plano de decisión separa un conjunto de objetos que tienes diferentes etiquetas de clases. Las SVM no están restringidas a los problemas lineales debido a las *funciones Kernel.*

Funciones Kernels

Las SVM pueden tener distintos tipos de kernels que tienen como objetivo tomar la data y transformarla. Algunas funciones kernels conocidas:

- Lineal: $\ker(x_i, x_j) = x_i \cdot x_j$
- Polinomial: $\ker(x_i, x_j) = (\gamma x_i \cdot x_j + C)^d$
- Radial: $\ker(x_i, x_j) = e^{(\gamma|x_i - x_j|)}$
- Sigmoidal: $\ker(x_i, x_j) = \tanh(\gamma x_i \cdot x_j + C)$

En la figura 3.5 muestra el efecto de las funciones kernels en un conjunto de datos para que este sea linealmente separable sin necesidad de construir curvas complejas.

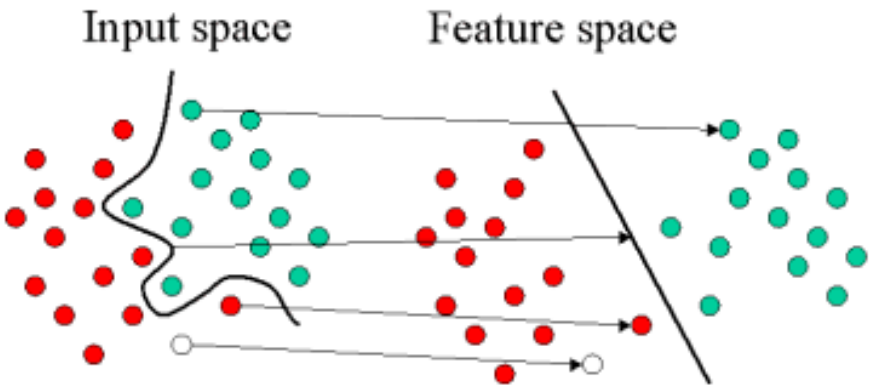


FIGURA 3.5: transformación con la función kernel
Fuente: *www.statsoft.com*

Podemos dividir SVM en 2 categoríasas:

Super Vector Classificacion

Los SVC realizan la tarea de clasificación encontrando un hiperplano que maximicé el margen entre 2 clases.Los vectores que definen el hiperplano son llamados *support vector*.

Para la clasificación es necesaria mapear los datos a un espacio de características de mayor dimensión donde resulte más fácil la separación lineal. La imagen de la Figura 3.5 muestra de manera gráfica que cambio de espacio nos permite separar clases de manera más sencilla.

Super Vector Regression

SVR trata de mapear los datos de entrenamiento $x \in X$, a un espacio de mayor dimensión mediante una mapeo no lineal $\varphi : X \to F$.

Las SVR son parecidas a las máquinas de soporte Vectorial para la clasificación pero con la diferencia de que la salida es un número real que es difícil de predecir con la información que se posee además de que tiene infinitas posibilidades. Para los problemas de regresión se usan los kernels Radial y polinomial. La figura 3.6 muestra un ejemplo de problema de regresión para un caso no lineal, mediante la mapeo φ se cambia el espacio.

- caso lineal: $y = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\langle x_i, x\rangle + b$
- caso no lineal: $y = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\ker(x_i, x) + b$

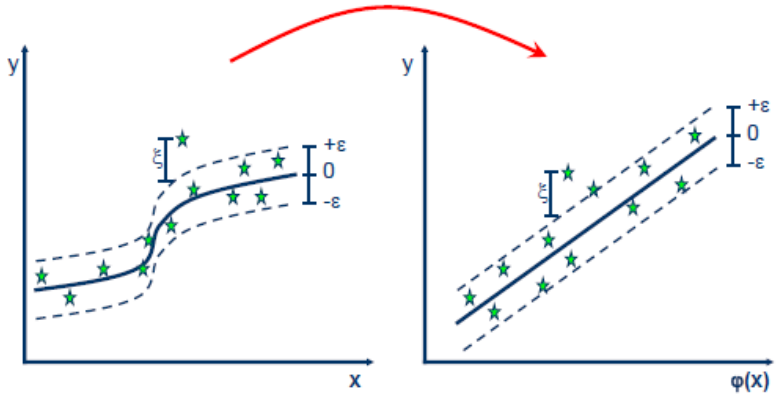


FIGURA 3.6: transformación para un problema de regresión
Fuente: *www.saedsayad.com*

Naive Bayes

Esta basado en el Teorema de Bayes donde se asume la independencia entre los predictores."El modelo Naive Bayes es fácil de construir, debido a que no tiene la necesidad de estimar los parámetros iterativamente por lo cual es particularmente útil para un gran conjunto de datos "[13].
Como la técnica de clasificación se basa en el Teorema de Bayes supone que cada característica de una clase no esta relacionada con otras características.
Podemos ilustrar mejor lo dicho mediante el siguiente ejemplo "Una fruta puede considerarse una manzana si es roja, redonda y de aproximadamente 3 pulgadas de diámetro. Incluso si estas características dependen unas de otras o de la existencia de otras características, todas estas propiedades contribuyen de forma independiente a la probabilidad de que esta fruta sea una manzana y es por eso que se la conoce como **Naives** "[14]

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$
$$P(c|X) = P(x_1|c) \times P(x_2|c) \times ... \times P(x_n|c) \times P(c) \tag{3.7}$$

- $P(c|x)$: Probabilidad condicional de una clase(target) dado un predictor(atributo)
- $P(c)$: Probabilidad de la clase

- $P(x|c)$:Probabilidad condicional de la clase dada por el predictor.
- $P(x)$: Probabilidad del predictor.

Arboles de decisión

Los arboles de decisión son modelos usados para tareas de regresión y clasificación, estos utiliza la estructura de un árbol. La idea se basa en dividir el conjunto de datos en subconjuntos cada vez más pequeños que a su vez desarrolla un árbol de decisión. El resultado final será un árbol con nodos de decisión y nodos hojas.
Los nodos de decisión tienen 2 o más ramas y los nodos hojas representa una clasificación o decisión. La raíz corresponde al mejor predictor. Un árbol de decisión puede ser usado para manejar datos categóricos o numéricos.

Árbol de decisión para clasificación.
El algoritmo principal para construir un árbol de decisión es el ID3 que fue desarrollado por J. R. Quinlan.
"ID3 realiza una algoritmo greeady para realizar un búsqueda de arriba hacia abajo a través del espacio de posibles ramas . ID3 usa entropía e información ganada para construir el árbol de decisión ".
A diferencia de Naives Bayes, en un árbol de decisión es importante asumir la dependencia de los predictores.

- **Entropía**: Para el algoritmo ID3 es importante tener la entropía para calcular la homogeneidad de la muestra.
- **Información ganada**: se basa en la disminución de la entropía después que un conjunto de datos se divide en atributos.

Árbol de decisión para regresión.
Al igual que el caso de clasificación, se usa el algoritmo ID3 con la diferencia que se usa la información ganada por la desviación estándar de reducción. Este

tipo de desviación se basa en la disminución de la desviación estándar cuando el conjunto de datos se divide en un atributo.

3.1.2. Aprendizaje No Supervisado

Mientras que el aprendizaje supervisado aprende en base a un conjunto de datos de entrenamiento que contienen respuestas o etiquetas correctas. En el aprendizaje no supervisado los datos de entrenamiento no poseen ninguno tipo de etiqueta, por lo tanto los sistemas deben de interpretar los datos por si mismos. El aprendizaje no supervisado es usado principalmente para el reconocimiento de patrones y modelado descriptivo. A continuación discutiremos algunos algoritmos del Aprendizaje No Supervisado.

Clustering

Clustering se refiere a agrupar objetos con características similares, es decir buscar la relación entre ellos sin necesidad de que exista un conocimiento a priori de esos grupos.

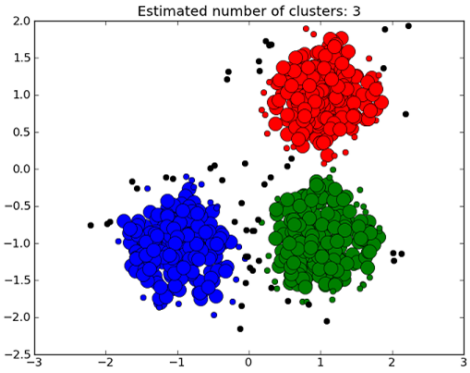


FIGURA 3.7: Clustering
Fuente: *https://medium.com/*

K-means Clustering

El algoritmo realiza la partición de N objetos en k clusters en los cuales cada objeto pertenece al cluster que posee la media más cercana. Este método produce k clústeres con la mayor distinción posible. El k adecuado no se conoce

a priori por lo cual debe calcularse a partir de datos. El objetivo de k-means es minimizar la función de error al cuadrado mostrada en la ecuación 3.8.

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \| x_i^j - c_j \|^2 \tag{3.8}$$

- J: Función de error cuadrado.
- k : Número de cluster.
- n : Número de casos.
- x_i^j : Objeto i
- c_j : Centroide del cluster j

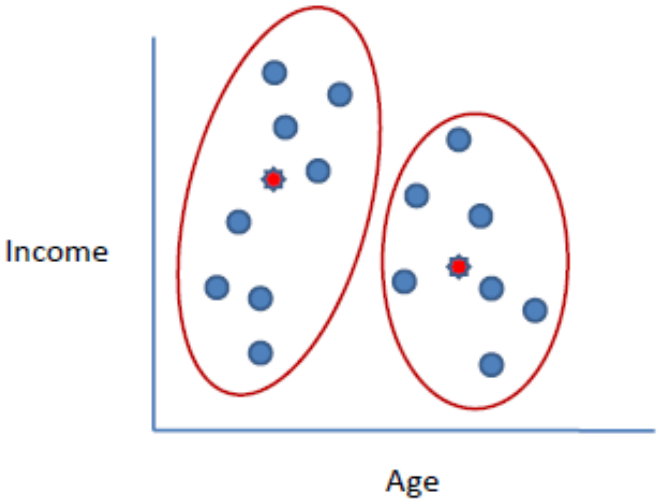


FIGURA 3.8: k means clustering
Fuente: *www.saedsayad.com*

3.1.3. Aprendizaje por refuerzo

Este tipo de aprendizaje fue inspirado por la psicología conductista, este busca determinar que tipo de acciones tomar en un entorno dado. "El objetivo del método es recopilar la interacción con el entorno para tomar acciones que maximicen el beneficio o minimicen el riesgo. "[9]

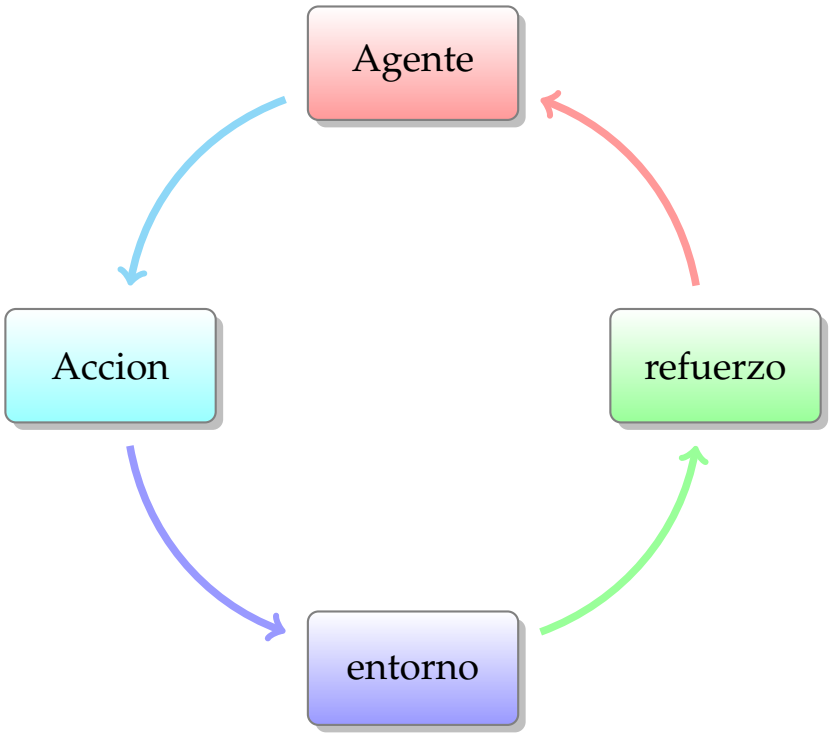


FIGURA 3.9: Esquema de aprendizaje por refuerzo
Fuente: *Fuente Propia*

3.2. Redes Neuronales

3.2.1. Neuronas

En la biología, la neurona es conocida como la unidad fundamental del cerebro humano, el cual está compuesto por millones de neuronas interconectadas entre si(sinapsis). El trabajo de las neuronas consiste en recibir información, procesarla y enviarla a otras células.

Este modelo fue copiado en 1943 por Warren S. McCulloch y Walter H. Pitts para poder diseñar un neurona artificial que es análoga a las neuronas del cerebro humano, la neurona artificial tomará una cantidad n de entradas $x_1, x_2, x_3, .., x_n$ estas entradas serán multiplicadas por pesos $w_1, w_2, w_3, .., w_n$ además se puede añadir una constante que llamaremos bias(b) para producir un salida.

La entrada a la neurona será la suma total de los productos z= $\sum_{i=1}^{n} w_i x_i + b$, el valor de z , esta se evaluará con una función f de tal forma que nuestra salida será $y = f(z)$.
En la ecuación 3.9 observamos la misma salida expresada en forma vectorial para nuestros vectores $x = [x_1 x_2 x_3 ... x_n]$ y $w = [w_1 w_2 w_3 ... w_n]$

$$y = f(x \cdot w + b) \tag{3.9}$$

Funciones de Activación

La función f anteriormente mencionada es una función no lineal, conocida como función de activación.
La tarea principal de la función de activación es introducir no linealidad a la salida de una neurona. Esto es importante debido a que la vida real no trabajamos con datos no lineal y de esta forma la neurona puede aprender representaciones no lineales.
Entre funciones de activación tenemos algunas comúnmente usada como:

- **Sigmoid**: Toma un valor real, y lo transforma en una valor en el rango de 0 a 1.
  $\sigma(x) = \frac{1}{1+e^{-x}}$
- **tanh**: Toma por entrada un valor real y lo transfroma a un número en le rango de -1 a 1.
  $\tanh(x) = \frac{2}{1+e^{-x}} - 1$
- **ReLu**: o Unidad lineal rectificada es una función que para valores menores que 0 asigna 0 y para valores mayores

$$f(x) = \begin{cases} 0 & x < 0 \\ \text{x} & x \geq 0 \end{cases}$$

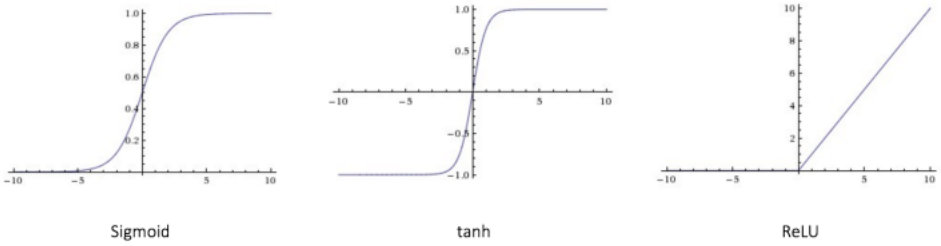


FIGURA 3.10: Funciones de activación
Fuente: *https://ujjwalkarn.me*

3.2.2. Redes Neuronales Artificiales

Las redes neuronales artificiales(ANN)toman de ejemplo la arquitectura del cerebro como inspiración para la construcción de sistemas inteligente. Actualmente son la base para el desarrollo de la inteligencia artificial.
Una red neuronal está constituida por las uniones de neuronas.
En la figura 3.10 podemos ver la comparación entre una neurona biológica y un artificial etiquetadas con A y B respectivamente. Además observamos que las redes neuronales artificiales (etiqueta D) imitan el la unión biológicas de las neuronas o sinapsis(Etiqueta C).

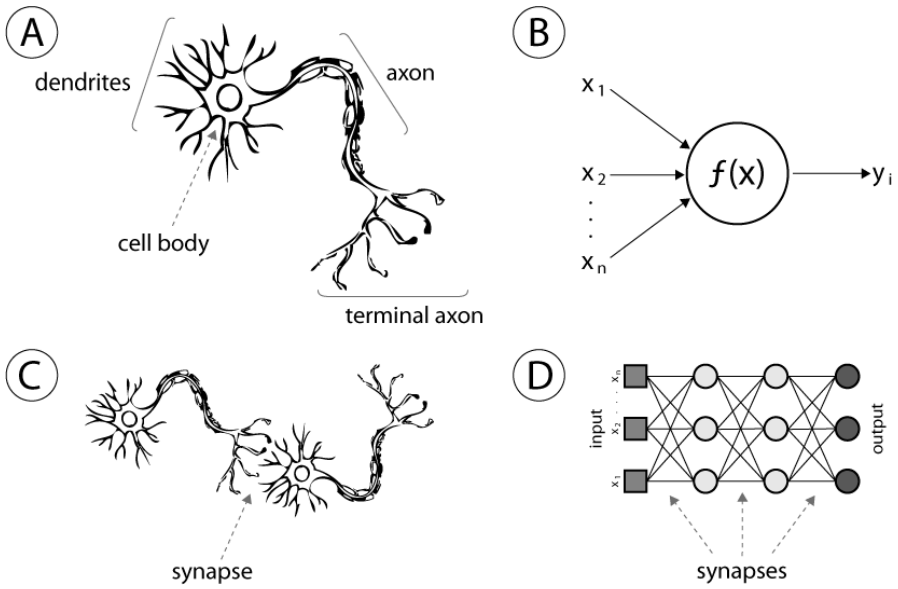


FIGURA 3.11: Redes neuronales biológicas y artificiales
Fuente: *https://medium.com*

Redes Neuronales Prealimentadas

Estás redes fueron de las primeras y más simples de los tipos de redes neuronales que fueron desarrolladas. Contienen múltiples neuronas (nodos) ordenadas en capas de modo que los nodos en capas adyacentes se conectan. Cada una de estas conexiones poseen un peso asociado a dicha conexión.
En la figura 3.12 mostramos el esquema de Redes prealimentadas con sus distintas capas(layer).

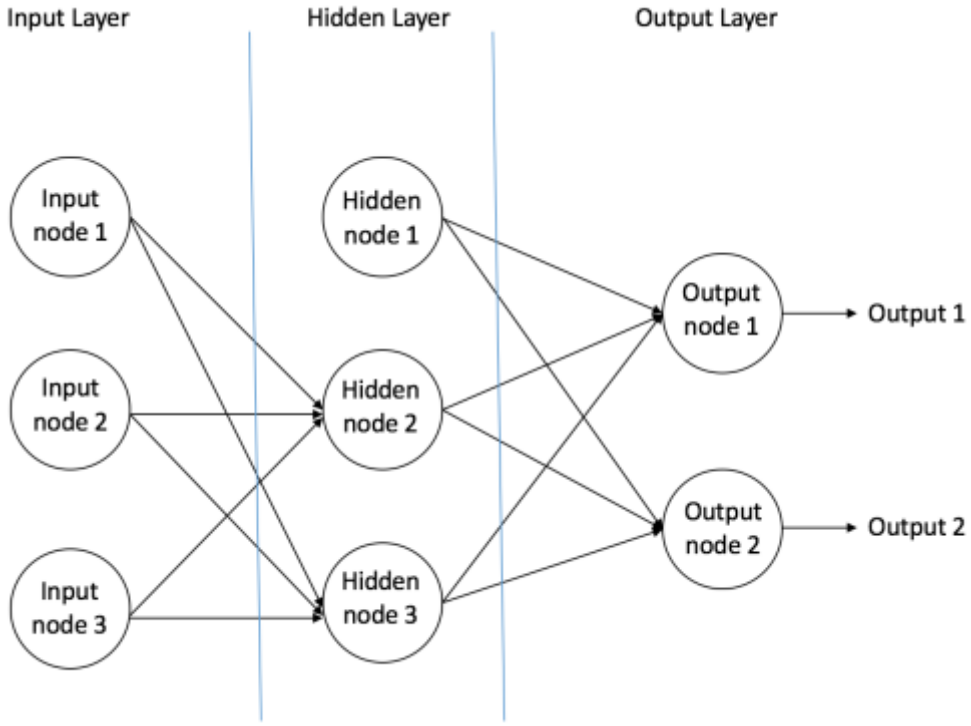


FIGURA 3.12: Esquema de Redes Neuronales Prealimentadas
Fuente: *https://ujjwalkarn.me*

- **Nodo de entrada(Input Node):** Proveen información a la red. En conjunto representan la capa de entrada, ningún calculo es realizado en esta capa solo se transfiere la información a la capa oculta.
- **Nodo Oculto(Hidden Node):** El trabajo de los nodos ocultos es calcular y transferir la información hacia a el nodo de salida. Una Red prealimentada tiene solo una capa de entrada y una salida pero puede tener múltiples capas ocultas.

- **Output Node:**Su tarea principal es realizar cálculos y transferir la información fuera de la red.

En las redes neuronales prealimentadas, la información solo se propaga en una dirección hacia adelante desde los nodos de entradas pasando por los nodos ocultos hacia los nodos de salida. No existen ciclos en este tipo de red.
Dentro de las redes neuronales prealimentadas tenemos algunos ejemplos:

- **Perceptron Simple:** Es un red prealimentada simple que no posee capa oculta.Solo puede aprender de funciones lineales.
- **Perceptron Multicapas:** Esta red posee una o más capas ocultas. Este perceptron puede aprender de funciones no lineales.
- **Redes neuronales de convolución:** Este tipo de redes neuronal será explicada con más detalle en el capítulo 4 sección 1.

Algoritmo de propagación hacia atrás

La propagación hacia atrás trata de aprender de los errores, como ya hemos visto en el aprendizaje supervisado los conjuntos de entrenamiento se encuentran etiquetados. Por lo cual podemos saber cual es la salida esperada.
El algoritmos se aplica de la siguiente forma:

1. Se toma un ejemplo y se asignan pesos aleatorios a todas las conexiones de la red. Luego por medio de las conexiones y funciones de activación se calcula la salida en las capas ocultas y de salida.
2. Se calcula el error total y se propagan estos errores hacia atrás a través de la red y se calcula la gradiente, luego se usan métodos como gradientes de descenso para ajustar los pesos y reducir el error en la capa de salida. La técnica de gradiente de descenso será explicada con más detalle en el siguiente capítulo.
3. Se repite el proceso con los otros ejemplos

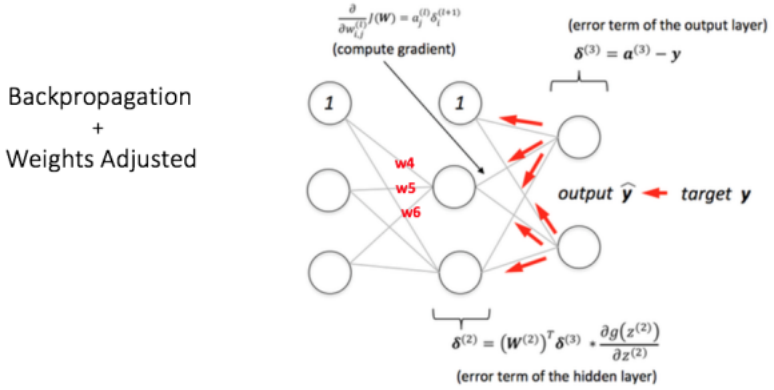


FIGURA 3.13: Propagación hacia atrás
Fuente: *https://ujjwalkarn.me*

Conjunto de datos para una red neuronal

Para alimentar nuestra red es importante tener información que le permita aprender. Esta información es llamada conjunto de datos o dataset, el cual nos permite entrenar a nuestra red y testearla.
Una tarea principal para construir nuestro modelo es separar este dataset en 3 categorías.

- **Datos de entrenamiento:** Permite entrar nuestro modelo.
- **Datos de Validación:** Permite evaluar y actualizar los parámetros de entrenamiento
- **Datos de testeo:** Permite testear el funcionamiento del modelo al ingresar datos nuevos.

Con esta división es posible entrenar nuestra red y verificar su funcionamiento. La figura 3.14 nos muestra el esquema de la división de los datos.

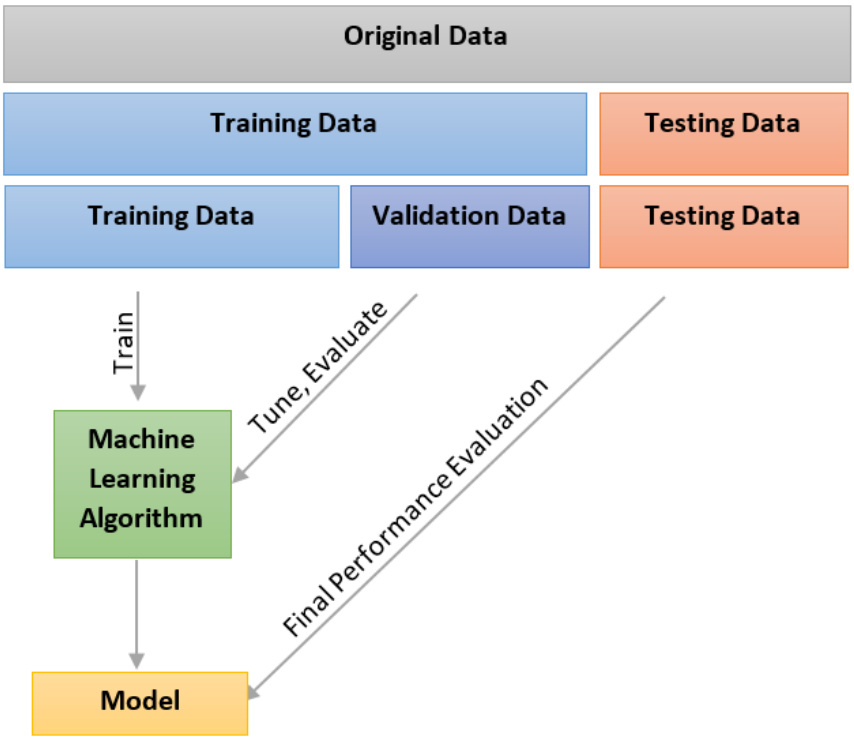


FIGURA 3.14: División del dataset
Fuente: *http://magizbox.com*

Capítulo 4

Optimizadores para la gradiente de descenso en una Red Neuronal Convolucional

En este capítulo se detallarán algunos algoritmos de optimización de Aprendizaje automático y principalmente se enfocará en aquellos que serán utilizados en nuestra red neuronal convolucional.
Al inicio de este capítulo veremos una introducción a este tipo de redes de prealimentación.

4.1. Redes Neuronales Convolucionales

Las CNN son un tipo de redes neuronales especiales para procesar datos como imágenes las cuales son más difíciles de tratar en una red neuronal tradicional, como por ejemplo en el caso del perceptron multicapas.
El término *convolucional* hace referencia a la operación lineal matemática usada. Las redes neuronales convolucionales usan esta operación para aprender de las características de mayor orden presente en los datos. La primera CNN fue creada por Yann LeCun. Entre sus usos más comunes tenemos el reconocimiento de imágenes y lenguaje natural.
Las redes neuronales convolucionales fueron inspiradas en la corteza visuales de los animales. Las células de la corteza visual, estas se activan para realizar tareas como el reconocimiento de patrones.

4.1.1. Estructura de una imagen

Debido a que las redes neuronales convolucionales trabajan principalmente con imágenes, es importante conocer cual es la estructura de una imagen y cómo es que la computadora comprende y utiliza esta información.
Las imágenes están constituidas por una sucesión de píxeles, podemos entender el pixel como la menor unidad homogénea en color de una imagen digital. Teniendo este concepto, podemos dividir la información de una imagen de la siguiente forma:

- **Width**: El ancho de la imagen medido en pixeles
- **Height**: El alto de la imagen medida en pixeles.
- **Canales RGB**: Estos canales contiene la información de los colores y profundidad de una imagen. Este canal guarda la información en tres canales Red, Green y Blue.

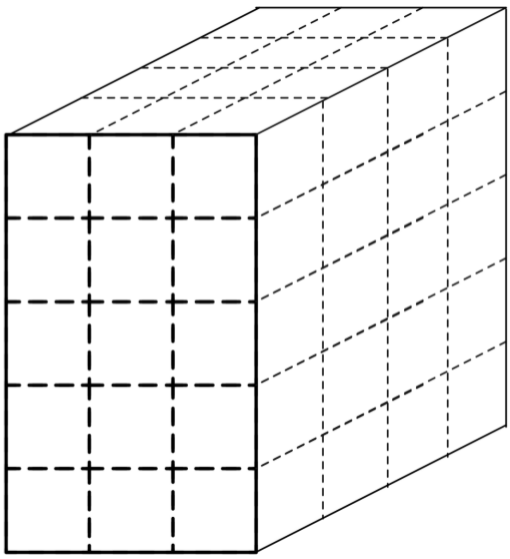


FIGURA 4.1: Estructura de la imagen de entrada
Fuente: *Deep Learning by Adam Gibson, Josh Patterson*

Teniendo en cuenta esta forma de guardar una podemos resaltar la ventaja de usar Redes convolucionales en lugar de usar una red neuronal multicapas.

Las redes multicapas toman un vector de una dimensión como entrada, si quisiéramos entrenar un perceptron multicapas con imágenes de 32x32 píxeles y con 3 canales RGB necesitaríamos crear 3072 pesos (w_i) para una sola neurona en la capa oculta. Esta generación excesiva de peso hace que la tarea resulte complicada usando redes multicapas.
De esta forma surge la idea de recurrir a un tiempo tipo de redes neuronales que faciliten la tarea sin consumir muchos recursos.

4.1.2. Capas de una CNN

Las redes neuronales convoluciones pueden ser dividas en distintas capas, cada una con una tarea específica para el tratamiento de la información. En esta sección describiremos cada una de estas capas.

Input layer

Esta capa es la encargada de cargar y almacenar la información de las imágenes para luego procesarlas en la red. Esta información contiene detalles de ancho, alto en píxeles y el número de canales de imagen. Las entradas de esta capa corresponden a la imagen vista en la figura 4.1.

Convolutional layers

Es una de las capas más importante en el diseño de las CNNs, esta capa es la encargada de transformar la entrada(imagen o convolución anterior) usando las conexiones de las neuronas en capas anteriores. La capa calculará el producto punto entre la región de las neuronas de la capa de entrada y los pesos a los que están colocados localmente en la capa de salida. Esta salida tendrá la misma dimensión de espacios o una dimensión menor.

Para entender más a fondo esta capa debemos definir la operación de *convolución*. La *convolución* es una operación matemática que describe una regla de como fusionar 2 conjuntos de información."Esta operación tiene importancia en campos como la matemática y la física debido que permite definir un puente

entre el domino del espacio/tiempo y el dominio de la frecuencias a través del uso de la transformada de fourier.

La convolución toma la entrada, aplica un kernel de convolución y nos da un mapa de características como salida "[15] .

Las convoluciones son usadas principalmente como un detectores de características cuyas entradas son la capa de entrada u otra convolución. En la figura 4.2 observamos la operación de convolución que por medio del uso de un kernel o filtro de convolución extrae características de la imagen, por ejemplo detalles como bordes de una imagen.

Haciendo analogía con los pesos en las redes neuronales convencionales, las redes poseen el *filtro o kernel* , esto resulta beneficioso, ya que no se tendrá definir un peso para cada neurona.

En la figura 4.2 vemos como se aplica el kernel para producir datos de característica, este kernel será desplazado a lo largo de las dimensiones espaciales. En el desplazamiento el kernel se multiplicará por los datos de entrada dentro de su limite, produciendo una sola salida al mapa de características.

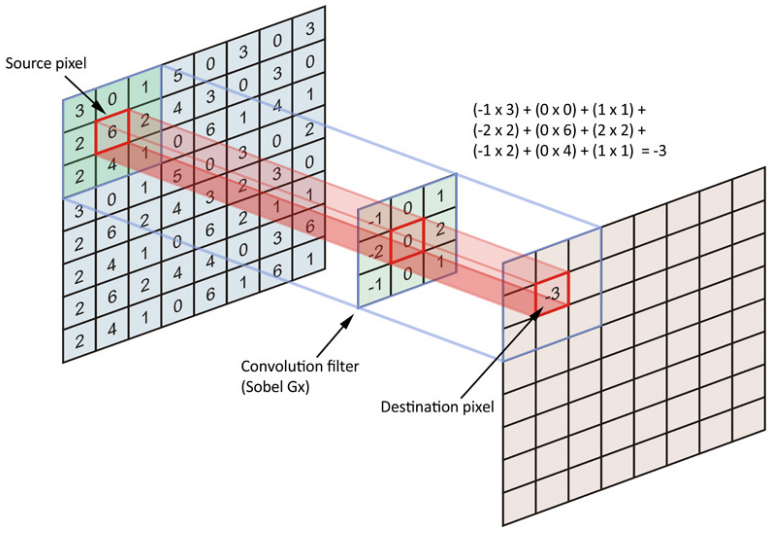


FIGURA 4.2: Operacion de convolución
Fuente: *www.openresearch.ai*

Las capas convolucionales aplican transformaciones o funciones de activación al conjunto de entrada, luego el mapa de activación generado se

apilará a lo largo de dimensión de profundidad para construir el volumen de salida.

Componentes de la capa de convolución.

Las capas convolucionales poseen parámetros e hiperparámetros. La gradiente de descenso tiene la funcion de entrenar los parámetros de modo que las clases sean consistentes con las etiquetas en el conjunto de entrenamiento. Entre estos parámetros tenemos: **Filtros**

Los filtros son una función que posee ancho(width) y alto (height) más pequeños que la entrada. Los filtros son aplicados a través de del ancho y alto de la entrada, pero también pueden ser aplicados a lo largo de la profundidad.

Hiperparámetros de una capa de convolución.

A continuación veremos algunos hiperparámetros que determinan la disposición espacial y tamaño del volumen de salida de una capa convolucional.

- **Filter size:** Cada filtro es pequeño con respecto al ancho(width) y alto(height) del la capa anterior. Por ejemplo podemos tener un filtro de tamaño $[5x5x3]$, lo representa 5 de ancho x 5 de alto x 3 de los canales RGB.

- **Output depth:** Este hiperparámetro controla el número de neuronas en la capa convolucional que están conectadas al mismo del volumen de entrada. Este parámetro puede ser elegido manualmente.

- **Stride:** Se encarga de configurar el tamaño de desplazamiento de la ventana de filtro. Cada filtro aplicado a la columna de entrada asignará más profundidad en el volumen de salida. Un stride grande creará un volumen de salida más grande y uno valor pequeño obtendrá un volumen menor.

- **Zero-padding:** Con este parámetro se puede controlar el volumen de salida. Es usado para mantener el tamaño espacial de entrada en la salida.

Pooling layers

Este tipo de capas se encuentran entre las capas convolucionales. Se encarga de reducir el tamaño espacial(ancho,alto) de los datos de representación. Esta capa reduce la representación de los datos progresivamente a través de la red y ayuda a controlar el *overfitting*.
Esta capa utiliza la operación *max()* para cambiar el tamaño de los datos de entrada espacialmente, a esta operación se le conoce como max pooling. Esta funciona de siguiente forma toma un filtro de nxn, y la operación max toma el mayor de los números en el área de filtro.
Por ejemplo en caso tener una imagen de entrada 32×32 píxeles y se aplica un filtro de 2×2, como resultado obtendremos una salida de 16×16 píxeles. Esto reduce cada segmento de profundidad en el volumen de entrada por un factor de 2.

Fully Connected Layers

Esta capa se calcula el puntaje de las clases que usaremos como salida de red, esta será la encargada de reconocer a que clase pertenece una imagen de prueba de acuerdo a su puntaje o probabilidad. Las dimensiones del volumen de la salida son [1x1xN], donde el valor de N corresponde al número de clases de salida que se están evaluando. En el caso del MNIST (dataset para reconocimiento de dígitos), el valor de N es igual a 10, número que corresponde a los 10 dígitos distintos que posee el dataset($0, ..., 9$).
Esta capa tiene conexión entre todas sus neuronas y las de la capa anterior. Esta capa realiza las transformaciones del volumen de datos de entrada. Estas son funciones de activación en el volumen de entrada y los parámetros (pesos y bias de las neuronas).

4.1.3. Arquitecturas conocidas

Actualmente existen algunas arquitecturas de CNN ya diseñadas que son aplicadas para el trabajo de reconocimiento de imágenes.

El proyecto ImageNet, posee una gran base de datos de imágenes. Este proyecto realizá una competición llamada *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* donde compiten distintos programas de software para detectar y clasificar objetos.

A continuación mostraremos algunas de las arquitecturas más importante de esta competencia:

- **LeNet-5 (1998)** "Arquitectura propuesta por LeCun, consiste 2 capas de convolución, activación y capas pooling seguidas por a fully conected layer"[16]

- **AlexNet (2012)** Fue propuesta por Alex Krizhevsky, esta arquitectura posee 5 capas de convolución seguida por 3 fully connected layers.

- **VGGNet (2014)** Fue desarrollada para Sigmoyan y Zisserman para la competición ILSVRC. "VGG consta de 16 capas convolucionales y es muy atractivo debido a su arquitectura uniforme. Consta de convoluciones de 3x3 y utiliza múltiples filtros "[17]

4.2. Métodos de Optimización

En el campo del Aprendizaje Profundo es recomendable la elección de un buen algoritmo de optimización, debido a que este puede representar la diferencia entre minutos,horas , etc. La tarea principal de este algoritmo es reducir un función objetivo, en nuestro caso nuestra función objetivo será la función de perdida $J(\theta)$.

4.2.1. Gradiente de descenso

La gradiente de descenso es un algoritmo más común para optimizar redes neuronales. La gradiente de descenso es una forma de minimizar la función de costo $J(\theta)$ parametrizada por los parámetros $\theta \in \Re^d$.
Esta función nos permitirá determinar que tan precisa es el rendimiento de nuestra red. La gradiente actualiza los parámetros en la dirección opuesta a la gradiente de nuestra función objetivo en este caso, la función de costo $\nabla_\theta J(\theta)$.
En la figura 4.3 observamos una función de costo con solo 2 parámetros, la tarea de la gradiente de descenso es encontrar valores particulares de θ que nos permitan llegar de el punto A al punto B donde la función alcanza un valor mínimo.

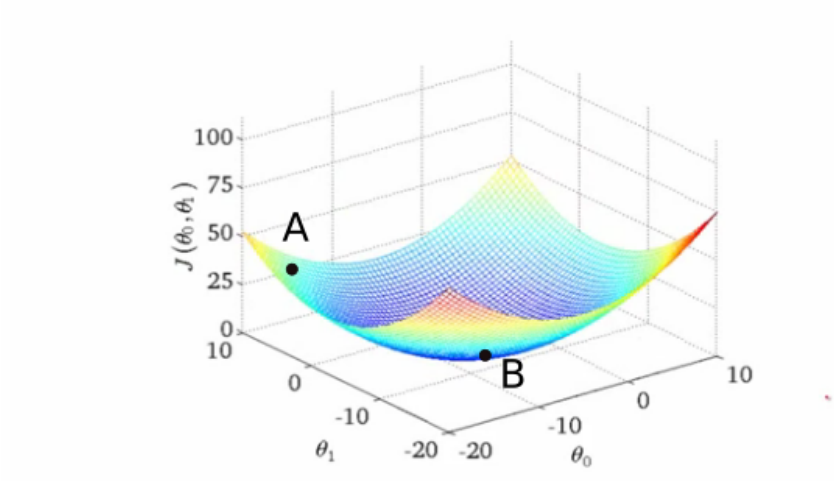


FIGURA 4.3: Estructura de la imagen de entrada
Fuente: *https://blog.paperspace.com/*

Dentro de la gradiente de descenso podemos diferenciar 3 variantes de acuerdo al la cantidad de datos que se usan para calcular la gradiente de nuestra función objetivo entre estas variantes tenemos a:

Batch gradient descent

Esta variante calcula la gradiente de descenso de una función de costo, con respecto a un parámetro θ, *para todo el conjunto de datos*. En la ecuación 4.1 podemos observar la actualización que se dará para cada ejecución. η representa la tasa o tamaño de los pasos para encontrar el mínimo local.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{4.1}$$

La ecuación 4.1 asegura la convergencia para mínimo global en una superficie convexa y mínimo local para una superficie no convexa. Entre las dificultades de este método tenemos que puede llegar a ser lento y que esta limitado por la cantidad de datos, ya que esta puede superar a la memoria nuestro computador.

Stochastic gradient descent

Otra variante es *Stochastic gradient descent* que a diferencia del método anterior, realiza las actualizaciones para cada ejemplo de entrenamiento de (x^i, y^i) de esta forma se evitan problemas como la generación de redundancia debido a que se realiza una actualización por cada ejemplo de entrenamiento.

$$\theta = \theta - \eta \nabla_\theta J(\theta, x^i, y^i) \tag{4.2}$$

En la figura 4.4 vemos que la función de costo usando SGD fluctúa demasiado esto podría representar un problema pero por el contrario, esta figura representa que el método SGD es capaz de saltar de un mínimo local a otro. Esto permite encontrar mínimos locales potencialmente mejores.

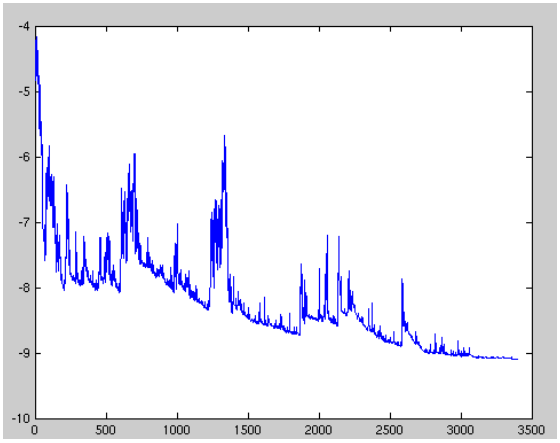


FIGURA 4.4: Función costo en SGD
Fuente: *www.doc.ic.ac.uk*

Mini-batch gradient descent

Este método pude verse como una mezcla de los 2 métodos anteriores, en lugar de aplicarlo para un conjunto entero de datos, los datos se dividen en pequeños conjuntos o mini batches, estos conjunto alimentan nuestro modelo de nuestra red.
Este método nos permite reducir la varianza de las actualizaciones de los parámetros lo cual nos permite una convergencia más estable. El tamaño de los mini-batches oscilan entre 50-250 y varían de acuerdo a su aplicación.

$$\theta = \theta - \eta \nabla_\theta J(\theta, x^{i:i+n}, y^{i:i+n}) \tag{4.3}$$

Mini-batch gradient descent es necesario elegir un ∇ adecuado. Debido que uno pequeño puede ocasionar una convergencia lenta y una grande puede ocasionar que la fluctué entre los valores mínimos y no converga.
Una de sus ventajas principales es que aprovecha el rendimiento de las GPUs para realizar cálculos más rápidos, esto se debe a que la información se guarda como tensores(Matrices de gran dimensión) y las GPUs realizan cálculos de operaciones de matrices más rápidos que una CPU. Es común encontrar a este en otras lecturas como *Stochastic gradient descent* sin realizar distinción alguna del método anteriormente descrito.

4.2.2. Optimizadores

En esta sección analizaremos y entenderemos como funcionan algunos optimizadores en el proceso de mejorar la convergencia de la gradiente de descenso.

Momentum

Las SGD tienen problemas para desplazarse en áreas con donde la superficie se curva más en una dimensión que en otra, estos lugares son los alrededores de los óptimos locales. En este escenario la SGD oscilará en la curvatura y descenderá lentamente hacia el óptimo como se muestra en la figura 4.5.

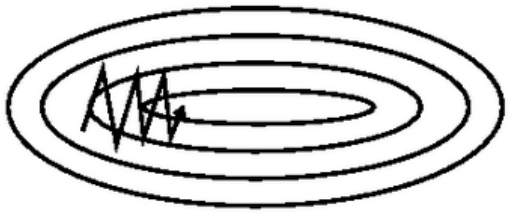


FIGURA 4.5: Actualización sin momentum
Fuente: *www.doc.ic.ac.uk*

El momentum es un método que ayuda a la SGD a acelerar en la dirección correcta, mientras evitas las oscilaciones. El momentum lográ esto añadiendo una fracción γ del vector de actualización pasado a nuestro vector presente tal como se muestra en las ecuaciones 4.4.
Un valor comúnmente elegido es $\gamma = 0.9$, en las actualización el valor del momentum aumenta para dimensiones cuyos gradientes apuntan en la misma dirección y disminuye para dimensiones en la que la gradiente cambia de dirección. Esto nos asegura que tendremos una convergencia más rápida con una oscilación reducida.En la figura 4.6 se observa gráficamente la aceleración de la convergencia en la SGD.

$$\begin{aligned} \nu_t &= \gamma\nu_{t-1} + \eta\nabla_\theta J(\theta) \\ \theta &= \theta - \nu_t \end{aligned} \tag{4.4}$$

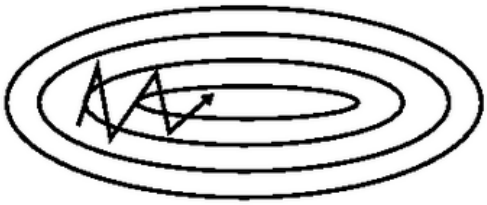


FIGURA 4.6: Actualización con momentum
Fuente: *www.doc.ic.ac.uk*

Nesterov accelerated gradient (NAG)

Este método permite que nuestro descenso sea más controlado, ya que reduce la velocidad antes de volver a subir una pendiente en nuestra superficie de la función de costo. Esta técnica es una variante de momentum donde se usa el término $\gamma\nu_{t-1}$ para mover los parámetros de θ. Al calcular el valor de $\theta - \gamma\nu_{t-1}$ nos permite tener una aproximación de donde se encontrá la siguiente posición de los parámetros. Por lo tanto no calculamos la gradiente en el parámetro θ actual sino que se calcula en una posición futura aproximada.

$$\begin{aligned} \nu_t &= \gamma\nu_{t-1} + \eta\nabla_\theta J(\theta - \gamma\nu_{t-1}) \\ \theta &= \theta - \nu_t \end{aligned} \tag{4.5}$$

En la figura 4.7 observamos el proceso. Primero el momentum calcula la gradiente actual(vector azul pequeño) y luego da un gran salto en la dirección de la gradiente actualizada acumulada (gran vector azul), el NAG primero realiza un gran salto en dirección del gradiente acumulado previo(vector marrón), luego realiza un corrección(vector rojo), esto nos da como resultado la actualización completa de NAG(vector verde). Este calculo anticipado es muy importante debido a que nos impide ir demasiado rápido y mejora la capacidad de respuesta lo cual aumenta el rendimiento de las CNN.

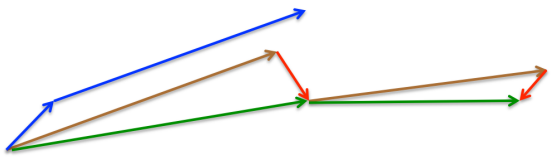


FIGURA 4.7: Convergencia Nesterov
Fuente: *www.doc.ic.ac.uk*

Adagrad

Es un algoritmo optimización basado en la gradiente de descenso, este adapta la tasa de aprendizaje. A este tipo de método se les conoce como *adaptativos*. Adagrad realiza actualizaciones más pequeñas para parámetros con características que se repiten con más frecuencia y una tasa alta para parámetros con características pocas frecuentes. Este método mejor en gran forma nuestra SGD, entre sus ventajas tenemos que es usado para entrenar redes neuronales a gran escala.

En métodos anteriores se usaba la actualización de todos los parámetros θ al mismo tiempo esto debido a que se usaba la misma tasa de aprendizaje η. Adagrad usa una tasa de aprendizaje diferente para cada parámetro θ_i en cada paso de tiempo t. En la ecuación 4.6 mostramos los cálculos de la gradiente en un tiempo t.

$$\begin{aligned} g_{t,i} &= \nabla_\theta J(\theta_{t,i}) \\ \theta_{t+1,i} &= \theta_{t,i} - \eta \cdot g_{t,i} \end{aligned} \tag{4.6}$$

El termino $\cdot g_{t,i}$ representa el valor de la gradiente en el paso de tiempo t, el cual es la derivada de la función objetivo con respecto al termino θ_i.

Adagrad modifica la idea de utilizar una tasa η fija, podemos observar el la ecuación 4.7 es una variante de la ecuación 4.6. En donde se modifica la tasa de aprendizaje en cada paso de tiempo t para todos los parámetros θ_i basándonos en los valores de las gradientes pasadas que fueron calculas para θ_i

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{4.7}$$

- $G_{t,ii}$: representa la suma de los cuadrados de las gradientes pasadas con respecto a θ_i
- ϵ es un término pequeño para evitar la división por 0. ϵ encuentra en el orden de 10^{-8}.

Como $G_t \in \Re^{dxd}$contiene la suma de los cuadrados de las gradientes pasados con respecto a todos los parámetros de θ a lo largo de la diagonal de su matriz.

Esto permite que se pueda realizar el producto matriz-vector.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_{t,i} \tag{4.8}$$

El método Adagrap en palabras de sus autores: "Informalmente nuestros procedimientos dan a las características más frecuentes tasas de aprendizaje bajas y para características poco frecuentes tasas de aprendizaje altas. Por lo tanto, la adaptación permite identificar características predictivas pero comparativamente raras. '[18]

De esta afirmación notamos que el principal beneficio de Adagrad es que nos evita el hecho de trabajar con una tasa fija por otro lado su principal desventaja se basa en el la suma de los gradientes al cuadrado aumentará en cada iteración lo cual provocará que su tasa sea cada vez más pequeña.

RMSprop

Es un método de aprendizaje por adaptación de la tasa que fue propuesto por Geoff Hinton. Este modelo se desarrollo con el objetivo resolver el problema de disminuir radicalmente la tasa de aprendizaje en Adagrad.
RMSprop divide la tasa de aprendizaje mediante el decaimiento del promedio de la suma de las gradientes al cuadrado.

$$\begin{aligned} E[g^2]_t &= \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \tag{4.9}$$

- $E[g^2]$: Promedio de la raíz de nuestro gradiente en cada peso.
- γ : Parámetro de decaimiento
- η : Tasa de aprendizaje

Divide la gradiente g_t por la raíz $\sqrt{E[g^2]_t + \epsilon}$ hace que el aprendizaje trabaje mucho mejor.

Adam

Adam es un algoritmo de optimización que desarrollado por Diederik Kingma y Jimmy ba[6] . Adaptative moment estimation o Adam, calcula una tasa de aprendizaje adaptativo para cada parámetro. Este método mantiene un decaimiento exponencial del promedio de las gradientes anteriores. El método tiene refiere los mínimos en las superficies de error. En la ecuación 4.10 mostramos el cálculo del promedio de decaimiento de las gradientes pasadas m_t y el cuadrado de las gradientes pasadas v_t

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{4.10}$$

- m_t : Primer momento (media)
- v_t : Segundo momento de la gradiente
- β_1 : Taza de decaimiento del primer momento.
- β_2 : Taza de decaimiento del segundo momento.

En la ecuación 4.11 mostramos la forma de calcular estimado de la primer y segundo momento.

$$\begin{aligned} \hat{m_t} &= \frac{m_t}{1-\beta_1^t} \\ \hat{v_t} &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{4.11}$$

- $\hat{m_t}$: Estimación del Primer momento (media)
- $\hat{v_t}$: Estimación del Segundo momento de la gradiente.

La ecuación 4.12 muestra la regla de actualización en Adam. Se utiliza el ϵ para prevenir una división por cero.

$$\theta_{t+1} = \theta_{t+1} - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t} \tag{4.12}$$

Capítulo 5

Resultados

En este capítulo se discutirán los resultados obtenidos durante el uso de los optimizadores mencionados en el capítulo anterior, con el propósito de acelerar el proceso de entrenamiento de una red neuronal convolucional.
Para estas pruebas usamos los datasets CIFAR10 y CIFAR100 que contienen 10 y 100 clases de objetos en su dataset respectivamente. Estos resultados fueron obtenidos usando una computadora con tarjeta gráfica NVIDIA GTX950M con un total de 640 núcleos y una memoria de 4GB.
Nuestra red neuronal convolucional contiene 2 capas de convolución , 2 capas pooling y un fully conected layer(ver Figura A.1 del Apéndice A.1)

5.1. Precisión

En esta sección se mostrarán los resultados de precisión para la tarea de clasificación,usando los dataset CIFAR 10 y CIFAR 100.

5.1.1. Precisión en dataset CIFAR-10

En la tabla 5.1 podemos ver la precisión para distintos optimizadores(ver Figura A.2 del apéndice A.2), en el cuadro observamos que el método Nesterov al ser una mejora de momentum obtiene una mejor precisión.
Además se observa que los métodos Adam y RMSprop obtienen los mejores resultados.(ver Figura A.5 del apéndice A.2)

CUADRO 5.1: Precisión para 5000 epochs

Método	Precisión(%)	Tiempo de ejecución(s)
Momemtum	53.04	303.07
Nesterov	54.09	305.59
Adagrad	44.42	310.20
RMSprop	68.91	322.20
Adam	68.02	316.51

En el cuadro 5.2 se realizaron 10000 iteraciones en proceso de entrenamiento de nuestra red. Los resultados de la precisión de cada método se muestra este cuadro(ver Figura A.3 del apéndice A.2).
El método de ADAM superó al RMSprop que había obteniendo mejores resultados hasta el momento, pero también se pudo observar que el método Adam produjo un resultado ligeramente mejor que el RMSprop pero también empleo un tiempo de ejecución ligeramente mayor.(ver Figura A.5 del apéndice A.2)

CUADRO 5.2: Precisión para 10000 epochs

Método	Precisión(%)	Tiempo de ejecución(s)
Momemtum	60.80	585.41
Nesterov	62.37	605.19
Adagrad	47.62	598.54
RMSprop	70.31	621.17
Adam	70.42	622.18

5.1.2. Precisión en dataset CIFAR-100

Para obtener más información respecto a los optimizadores se utilizó otro dataset el CIFAR-100 donde obtuvimos los siguientes resultados, (ver Figura A.6 del apéndice A.2) para entrenar la red neural de manera que esta sea capaz de reconocer 100 clases de objetos.

En el cuadro 5.3 observamos que los métodos Momentum y Nesterov obtuvieron resultados muy cercanos, pero el método Nesterov lo logra en un menor tiempo. Adagrad obtiene una precisión de 7.73 % la cual fue la más baja de todos los métodos. RMSprop volvió a superar a Adam para este nuevo conjunto de datos, ambos obtuvieron los mejores resultados. (ver Figura A.7 del apéndice A.2)

CUADRO 5.3: Precisión para 10000 epochs

Método	Precisión(%)	Tiempo de ejecución(s)
Momemtum	17.33	597.75
Nesterov	17.33	595.53
Adagrad	7.73	628.75
RMSprop	36.66	622.50
Adam	35,13	624.64

5.2. Función de costo

El objetivo principal de utilizar estos métodos es el de optimizar la gradiente de descenso para así reducir la función de costo o perdida. En el programa de utilizo *the cross entropy* como función de pérdida.

5.3. Función de costo en CIFAR-10

El cuadro 5.4 nos muestra los resultados de la reducción de la función de costo mostrando la disminución del error en el dataset CIFAR-10 para 5000 iteraciones(ver Figura A.8 del apéndice A.3).
En la figura A.9 del apéndice A.3, podemos observar el comportamiento de la reducción de la función de costo en 50 iteraciones o epochs.
La gráfica nos muestra que Adam empieza con un erro alto pero rápidamente logra disminuir y obtener el menor error.

La figura A.10 del apéndice A.3, muestra la tendencia final de los métodos de optimización, mostrando que los mejores métodos continúan siendo Adam y RMSprop.

CUADRO 5.4: Función costo para 5000 epochs

Método	Error en función de costo
Momemtum	1.26
Nesterov	1.38
Adagrad	1.73
RMSprop	0.19
Adam	0.30

En el cuadro 5.5 mostramos los resultados obtenidos en la reducción del error para 10000 iteraciones usando el CIFAR-10.(ver Figura A.11 del apéndice A.3). En el cuadro se observa que el método Adam obtiene el menor error en comparación a los demás métodos.

CUADRO 5.5: Función costo para 10000 epochs

Método	Error en función de costo
Momemtum	1.10
Nesterov	1.08
Adagrad	1.64
RMSprop	0.19
Adam	0.14

5.4. Función de costo en CIFAR-100

El cuadro 5.6 muestra los resultados al reducir el error en la función de costo. El cuadro muestrá que Adam y RMSprop tienen los menores errores para clasificar 100 clases de imágenes.

CUADRO 5.6: Función costo para 10000 epochs

Método	Error en función de costo
Momemtum	3.46
Nesterov	3.48
Adagrad	4.09
RMSprop	0.33
Adam	0.34

Capítulo 6

Conclusiones y Trabajo Futuro

En este capítulo se describirán las conclusiones generales que se encontraron al probar y estudiar los distintos métodos de optimización utilizados para el proceso de acelerar el entrenamiento de nuestra 2 redes neuronales convolucionales en los dataset CIFAR-10 y CIFAR-100.
Además, se propondrán algunas mejoras para que el trabajo obtenga mejores resultados.

6.1. Conclusiones

- Los métodos de optimización Adam y RMSprop obtuvieron los mejores resultados de precisión en ambas pruebas.
- A pesar de que el método de optimización Adam fue propuesto a partir del RMSprop. Adam fue superado en algunas de pruebas realizadas.
- Entre los métodos adaptativos Adam, RMSprop y Adagrad . Solo este último obtuvo los peores resultados, esto se debió a su dificultad de trabajar con la suma de las gradientes al cuadrado lo cual poco a poco redujo su tasa de aprendizaje.
- El RMSprop como una mejora del Adagrad, obtuvó mejores resultados que este último. Esto debido a que RMSprop trabaja con el promedio de la raíz de la gradiente anterior y tasas de decaimiento para controlar el problema de la disminución de la tasa de aprendizaje del método Adagrad.

- Adam es el método que tiene un decaimiento más acelerado al calcular el error en la función de costo cross-entropy.

6.2. Trabajo Futuro

El propósito general de este seminario I fue adquirir el conocimiento y experiencia necesarios para poder trabajar con redes neuronales profundas. Los métodos de optimización fueron una manera de introducirme al área de las redes convolucionales y comprender las ventajas y desventajas de algunos métodos.

Los temas de aprendizaje automático y en particular del aprendizaje profundo son muy amplios y en este seminario se trato de acoplar los temas pero no se realizó un análisis más detallado debido a la amplitud del área.

En el Seminario II se trabajará con más detalle el campo de redes neuronales convolucionales, además se tratará de diseñar un red neuronal convolucional y comparar este modelo con algunos actualmente usados. Además de realizar un implementación más interactiva .

Este seminario fue limitado debido a la capacidad de la tarjeta gráfica usada ya que en algunos ensayos la memoria era insuficiente para el futuro seminario se planea realizar las pruebas en mejores equipos como por ejemplo contratar servicios de máquinas virtuales de Amazon u otro proveedor.

Bibliografía

[1] John Nickolls William J. Dally. The gpu computing era. *IEEE Micro*, 30:56–69, 2010.

[2] César Estrebou Franco Ronchetti, Facundo Quiroga and Laura Lanzarini. Handshape recognition for argentinian sign language using probsom. *JCST*, 16(1):1–5, 2010.

[3] Derek Rose Itamar Arel and Thomas Karnowski. Deep machine learning - a new frontier in artificial intelligence research [research frontier]. *IEEE Computational Intelligence Magazine*, pages 1–7, 2010.

[4] Vadim Smolyakov. Neural network optimization algorithms a comparison study based on tensorflow. `https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5`, Ene 2017. Accessed on 2018-06-11.

[5] Adam Coates Abhik Lahiri Bobby Prochnow Quoc V. Le, Jiquan Ngiam and Andrew Y. Ng. On optimization methods for deep learning. *International Conference on Machine Learning 2010*, pages 1–8, 2010.

[6] Diederik P. Kingma and Jimmy Lei Ba. Adam : A method for stochastic optimization. *ICLR*, pages 1–15, 2015.

[7] Timothy Dozat. Incorporating nesterov momentum into adam. *ICLR*, pages 1–3, 2016.

[8] Machine learning 101 | supervised, unsupervised, reinforcement and beyond. `https://medium.com/deep-math-machine-learning-ai/different-types-of-machine-learning-and-their-types-34760b9128a2`, Sep 2017. Accessed on 2018-05-11.

[9] David Fumo. Types of machine learning algorithms you should know. https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861, Jun 2017. Accessed on 2012-05-11.

[10] Jason Brownlee. Linear regression for machine learning. https://medium.com/deep-math-machine-learning-ai/different-types-of-machine-learning-and-their-types-34760b9128a2, Mar 2016. Accessed on 2018-05-12.

[11] Essentials of machine learning algorithms (with python and r codes). https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/, Sep 2017. Accessed on 2018-05-14.

[12] K nearest neighbors - classification). http://www.saedsayad.com/k_nearest_neighbors.htm. Accessed on 2018-05-15.

[13] Naive bayesian). http://www.saedsayad.com/naive_bayesian.htm. Accessed on 2018-05-15.

[14] Sunil Ray. 6 easy steps to learn naive bayes algorithm (with codes in python and r). https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/, Sep 2017. Accessed on 2018-05-13.

[15] Josh Patterson and Adam Gibson. Major architectures of deep networks. In *Deep Learning A Practitioner's Approach*, pages 132–135. O'Reilly Media, 2017.

[16] Adrian Rosebrock. Lenet – convolutional neural network in python. https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python, Ago 2016. Accessed on 2018-06-15.

[17] Siddharth Das. Cnn architectures: Lenet, alexnet, vgg, googlenet, resnet and more .... https://medium.com/@siddharthdas_32104/cnns-

architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5, Nov 2017. Accessed on 2018-06-15.

[18] Elad Hazan John Duchi and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12*, pages 1–2, 2011.

Apéndice A

Arquitectura de la red y Resultados obtenidos

A.1. Capas de la red neuronal convolucional

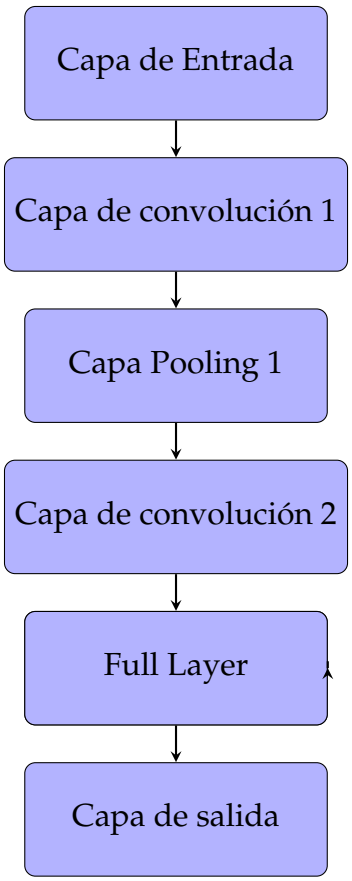
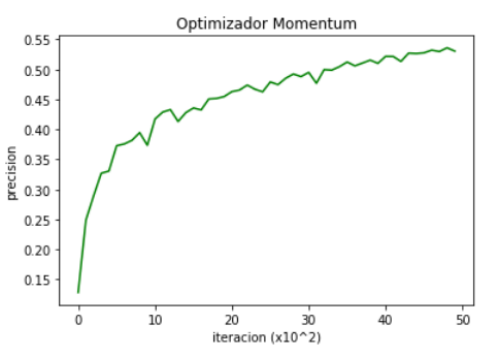


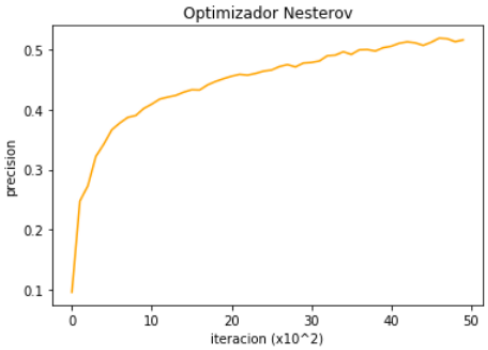
FIGURA A.1: Capas de la red neuronal usada
Fuente: *Fuente Propia*

A.2. Resultados de precisión de entrenamiento

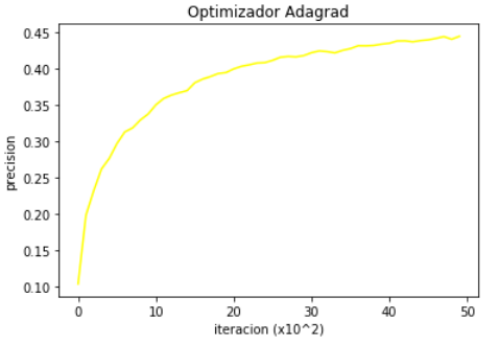
A.2.1. CIFAR-10



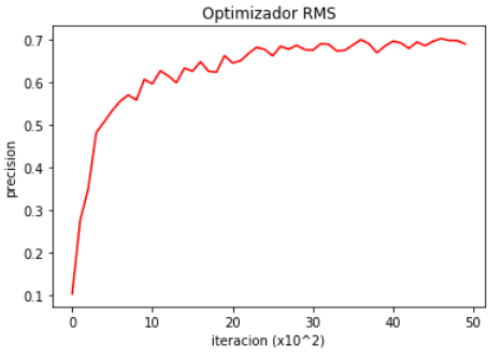
(A) fig 1



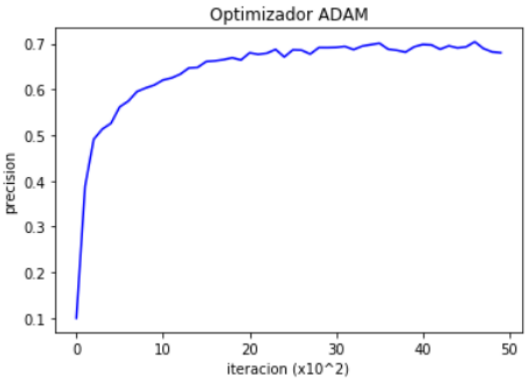
(B) fig 2



(C) fig 3

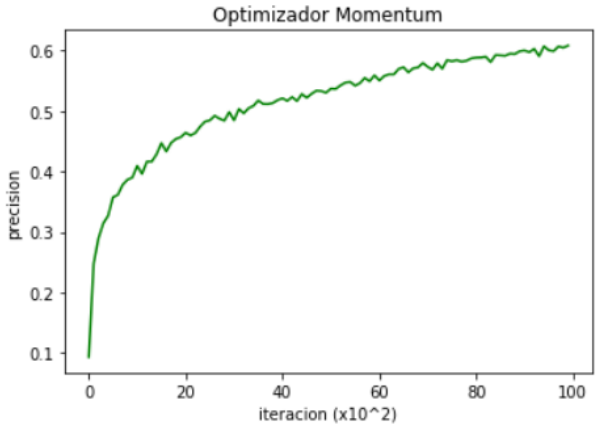


(D) fig 3

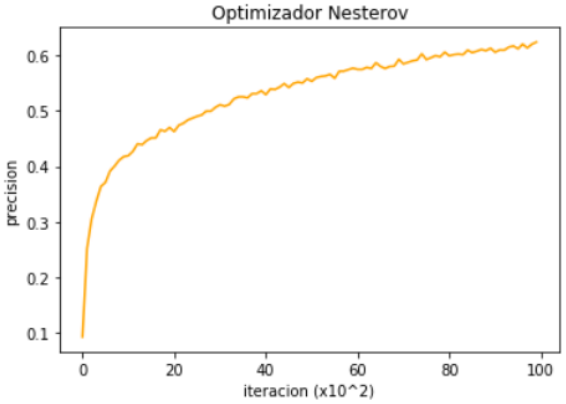


(E) fig 4

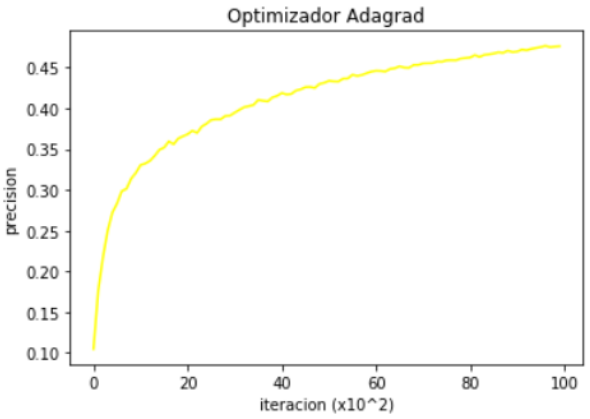
FIGURA A.2: optmizadores 5000 epochs
Fuente :*Fuente Propia*



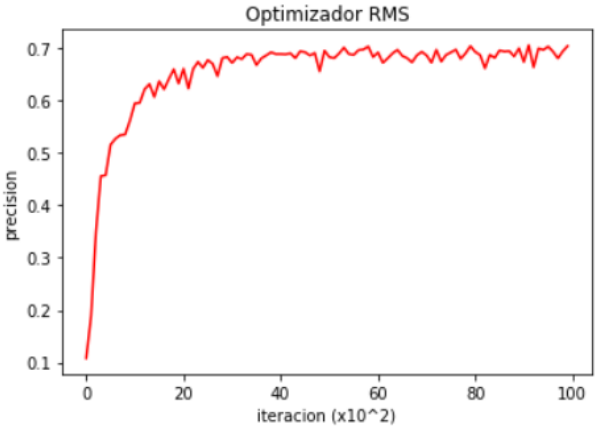
(A) fig 1



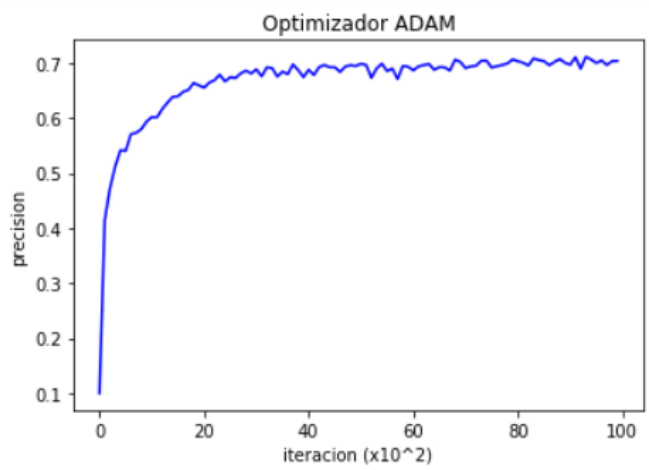
(B) fig 2



(C) fig 3

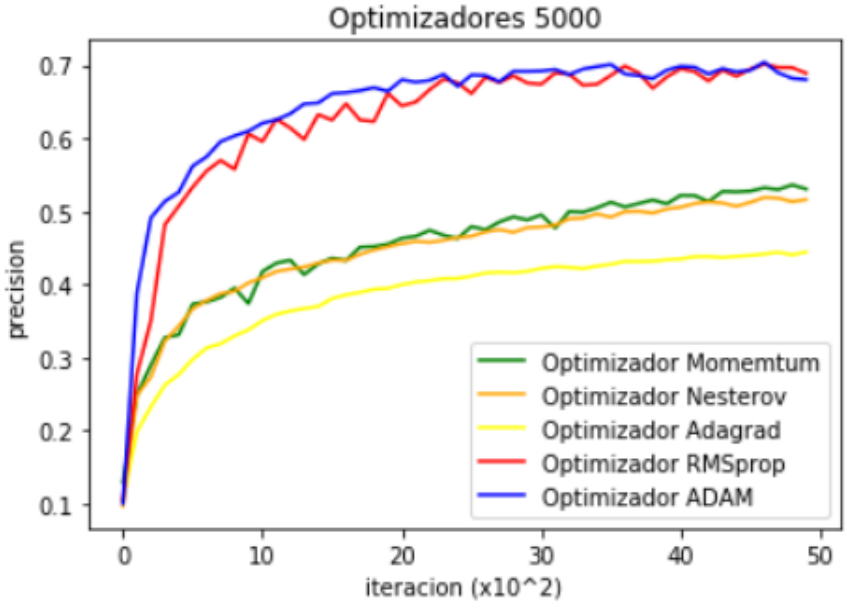


(D) fig 3



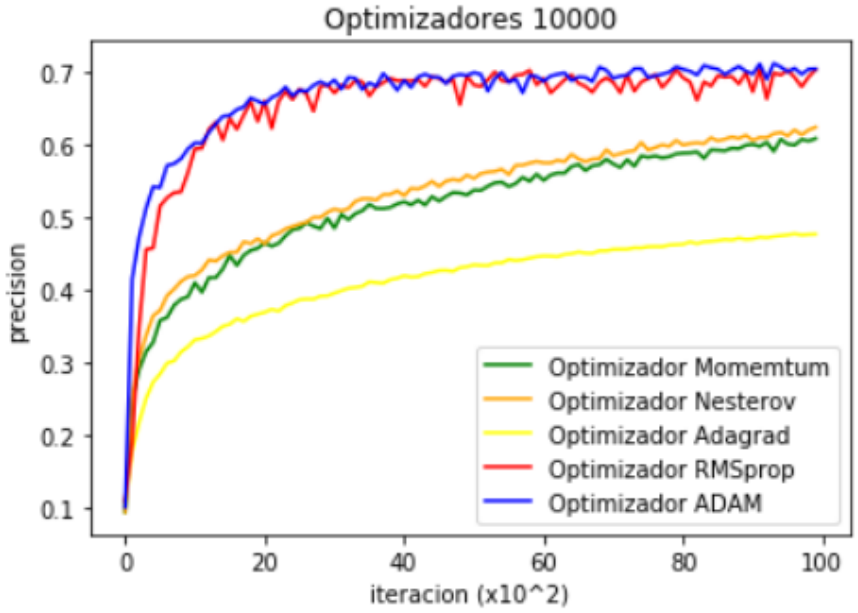
(E) fig 4

FIGURA A.3: optmizadores 10000 epochs
Fuente: *Fuente Propia*



(A) fig 1

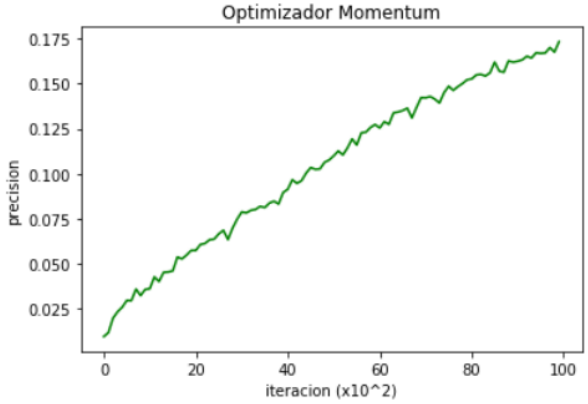
FIGURA A.4: Comparación de precisión de optimizadores para 5000 epochs
Fuente: *Fuente Propia*



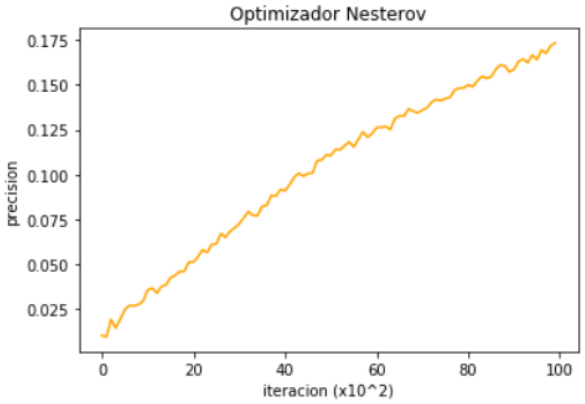
(A) fig 1

FIGURA A.5: Comparación de optimizadores para 10000 epochs
Fuente: *Fuente Propia*

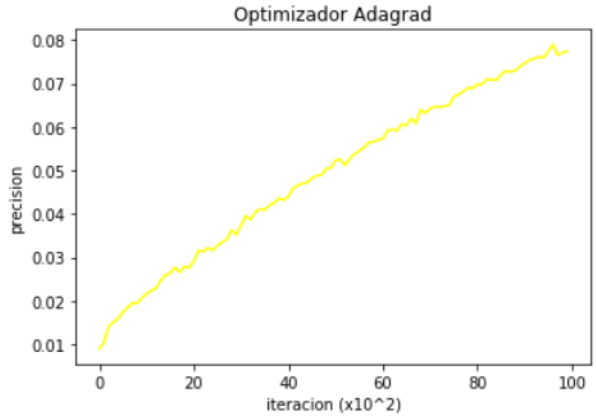
A.2.2. CIFAR-100



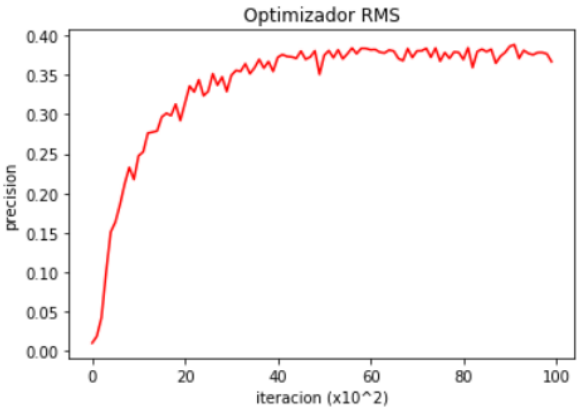
(A) fig 1



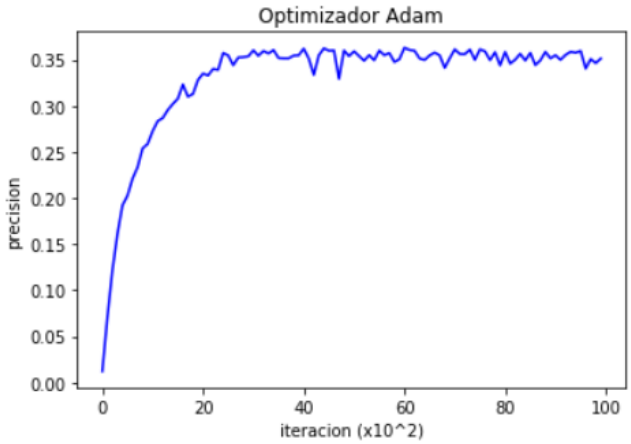
(B) fig 2



(C) fig 3

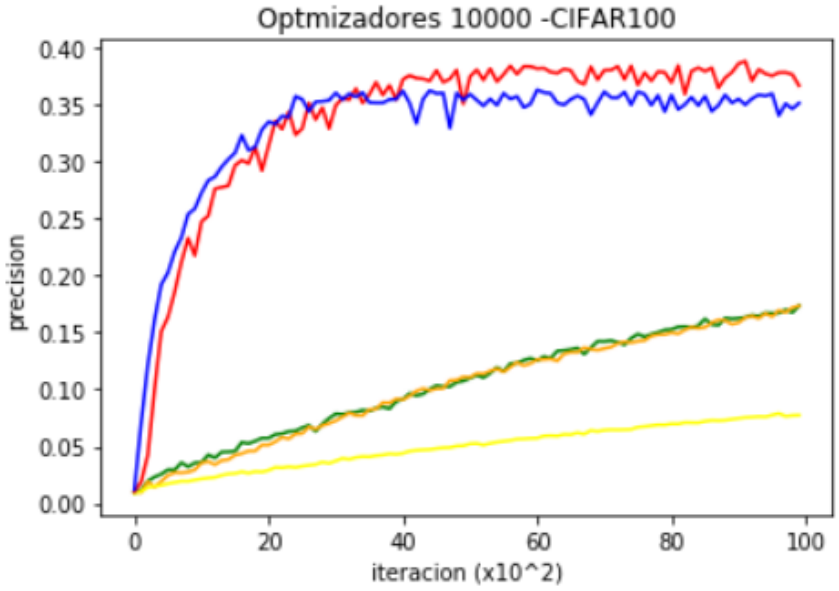


(D) fig 3



(E) fig 4

FIGURA A.6: optmizadores 10000 epochs
Fuente: *Fuente Propia*

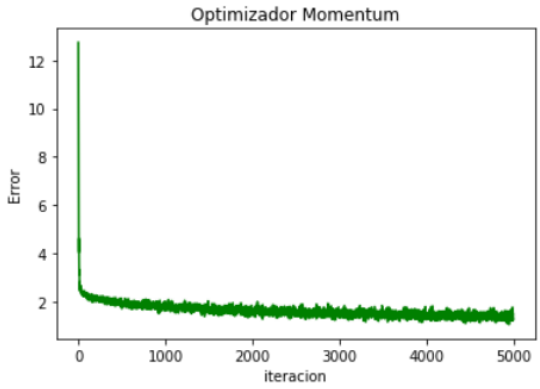


(A) fig 1

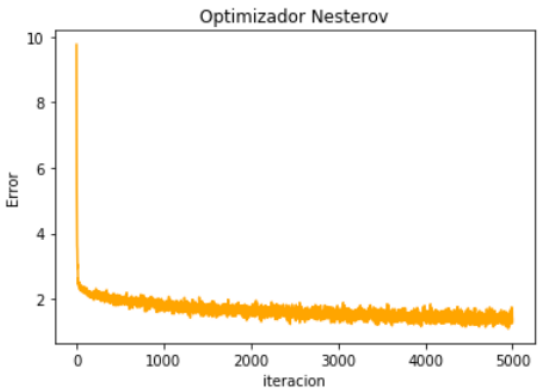
FIGURA A.7: Comparación de optimizadores para 10000 epochs - CIFAR100
Fuente: *Fuente Propia*

A.3. Resultados del error en el entrenamiento

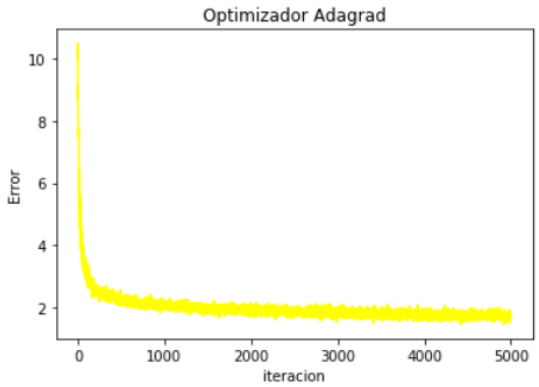
A.3.1. CIFAR-10



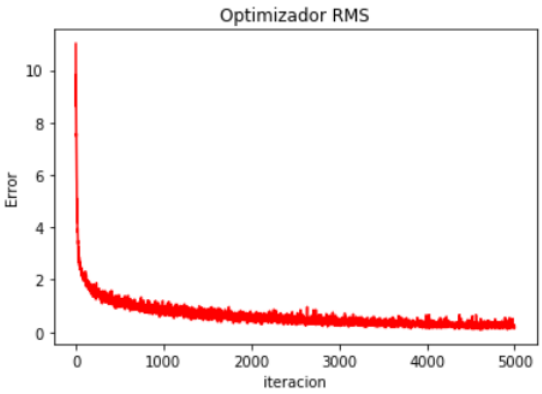
(A) fig 1



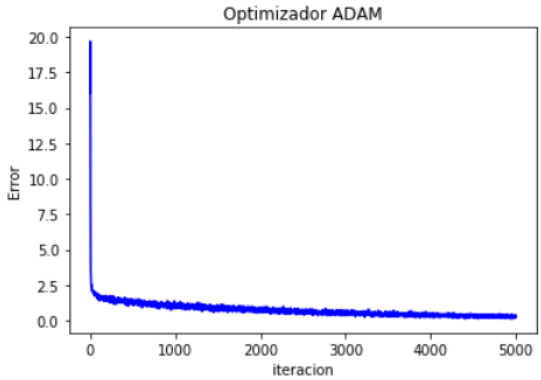
(B) fig 2



(C) fig 3

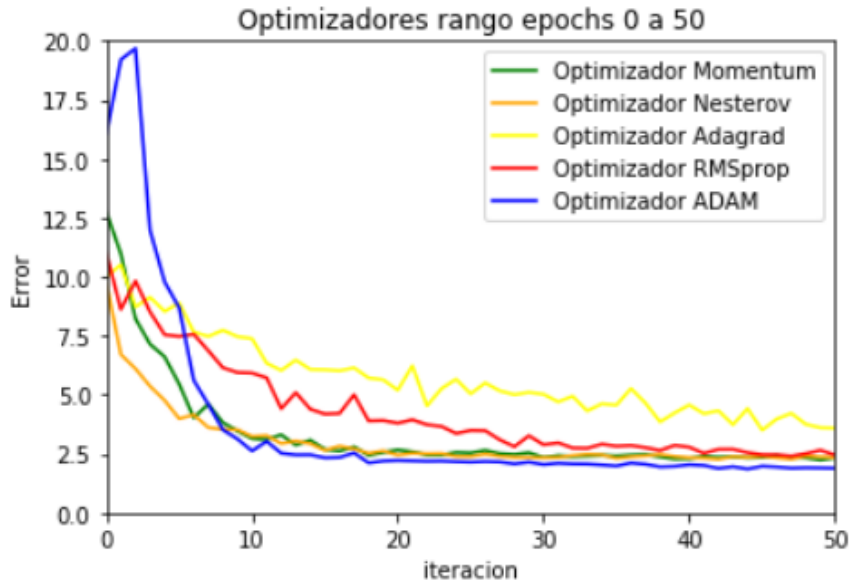


(D) fig 3



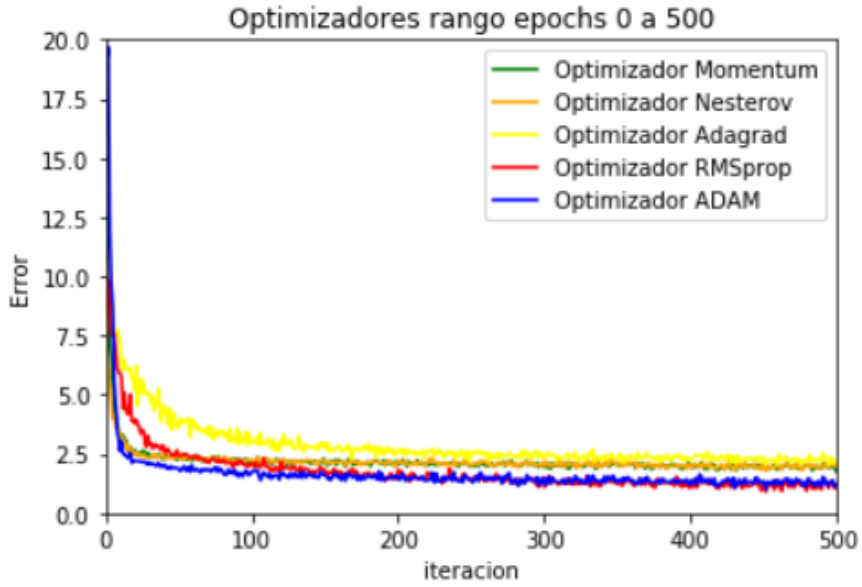
(E) fig 4

FIGURA A.8: Error en los optimizadores 5000 epochs
Fuente: *Fuente Propia*



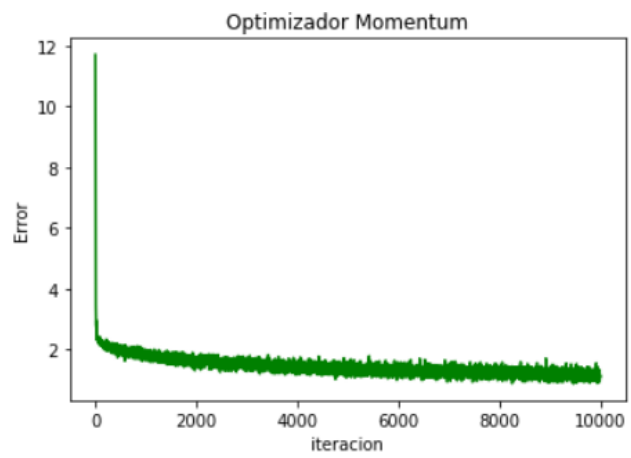
(A) fig 1

FIGURA A.9: Comparación de las funciones de costo rango 0-50
Fuente: *Fuente Propia*

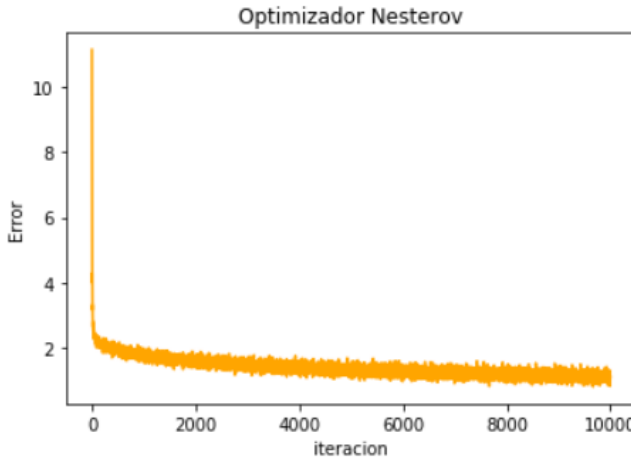


(A) fig 1

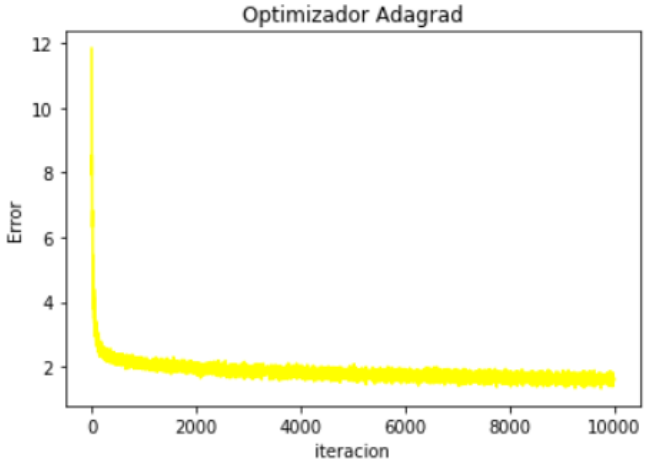
FIGURA A.10: Comparación de las errores rango 0-500
Fuente: *Fuente Propia*



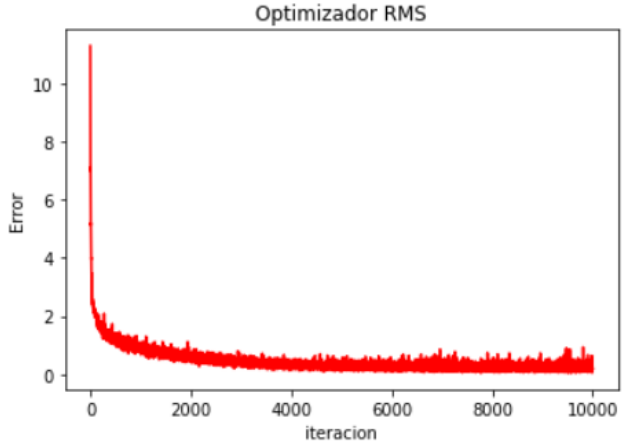
(A) fig 1



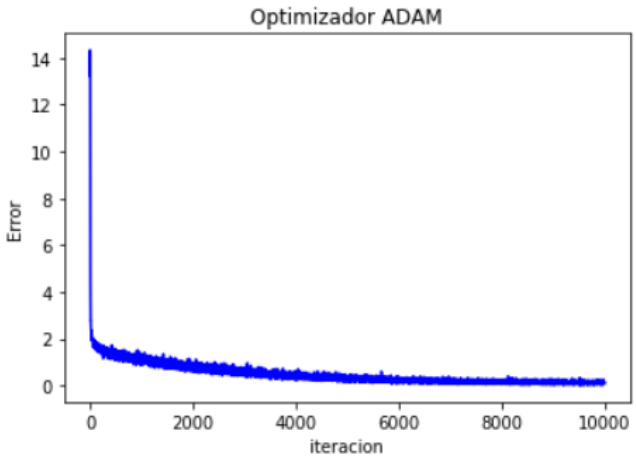
(B) fig 2



(C) fig 3



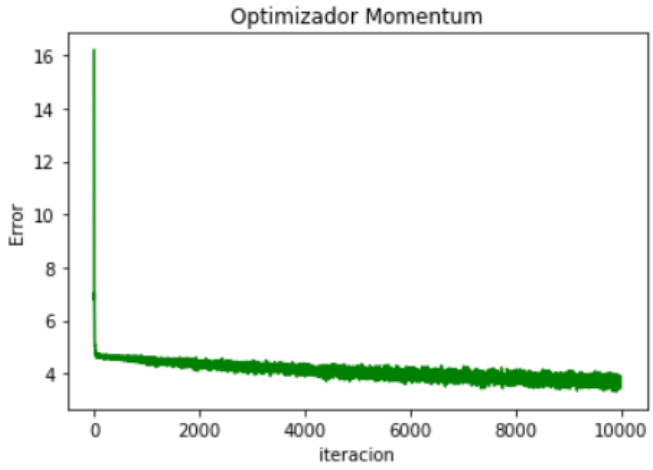
(D) fig 3



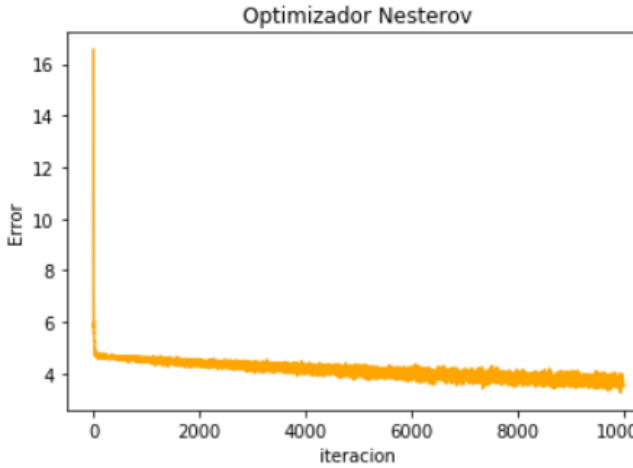
(E) fig 4

FIGURA A.11: Error en los optimizadores 10000 epochs -CIFAR10
Fuente: *Fuente Propia*

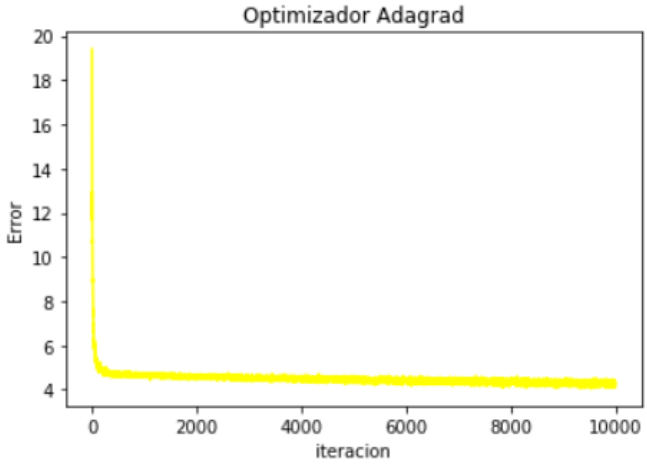
A.3.2. CIFAR-100



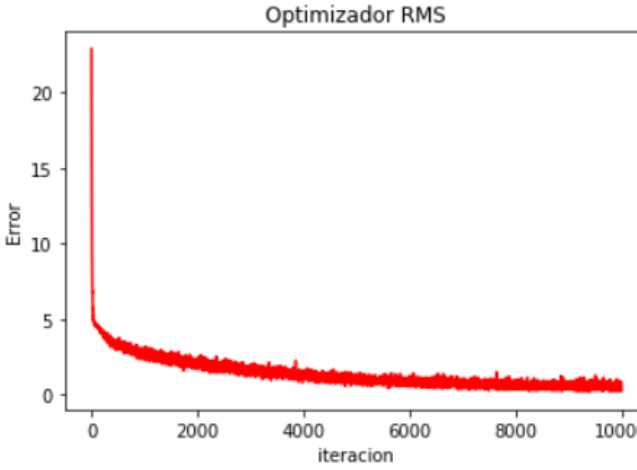
(A) fig 1



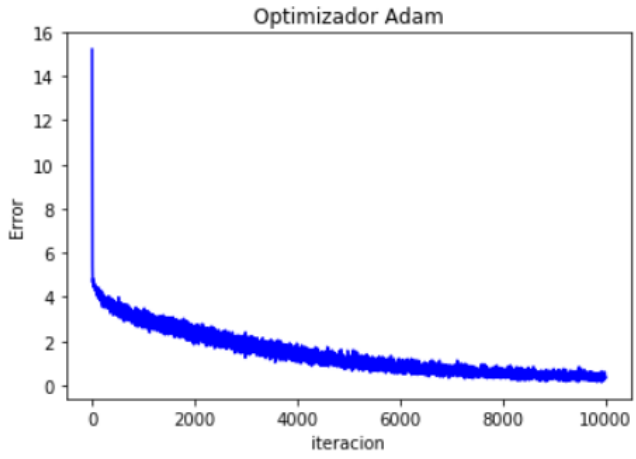
(B) fig 2



(C) fig 3

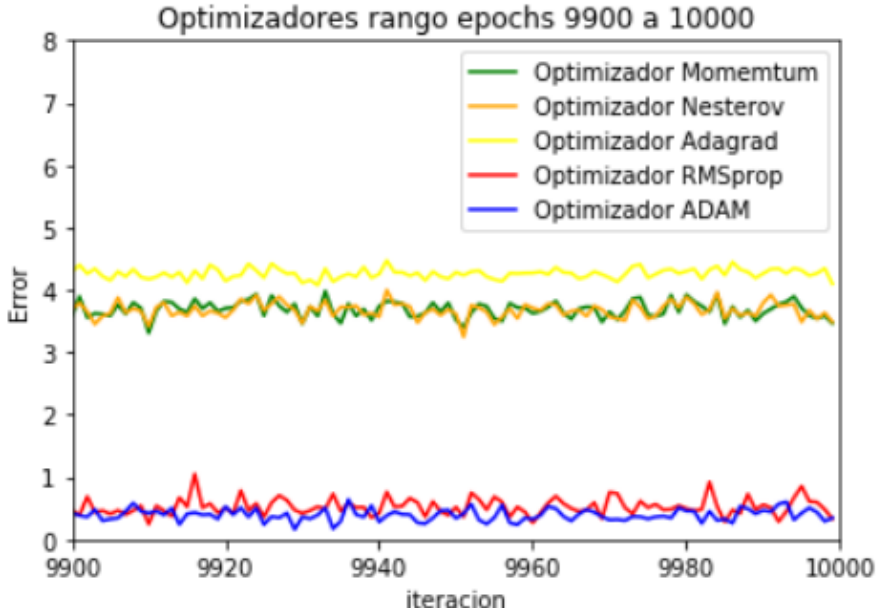


(D) fig 3



(E) fig 4

FIGURA A.12: Error en los optimizadores con 10000 epochs - CIFAR 100
Fuente: *Fuente Propia*



(A) fig 1

FIGURA A.13: Comparación los errores rango 9900-10000
Fuente: *Fuente Propia*