



UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

Comparación de métodos de optimización en una red neuronal convolucional para tareas de clasificación

SEMINARIO DE TESIS 1

Autor: Víctor Jesús Sotelo Chico

Asesor: Víctor Melchor Espinoza

Junio, 2018

Resumen

Índice general

Resumen	III
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del Seminario	3
2. Estado del Arte	4
2.1. GPU computing	4
2.1.1. The GPU computing Era	4
2.2. Machine learning	5
2.2.1. Uso de redes neuronales para encontrar el rendimiento de una GPU	5
2.3. Deep Learning	5
2.3.1. Deep Machine Learning - A New Frontier in Artificial Intelligence	5
2.4. Métodos de optimización	6
2.4.1. Neural Network Optimization Algorithms: A comparison study based on TensorFlow	6
2.4.2. On Optimization Methods for Deep Learning	6
2.5. Conclusiones	7
3. Machine Learning y Redes Neuronales	8
3.1. Aprendizaje Automático	8
3.1.1. Aprendizaje Supervizado	9
3.1.2. Aprendizaje No Supervizado	19

3.1.3.	Aprendizaje por refuerzo	20
3.2.	Redes Neuronales	21
3.2.1.	Neuronas	21
	Funciones de Activación	22
3.2.2.	Redes Neuronales Artificiales	23
	Redes Neuronales Prealimentadas	24
	Algoritmo de propagación hacia atrás	25
	Conjunto de datos para una red neuronal	26
4.	Algoritmos de optimización una Red Neuronal convolucional	29
4.1.	Redes Neuronales Convolucionales	29
4.1.1.	Estructura de una imagen	30
4.1.2.	Capas de una CNN	31
	Input layer	31
	Convolutional layers	31
	Pooling layers	33
	Fully Connected Layers	33
4.1.3.	Arquitecturas conocidas	34
4.2.	Métodos de optimización	34
4.2.1.	Gradiente de descenso	34
	Batch gradient descent	34
	Stochastic gradient descent	35
	Mini-batch gradient descent	35
4.2.2.	Optimizadores	36
	Momentum	36
	Nesterov accelerated gradient	37
	Adagrad	38
	RMSprop	39
	Adam	39
	AdaMax	40
	Nadam	40

5. Resultados	41
5.1. Conclusiones	41
6. Conclusiones y Trabajo Futuro	42
6.1. Conclusiones	42
6.2. Trabajo Futuro	42
A. Título del apéndice	47

Índice de figuras

3.1. Regresión y clasificación Fuente: https://medium.com/	10
3.2. Regresión Lineal Fuente: www.forexmt4indicators.com/	11
3.3. Regresión Logística Fuente: www.analyticsvidhya.com	13
3.4. knn Fuente: www.medium.com/	14
3.5. transformación con la función kernel Fuente: www.statsoft.com	15
3.6. transformación para un problema de regresión Fuente: www.saedsayad.com	17
3.7. Clustering Fuente: https://medium.com/	19
3.8. k means clustering Fuente: www.saedsayad.com	20
3.9. Esquema de aprendizaje por refuerzo Fuente: https://towardsdatascience.com	21
3.10. Funciones de activación Fuente: https://ujjwalkarn.me	23
3.11. Redes neuronales biológicas y artificiales Fuente: https://medium.com	23
3.12. Esquema de Redes Neuronales Prealimentadas Fuente: https://ujjwalkarn.me	24
3.13. Propagación hacia atrás Fuente: https://ujjwalkarn.me	26
3.14. División del dataset Fuente: http://magizbox.com	27
4.1. Estructura de la imagen de entrada Fuente: <i>Deep Learning by Adam Gibson, Josh Patterson</i>	30
4.2. Operacion de convolución Fuente: www.openresearch.ai	32
4.3. Función costo en SGD Fuente: www.doc.ic.ac.uk	35
4.4. Actualización sin momentum Fuente: www.doc.ic.ac.uk	36
4.5. Actualización con momentum Fuente: www.doc.ic.ac.uk	37
4.6. Convergencia Nesterov Fuente: www.doc.ic.ac.uk	38

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 en estos años de estudio

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. 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 hubiese sido considerado imposible. El hecho de lograr que las computadoras sean capaces de reconocer objetos, clasificarlos lo cual ha permitido que la industria de la robótica desarrolle de manera acelerada en las últimas décadas. 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 incrementa lo cual se convierte un problema importante.

Una de la soluciones que apareció fue el uso de la GPU's para acelerar procesos como el entrenamiento de una red neural con muchas capas ocultas, las GPU's representa 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 esta dominado por NVIDIA y AMD esta competencia y la alta demanda permite que las GPU's tengan mejor rendimiento lo cual puede ser usado obtener mejores resultados en el campo de machine learning.

Por otro lado la optimización no solo se basa el uso de hardwares más potentes sino también depende de la elección de métodos adecuados para nuestro modelos esta elección dependerá mucho del problema a tratar métodos un método usado comúnmente en el campo de machine learning es la gradiente de descenso estocástica pero realmente es adecuado para toda variedad de problemas.

Respecto a la problemática de encontrar métodos más eficientes de optimización que obtengan un mejor rendimiento, este presente seminario se centra en la búsqueda y comparación de estos métodos con el fin de encontrar aquellos que sean más rápidos y eficientes.

1.2. Objetivos

El objetivo de este seminario es el de mostrar las ventajas del uso de distintos métodos de optimización en el entranimiento de una deep neural network 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 machine learning.
- Conocer las ventajas y desventajas de diferentes métodos de optimización.
- Mostrar los resultados de distintos métodos de optimización en el entramiento de una red neuronal profunda.

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 deep learning.
- Obtener un conocimiento de las herramientas y recursos que existen actualmente para abordar problemas de deep learning, 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:**

En este capítulo muestra los trabajos e investigaciones ya realizadas, además de algunas aplicaciones que motivaron al presente seminario y muestra la importancia del seminario.

Capítulo 2

Estado del Arte

En este capítulo describiremos anteriores investigaciones de machine learning, además de sus aplicaciones. Además veremos algunas investigaciones GPU como un modo de obtener un mejor rendimiento y nos enfocaremos principalmente en los estudios de los métodos de optimización.

También mostraremos investigaciones referentes a deep learning exclusivamente nos enfocaremos a Convolutional Neural Network(CNN) ya que son parte del tema de estudio en este seminario.

2.1. GPU computing

El uso de las GPU's han permitido lograr aplicaciones que antes podriamos imaginar que eran imposibles debido a su largo tiempo de ejecución. Hoy en día las GPU's son altamentes usadas debido 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

En artículo se enfoca principalmente en describir la evolución que sufrieron las arquitecturas de GPU's, además de mostrar la importancia del uso de las GPU's para un mayor rendimiento y eficiencia que antes hubiesen sido consideradas imposibles de bido al alto tiempo de ejecución que requerian.

Además nos muestra que la escalabilidad es la principal característica que ha permitido que las GPU's aumenten su paralelismo y rendimiento.

2.2. Machine learning

El uso de machine learning 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

En la actualidad existen empresas dedicadas a la creación de GPU's, en el proceso una parte fundamental es la verificación del rendimiento de las GPU's actualmente existen simuladores conocidos como GPGPU-SIM que permiten estimaciones precisas pero estos poseen algunas dificultades como el tiempo empleado en configurarlos en base al hardware real además que este proceso esta propenso a errores. Un equipo de equipo conformado por investigadores de AMD y The University of Texas at Austin, quienes propusieron el uso de redes neuronales para predecir el rendimiento.

2.3. Deep Learning

Dentro del área de machine learning encontramos deep learning o aprendizaje profundo el cual consiste en un conjunto de algoritmos que modela abstracciones de alto nivel.

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

Este trabajo de investigación fue realizado por investigadores oak Ridge National Laboratory y University of tennessee, el objetivo principal de

este trabajo fue presentarnos el aprendizaje profundo como un camino para la imitación del cerebro humano y sus principales cualidad como el reconocimientos de objetos, rostros, etc.

Además de mostrarnos las aplicaciones del aprendizaje profundo, como : análisis de documentos, detección de voz, rostro, procesamiento natural del lenguaje, etc.

Actualmente existen algunas empresas privadas que apoyan el campo de deep learning con el objetivo de buscar sus aplicaciones comerciales entre estas empresas tenemos: Numenta y Binatix.

2.4. Métodos de optimización

El campo de machine learning 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 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 realizo una prueba comparativa con una arquitectura simple de CNN usando el conjunto de datos del MNIST. "Se comparo diferentes optimizadores y se obtuvo que SGD con Nesterov y Adam producen mejores resultados en el entrenamiento de una CNN simple usando tensorflow "[1]

2.4.2. On Optimization Methods for Deep Learning

Un equipo de la universidad de standford realizó una pruebas con el objetivos de encontrar métodos adecuados para un entrenamiento en deep

learning. El equipo se percató de lo común que resulta el uso de Gradiente de descenso estocástico o SGD por sus siglas en inglés en deep learning . Se realizaron pruebas con otros métodos de optimización como la gradiente conjugada y Limited memory BFGS(L-BFGS) los cuales permitieron acelerar el proceso de entrenamiento de algoritmos de deep learning 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. ”

2.5. Conclusiones

A medida que tratamos muchos problemas vemos la necesidad de encontrar optimizadores adecuados para los tipos de problemas. En el área de deep learning comumente se trabaja en el campo de reconocimiento de imágenes a pesar de la mejora mediante el uso de GPU's este tipo de problemas necesitan métodos óptimos para obtener una mejor performance.

Capítulo 3

Machine Learning y Redes Neuronales

En este capítulo trataremos los principales conocimientos de Aprendizaje automático como su clasificación y su 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 realice 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: Supervizado, No supervisado, Aprendizaje con refuerzo.[2]

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 sus etiquetas.

“El aprendizaje supervisado trata de modelar la relación entre el resultado de la predicción y las características de las entradas de manera que se puede predecir nuevos valores para un nuevo conjunto de datos ”[3]

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. ”[2]

“El objetivo de los problemas de clasificación es asignar las observaciones en categorías discretas en lugar de estimar valores continuos. ”[3]

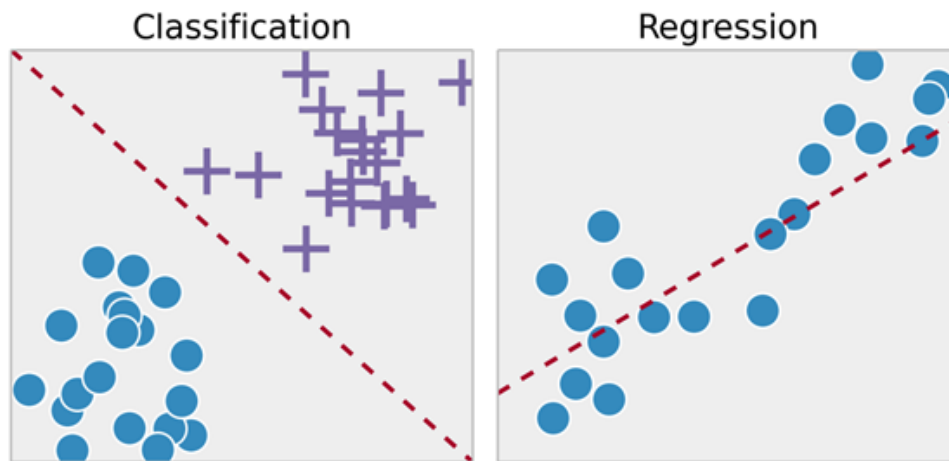


FIGURA 3.1: Regresión y clasificación

Fuente: <https://medium.com/>

Algoritmos de Aprendizaje Supervizado

Regresión Lineal

“El algoritmo de regresión lineal asume que existe una relación entre las variables de entrada $x = (x_1, \dots, x_n)$ 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. ”[4] .Es comúnmente usado para estimar valores reales en base a variables continuas. La figura 3.2 muestra una regresión lineal simple.

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n \quad (3.1)$$

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 b_0 y b_1 de manera que se reduzca la siguiente función de costo.

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

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

Un método muy importante es la gradiente de descenso que se usa para actualizar los valores de los b_i de manera que se reduzca la función de costo J de la ecuación 3.2.

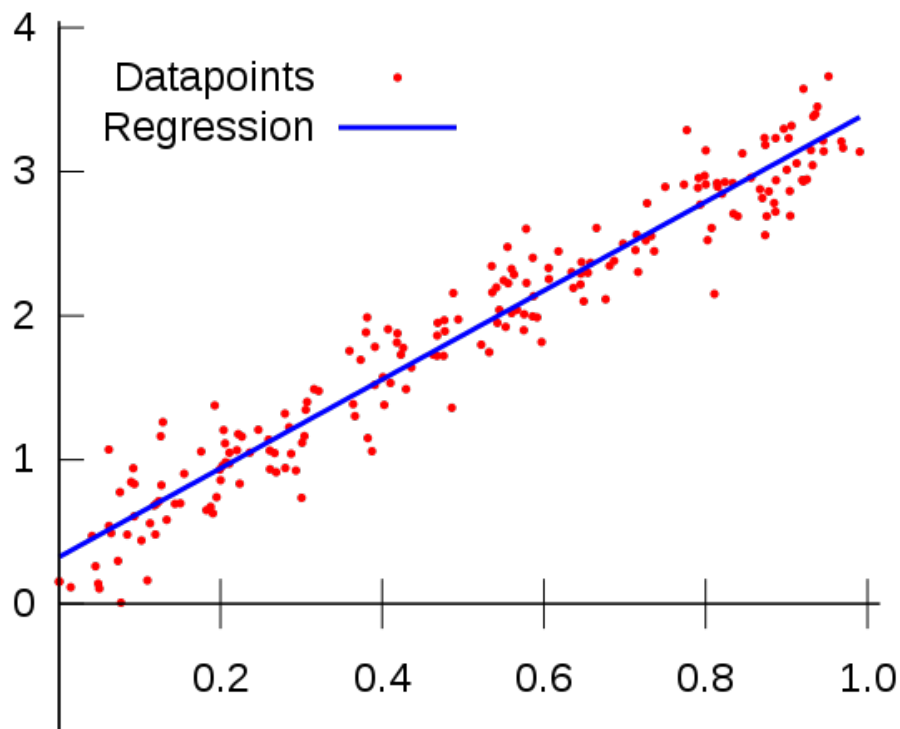


FIGURA 3.2: Regresión Lineal
Fuente: www.forexmt4indicators.com/

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 describen por un número finito de categorías.

La regresión Logística es usado para problemas de clasificación lo hace mediante la predicción de que una salida Y sea dicotoma es decir que solo tenga 2 posibles valores. la regresión produce una curva la cual produce valores entre 0 y 1. "Matemáticamente podemos como que las salidas están modeladas como una combinación de los predictores lineales." [5]

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

- 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:

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

En la ecuación 3.3, Y define la función logística que se muestra en la figura 3.3. Esta forma también se puede conocer como la función sigmoideal 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.

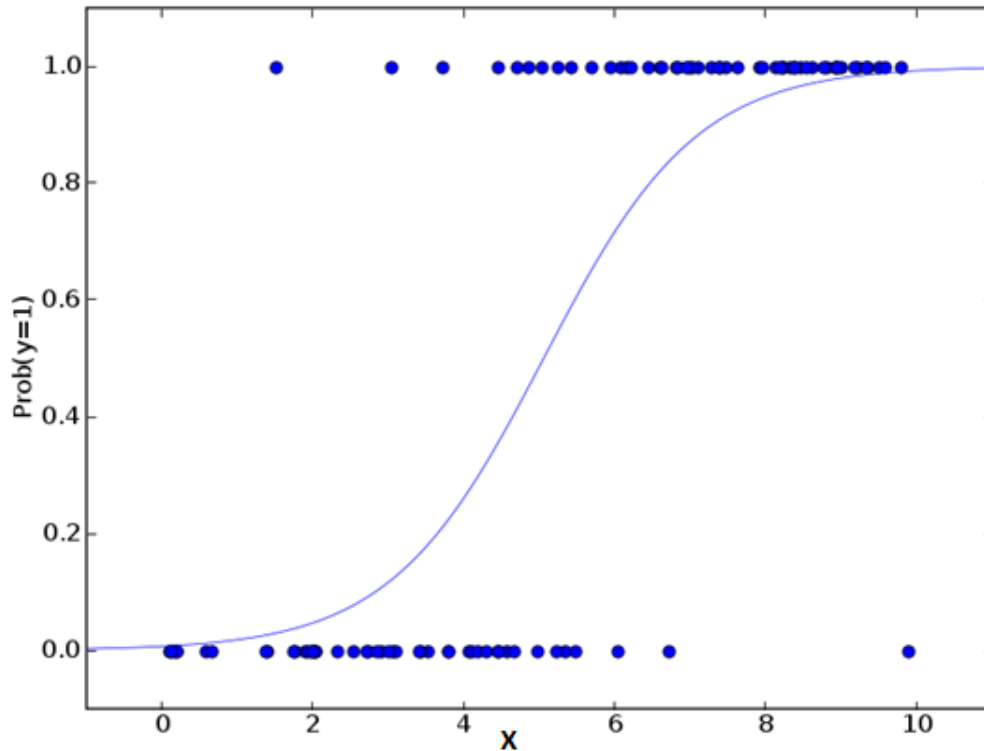


FIGURA 3.3: Regresión Logística

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.

$$x^i = (x_1^i, x_2^i, \dots, x_n^i) \quad (3.5)$$

$$d_E(x^i, x^j)$$

- x^i : objeto con n características.

Definimos d_E como la función distancia entre los vectores x_i y y_i están función distancia pueden una de las siguientes clasificaciones:

- 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 para variables continuas. Para el caso de variables categóricas debería usarse la distancia de Hamming cuya definición se muestra en la ecuación 3.6

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

$$x = y \implies D = 0$$

$$x \neq y \implies D = 1$$

“La elección de un valor óptimo de 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.”[6]

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

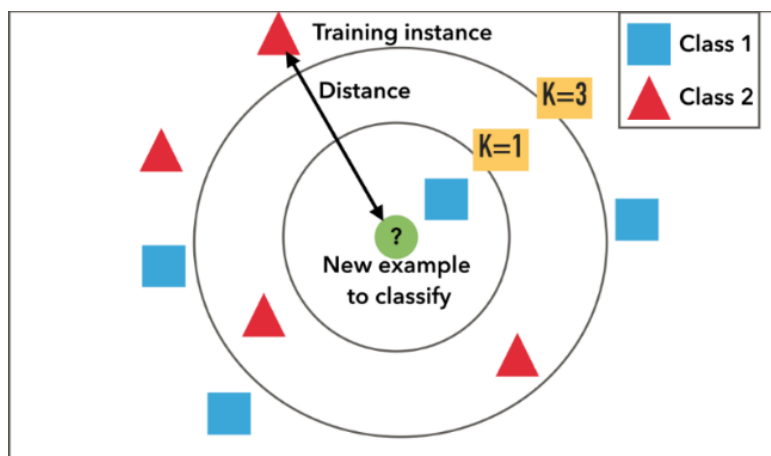


FIGURA 3.4: knn
Fuente: www.medium.com/

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 transforma una forma requerida algunas de estas funciones son:

- 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.

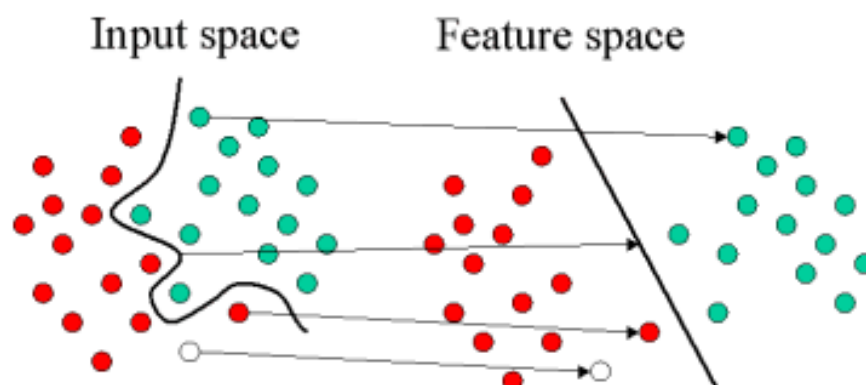


FIGURA 3.5: transformación con la función kernel

Fuente: www.statsoft.com

Podemos dividir SVM en 2 categorías:

Super Vector Classificacion

Los SVC realizan la tarea de clasificación encontrando un hiperplano que maximeze 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

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

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 el 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: $y = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \text{ker}(x_i, x) + b$

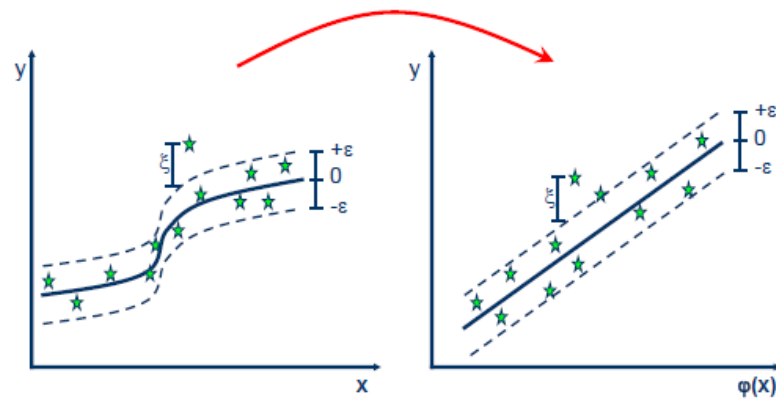


FIGURA 3.6: transformación para un problema de regresión

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 que estimar los parámetros iterativamente lo cual lo hace particularmente útil para un gran conjunto de datos ”[7].

Como la técnica de clasificación esta basada en el teorema de bayes supone que las características 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** ”[8]

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \quad (3.7)$$

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

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

- $P(x|c)$: es la 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 utilizando la estructura de un árbol. La idea se basa en dividir el conjunto de datos en subconjuntos cada vez más pequeños a su vez se 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 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 los arboles 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 la información ganada por la desviación standard de reducción. Este tipo

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

Neural Networks

Los modelos de redes neuronales fue inspirado de las redes neuronales biológicas y realizan tareas complejas como reconocimiento de imágenes El tema de Neural Network será tratado con más detenimiento en la siguiente sección.

3.1.2. Aprendizaje No Supervizado

Mientras que el aprendizaje supervisado aprende de un conjunto de datos de entrenamiento con respuestas o etiquetas correctas. En el aprendizaje no supervisado los datos de entrenamiento no poseen ninguno tipo de etiqueta, el sistema debe de interpretar los datos por si mismo. El aprendizaje no supervisado es usado principalmente para el reconocimiento de patrones y modelado descriptivo.

Clustering

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

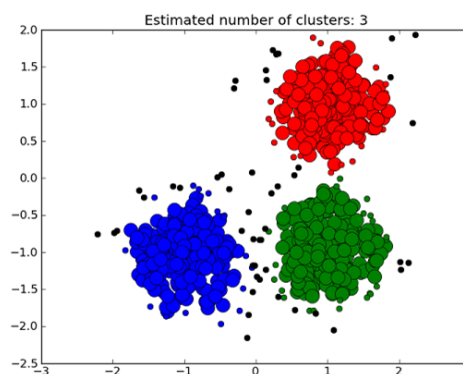


FIGURA 3.7: Clustering
Fuente: <https://medium.com/>

K-means Clustering

El algoritmo intenta particionar 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.

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

- J : función de error cuadrado
- k : número de cluster
- n : número de casos.
- x_i^j : caso i
- c_j : centroide del cluster j

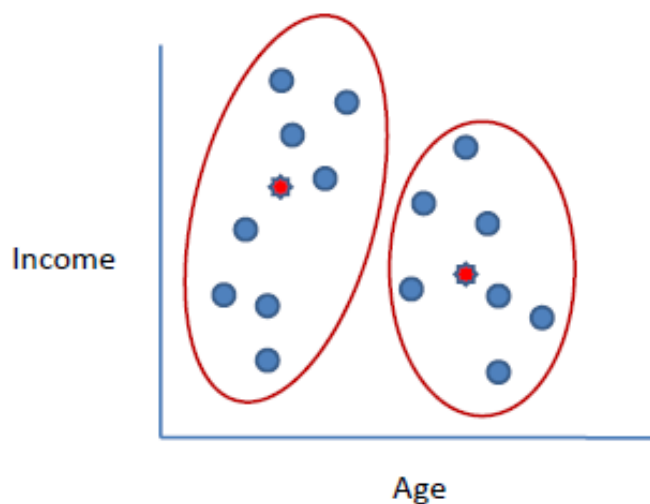


FIGURA 3.8: k means clustering
Fuente: www.saedsayad.com

3.1.3. Aprendizaje por refuerzo

Este tipo de aprendizaje fue inspirado por la psicología conductista, este tipo 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. "[3]

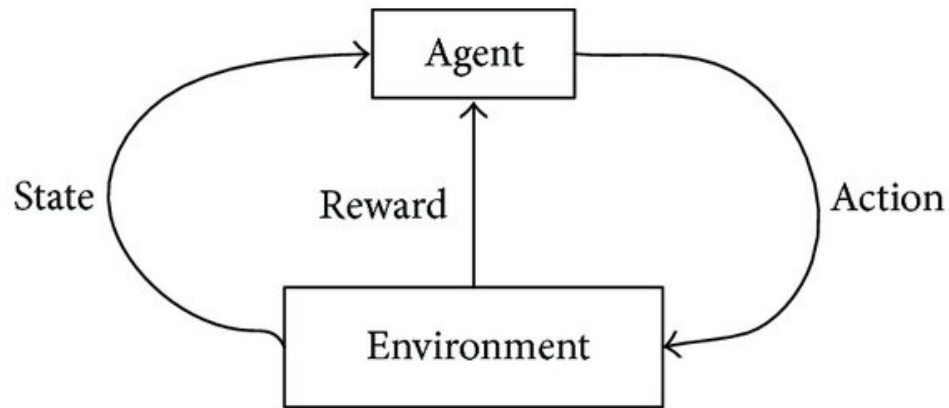


FIGURA 3.9: Esquema de aprendizaje por refuerzo

Fuente: <https://towardsdatascience.com>

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 análogamente con las neuronas del cerebro humano, la neurona artificial tomará una cantidad n de entradas $x_1, x_2, x_3, \dots, x_n$ estas entradas serán multiplicadas por pesos $w_1, w_2, w_3, \dots, w_n$ además se puede añadir una constante que llamaremos bias(b) para producir un salida.

La entrada a de la neurona será la suma total de los productos $z = \sum_{i=1}^n w_i x_i + b$, el valor de z será la entrada a la neurona la cual la evaluará con una función f de tal forma que nuestra salida sea $y = f(z)$.

En la ecuación 3.9 observamos la misma salida expresada en forma vectorial para nuestros vectores $x = [x_1 x_2 x_3 \dots x_n]$ y $w = [w_1 w_2 w_3 \dots w_n]$

$$y = f(x \cdot w + b) \quad (3.9)$$

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 data 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 un valor en el rango de 0 a 1.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- **tanh:** Toma por entrada un valor real y lo transforma a un número en el 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

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

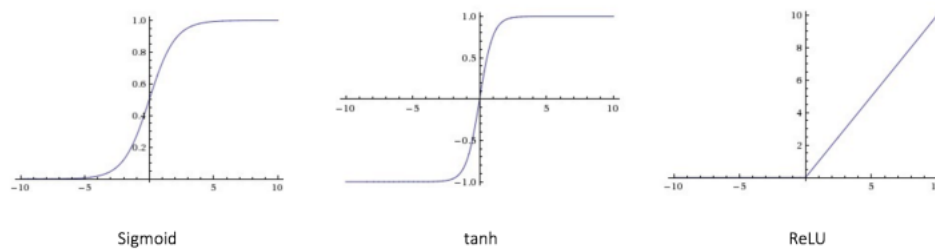


FIGURA 3.10: Funciones de activación

Fuente: <https://uijwalkarn.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. Las redes neuronales están constituidas de las uniones de las neuronas.

En la figura 3.10 podemos ver la comparación entre una neurona biológica y un artificial etiquetas 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).

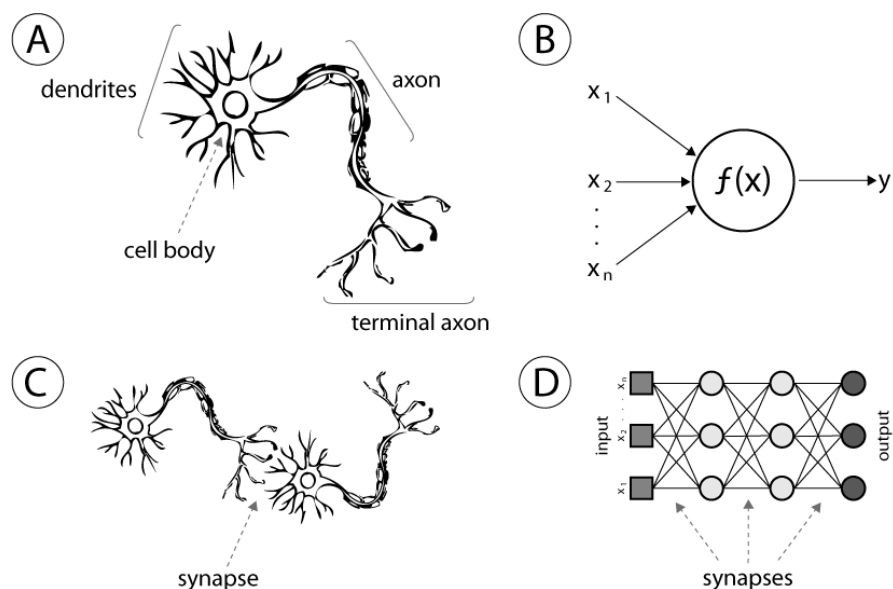


FIGURA 3.11: Redes neuronales biológicas y artificiales

Fuente: <https://medium.com>

Redes Neuronales Prealimentadas

Fue uno de los primeros y más simples tipos de redes neuronales que se desarrollaron. Contiene 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).

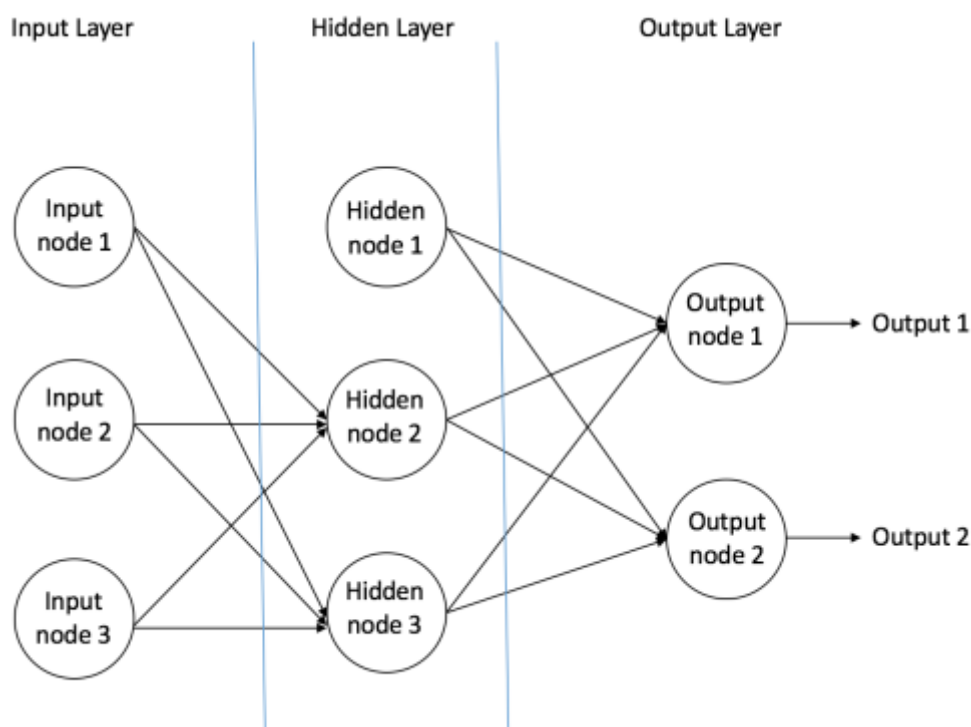


FIGURA 3.12: Esquema de Redes Neuronales Prealimentadas

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 una 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 red 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 algoritmo 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 las 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

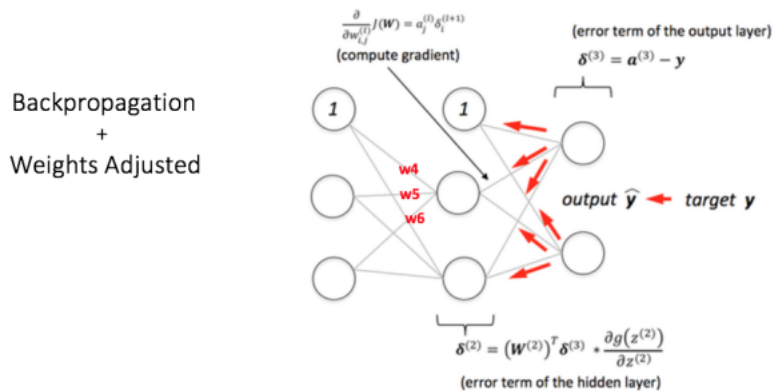


FIGURA 3.13: Propagación hacia atrás

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 testear.

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.

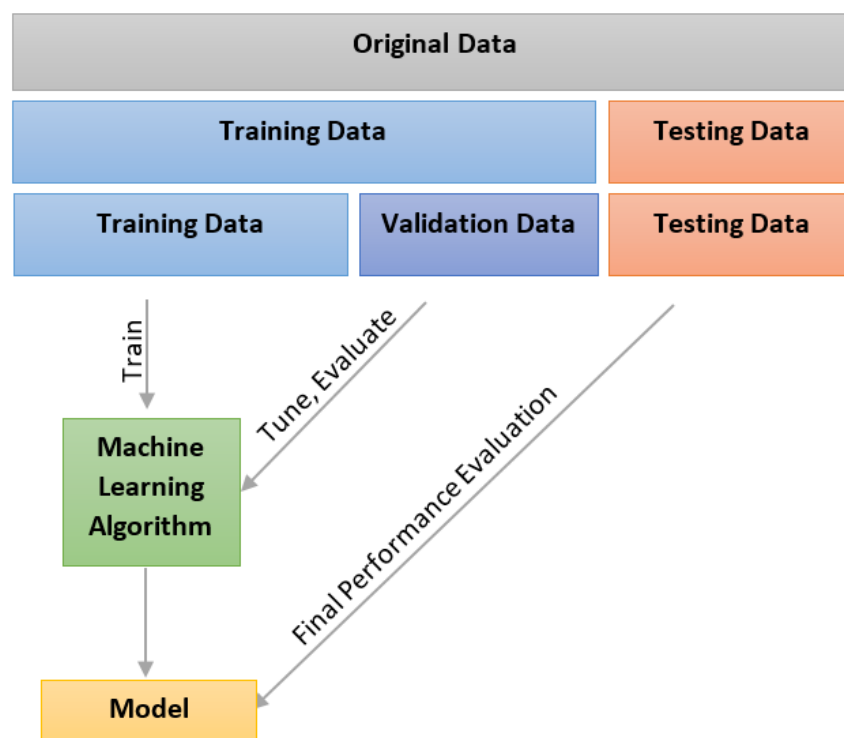


FIGURA 3.14: División del dataset

Fuente: <http://magizbox.com>

Capítulo 4

Algoritmos de optimización una Red Neuronal convolucional

En este capítulo se detallarán los algoritmos de optimización de Aprendizaje automático y principalmente se enfocará en aquellos aplicados a las redes neuronales convolucionales por lo cual 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 procesar en una red neuronal tradicional como es el caso del perceptron multicapas.

El termino convolucional hace referencia a la operación lineal matemática usada. Las redes neuronales convolucionales usan esta operación para aprender de las características de mayor orden en la data. 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 principalmente 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 como la computadora comprende y utiliza esta información.

Las imágenes están constituidas por la sucesión de píxeles, podemos entender el pixel como la menor unidad homogénea en color de una imagen digital. Teniendo este concepto la información de una imagen puede dividirse 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.

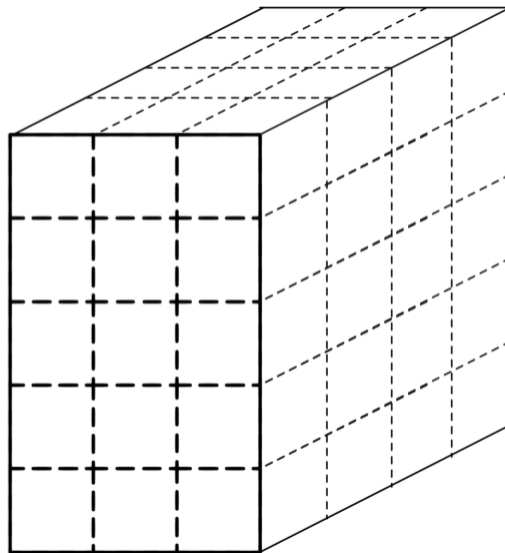


FIGURA 4.1: Estructura de la imagen de entrada
Fuente: *Deep Learning by Adam Gibson, Josh Patterson*

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. La creación hace que la tarea resulte complicada usando redes multicapas.

4.1.2. Capas de una CNN

Input layer

Esta capa es donde se carga y almacena la información de las imágenes para procesarlas en la red. Esta información contiene detalles de ancho, alto y número de canales de imagen.

Convolutional layers

Son una capa importante en el diseño de las CNN's, las capas convolucionales transformarán la entrada de la data usando las conexiones de las neuronas de las capas anteriores. La capa calculará el producto punto entre la región de la neurona 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 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 dominio del espacio/tiempo y el dominio de la frecuencias a través del uso de transformada de fourier. Toma la entrada un entrada, aplica un kernel de convolución y nos da un mapa de características como salida "[9] .

Las convoluciones son usadas principalmente como un detector de características cuyas entradas principalmente son la capa de entrada u otra convolución. En la figura 4.1 observamos la operación de convolución que por medio del uso de un kernel o filtro de convolución extrae características de

la data 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 lo cual es beneficioso ya que no se debe definir un peso para cada neurona.

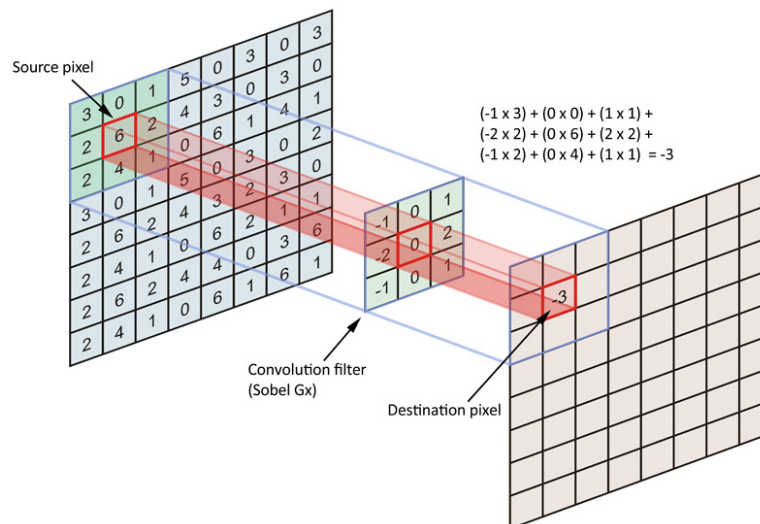


FIGURA 4.2: Operación de convolución

Fuente: www.openresearch.ai

Componentes de la capa de convolución.

Las capas convolucionales poseen parámetros e hiperparámetros. La gradiente de descenso se usa principalmente para 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. También pueden ser aplicados a 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). Por ejemplo podemos tener un filtro de tamaño $[5 \times 5 \times 3]$, 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 a la misma 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. Puede ser 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. La operación funciona de siguiente forma toma un filtro de $n \times n$, la operación \max el mayor de los números en el área de filtro.

Fully Connected Layers

Esta capa es usada para calcular el puntaje de la clase que usaremos como salida de red. Las dimensiones del volumen de la salida son $[1 \times 1 \times N]$, 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 neuronas de la capa

anterior. Esta capa realiza las transformaciones del volumen de datos de entrada. Estas transformaciones son funciones de activación en el volumen de entrada y los parámetros (pesos y bias de las neuronas).

4.1.3. Arquitecturas conocidas

4.2. Métodos de optimización

Los métodos de optimización dentro del campo de deep learning son muy importantes debido a que existen gran cantidad de parámetros es

4.2.1. Gradiente de descenso

La gradiente de descenso es un algoritmo común para optimizar redes neuronales. La gradiente de descenso es una forma de minimizar la función de costo $J(\theta)$ para metrizada por los parámetros $\theta \in \mathbb{R}^d$ actualizando los parámetros en la dirección opuesta a la gradiente de la función objetivo en este caso a nuestra función de costo $\nabla_{\theta} J(\theta)$ 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 la función objetivo entre estas variantes tenemos a:

Batch gradient descent

Esta variante calcula la gradiente de descenso de la 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.

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

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 del computador.

Stochastic gradient descent

A diferencia del método anterior las actualización se realizan para cada ejemplo de entrenamiento de (x^i, y^i) de esta manera se evitan problemas como la generación de redundancia debido a que se realiza una actualización por ejemplo de entrenamiento.

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

En la figura 4.2 vemos que la función de costo en SGD fluctúa demasiado esto podría representar un problema pero al contrario de la figura representa que el método SGD es capaz de saltar de un mínimo local con lo cual puede encontrar mínimos locales potencialmente mejores.

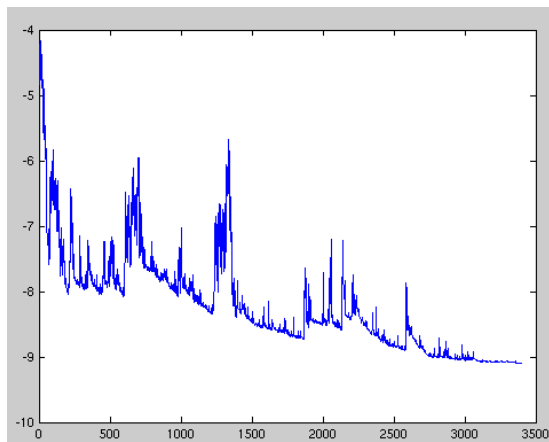


FIGURA 4.3: Función costo en SGD

Fuente: www.doc.ic.ac.uk

Mini-batch gradient descent

Este método puede verse como una mezcla de los 2 métodos anteriores, en lugar de aplicarlo para un conjunto entero de datos, los datos se particionan en pequeños conjuntos o mini batches. 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.

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

4.2.2. Optimizadores

En la siguiente sección analizaremos algunos optimizadores que acelerarán el proceso de 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 4.3.

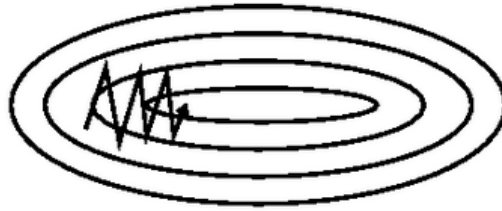


FIGURA 4.4: Actualización sin momentum
Fuente: www.doc.ic.ac.uk

El momentum es un método que ayuda a la SGD a acelerar en la dirección correcta, mientras evitas las oscilaciones. El momentum logrará esto añadiendo una fracción γ del vector de actualización pasado al vector presente tal como se muestra en las ecuaciones 4.4. Un valor comunmente elegido de $\gamma = 0.9$, en las actualización el valor del momentum aumenta para dimensiones cuyos gradientes apuntan en la misma dirección y disminuye para dimensional en la que la gradiente cambia de dirección. Esto nos asegura que tendremos una

convergencia más rápida con una oscilación reducida. En la figura 4.4 se observa gráficamente la aceleración de la convergencia en la SGD.

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

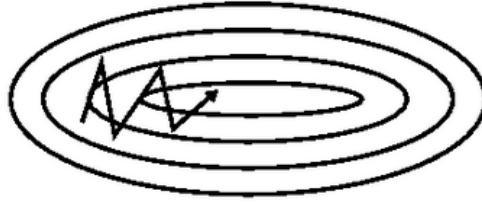


FIGURA 4.5: Actualización con momentum
Fuente: www.doc.ic.ac.uk

Nesterov accelerated gradient

Este método en el que nuestro descenso sea más controlado ya que reduce la velocidad antes de volver a subir una pendiente. En momentum usamos el término $\gamma\nu_{t-1}$ para mover los parámetros de θ . Al calcular el valor de $\theta - \gamma\nu_{t-1}$ nos da una aproximación de donde se encontrará la siguiente posición de los parámetros. De esta forma 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 4.5 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 una corrección (vector rojo), esto nos da como resultado la actualización completa de NAG (vector verde). Esta actualización anticipada 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.

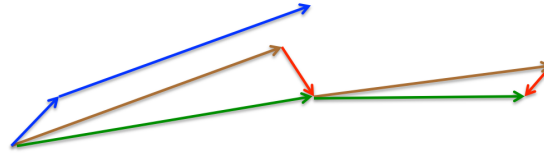


FIGURA 4.6: Convergencia Nesterov

Fuente: www.doc.ic.ac.uk

Adagrad

Es un algoritmo de optimización basado en el gradiente descendente, el algoritmo adapta la tasa de aprendizaje realizando actualizaciones más pequeñas para parámetros con características que se repiten con más frecuencia y una tasa alta para parámetros con características poco frecuentes. Adagrad mejora en gran manera a la SGD, este método 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

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

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

Adagrad modifica la idea de utilizar una tasa *eta* fija, podemos observar en 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 basadas en los valores de las gradientes pasadas que fueron calculadas para θ_i

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

- $G_{t,ii}$: 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 $G_t \in \mathbb{R}^{d \times d}$ contiene la suma de los cuadrados de las gradientes pasados con respecto a todos los parámetros de θ a lo largo de su diagonal. A lo largo de su diagonal por lo cual se puede realizar ahora el producto matriz- vector.

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

El principal beneficio de adagrad 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 cuadradas.

$$\begin{aligned} E[g^2]_t &= 0.9E[g^2]_{t-1} + 0.1g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \quad (4.9)$$

Adam

Adaptative moment estimation o Adam que calcula la tasa de aprendizaje adaptativo para cada parámetro. Este método mantiene un decaimiento exponencial del promedio de las gradientes pasadas. El método adam prefiere los mínimos en las superficies de error. En la figura 4.9 mostramos el calculo de 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.

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

La ecuación 4.12 muestra la regla de actualización en Adam.

$$\theta_{t+1} = \theta_{t+1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{4.12}$$

AdaMax

Nadam

Capítulo 5

Resultados

En este capítulo discutiremos los resultados obtenidos durante el uso de optimizadores para acelerar el proceso de entrenamiento de red neuronal convolucional. Estos ejemplos fueron aplicados al dataset

5.1. Conclusiones

Capítulo 6

Conclusiones y Trabajo Futuro

ESTE CAPÍTULO ES UNO DE LOS MÁS IMPORTANTES, POR NO DECIR EL QUE MÁS. EN ÉL, EL JURADO VA A TENER CLARO QUÉ HA APRENDIDO EL ALUMNO Y CÓMO LO HA DESARROLLADO, LOS PROBLEMAS QUE HAN SURGIDO Y COMO LOS HA SOLUCIONADO... ADEMÁS DE QUE EL ALUMNO DEJARÁ CLARO QUE SE HA ESPECIALIZADO EN LA TEMÁTICA Y DEJARÁ EN ESCRITO TODO LO APRENDIDO Y COMO CONTINUARÁ CON LA TEMÁTICA EN POSTERIORES ESTUDIOS DEL MISMO TEMA

6.1. Conclusiones

- CONCLUSION 1: ASDFASDFASDFAS.
- CONCLUSION 2: ASDFASDFASDFAS.

Además de lo anterior

6.2. Trabajo Futuro

COMO SE VA A SEGUIR TRABAJANDO CON ESTA TEMÁTICA Y QUE FALTA POR DESARROLLAR. ADEMÁS SE ACONSEJARÁ SEGUIR UNA METODOLOGÍA PARA QUE LAS PERSONAS QUE QUIERAN SEGUIR TRABAJANDO ESTA TEMÁTICA

NASDFASDFASF

ADFASDFASDFA

NASDFASDFASF

ADFASDFASDFA

NASDFASDFASF

ADFASDFASDFA

Bibliografía

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] K nearest neighbors - classification). http://www.saedsayad.com/k_nearest_neighbors.htm. Accessed on 2018-05-15.
- [7] Naive bayesian). http://www.saedsayad.com/naive_bayesian.htm. Accessed on 2018-05-15.

-
- [8] 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.
- [9] Josh Patterson and Adam Gibson. Major architectures of deep networks. In *Deep Learning A Practitioner's Approach*, pages 132–135. O'Reilly Media, 2017.

Apéndice A

Título del apéndice

Un ejemplo de los apendices