



Eduardo Díaz Domínguez.  
Centro de Estudios Profesionales, CEU Andalucía.

Ciclo Superior en Desarrollo de Aplicaciones Multiplataforma, 2019-2021.

Tutor: D. Juan Luis Proenza Quirós

## Índice

[**Justificación**](#_dej5az80adn9) 4

[**Marco Teórico**](#_f88ci57fm748) **5**

[**Metodología**](#_32ihdycg8j8h) **9**

[**Objetivos**](#_21ls3gdkcgx4) **10**

[Objetivos Generales](#_ujkw7vfm32gi) 10

[Objetivos Específicos](#_toua5q8sakrh) 10

[**Desarrollo Resultado y Análisis**](#_1enpnziile0c) **10**

[Recopilación de Cuadernos](#_4wkpwvcid2b3) 10

[Tensors](#_gn8f9nu9tag0) 11

[Algoritmos básicos](#_kz1vper6avgw) 12

[Machine Learning: Linear Regression](#_dsjlr2y9gkm1) 12

[DNN: Classification](#_n37t1p69ciz9) 14

[Machine Learning: Clustering(K-means-Clustering)](#_6zi1qbwwitc3) 15

[Core Learning Algorithms: Hidden Markov Models](#_fdc0ttax4k5i) 16

[Deep Neural Networks with TensorFlow](#_r8xf557mu029) 19

[Conclusiones](#_7ih6ipo2szud) 25

[Bibliografía / Referencias](#_4gbwh97rs51e) 26

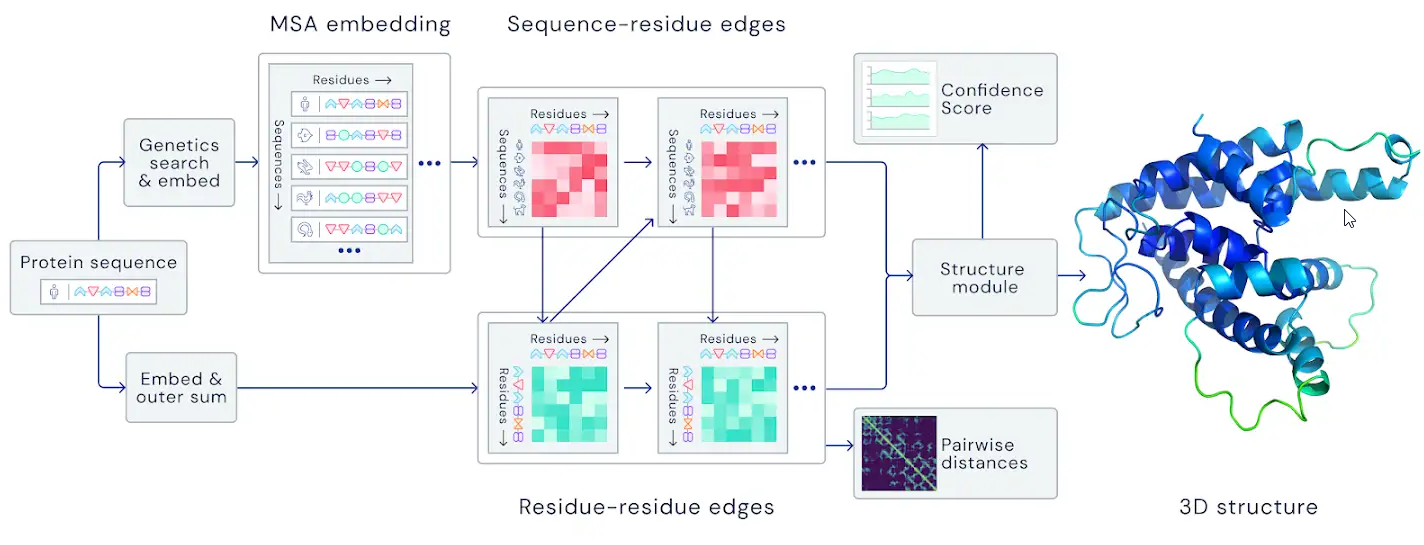
[**Anexos**](#_obgw2sly3x8f) **27**

## 

## Justificación

Encuentro las tecnologías de la inteligencia artificial un tema apasionante que además está muy demandado y tiene infinidad de aplicaciones. [Tier1-Comerzzia](https://www.tier1.es), empresa donde estoy realizando mis prácticas laborales, que desarrolla software aplicado al sector retail ya viene un tiempo aplicándolas. El potencial uso de estas tecnologías aplicadas redundan en un aumento de la competitividad de sus productos. Estos son algunos ejemplos de uso:

* Segmentación de clientes: Si somos capaces de clasificar a un cliente, dadas sus características, en un segmento (jóvenes entusiastas de las tecnología, padre de familia interesado en los deportes etc.), podemos conocer mejor a nuestros compradores y elaborar campañas de promoción de ciertos productos. Mejorando la estrategia de mercado.
* Sistemas de recomendación: Basándonos en compras pasadas, podríamos ser capaces, usando Machine Learning, de predecir en qué tipo de artículos estaría interesado cada cliente. Al ofrecer estos productos, los anuncios ganan en efectividad.
* Análisis facial: Conocimiento de la satisfacción de un servicio.
* Otros: predicción de ventas, chatbox, probadores virtuales, cartelería inteligente.

Además de las aplicaciones de la IA en el sector retail, existen otras muchas aplicaciones muy demandadas, como por ejemplo en investigación biomédica. Un ejemplo del uso de IA en biotecnología sería la solución a un problema crucial a día de hoy: la **predicción del plegamiento de proteínas**. El problema consiste en, a partir de la secuencia de aminoácidos de una proteína, ser capaz de predecir la estructura tridimensional de esta. Debido a que la forma y la función de una proteína están estrechamente relacionadas, conocer qué forma tendrá una proteína nos puede ayudar a comprender y a modificar su funcionamiento. Lograr modificar el funcionamiento de una proteína tiene infinitas aplicaciones: diseño de fármacos para destruir virus o bacterias , descubrimiento de nuevas dianas contra células cancerígenas o simplemente, el avance del conocimiento básico.[[1]](#footnote-0)

En este sentido, ya existen un grupos de investigación que viene desarrollando comercialmente este tipo de aplicaciones, como por ejemplo, [alphafold](https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology).

Es por estos motivos que he elegido este tema como proyecto de fin de grado. Realizándolo, pretendo conocer y presentar, al menos, una visión general de este fascinante mundo.

## Marco Teórico

Las tecnologías conocidas como Machine Learning (ML) y Deep Neural Networks (DNN) son desarrollos modernos de una disciplina madre que engloba a ambas, conocida como Inteligencia Artificial (IA).

*¿Qué es la IA?* Una posible definición sería:

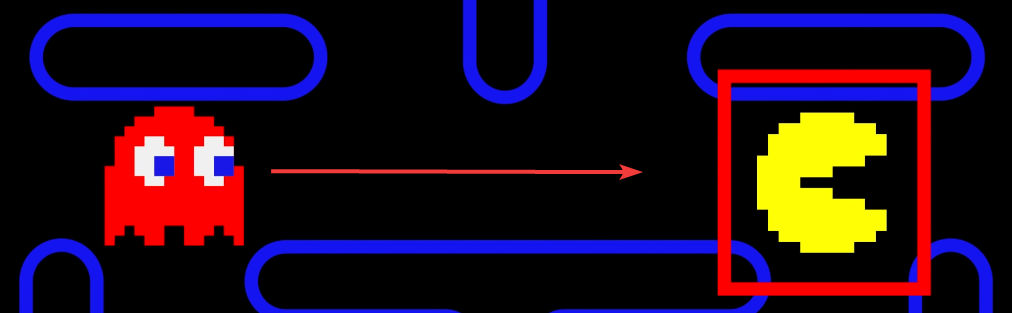
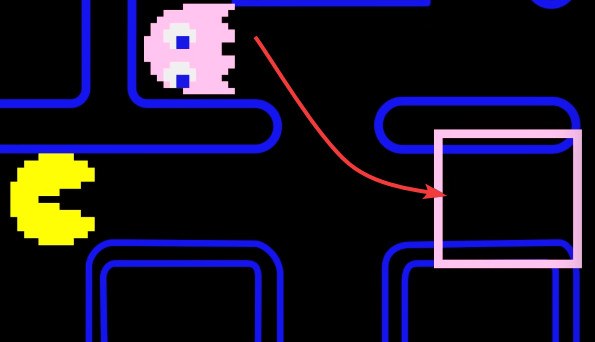
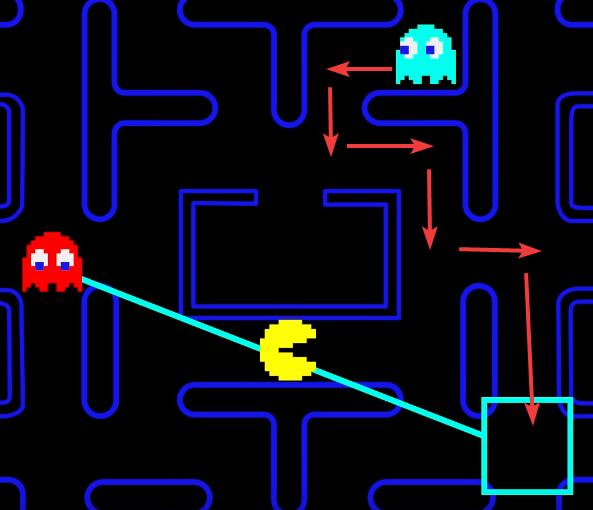
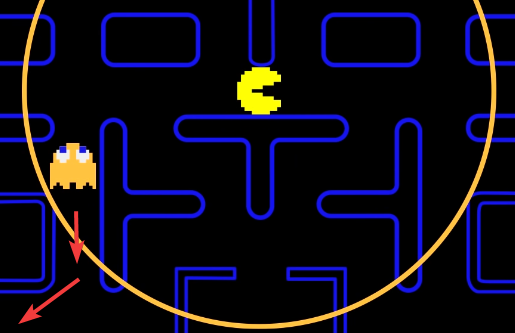
*La automatización de tareas intelectuales humanas.*

Esta no deja de ser una definición muy general por lo que para entenderla, sería interesante conocer un poco la historia del desarrollo de la IA.

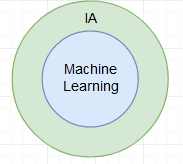
*¿Pueden las máquinas pensar?*

Históricamente, antes de la llegada de ML o las DNN la implementación de la IA ha consistido en generar instrucciones del tipo: Si ocurre A -> entonces haz B. Esto se llaman comúnmente instrucciones y, con ellas, una máquina puede tener una IA que consista en unas pocas líneas de código o millones de ellas. La IA, siguiendo esta arquitectura, puede ser tan potente como bien diseñadas y pormenorizadas estén dichas instrucciones. Veamos, a través de un ejemplo, un tipo de IA implementada de esta manera a través del clásico videojuego de los años 80, Pacman.

Nuestro come cocos tenía 4 enemigos, cada uno de los cuales contaba con su propia IA:

* Rojo (Blinky): El más agresivo, se dirige directamente a Pacman en línea recta.  
  
* Rosa (Pinky): Intenta emboscar a Pacman dirigiéndose, 4 pasos adelante, en la dirección que lleva Pacman.  
   
* Azul (Inky): El más complejo, se dirige a la posición opuesta entre Blinky y Pacman, intentando una "maniobra de pinza" para atrapar a Pacman.  
  
* Naranja(Clyde): Se dirige hacia Pacman directamente y, cuando se acerca mucho, se da la vuelta y se dirige a su esquina original.  
  

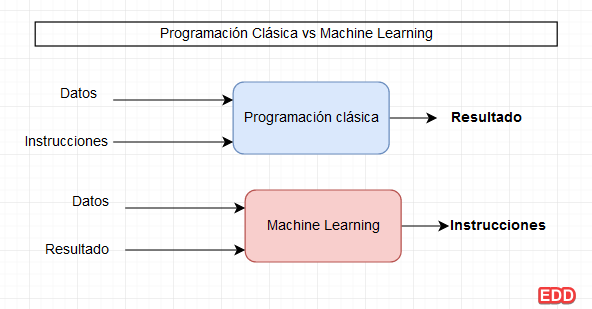
[Tómate un descanso, y reflexiona sobre la IA mientras te echas una partida a Pacman](https://www.google.com/logos/2010/pacman10-i.html)

*¿Qué es Machine Learning?*

Como mencionamos antes, Machine Learning es una subdisciplina englobada dentro de la IA.

 *¿Cuál es la diferencia entre la programación clásica y Machine Learning?*

Sin entrar mucho detalle, la abstracción que recibe la información se llama modelo. La misión del desarrollador o programador consiste en elegir el modelo que mejor corresponda al problema que se quiere solucionar así como alimentar al modelo con información de una manera muy determinada que veremos a lo largo de este proyecto.

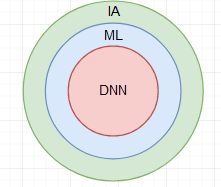
Existe un interesante cambio de paradigma a la hora de implementar este software. En la programación clásica, nosotros dictamos las reglas o instrucciones. Con dichas reglas se transforma una "información de entrada" o **input** en una "información de salida" o **output**. En Machine Learning, en cambio, lo que suministramos son los inputs y los outputs y es el propio software, el que se encargará de analizar la información y general las instrucciones por nosotros.  




Machine Learning : Se llama así porque la máquina necesita **aprender**. Esto implica el suministro de grandes cantidades de datos para que el modelo genere las reglas en un proceso llamado: Training o entrenamiento del modelo. El modelo, una vez entrenado, será capaz de predecir, dentro de un intervalo de confianza, futuros **outputs** a partir de **inputs** problema. La precisión del modelo no es perfecta. Esto quiere decir que no obtendremos la información de salida correcta el 100% de las veces.

El objetivo del desarrollador humano del modelo, mediante el pre-procesamiento de los datos, elección de variantes del modelo y ajuste del proceso de entrenamiento, producir un modelo lo más preciso posible.

No deja de ser irónico que aquello que intenta imitar al ser humano, cometa, por definición, múltiples errores.

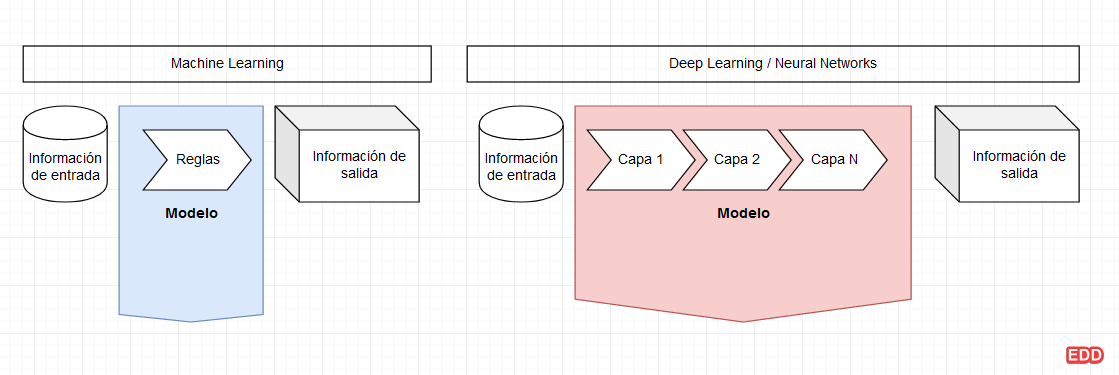


*¿Qué son las Deep Neural Networks?*

Una aplicación de Machine Learning que usa una representación de la información en capas.

*¿En qué se diferencian las Deep Neural Networks del Machine Learning?*

En Machine Learning el sistema aprende usando la información suministrada para producir una serie de reglas que transforman la información en su estado final: la información de salida.

En **Deep Learning** se suministra una capa de información al sistema, que produce otra capa a partir de esta, después, con esa información se produce la siguiente capa y así sucesivamente. A diferencia del Machine Learning en el que los datos sólo sirven para producir unas reglas o modelo, en DNN la propia información ES el modelo. Unas capas afectan a las siguientes en una dirección y, cuando esta llega al final, se produce una retroalimentación y ajuste del conjunto de capas que, poco a poco, van conformando un modelo cada vez más refinado.  


En este proceso, la información queda transformada formando parte de una red. Si no le ha quedado claro todavía no se preocupe, es completamente normal, lo explicaremos más adelante en más detalle.



*¿Qué es* [*TensorFlow*](https://www.tensorflow.org/)*?*

Es un framework o ecosistema integrado que incluye todas las herramientas necesarias, para crear modelos de Machine Learning y DNN de manera relativamente sencilla y exportable. Es open source, por lo que es posible tener acceso al código, modificarlo y distribuirlo.  


*¿Qué es* [*Keras*](https://keras.io/)*?*

Un conjunto de librerías que permite crear DNN sin necesidad de hacerlo desde cero. Es muy útil ya que encapsula y permite abstraernos de gran parte de la complejidad matemática necesaria para crear estos modelos. Está escrito en Python y es compatible con TensorFlow.

## Metodología

El proyecto tendrá dos partes diferenciadas que se apoyarán la una en la otra. Por un lado una parte teórica en la que explicaremos, de la manera más asequible posible para cualquier lector, tanto definiciones, como desarrollo y procedimientos generales a la hora de usar ML y DNN. Para ello nos apoyaremos en la bibliografía referenciada, así como en diagramas bien creados por nosotros o bien libres de derechos de autor para hacer más comprensible todos los conceptos.

Por otro lado, el presente trabajo presentaremos una parte práctica nos valdremos de un entorno de desarrollo en la nube de Google llamado [Colaboratory](https://colab.research.google.com). Se trata de un entorno de desarrollo completo, que pondrá a nuestra disposición la capacidad de instalar en una máquina todas las librerías y módulos necesarios. Usando el lenguaje Python podremos ejemplificar, demostrar y explicar la aplicación práctica en código fuente de estos modelos. En esta parte nos apoyaremos en ejemplos reales. Recomendamos echar un vistazo a los cuadernos y ejecutar el código ya que gran parte del tiempo dedicado al proyecto se destinó a estas implementaciones.

Es importante mencionar que gran parte del entendimiento y ejemplos utilizados provienen de la certificación en Machine Learning de [Freecodecamp](https://www.freecodecamp.org/).

## Objetivos

### Objetivos Generales

El objetivo del proyecto es la comprensión básica de las disciplinas Machine Learning y Deep Neural Networks. Deberemos ser capaces de comprender y aplicar los principales elementos tanto teóricos como prácticos de estas ramas de la Inteligencia Artificial.

### Objetivos Específicos

1. Comprender el uso de los principales módulos y métodos de Tensorflow.
2. Construir y aplicar un modelo de regresión lineal. (Machine Learning).
3. Construir y aplicar un modelo de clasificación (DNN).
4. Construir y aplicar un modelo de tipo Hidden Markov (Machine Learning).
5. Construir y aplicar un modelo de clasificación de imágenes (DNN).
6. Usar el lenguaje de programación Python.
7. Usar los módulos y librerías: TensorFlow, Keras, Pandas y Numpy entre otros.

## Desarrollo Resultado y Análisis

En esta sección aumentaremos el nivel de detalle, lo que nos permitirá entender mejor tanto la implementación programática como los conceptos matemáticos que subyacen a ML y DNN.

### Recopilación de Cuadernos

Se recomienda visitar y jugar con el código mostrado. Está explicado y facilita la comprensión de las tecnologías:

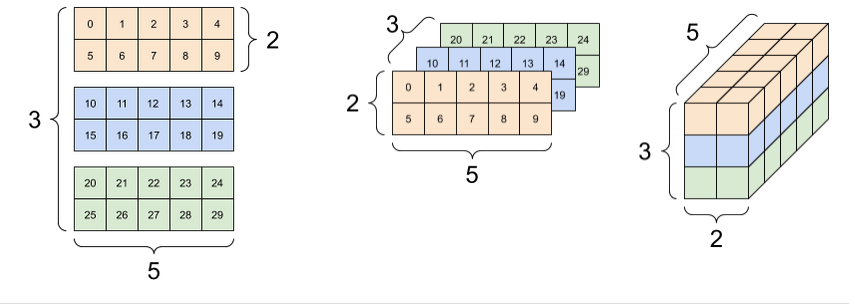
1. [Introducción a Python, Tensorflow y Collaboratory - Tensores y módulos esenciales](https://colab.research.google.com/drive/11B4wTn5yDh3GO56hdLfjCCrXeaYc-LDa#scrollTo=2Jry-Cohl7mm)
2. [Regresión Lineal - Supervivencia de los Pasajeros del Titanic](https://colab.research.google.com/drive/1v3uWoh1cjB7iq9ZaQC6qjYkIT04c-C4T)
3. [Classification - Infirendo especies de plantas](https://colab.research.google.com/drive/1UOOsY4sQjSTVp0BL3X3zAWouOGr1byyJ)
4. [Hidden Markov Models - Predicción del tiempo](https://colab.research.google.com/drive/1M-EKpqVIECf6rXNhLuuzwIDQQUGnSF-W)
5. [Deep Neural Networks - Entrenando un modelo capaz de clasificar imágenes de distintas prendas de ropa](https://colab.research.google.com/drive/1s4T_3bQ4lPY9BBDNYVZ4X_t572aN0OFu).

### Tensors

¿Qué es un tensor?

Traduciendo directamente de la documentación de TensorFlow:

"Un tensor es una generalización de los vectores y las matrices añadiendo n-dimensiones. Internamente, TensorFlow representa dichos tensores como arrays de datos"

Cabe mencionar, además, que un tensor también representa una operación o computación parcialmente definida, que eventualmente producirá un valor o resultado. Es decir, un tensor alberga información y al mismo tiempo está definiendo operaciones que la máquina puede procesar usando métodos de las librerías de TensorFlow. Esto se entenderá mejor a través de los ejemplos prácticos de los cuadernos.  
[[2]](#footnote-1)

*Propiedades de un tensor*

* DataType: el tipo de información que contiene el tensor (entero, float, strings etc.)
* Rank: nº de dimensiones
* Shape: Forma de un tensor. Mencionar que un mismo tensor puede tener diferentes formas. Mirar el cuaderno de la sección.

*Tipos de tensor*

* Mutables: no cambian durante la ejecución. Existen diferentes tipos pero no los declararemos durante el proyecto.
* Variables: sí cambian durante la ejecución.

### Algoritmos básicos

En esta sección, vamos a ver 4 algoritmos fundamentales de Machine Learning:

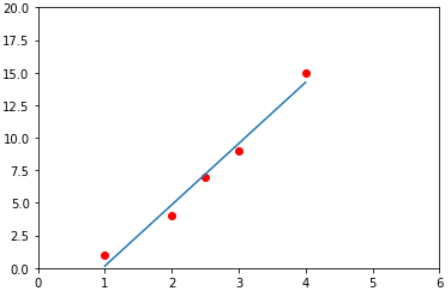
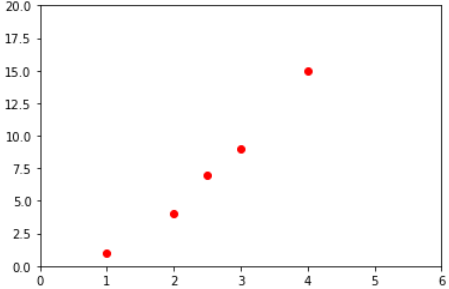
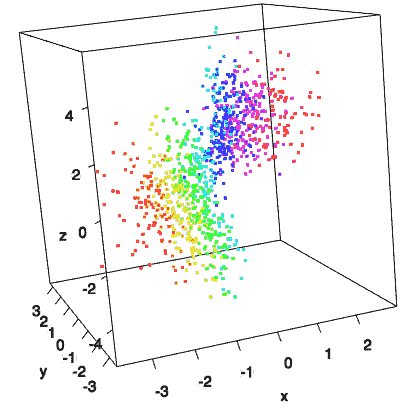
* Linear Regression
* Classification
* Clustering
* Hidden Markov Models

Algunos, como la regresión lineal, son sencillos pero no por ello menos usados y potentes. Debido a su sencillez, nos servirá para ilustrar los aspectos fundamentales del ML. Cada uno de estos modelos tiene muchas variantes e implementaciones.

#### Machine Learning: Linear Regression

[Regresión Lineal - Supervivencia de los Pasajeros del Titanic](https://colab.research.google.com/drive/1v3uWoh1cjB7iq9ZaQC6qjYkIT04c-C4T)

Este modelo busca, en última instancia, encontrar la ecuación de la recta que mejor se ajusta a una nube de puntos. Dicha nube de puntos es un conjunto de datos representados en coordenadas en n-dimensiones. Si estuviésemos trabajando con tan sólo 2 dimensiones la ecuación sería la siguiente: y = m x + n.

[[3]](#footnote-2)

*[[4]](#footnote-3)*

Este ejemplo es muy sencillo, pero debemos imaginar un escenario en el que la nube de puntos no tiene un patrón que se asemeja claramente a una recta y, tengamos más de 2 dimensiones. Es entonces cuando este modelo adquiere su valor. Imagina cuántos cálculos serían necesarios para obtener la recta que mejor se ajusta a esta nube de puntos...



##### TRATAMIENTO DE DATOS

En todo modelo de ML es necesario separar los datos en 2 conjuntos de datos: los de **entrenamiento** y los de test o **evaluación del modelo**. Los primeros sirven para producir y mejorar el modelo y suelen ser mucho más numerosos. Los segundos sirven para evaluar la **precisión** de nuestro modelo. Lo que nos permitirá ir ajustando sus parámetros.

Como puede verse en la implementación, los datos se guardan en objetos DataFrame de las librerías de Pandas.

Tras su obtención, se separan las columnas o campos de predicción o **labels**.

Si observamos los [datos](https://storage.googleapis.com/tf-datasets/titanic/train.csv), vemos que existen dos tipos de datos, numéricos en las llamadas numeric columns, con infinidad de valores posibles y, por otro lado, valores discretos, y con una determinada cantidad de valores que en la mayoría de los casos consisten en cadenas de texto, por ejemplo, hombre/mujer. Para poder alimentar el modelo es necesario transformar todos ellos en datos numéricos. Esto se puede automatizar usando los módulos de TensorFlow 2.0. El resultado son los datos tabulados y homogeneizados en las llamadas **feature columns.** Que el modelo podrá comprender.

##### El proceso de entrenamiento

El proceso de entrenamiento suele usar una ingente cantidad de datos, del orden de terabytes o más. Como esta cantidad de información no cabe en RAM es necesario dividirla en lotes o **batches**. Además, para refinar mejor el modelo, los datos se presentan muchas veces. El número de veces que los datos de entrenamiento se presentan al modelo se llama **epochs**. Para lograr que los datos se suministren al modelo en batches y por epochs es necesario definir las llamadas **input functions**. Hay un tamaño óptimo de batch para que el proceso no sea demasiado lento.

El modelo aumentará su precisión si le presentamos el mismo conjunto de datos varias veces. Hay que tener cuidado porque puede ocurrir que sobrepasemos el número óptimo de veces que se presentan los datos, de tal manera que el modelo "memorice" los datos y pierda capacidad predictiva en un fenómeno que se conoce como **overfeeding**. Lo que ocurre es que el modelo se vuelve muy bueno en clasificar el conjunto de datos de entrenamiento pero se vuelve muy malo en clasificar nuevos datos.

La manera de evitar el overfeeding es calibrar el modelo usando diferentes epochs y batches hasta alcanzar mejor precisión.

Además de los dos parámetros mencionados, en la función también indicamos si los registros se dan de forma aleatoria o no (randomizado).

Cada conjunto de datos, así como la forma en la que se administrará al modelo quedará codificado en una input function (objetos y funciones se pueden almacenar en variables en Python), con la diferencia de que en el caso de los datos de evaluación, no es necesario separar en batches los datos (la cantidad de información es mucho menor) y no es necesario randomizar los datos.

Finalmente entrenamos el modelo con la input function de los datos de entrenamiento.

##### Evaluación del modelo

La evaluación del modelo usando las input function de los datos de test. La evaluación va a arrojar una serie de datos estadísticos, pero el valor que más nos interesa es el accuracy o precisión. Este dato es un dato agregado y se obtiene al comparar todos las predicciones del modelo con los valores reales y así obtener un valor de precisión global.

#### DNN: Classification

De entre los modelos de clasificación que TensorFlow tiene implementados podemos elegir entre dos principales: DNNClasiffier (Deep Neural Network) o LinearClassifier. El último, funciona de manera muy similar a la regresión lineal. En este caso vamos a usar **DNNClasiffier** porque lo recomienda TensorFlow para este tipo de problemas.

En esta parte del trabajo solamente explicaremos algunas diferencias de implementación, ya que más adelante abundaremos en la explicación teórica de los DNN.

Los pasos a seguir son similares al modelo anterior y están muy bien explicados en el cuaderno de esta sección.

1. Descarga de datos
2. Creación de los Dataframe de pandas
3. Definición de las Feature Columns
4. Definición de las Input Function
5. Definición del modelo del modelo DNN
6. Entrenamiento del modelo
7. Evaluación del modelo
8. Predicciones

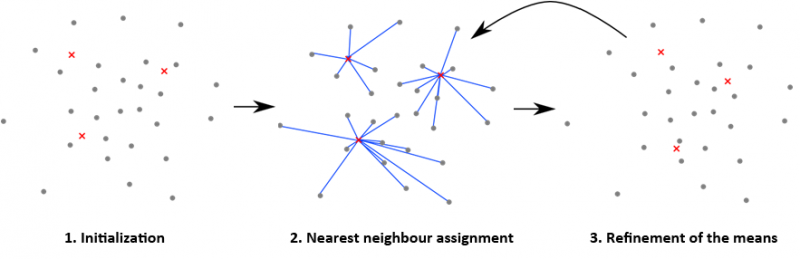
Cuando se trata de entrenar el modelo se hace igualmente con Input functions. La diferencia radica en que en este caso hay que definir los **steps** en lugar de los epochs. Los steps son el nº de pasos que utilizará el modelo para procesar los datos. No siempre un mayor número de pasos de como resultado un modelo con una mejor precisión. Si observamos el log nos va informando del número de pasos y del **loss** de cada uno de ellos. Explicaremos este parámetro más adelante, pero por el momento basta con comprender que cuanto más bajo sea, más exitoso está siendo el proceso de entrenamiento.

Una vez entrenado el modelo, podemos hacer predicciones a partir de un registro. Por ejemplo, le damos las medidas de partes anatómicas de una flor y el modelo nos devolverá a qué especie pertenece y con qué probabilidad. El modelo sólo entiende la información a partir de un tipo de objeto, que contiene una input function. En este caso no hay steps porque no estamos entrenando el modelo.

#### 

#### Machine Learning: Clustering(K-means-Clustering)

Es un modelo no supervisado. Esto quiere decir que una vez arranca no podemos hacer variar ningún parámetro. Se usa en una situación muy específica: cuando tenemos mucha información o inputs pero no tenemos información de salida o outputs. Lo que hace este algoritmo es agrupar datos similares. Vamos a centrarnos, en concreto en el modelo **K-Means**:

1. Asignación aleatoria de K-centroids: Asigna una posición inicial para los **centroides**. Estos son puntos imaginarios que sirven para hacer cálculos de distancia e ir clasificando los datos en grupos o **clusters**, en un proceso iterativo.
2. Asigna los datos al K-centroid más cercano por distancia.
3. Se mueven todos los centroid al centro de masa de todos los puntos de datos que se asignaron a dicho centroid.
4. Se vuelven a asignar los puntos de datos al centroide más cercano.
5. Se repiten los pasos 2-4. Hasta que los puntos de datos dejan de cambiar a qué cluster pertenecen.

[[5]](#footnote-4)

Si lo pensamos, lo único que se mueven son los centroides. Lo que hace el algoritmo es ajustar los centros de masa de cada cluster, para así agrupar datos de la manera más precisa posible.

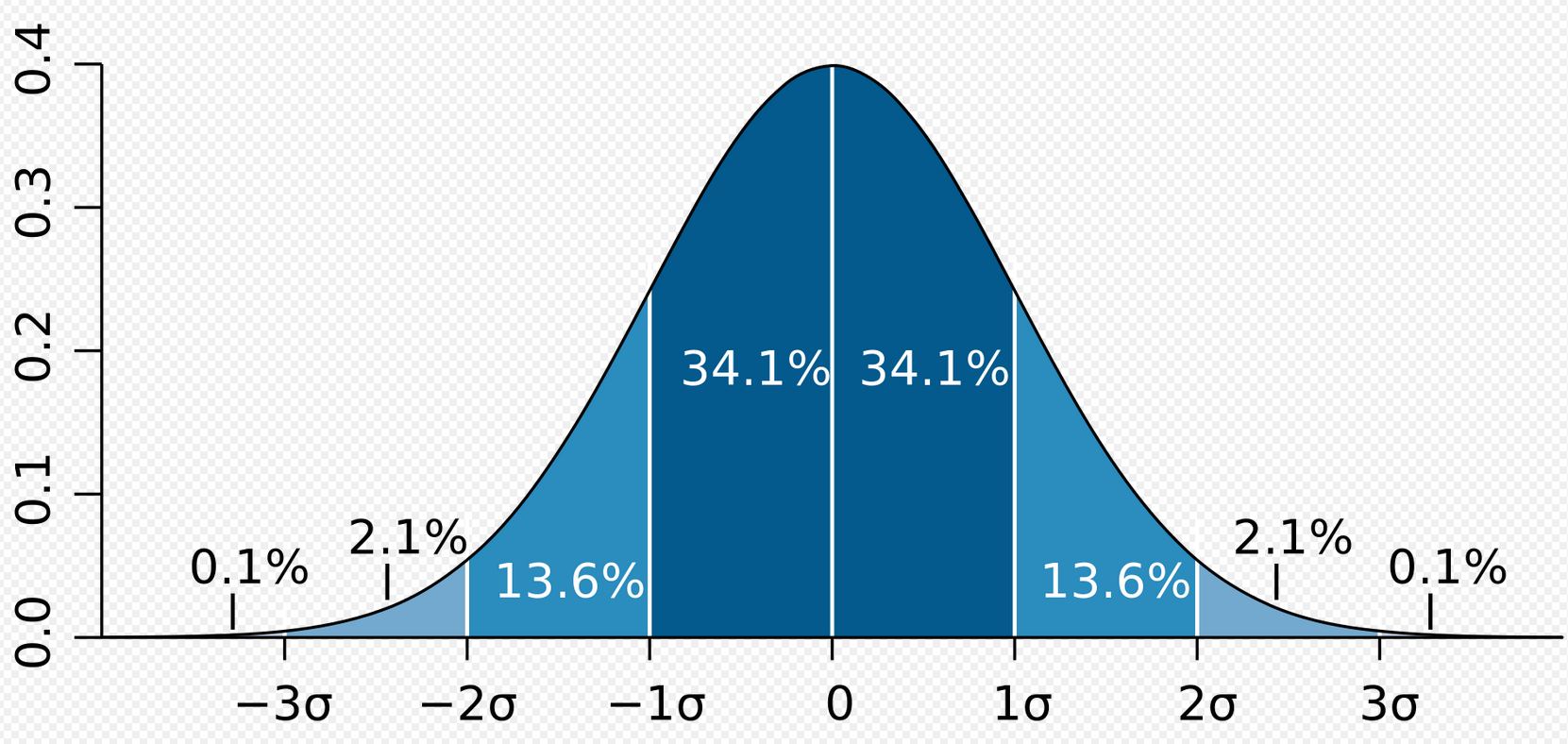
Una particularidad de este modelo es que necesitamos saber cuántos cluster queremos que busque el algoritmo (el número K). Existen otros algoritmos que ayudan a conocer cuál es el número óptimo de clusters, pero su uso se sale del objetivo de este epígrafe.

#### Core Learning Algorithms: Hidden Markov Models

[Cuaderno Hidden Markov Models](https://colab.research.google.com/drive/1M-EKpqVIECf6rXNhLuuzwIDQQUGnSF-W)

Este modelo se compone de una serie finita de estados, cada uno de los cuales está asociado con una **probabilidad de distribución**. Ésta probabilidad suele ser multidimensional. Cómo entenderemos a lo largo de esta sección, la diferencia entre este tipo de modelos y, por ejemplo, la regresión lineal o la clasificación es que la respuesta a la pregunta ¿Qué ocurrirá mañana? No es ni un valor, ni un porcentaje sino una función de distribución de probabilidades

*¿Qué es una distribución de probabilidades?*

Sin adentrarnos demasiado en el campo de la estadística, que no es el objetivo, esto quiere decir que dado un evento aleatorio (lanzamiento de dados), existe una función que nos indica la probabilidad de que uno de los eventos esperados suceda (el resultado la tirada de datos).   
[[6]](#footnote-5)

Adentrémonos en las particularidades de los Hidden Markov Models (HMM). Éstos se componen de las siguientes abstracciones de los datos y de otras abstracciones que veremos a continuación. Como haremos en el [cuaderno](https://colab.research.google.com/drive/1M-EKpqVIECf6rXNhLuuzwIDQQUGnSF-W), explicaremos el funcionamiento de este modelo a través de un ejemplo: la predicción del tiempo.

##### DATOS en HMM

En otros modelos usábamos conjuntos de datos con 100 o más registros, por contra, en HMM trabajaremos únicamente con una función de distribución que o bien es dada, o bien podemos adquirirla a partir de un conjunto de datos tras un análisis estadístico.

##### COMPONENTES de los HMM

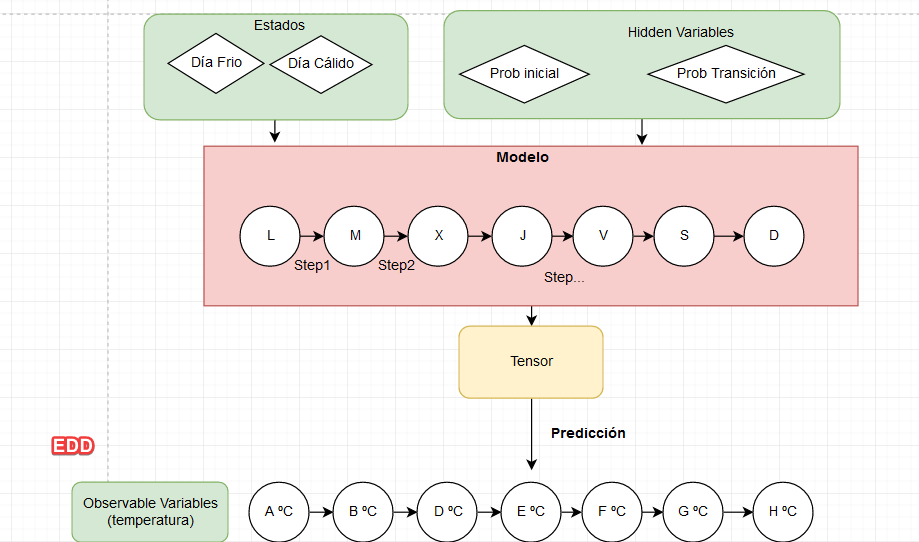
Usando el ejemplo de predicción del tiempo, definamos dichos componentes:

* Estados: cálido, frío, verde, azul, amarillo etc. Estos estados no se pueden observar de manera directa, están ocultos, (de ahí la nomenclatura del modelo).
* Observaciones de distribución: Cada estado tiene un resultado específico observable. Que ocurre siguiendo una distribución de probabilidad. Por ejemplo: En un día soleado, la probabilidad de que Eduardo está contento es del 80 % frente a un 20% de que esté triste.
* Transiciones de distribución: Cada estado tiene asociado una probabilidad de cambiar a otro estado. Por ejemplo: Un día lluvioso tiene un 70 % de probabilidades de ser sucedido por un día soleado y un 30% por otro día lluvioso más.

[[7]](#footnote-6)

Pasos

1. Definición de las variables de distribución a partir de observaciones.
2. Creación del modelo a partir de las variables de distribución y añadiendo el nº de STEPS.
3. Ejecutar el modelo y obtener el tensor
4. Ver los valores del tensor 

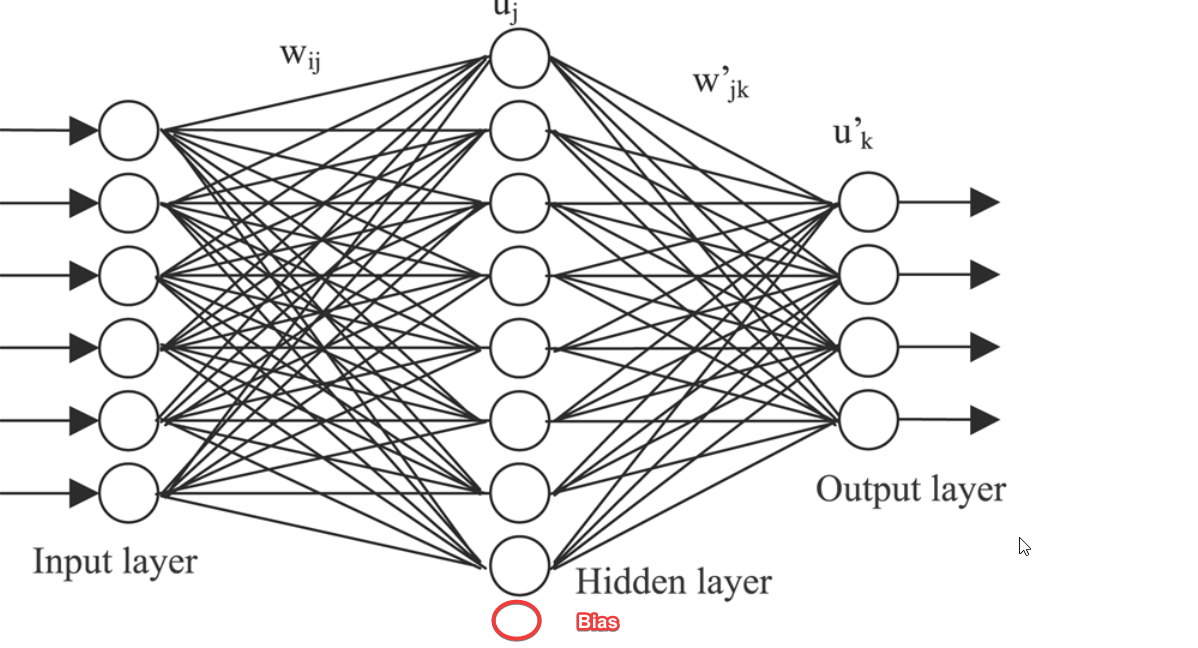
*Aconsejamos ver la implementación del modelo en el cuaderno adjunto*.

### Deep Neural Networks con TensorFlow

[Deep Neural Networks - Entrenando un modelo capaz de clasificar imágenes de distintas prendas de ropa](https://colab.research.google.com/drive/1s4T_3bQ4lPY9BBDNYVZ4X_t572aN0OFu).

Aunque anteriormente hemos visto un modelo sencillo de clasificación con DNN es ahora cuando vamos a adentrarnos en detalle para comprender en qué consisten este tipo de modelos. También podemos ver, a través del cuaderno, su implementación pormenorizada.

#### *Introducción*

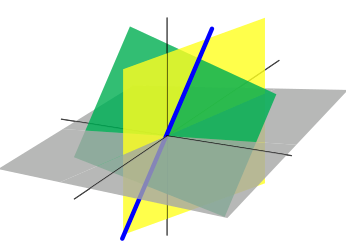
Una Deep Neural Network (DNN) es una representación de datos en capas. Cada capa está formada por un número de **neurons** o neuronas conectadas entre sí a modo de malla o red (network). El término deep atiende a la profundidad de la malla, debido a las múltiples capas de las que se componen estos modelos y, en las que se establecen los datos.  
[[8]](#footnote-7)

¿Qué diferencia las DNN de los modelos vistos anteriormente?

En anteriores modelos (Regresión lineal, Clasificación, Clustering y HMM ) los datos no sufrían ninguna transformación. Tras crear el modelo, prepararlo dándole unas características definitorias de los datos o features y entrenar modelo, éste, al recibir unos datos problema devolvía una solución después de un proceso matemático. Los datos, sin embargo, permanecían intactos después de este proceso, alojados en una capa. En las DNN, los datos se procesan de manera diferente. En estos modelos, los datos entran en la primera capa o input layer, para ir pasando a través de las capas intermedias o **hidden layers** hasta llegar a la capa de salida o output layer. Los datos, alojados en los nodos de la red o neurons van sufriendo una serie de transformaciones, de manera que según en qué capa nos encontremos su valores y propiedades irán cambiando.

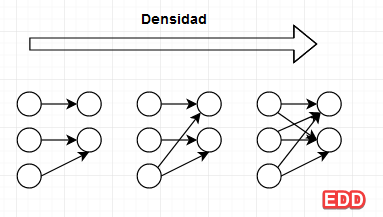
Será a través de estas transformaciones que el modelo extraerá una comprensión abstracta los datos, lo que le permitirá, después de un proceso de entrenamiento, aumentar su capacidad predictiva y ser capaz de obtener una solución con el intervalo de confianza más elevado posible a partir de un conjunto de datos problema.

#### *Cómo funcionan las DNN*

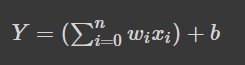
Empezaremos haciendo una definición con un nivel alto de abstracción sobre las matemáticas que gobiernan estos modelos, que, a bajo nivel se basan en operaciones básicas y álgebra lineal (vectores, matrices, sistemas de ecuaciones lineales, espacios vectoriales y sus transformaciones).  [[9]](#footnote-8)

Como veremos en mayor detalle más adelante, las DNN utilizan transformaciones algebraicas para "mover los datos" y entenderlos mejor, de igual manera que haría un niño cuando le dan un juguete de "lego": monta y desmonta las piezas del juguete para comprender mejor el sentido de la relación entre las piezas entre sí para poder, más adelante, añadir nuevas funciones al juguete respetando la lógica del conjunto.

Cada neural network consiste en una secuencia de layers. Estas capas están formadas por neuronas. Las neuronas de una capa están conectadas con las neuronas de la siguiente capa. Cada capa tiene, además, una neurona especial llamada **bias** o sesgo, que no tiene ninguna conexión con otras y simplemente aloja un valor numérico.

Cada capa puede tener diferentes niveles de densidad de conexión con la siguiente capa. Esto quiere decir que una neurona puede establecer conexiones con algunas o todas las neuronas de la capa siguiente. Se dice que una conexión es totalmente densa cuando todas las neuronas de una capa están conectadas con todas la neuronas de la siguiente capa.[[10]](#footnote-9)

Los datos comienzan su periplo en la input layer, la primera capa, y van pasando de capa en capa. Alojados en las neuronas de cada capa, quedan definidos mediante la siguiente ecuación:



* **w** es weight o "peso" de cada conexión con una neurona $x$ es el valor de de la ...
* **b** es el bias de cada capa
* **n** es el número de conexiones
* Cada neurona tiene un valor **Y** . Dicho valor, como veremos, influenciará al valor de Y de las neuronas siguientes con las que está conectada.

La ecuación, no está definida del todo aún. Hasta ahora, el modelo ha quedado definido con un sumatorio muy sencillo definido por los datos de partida, el peso de las conexiones entre neuronas, el sesgo de cada capa y el número de neuronas de cada capa. Sin embargo, por mucho que nos pese (o nos entusiasme) el modelo se complica. La función de Y está corregida por otra multitud de funciones, llamadas Activation Functions o funciones de activación-> F. Éstas son funciones actúan sobre los datos dándoles multidimensionalidad y complejidad. Las explicaremos en los siguientes epígrafes. Esta función se añade a la ecuación para añadir **complejidad** y **dimensionalidad** a los datos nuestra red. La ecuación, una vez añadida la función de activación quedaría de la siguiente manera

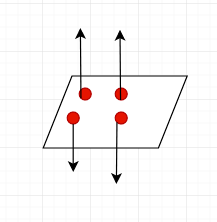


Nuestra red comenzará con un conjunto de activation functions predefinidas, con **weighs** y **bias** aleatorios para cada conexión y capa, respectivamente. A medida que entrenamos el modelo añadiendo nuevos datos, éste aprenderá y corregirá weights and biases usando una técnica llamada **backpropagation** propagación retrógrada (la explicaremos más adelante). Una vez que los weighs y bias de nuestra red son correctos, el modelo será capaz de producir predicciones que tengan sentido. Las predicciones se observan en la última capa, llamada la output layer o capa de salida.

##### Neural Networks: Activation functions

Las funciones de activación se aplican sobre cada capa de la red neuronal. Existen diferentes activation functions.

* RELU (Rectified Linear Unit): Devuelve el valor absoluto de un número. [0, + infinito]
* TANH (Hyperbolic Tangent) : Transforma los valores finales en un rango [-1,1]
* Sigmoid: Transforma los valores en el rango [0,1].

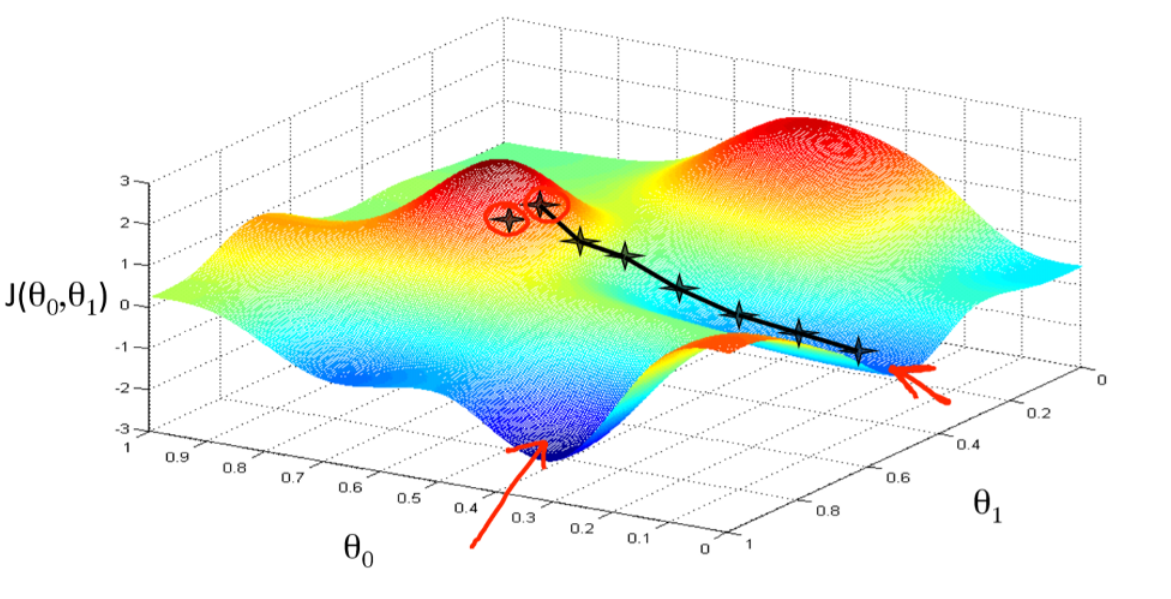
Nosotros podemos definir qué funciones de activación se aplican a cada neurona. La elección más importante es sobre la output layer que es la última neurona. Las funciones de activación introducen complejidad y dimensionalidad. Por ejemplo, imaginemos una serie de coordenadas que pertenecen a un plano. Si aplicamos una sigmoide lo que estamos haciendo es mover los puntos ortogonalmente hacia arriba y abajo del plano para obtener información de los puntos.

(esto tendrá sentido después como veamos el proceso de entrenamiento). 

Lo que hace, si lo pensamos, es cambiar las coordenadas desde 2 dimensiones a 3 dimensiones con la idea de extraer más detalles que antes no se podían extraer. Por ejemplo, imaginemos un cuadrado. A partir de él podemos extraer la altura, la anchura y el color. Si añadimos una dimensión, podremos conocer, el volumen, la profundidad, el número de caras etc.. Valga la simplificación, de lo que se trata es de extraer más información del mismo conjunto de datos, con la idea de entenderlos mejor y poder hacer mejores predicciones.

**Loss function**: Otro parámetro a controlar durante el entrenamiento del modelo es el loss, que podría traducirse como la pérdida de precisión o también cuánto se aleja el valor obtenido en el output layer o capa final del valor esperado. Por ejemplo si esperábamos un valor 0 o 1 y obtenemos un 0.7 la loss function nos da la diferencia. Esta función es una estimación de cómo de buena es la red de neuronas para predecir/procesar un valor. Este valor sirve para retocar las redes neuronales y mejorarlas. Queremos que nuestra función tenga un valor de loss lo más bajo posible.

##### Neural Networks: Optimizers

El proceso por el cual se consigue optimizar la red neuronal para conseguir un loss mínimo utiliza unas funciones conocidas como OPTIMIZERS, siendo la más común y la que usamos en el ejemplo práctico la llamada **gradient descent**.  


Imaginemos que hemos empezado la red neural, a partir de los puntos aleatorios en las crestas rojas del plano que se ve en la imagen. Tenemos un **loss** muy alto. El objetivo es movernos hacia las zonas bajas o azul marino. Lo que significa en realidad el gradient descended es la **dirección de evolución que tiene que tomar nuestra red neuronal para alcanzar ese valor con una mayor precisión**.

##### En resumen y recapitulando

La ecuación de suma de pesos define los valores que van tomando las neuronas. Se suministran datos a la red en la input layer. Los pesos y bias en la primera iteración son aleatorios. Los datos se transforman al paso de cada capa y llegan al final. Cuando se alcanza la última capa se ejecuta una retroalimentación de la red (backpropagation). Esta retroalimentación hace variar los pesos, vías y valores de Y de cada neurona ajustándose una y otra vez en un proceso de aprendizaje. Este proceso está gobernado por el "gradient descent", que armoniza la variaciones de estas variables de tal manera que la red neuronal se va refinando hasta alcanzar valores de precisión muy altos y valores de pérdida o loss muy bajos.

*Ver cuaderno.*

Existen otros optimizers muchas variantes:

* Gradient descent (la que usaremos)
* Stochastic Gradient Descent
* Mini-Batch Gradient Descent
* Nesterov Accelerated Gradient

## Conclusiones

* La prueba de concepto demuestra la versatilidad y la relativa simplicidad a la hora aplicar las tecnologías de Machine Learning y Deep Neural Networks para general modelos predictivos.
* Gracias a frameworks como TensorFlow es sencillo iniciarse en el entendimiento y uso de estas tecnologías con un nivel medio de programación.
* Un desarrollador experimentado usando estas herramientas tiene un gran potencial para mejorar el rendimiento de su empresa y su carrera profesional.

## 

## Bibliografía / Referencias

Doshi, S. (2020, 3 agosto). *Various Optimization Algorithms For Training Neural Network*. Medium - Algoritmos de Optimización Entrenamiento de DNN. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

Freecodecamp. (s. f.). *freeCodeCamp.org*. Certificación Machine Learning Freecodecamp. Recuperado 25 de mayo de 2021, de https://www.freecodecamp.org/learn/machine-learning-with-python/

*Gradient Descent — ML Glossary documentation*. (s. f.). Machine Learning Glossary - Gradient Descent - Optimización DNN. Recuperado 25 de mayo de 2021, de https://ml-cheatsheet.readthedocs.io/en/latest/gradient\_descent.html

*Keras: the Python deep learning API*. (s. f.). Keras - Documentación. Recuperado 25 de mayo de 2021, de https://keras.io/

*Line Of Best Fit*. (s. f.). Investopedia - Regresión Lineal. Recuperado 25 de mayo de 2021, de https://www.investopedia.com/terms/l/line-of-best-fit.asp

Seif, G. (2021, 25 enero). *The 5 Clustering Algorithms Data Scientists Need to Know*. Medium - Algoritmos de Clustering. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

TensorFlow. (s. f.-a). *Basic classification: Classify images of clothing | TensorFlow Core*. TensorFlow - Clasificación de Imágenes de Ropa Con Keras. Recuperado 25 de mayo de 2021, de https://www.tensorflow.org/tutorials/keras/classification

TensorFlow. (s. f.-b). *Introduction to Tensors | TensorFlow Core*. TensorFlow - Tensors. Recuperado 25 de mayo de 2021, de https://www.tensorflow.org/guide/tensor

TensorFlow. (s. f.-c). *tfp.distributions.HiddenMarkovModel | TensorFlow Probability*. TensorFlow - Hidden Markov Models. Recuperado 25 de mayo de 2021, de https://www.tensorflow.org/probability/api\_docs/python/tfp/distributions/HiddenMarkovModel

*tf.feature\_column.categorical\_column\_with\_vocabulary\_list*. (s. f.). TensorFlow - Categorical Columns - Procesamiento de Datos. Recuperado 25 de mayo de 2021, de https://www.tensorflow.org/api\_docs/python/tf/feature\_column/categorical\_column\_with\_vocabulary\_list?version=stable

Tier1. (s. f.). *¿Por qué la AI está revolucionando todo el sector del Retail? - visor-noticias*. comerzzia.com. Recuperado 25 de mayo de 2021, de https://www.comerzzia.com/portal/visor-noticias/-/asset\_publisher/dN6OY2hzi6K6/content/%C2%BFpor-que-la-ai-esta-revolucionando-todo-el-sector-del-retail-

## 

## Anexos

1. [Introducción a Python, Tensorflow y Collaboratory - Tensores y módulos esenciales](https://colab.research.google.com/drive/11B4wTn5yDh3GO56hdLfjCCrXeaYc-LDa#scrollTo=2Jry-Cohl7mm)
2. [Regresión Lineal - Supervivencia de los Pasajeros del Titanic](https://colab.research.google.com/drive/1v3uWoh1cjB7iq9ZaQC6qjYkIT04c-C4T)
3. [Classification - Infirendo especies de plantas](https://colab.research.google.com/drive/1UOOsY4sQjSTVp0BL3X3zAWouOGr1byyJ)
4. [Hidden Markov Models - Predicción del tiempo](https://colab.research.google.com/drive/1M-EKpqVIECf6rXNhLuuzwIDQQUGnSF-W)
5. [Deep Neural Networks - Entrenando un modelo capaz de clasificar imágenes de distintas prendas de ropa](https://colab.research.google.com/drive/1s4T_3bQ4lPY9BBDNYVZ4X_t572aN0OFu).

1. Figura: Esquema de la inferencia de estructura proteica a partir de su secuencia de aminoácidos [↑](#footnote-ref-0)
2. Un tensor cuya propiedad shape o forma es [2,3,5] [↑](#footnote-ref-1)
3. *Representación de la nube de puntos en 2 dimensiones* *Representación de la ecuación de recta que mejor se ajusta a los puntos* [↑](#footnote-ref-2)
4. *Nube de puntos en 3 dimensiones* [↑](#footnote-ref-3)
5. Esquema del algoritmo de clustering [↑](#footnote-ref-4)
6. Una de las funciones de distribución que más se da en la naturaleza: la Distribución Normal o Campana de Gaus. [↑](#footnote-ref-5)
7. Creación y entrenamiento del modelo [↑](#footnote-ref-6)
8. Esta de una DNN, se aprecian las diferentes capas, así como w (peso de las conexiones); En el esquema sólo hay una hidden layer pero puede haber infinidad de ellas. [↑](#footnote-ref-7)
9. Espacio vectorial en 3 dimensiones (euclídeo). Los datos están representados mediante coordenadas. Pueden incluirse en planos o vectores y las transformaciones algebraicas representan movimientos de dichos datos en el espacio vectorial [↑](#footnote-ref-8)
10. Diferentes niveles de densidad de conexión entre neuronas. [↑](#footnote-ref-9)