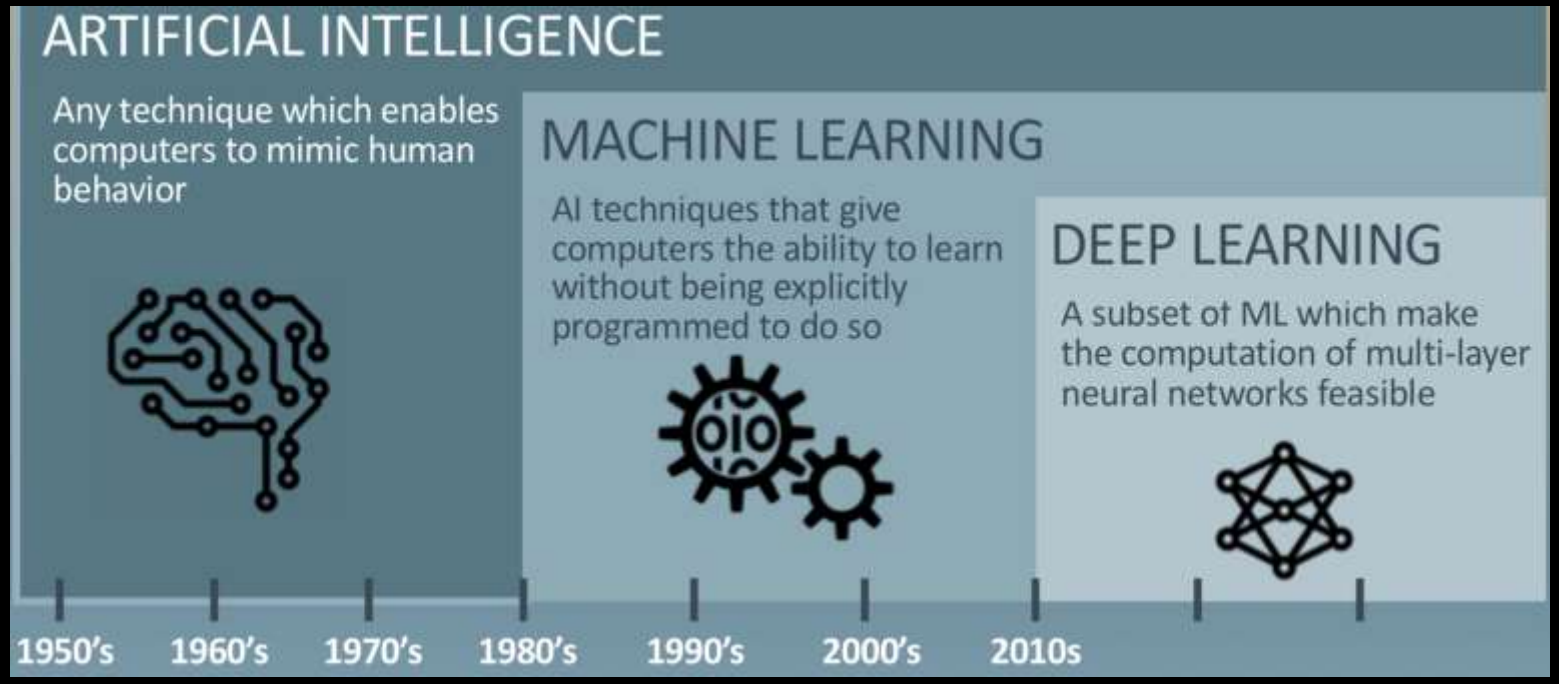
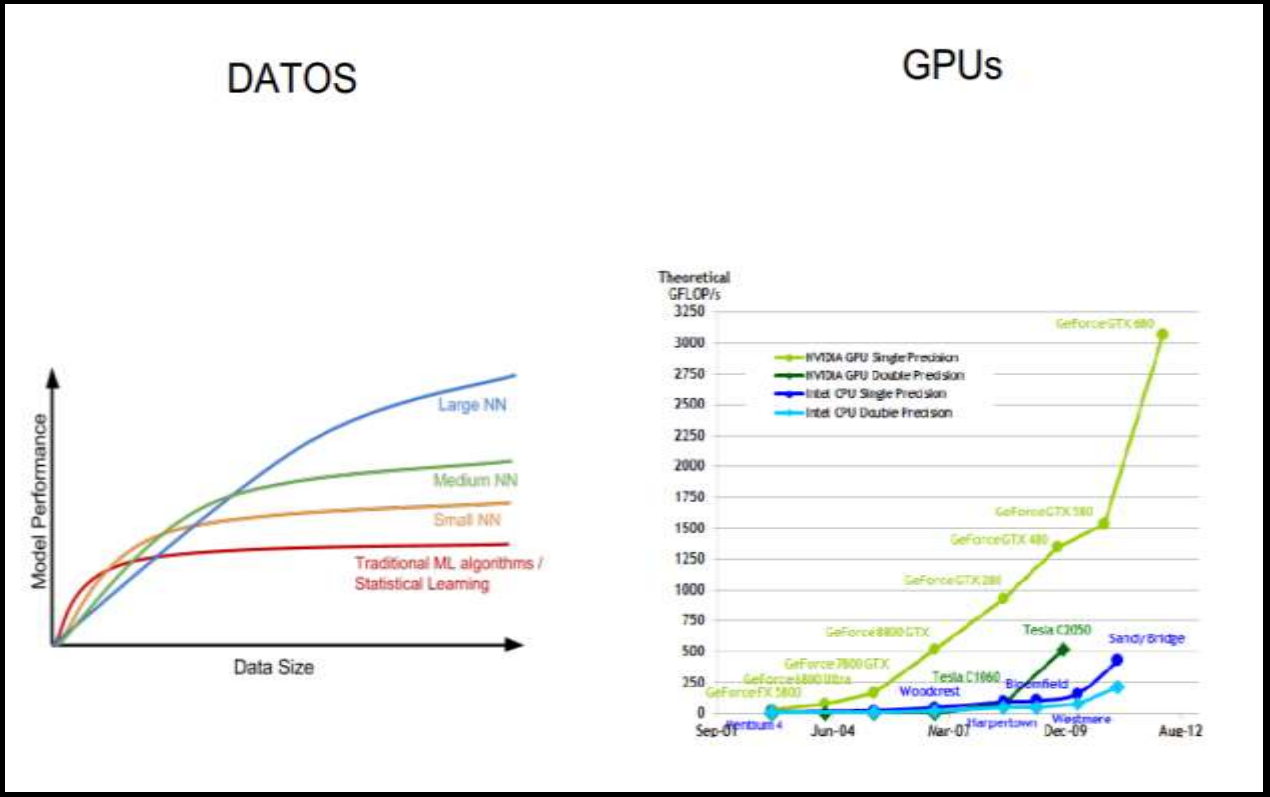
Deep Learning - Introduction

Deep Learning



¿Por qué ahora?

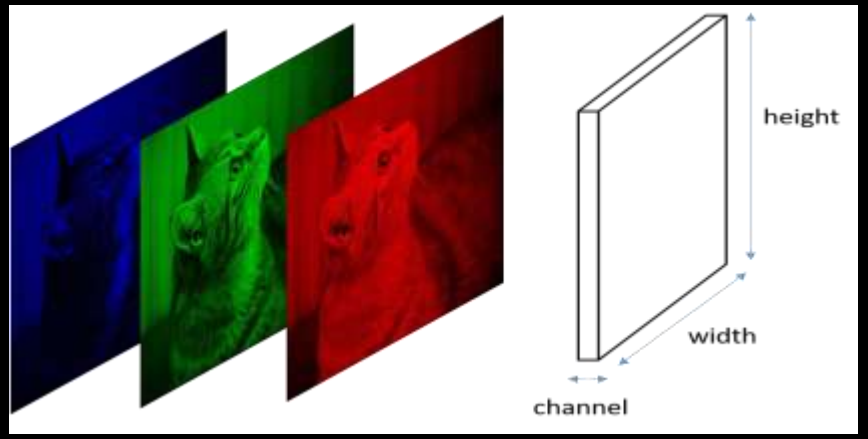
Los algoritmos tradicionales no escalan bien con gran cantidad de datos.
Ha crecido exponencialmente la capacidad de cómputo



¿Por qué ahora?

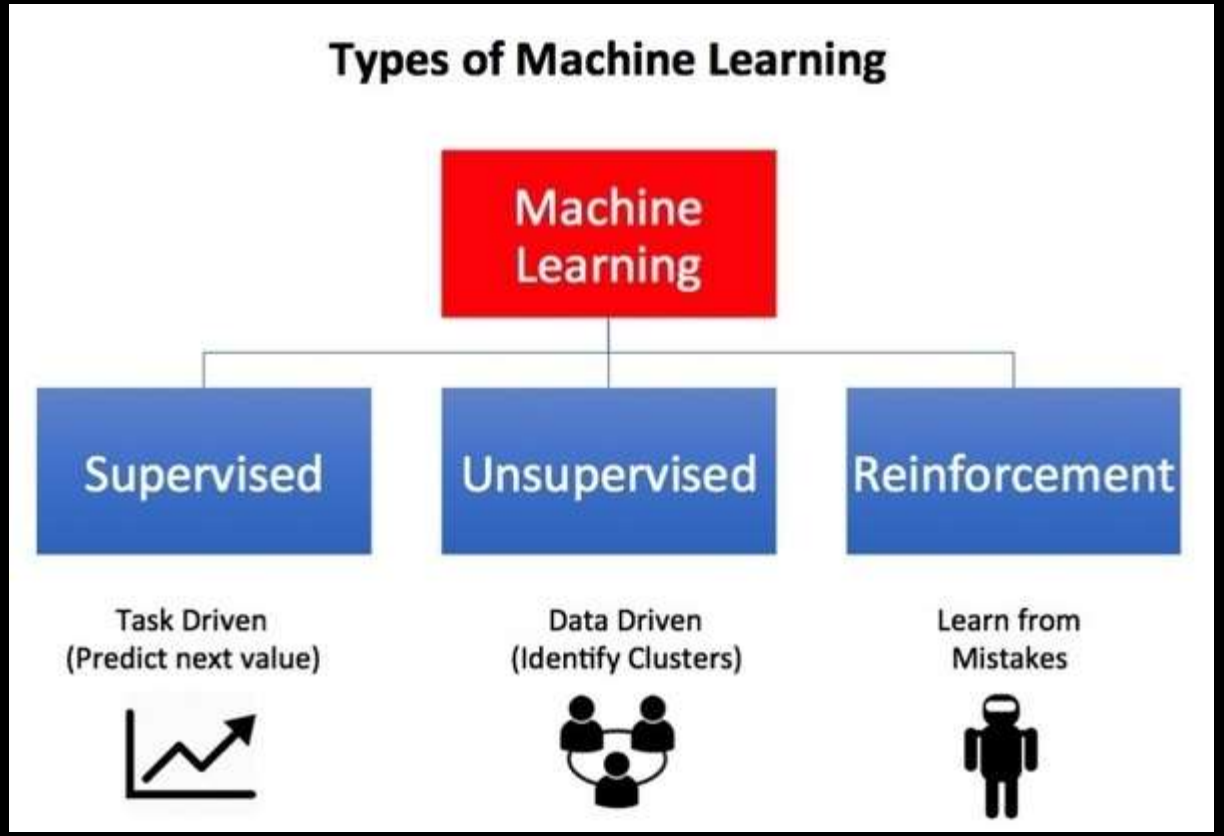
Complejidad de los datos

	A	B	C	D	E	F
1	Country	Salesperson	Order Date	OrderID	Units	Order Amount
2	USA	Fuller	1/01/2011	10392	13	1,440.00
3	UK	Gloucester	2/01/2011	10397	17	716.72
4	UK	Bromley	2/01/2011	10771	18	344.00
5	USA	Finchley	3/01/2011	10393	16	2,556.95
6	USA	Finchley	3/01/2011	10394	10	442.00
7	UK	Gillingham	3/01/2011	10395	9	2,122.92
8	USA	Finchley	6/01/2011	10396	7	1,903.80
9	USA	Callahan	8/01/2011	10399	17	1,765.60
10	USA	Fuller	8/01/2011	10404	7	1,591.25
11	USA	Fuller	9/01/2011	10398	11	2,505.60
12	USA	Coghill	9/01/2011	10403	18	855.01
13	USA	Finchley	10/01/2011	10401	7	3,868.60
14	USA	Callahan	10/01/2011	10402	11	2,713.50
15	UK	Rayleigh	13/01/2011	10406	15	1,830.78
16	USA	Callahan	14/01/2011	10408	10	1,622.40
17	USA	Farnham	14/01/2011	10409	19	319.20
18	USA	Farnham	15/01/2011	10410	16	802.00



Redes Neuronales: tipos de aprendizaje

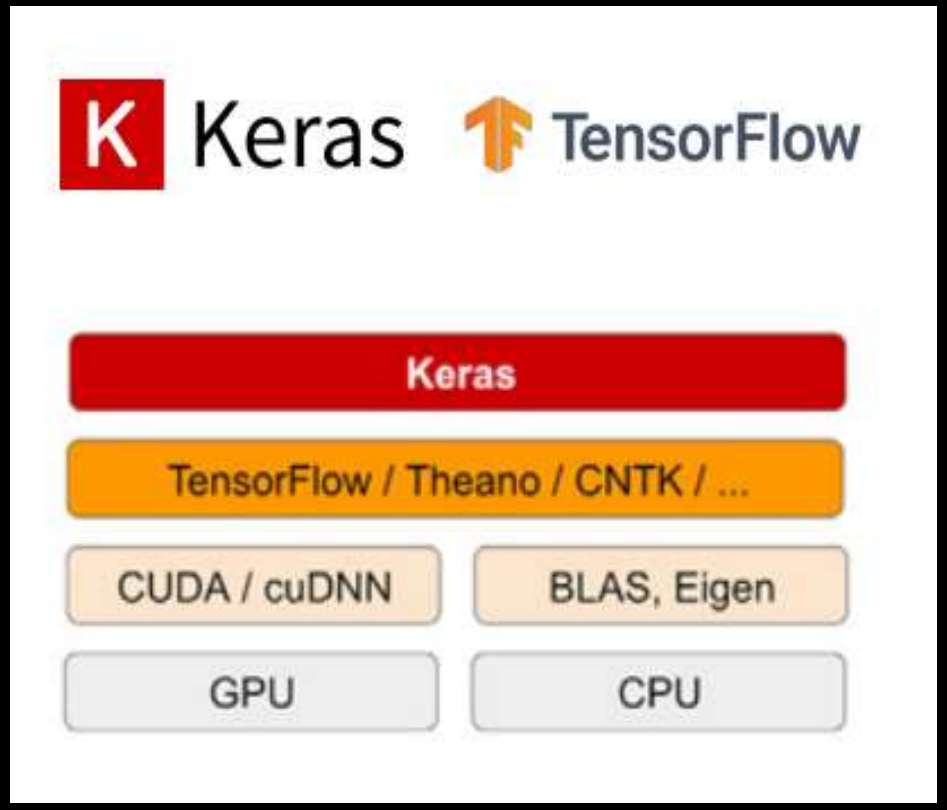
- La red neuronal funciona con cualquier tipo de problema: supervisado, no supervisado, por refuerzo, etc.
- Normalmente se usa para aprendizaje supervisado y por refuerzo.



Tensores

- ***Vector data*** — 2D tensors of shape (samples, features)
- ***Timeseries data or sequence data*** — 3D tensors of shape (samples, timesteps, features)
- ***Images*** — 4D tensors of shape (samples, height, width, channels) or (samples, channels, height, width)
- ***Video*** — 5D tensors of shape (samples, frames, height, width, channels) or (samples, frames, channels, height, width)

Herramientas

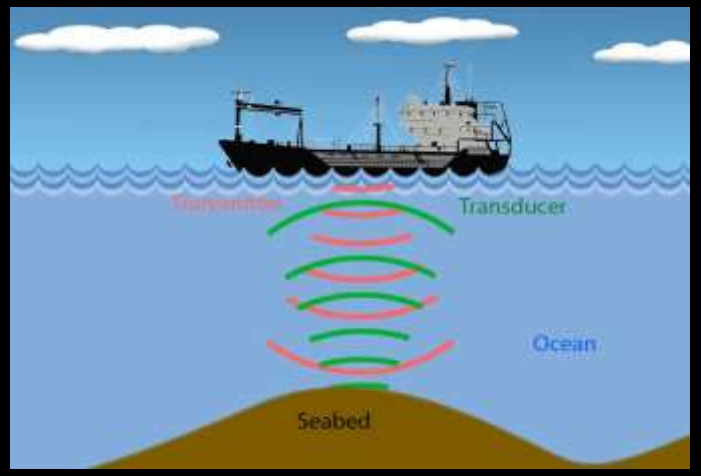


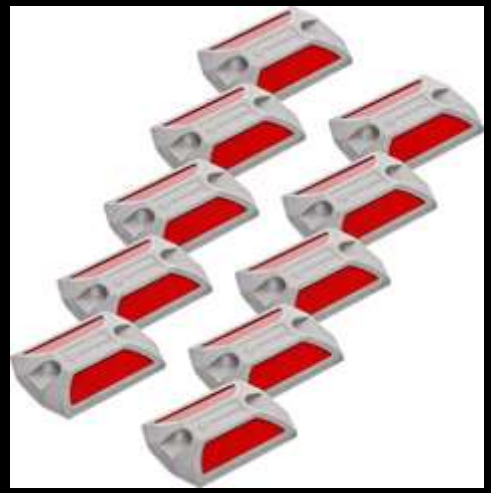
Algunas aplicaciones

Real time traductor for conference

[Trailers hechos por una IA]

Neural Networks

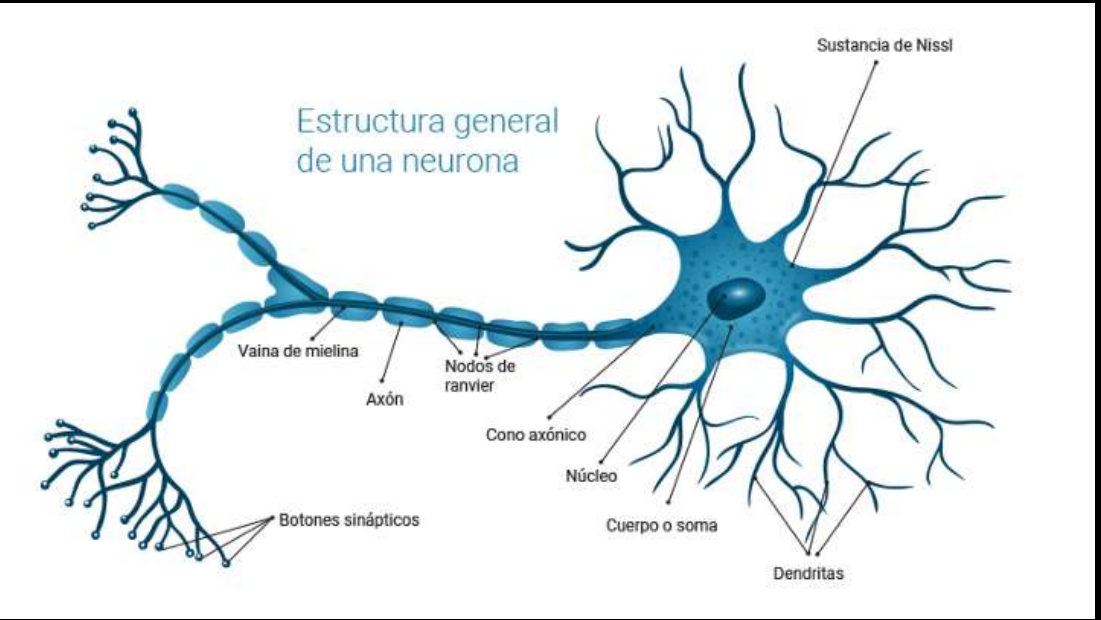

Transducer
Ocean
Seabed




CRESSI
CRESSI

Inspiración biológica

Las redes neuronales están inspirados en el concepto biológico de una neurona. Las neuronas recogen información de otras neuronas, a través de las dendritas. Estas señales producen cambios en el soma (cuerpo de la célula) mediante pequeñas señales eléctricas, y si la señal supera cierto umbral, se propaga a las siguientes neuronas a través del axón.



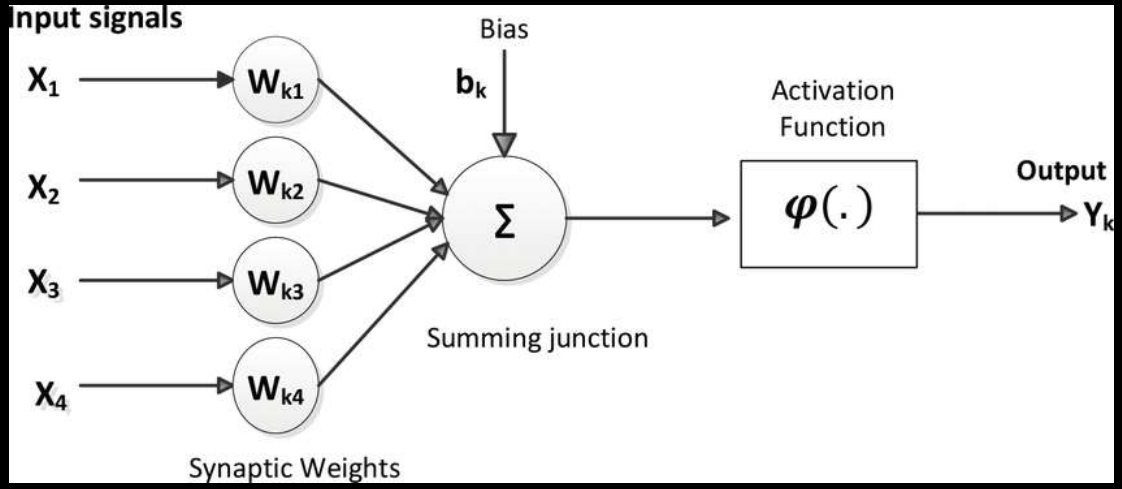
La neurona

McCulloch-Pitts neuron

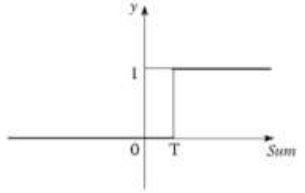
Componentes de la neurona:

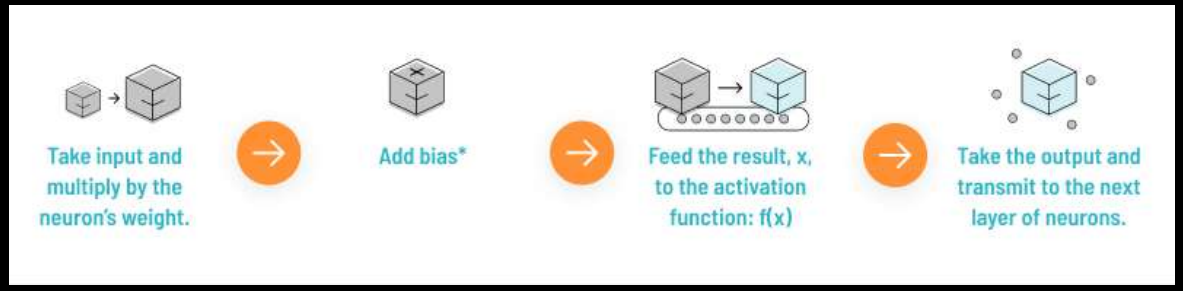
- **Inputs**
- **Pesos** que recibe cada input, se combinan como si de una regresión lineal se tratase
- Problemas binarios
- **Bias**
- **Activation function:** si la señal cumple cierto threshold, se activa la salida

Comportamiento similar a una regresión lineal



$$o_j = \begin{cases} 0 \ if \ \sum_i w_{ij}x_i < T \\ 1 \ if \ \sum_i w_{ij}x_i \geq T \end{cases}$$

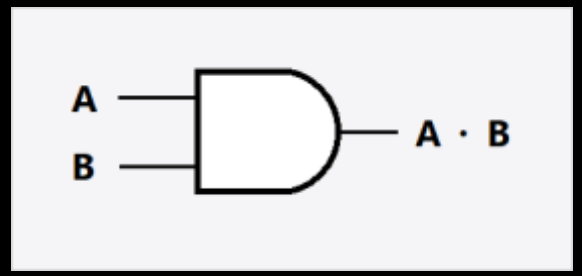




¿Qué resuelve una neurona?

Las neuronas en solitario no resuelven problemas complejos. Veamos algunos ejemplos

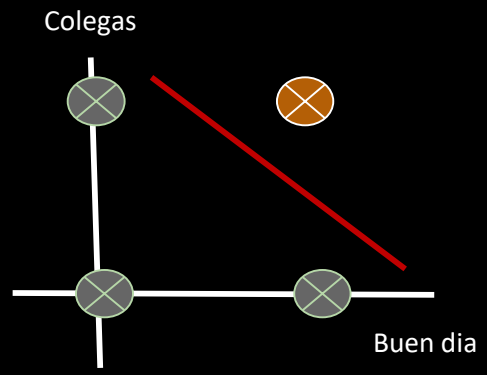
Puerta AND



Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Vamos a determinar si salimos de excursión el fin de semana mediante una puerta AND

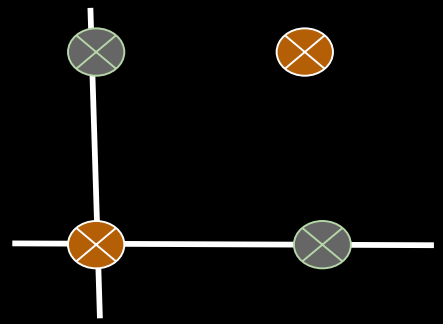
Buen dia	Colegas	Excursión
0	0	0
0	1	0
1	0	0
1	1	1



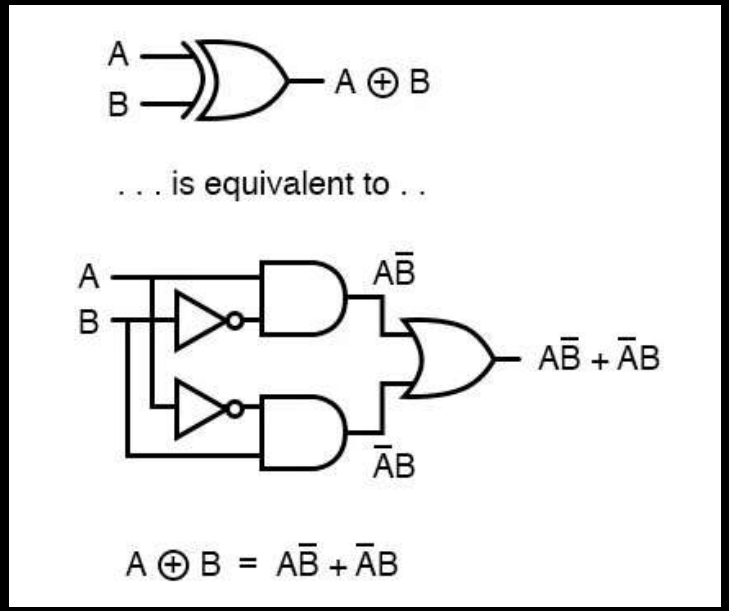
Problemas más avanzados

Vamos a intentar resolver una puerta XOR

Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0



Es imposible separar los puntos con un único hiperplano.
¿Solución? Combinar varias neuronas



Entonces si tenemos puertas lógicas, ¿para qué necesitamos las neuronas?

Perceptrón

A diferencia de una simple neurona o una puerta lógica, aprende automáticamente los pesos de la neurona de McCulloch-Pitts para minimizar los errores. Se utilizan para problemas separables linealmente. Ya no son datos binarios, sino que pueden ser números.

Inventado por Frank Rosenblatt en 1957

¿Cómo se entrena?

1. El perceptrón se inicializa con unos pesos a 0 o aleatorios
2. Calcula las salidas, a través de la combinación lineal de entradas y pesos + bias
3. Atraviesa la función de activación
4. Evalúa errores en la salida
5. Vuelve a ajustar los pesos hasta que minimiza el error.

No muy efectivo para problemas con separación no lineal

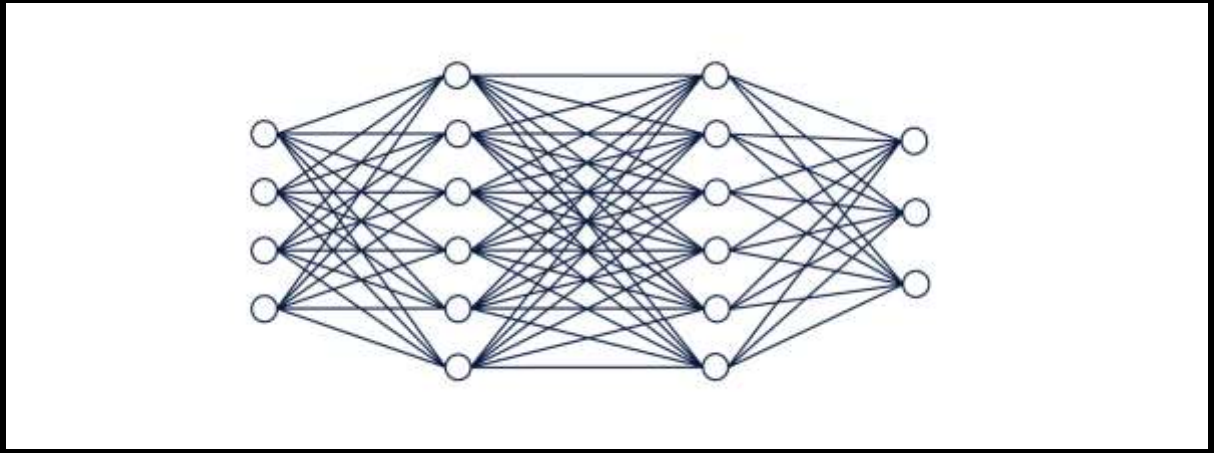
Si los perceptrones resuelven problemas simples, ¿qué están aportando?

ANN (Redes Neuronales Artificiales)

Agregación de neuronas en capas, de tal manera que los outputs de una capa son los inputs de la siguiente. Cada conjunto de neuronas se va entrenando y especializando en resolver partes del problema, como a solventar una puerta XOR.

Algoritmos de entrada/salida, que simulan el comportamiento del cerebro humano. Se utilizan una serie de neuronas artificiales o nodos para solucionar problemas.

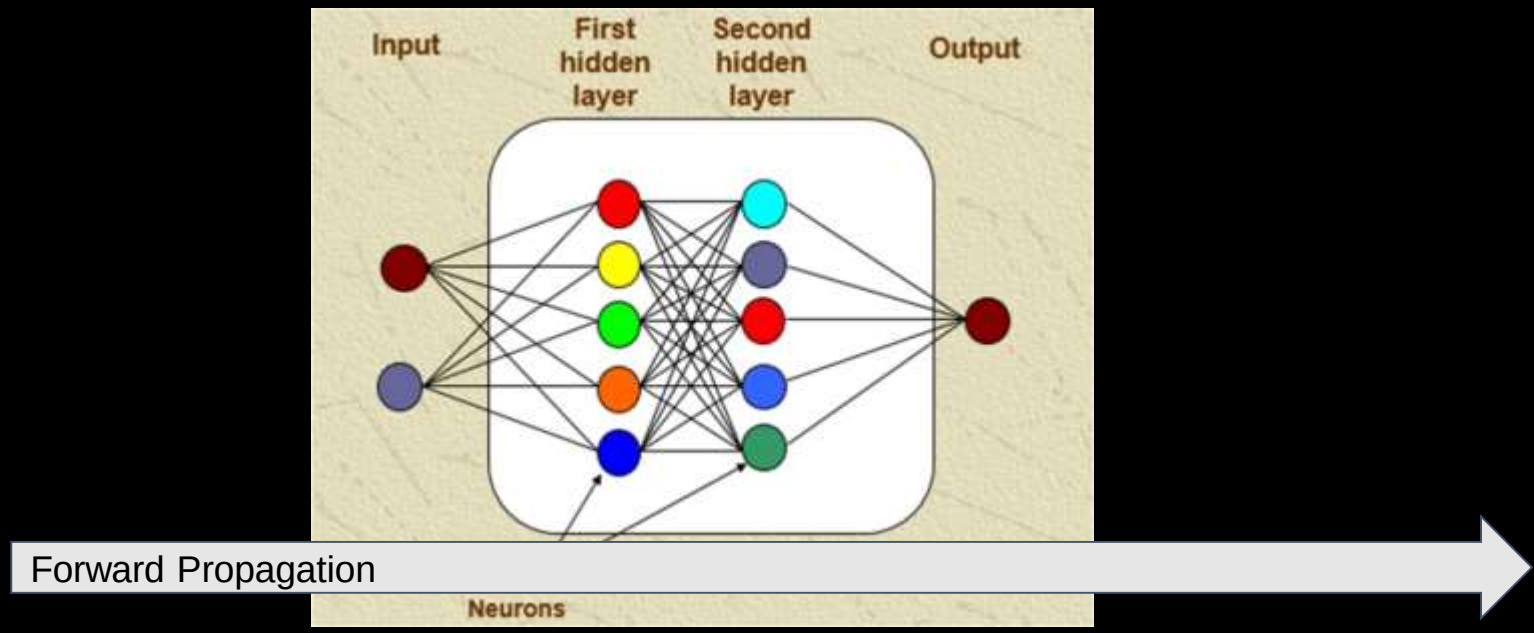
Son algoritmos de caja negra



MLP
MultiLayer Perceptron

Componentes MLP

1. **Input layer**: NO tienen capa de activación. Cada nodo de entrada es una feature.
2. **Hidden layers**: capas intermedias. Puede ser una o varias. Tienen función de activación y término bias. Con una capa se suele llamar MLP y si tiene múltiples **Deep Neural Network.**
3. **Output layer**: posibles salidas de una red neuronal.
    a. Una neurona: clasificación binaria o regresión
    b. Varias neuronas: clasificación multiclase. Tantas neuronas como clases.



Deep Learning, Artificial Neural Network, Perceptron, MultiLayer Perceptron...

¿Quién es quién?

Algunos conceptos

Neurona artificial
Unidad lógica inspirada en el comportamiento de las neuronas cerebrales. Permite resolver problemas sencillos como una puerta lógica AND u OR.

Perceptrón
La red neuronal artificial más simple. Emplea una neurona artificial para resolver problemas de clasificación y regresión. El perceptrón entrena con los datos de entrada como cualquier algoritmo de machine learning, intentando minimizar sus errores.

ANN - Red Neuronal Artificial
Modelo matemático que simula la forma en que el cerebro responde a estímulos. Está compuesta de una o más neuronas artificiales. Se utiliza en problemas de clasificación o regresión. Un diagrama de red define las relaciones entre las señales de entrada (x) y salida (y).

MLP - MultiLayer Perceptron
Se trata de una ANN compuesta por varias neuronas, pero en este caso se distribuyen en capas. Tiene una input layer, varias hidden layers y una output layer. Se suele denominar MLP a las redes neuronales con una sola hidden layer. También se les llama Shallow Neural Network.

Deep Neural Network
Las redes con una única hidden layer se suelen denominar MLP, mientras que las que tienen varias hidden layers son Deep Neural networks.

Deep Learning
Algoritmos de machine learning que utilizan redes neuronales para aprender patrones en los datos y resolver problemas complejos como tratamiento de imágenes o de sonido.

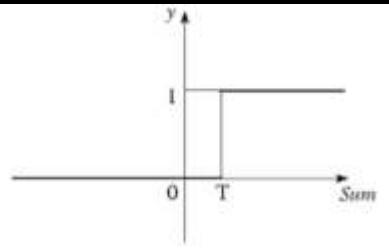
Funciones de activación

Hasta ahora hemos usado la *step function* como función de activación, pero existen otras que permiten obtener un resultado contínuo. Permite a la red neuronal **aprender patrones no lineares** y adaptarse a problemas más complejos.

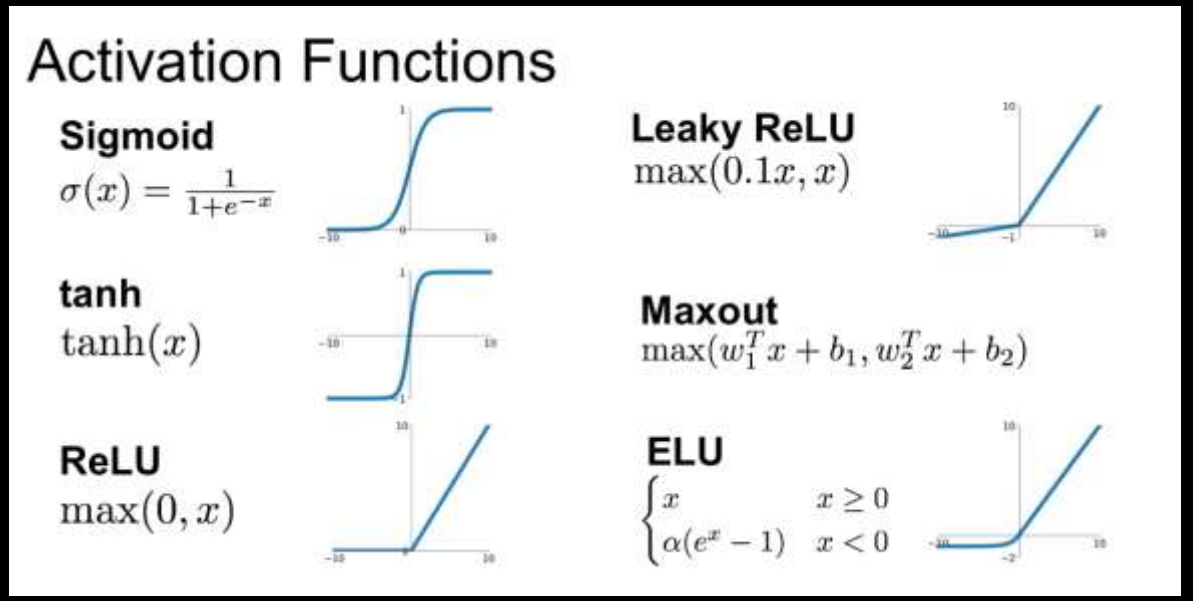
Dado que la *step function* no es derivable, hay que acudir a otras funciones de activación para poder recurrir en el entrenamiento a métodos como Gradient Descent.

La que más se suele utilizar es la ReLu (Rectified linear), a pesar de no ser derivable en 0. Computacionalmente es eficiente

$$o_j = \begin{cases} 0 \ if \ \sum_i w_{ij} x_i < T \\ 1 \ if \ \sum_i w_{ij} x_i \geq T \end{cases}$$



[Buen artículo de funciones de activación]



Elección de funciones de activación

- **¿Función de activación de la entrada?** NO LLEVA
- **¿Problema de regresión?** Función de activación lineal, como ReLu o Leaky Relu
- **¿Clasificación binaria?** Sigmoide. Funciona bien. Computacionalmente es pesada y por eso solo se recomienda al final.
- **¿Clasificación multiclase?** Softmax es la mejor, acota los datos entre 0 y 1, que es lo que nos interesa, la probabilidad de pertenencia a una clase.
- **¿Capas intermedias?** ReLu es la que mejores prestaciones nos da. Buenos resultados y computacionalmente eficiente.

- **Sigmoide y tanh**: son funciones que van muy bien, pero lentas de entrenar. Sus derivadas parciales son muy pequeñas en los extremos de la curva, por lo que puede no llegar nunca a converger (vanishing gradient problem). Usar en la salida.

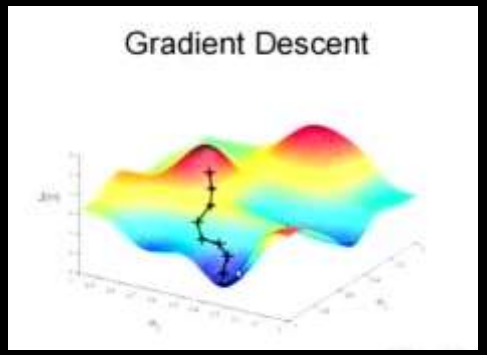
¿Cómo entrena la red neuronal?

Backpropagation

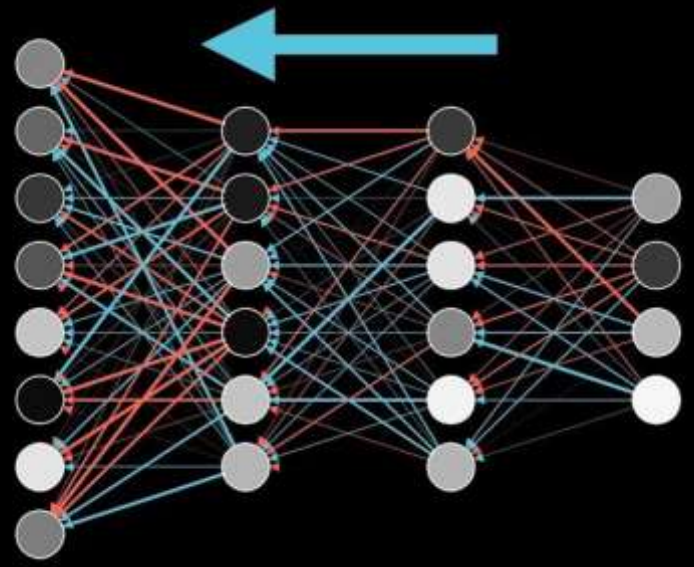
Algoritmo de entrenamiento de una red neuronal. Tras la forward propagation, la predicción se compara con el true label y obtenemos un error. Este error se propaga hacia atrás para corregir los valores de los pesos de las funciones de transformación de las neuronas

¿Cómo funciona?

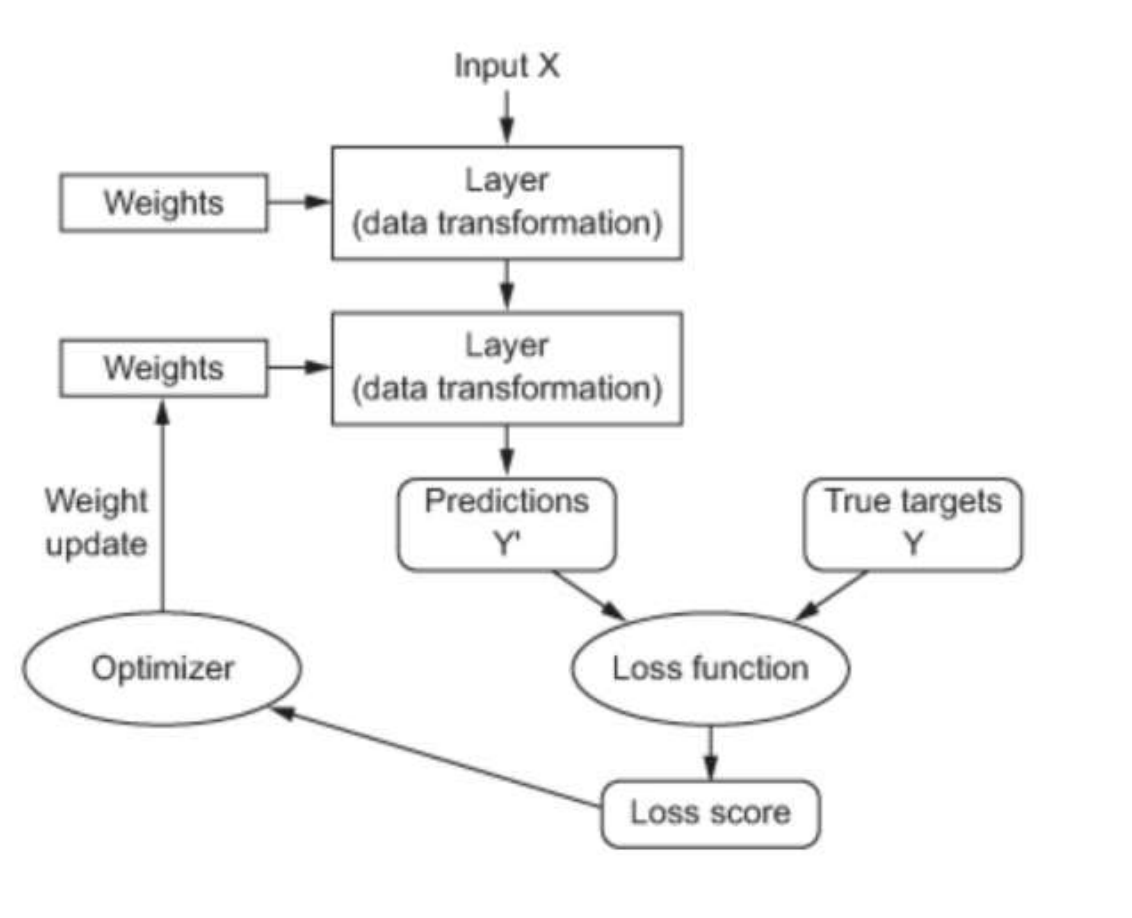
1. Se inicializan los pesos de las conexiones de manera aleatoria
2. Pasamos los datos de entrenamiento a través de la red
3. Comparamos a la salida con los true labels
4. Esto nos da un error (cost function), que depende de todos los pesos y los bias de la red. El error no es más que el MSE (Mean Squared Error)
5. ¿Objetivo? Ya nos lo sabemos. Minimizar ese error. → Optimizadores (Gradient Descent, SGD, Adam...)
6. Se propaga el error hacia atrás para ver qué neurona ha tenido mayor responsabilidad.
7. Se computa el gradiente para cada uno de los pesos, y se actualizan en la dirección de minimización del error
8. Cuando el gradiente sea 0, ya hemos alcanzado el mínimo de error, el algoritmo termina, y si no, volvemos al punto 6.



$$E(\boldsymbol{w},\boldsymbol{b}) = \frac{1}{2n}\sum_{i=1}^{n}\|\boldsymbol{y}_i - \boldsymbol{o}_i\|^2$$



Proceso entrenamiento



1. Entrena con una cantidad de muestras fijada por nosotros, llamada *batch_size*

2. Con cada fase del entrenamiento, va actualizando los pesos.

3. Cuando se le acaban las muestras del dataset, vuelve a realizar el mismo procedimiento.

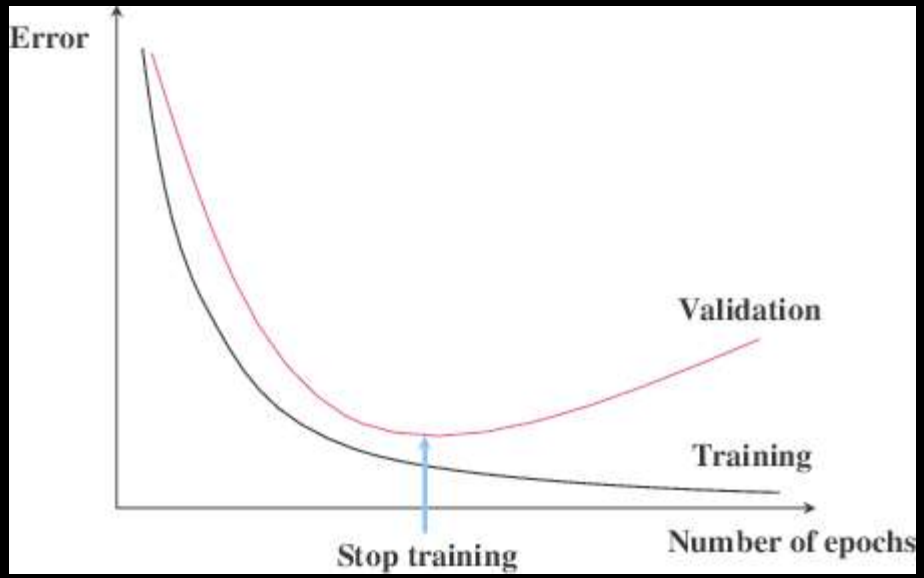
4. ¿Cuántas veces lleva a cabo este procedimiento? Lo definimos nosotros con el parámetro ***epochs***.

Early stopping

Como en todo algoritmo de machine learning podemos sufrir de **overfitting**. En particular las redes neuronales son bastante sensibles al overfitting ya que mediante el algoritmo de backpropagation se van actualizando los pesos hasta minimizar el error. Este proceso puede no acabar nunca.

Las iteraciones hacia atrás que realiza el algoritmo se denominan **epochs**.

Existe una técnica llamada **early stopping** que nos permite parar el entrenamiento cuando empiezan a empeorar los resultados en validación. En ese momento habremos encontrado el punto óptimo de bias vs variance.



Optimización

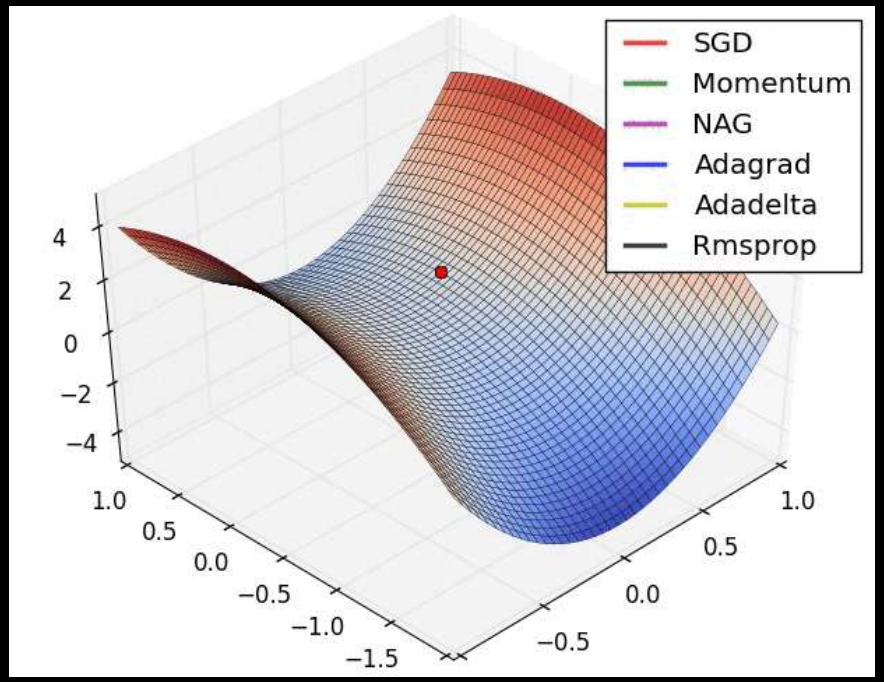
Los algoritmos de ML buscan siempre minimizar sus errores en los conjuntos de test y validación. Matemáticamente hablando esto es un problema de optimización, en el que tenemos una función de coste, como podría ser el MSE, y hay algoritmos iterativos que se encargan de llegar al mínimo de esa función, como el Gradient Descent.

Existen diferentes funciones de optimización, cada una con diferentes maneras de llegar al punto mínimo.

- Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent
- Momentum
- Nesterov Accelerated Gradient
- Adagrad
- AdaDelta
- Adam
- (…)

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001),
              metrics=['accuracy'])
```

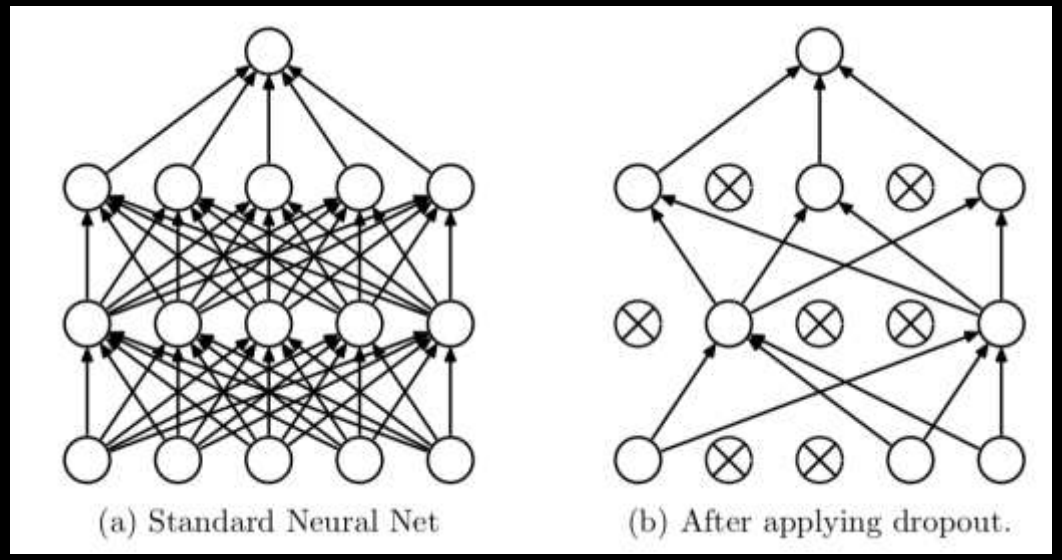


Detalle de optimizadores

Regularization

Como sabes, la regularización consiste en aplicar penalizaciones al modelo en la etapa de entrenamiento. Estas técnicas nos sirven para reducir el overfitting y en RRNN también las tenemos.

Un ejemplo de penalización muy típico en RRNN es el dropout, que consiste en eliminar algunas neuronas aleatoriamente en el proceso de entrenamiento. Las neuronas se van “apagando y encendiendo” aleatoriamente en cada step del algoritmo de entrenamiento.



(a) Standard Neural Net (b) After applying dropout.

Method	Test Classification error %
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

Demo

https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=&seed=0.14894&showTestData=false&discretize=false&percTrainData=50&x=true&y=false&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false