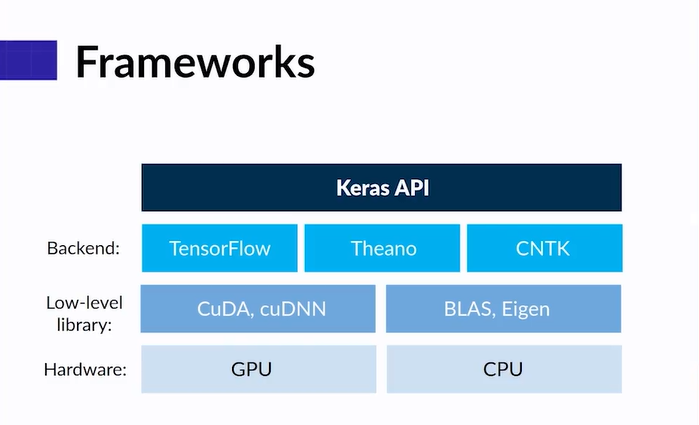
**Fundamentos de redes neuronales con Python y Keras**

* El perceptrón es la estructura base una red neuronal artificial, siguiendo la forma de una neurona natural.

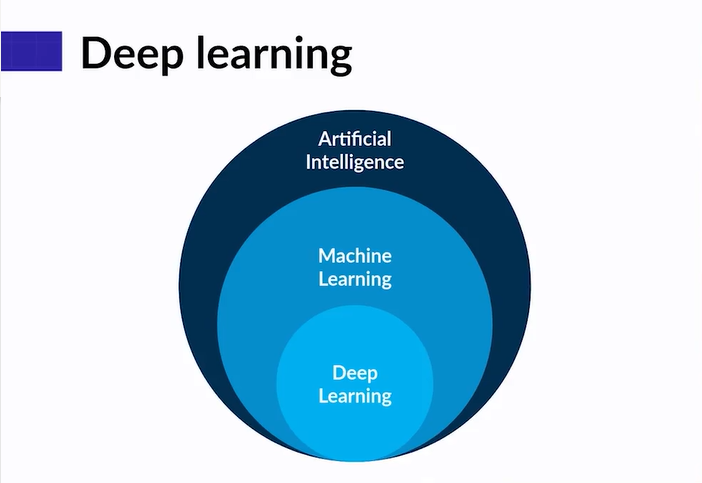
1. **Herramientas principales para el uso de redes neuronales:**

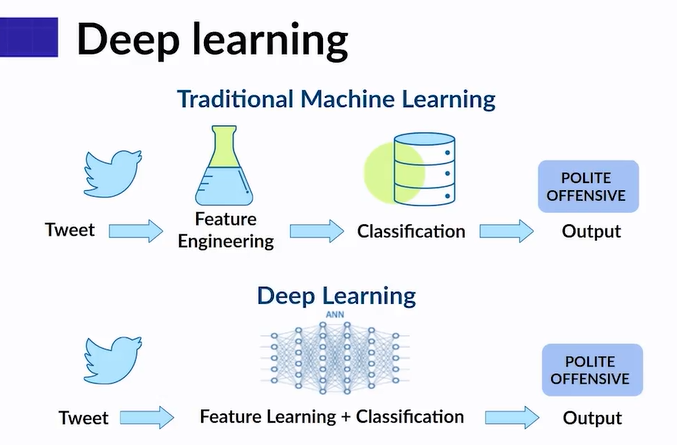
* TensorFlow y PyTorch
* Keras no es una biblioteca sino una API para optimizar el uso de TensorFlow.



1. **¿Qué es Deep learning?**

* A medida que tengo + capas conectadas entre si tendré resultados más profundos y exactos.

****

****

**2.1 Principales restricciones del Deep learning:**

* Overfitting: el modelo se aprende los datos más no aprende a generalizar. En redes es común porque los datos utilizados cada vez se vuelven más específicos dentro de las capas ocultas.
* Sistema de caja negra: no sabemos que pasa exactamente en las capas ocultas y por que toma ciertas decisiones.

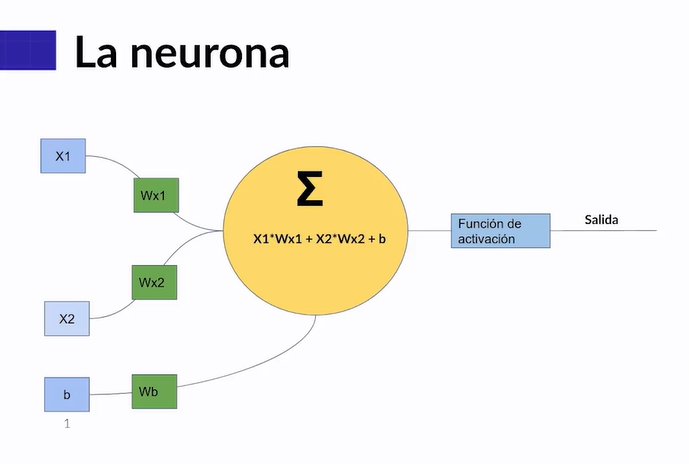
¿Es ético usar redes neuronales sabiendo que no sabemos que pasa dentro de ellas y como toman decisiones?

\*\*Herramienta para ver como funciona una red convolucional (para imágenes): <https://adamharley.com/nn_vis/cnn/3d.html>

1. **Red neuronal con Keras**

Ver notebook mi primera red neuronal:

1. **La neurona**

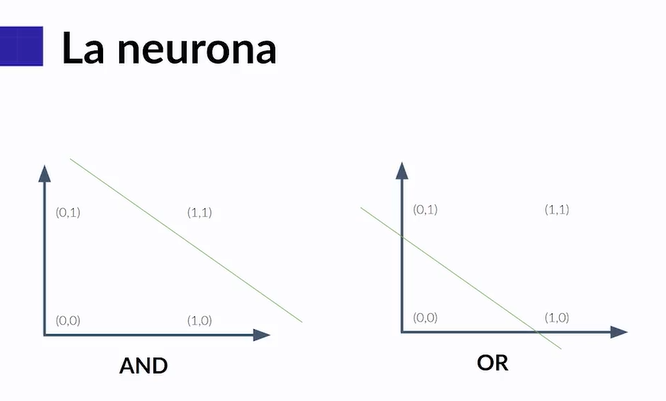


La neurona tiene entradas, pesos, sumas ponderadas (similar a un OLS). Esto pasa por una función de activación y entrega un resultado. Los pesos de iteran hasta que se resuelva el problema que la red neuronal busca resolver.

Hay compuertas AND y compuertas OR.

Compuerta AND: solo acepta 1 tipo de resultado

Compuerta OR: acepta 2 tipos de resultados.



Usualmente, las soluciones no se hallan de forma lineal. Por lo tanto, un solo perceptrón no es suficiente.

Gráfico, Gráfico de dispersión

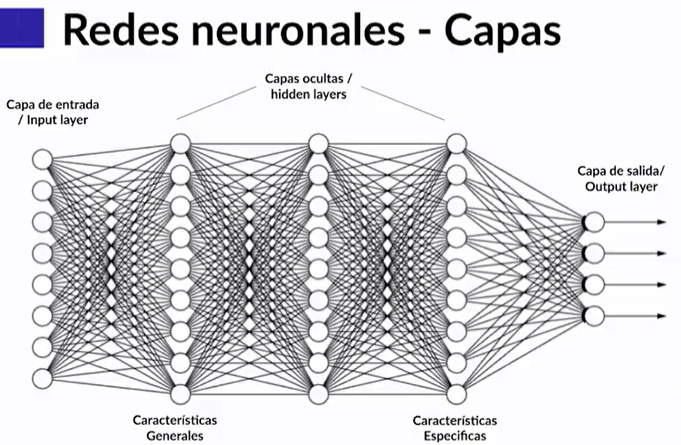
Descripción generada automáticamente

Al usar varios perceptrones (varias neuronas) estamos armando una red neuronal.

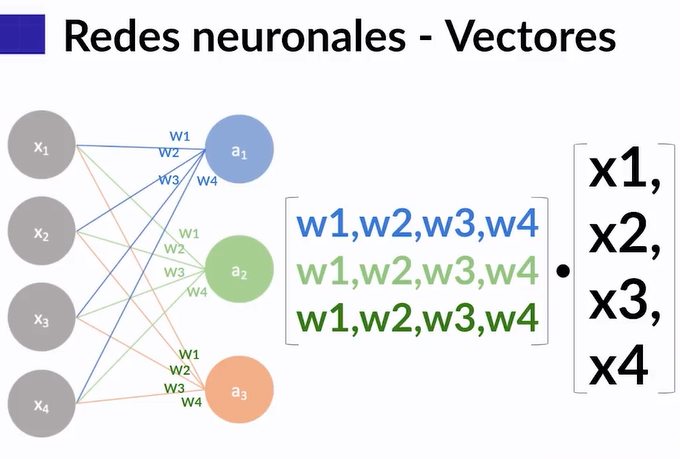
Diagrama

Descripción generada automáticamente

1. **Arquitectura de una red neuronal**
   * Una red neuronal se maneja x capas, cada capa pasa información a la siguiente capa.
   * Capa inicial: capa de entrada
   * Capas ocultas: capas intermedias
   * Capa de salida: resultados, predicciones.

****

* Se usa producto punto entre los valores de entrada y los pesos para resolver los problemas:

****

**Diagrama, Texto

Descripción generada automáticamenteTexto

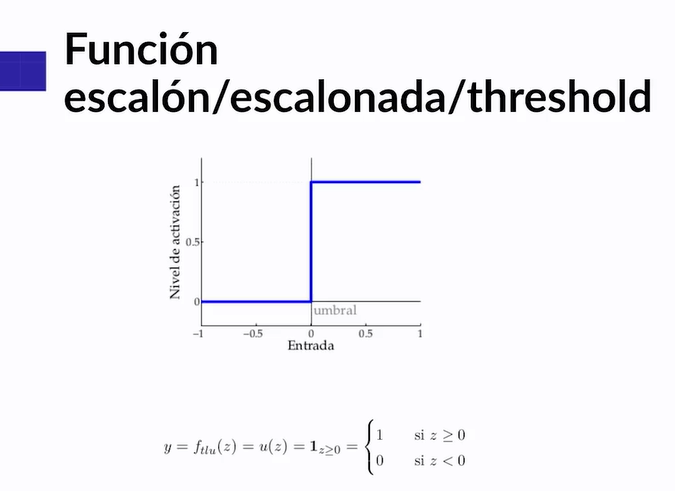
Descripción generada automáticamente**

* Las operaciones que se hacen en una red neuronal no son 100% lineales, pues si se maneja así el resultado también será lineal y se pierde la esencia de esta metodología. Por ello la función de activación cumple un importante papel.

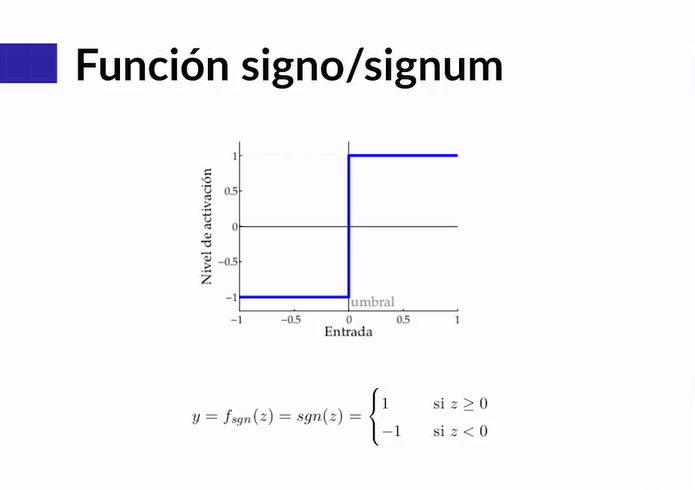
1. **Funciones de activación**

* Hay funciones de activación discretas y continuas.

**Tipos de funciones:**

* 

Se usa cuando el resultado solo toma 2 valores.

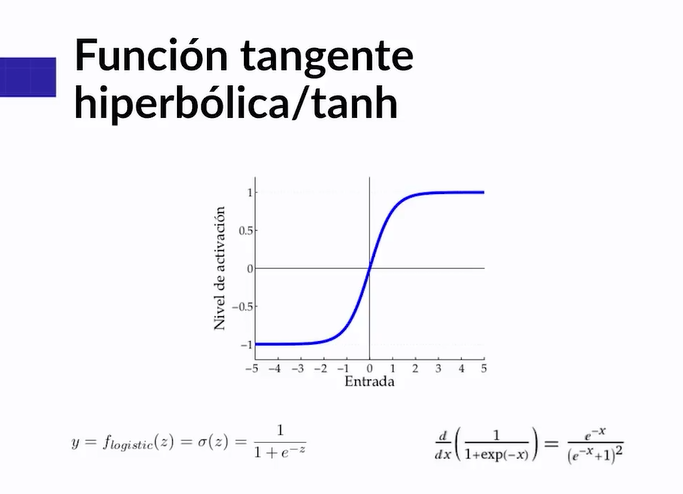


Esta admite valores negativos.

Gráfico, Gráfico de líneas, Histograma

Descripción generada automáticamente

Esta función usa derivadas, se usa más que todo para probabilidades. El problema es que los valores tienden a acumularse alrededor de 0 y 1.

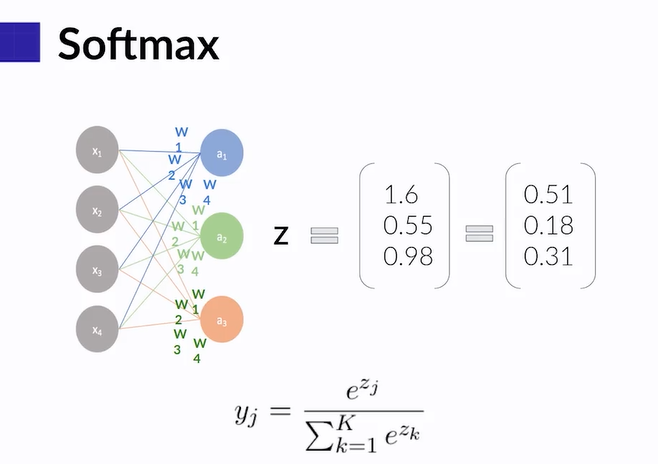


Similar a la sigmoide, pero admite valores negativos.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Esta es la función más utilizada. Si el valor es menor a cero, pone cero. Si no es cero, deja el valor que viene. No es una función lineal porque se deforma cuando los valores son negativos. Es la más utilizada en capas ocultas.



Esta función da la probabilidad de cada una de las posibles salidas. Se usa mucho para clasificación binaria y multiple.

1. **Función de perdida**

Al final, la red neuronal busca hacer una predicción, ya sea de una regresión o clasificación.

La red neuronal toma los valores reales y los compara con las predicciones, y las diferencias entre estos valores es la función de perdida.

Algunos tipos de función de perdida:

* MSE: castiga las diferencias amplias, es decir, las observaciones con predicciones con un error alto.
* Cross Entropy: evalua la distancia entre las predicciones, el valor real y arroja un score.

Las redes neuronales actualizan los pesos a partir de las derivadas de las funciones de activación y las funciones de perdida. La red toma los resultados de la función de perdida, los deriva y a partir de eso ajusta los pesos.

**Ver script Funciones**

1. **Descenso del gradiente**

* Al derivar, asumiendo que se quiere optimizar la función de perdida, se quiere obtener el resultado más bajo posible.
* **Learning rate:** la distancia de los pasos que hay que dar para encontrar el punto más bajo de la función. Un learning rate muy alto (pasos muy grandes) no podrá encontrar el mejor punto mínimo. Un learning rate muy bajo (pasos muy pequeños) tardara mucho en encontrar el mejor punto mínimo y requerirá potencia de cómputo.

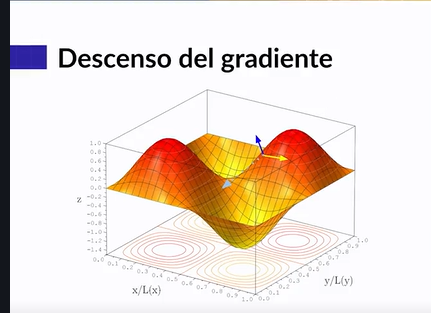


Las funciones a veces tienen mas de un mínimo, como mínimos locales y el mínimo global. Lo ideal es siempre llegar al máximo global. Para alcanzarlo, se hace uso del concepto físico **Momentum.**

Con la optimización adecuada, se pueden omitir los mínimos locales y llegar al mínimo local.

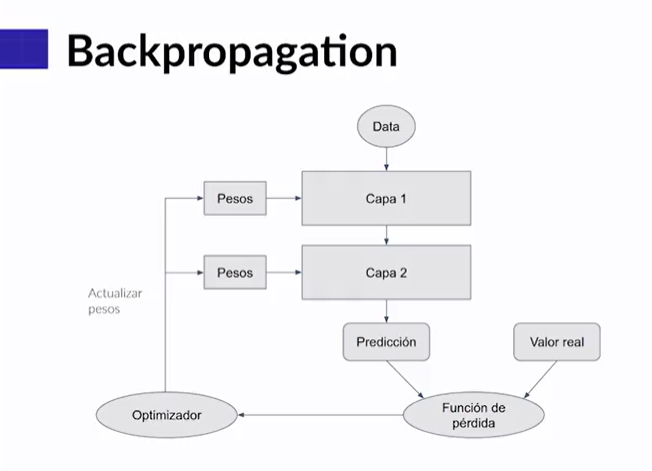
El optimizador dice como se deben optimizar los pesos al hacer el descenso de gradiente en la función de perdida.

Para llevar la función de perdida a los valores más bajos, se hace uso de las derivadas parciales, las cuales dan el gradiente, el cual debo multiplicar por (-1).



**9.Backpropagation**

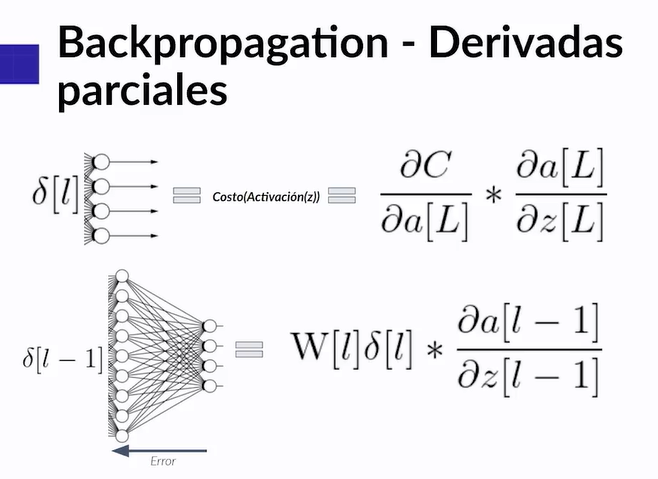
* Como se distribuye el error que se obtuvo en la función de perdida en la red neuronal.



Los pesos se van actualizando de acuerdo a la función de perdida y el descenso de gradiente.

Cuando los resultados arrojan una función de coste alta, no es posible saber con exactitud en que capa de la red neuronal se genera en mayor proporción este error. Por ello es necesario tener una técnica para distribuir el error entre las capas.

La solución es hacer uso de la técnica **backpropagation,**  la cual por medio del uso de derivadas parciales distribuye el error de atrás hacia adelante. (es decir, inicia en la última capa)



**Playground TensorFlow:** <https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.78652&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

1. **Entrenamiento, validación y test**

Lo ideal es dividir el set de datos en 3: entrenamiento, validación y test.

* Entrenamiento: entrena los datos.
* Validación: probar el entrenamiento, validamos predicciones y así ver si debemos cambiar hiperparametros.
* Test: se usa para evaluar el modelo.

Esto se hace por una cuestión ética, y es que el modelo se debe probar solo con datos que el modelo jamás haya visto o procesado.

1. **Soluciones al Overfitting**

Soluciones:

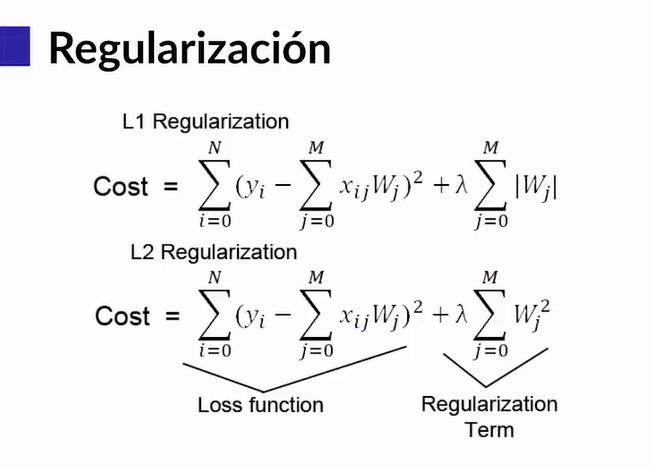
* Usar un modelo más sencillo
* Regularización
* Drop-out

**Regularización:** hacer los datos más regulares, usar la solución más simple. Es decir, hacer el modelo más simple usando valores más bajos en la red. La regularización le da más péso a la función de costo para que sea más castigada.

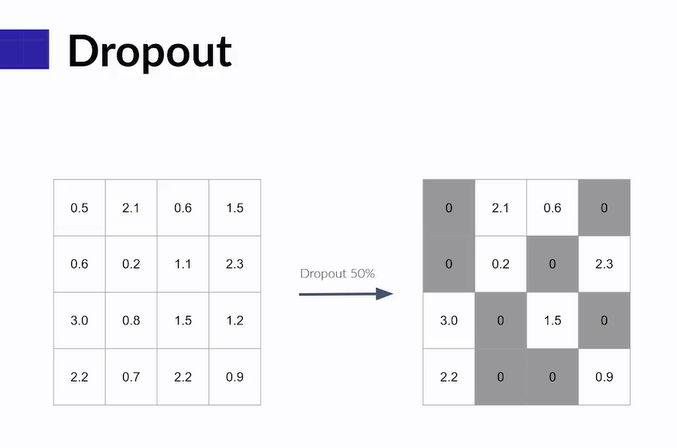
Hay 2 tipos:

* Regularización L1: usa los valores absolutos de los pesos de la neurona y se suman a la función de costo.
* Regularización L2: usa los valores al cuadrado de los pesos de la neurona y se suman a la función de costo.

Las 2 usan un valor delta, el cual si es muy bajo castiga poco y si es muy alto castiga mucho.



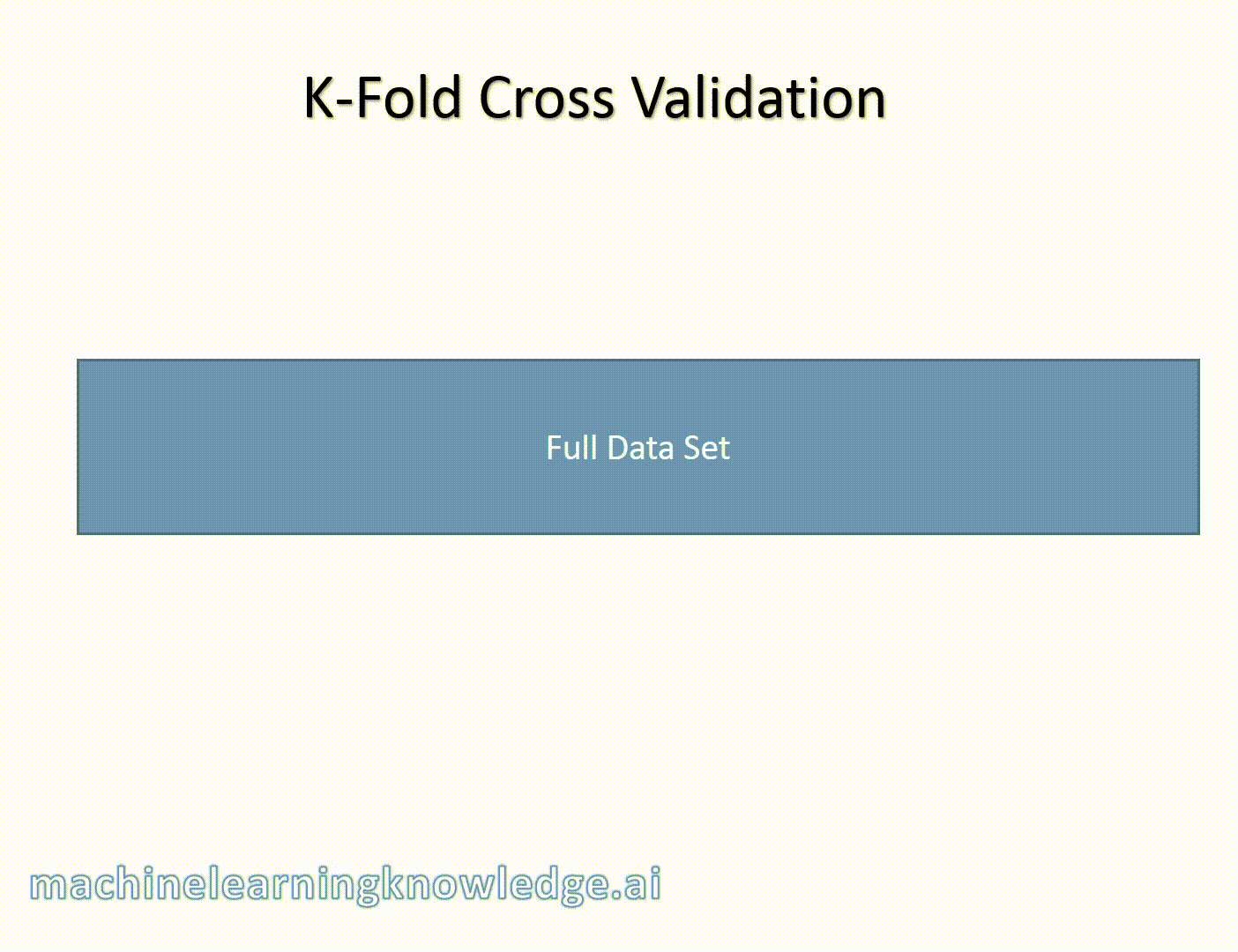
**Dropout:** apagar ciertas neuronas en la red neuronal. En cada iteración se apaga un porcentaje de la red neuronal de forma aleatoria.



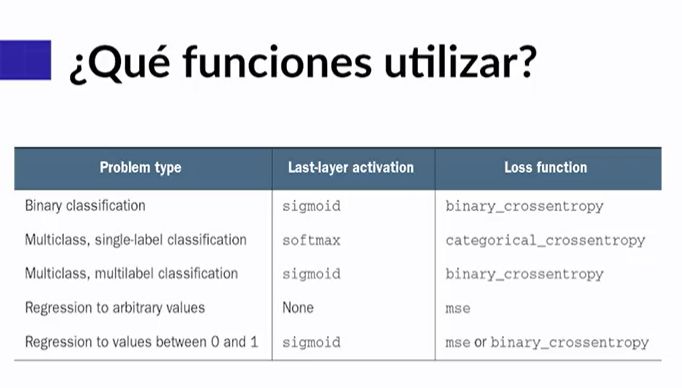
Diagrama

Descripción generada automáticamente

1. **Validación cruzada (K-Fold Cross Validation)**



1. **Funciones a usar en la ultima capa de la red dependiendo del caso**

****