

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.

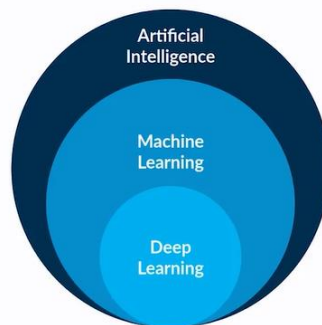
Frameworks



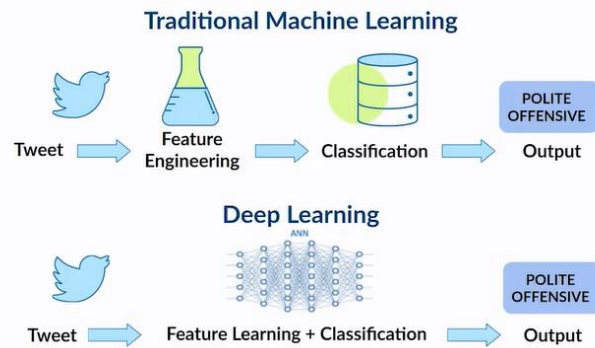
2. ¿Qué es Deep learning?

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

Deep learning



Deep learning



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):

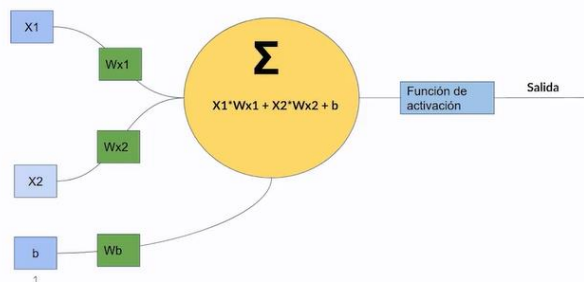
https://adamharley.com/nn_vis/cnn/3d.html

3. Red neuronal con Keras

Ver notebook mi primera red neuronal:

4. La neurona

La neurona



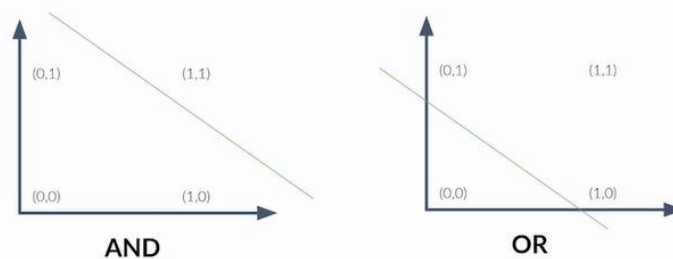
La neurona tiene entradas, pesos, sumas ponderadas (similar a un OLS). Esto pasa por una función de activación y entrega un resultado. Los pesos se iteran hasta que se resuelva el problema que la red neuronal busca resolver.

Hay compuertas AND y compuertas OR.

Compuerta AND: solo acepta 1 tipo de resultado

Compuerta OR: acepta 2 tipos de resultados.

La neurona



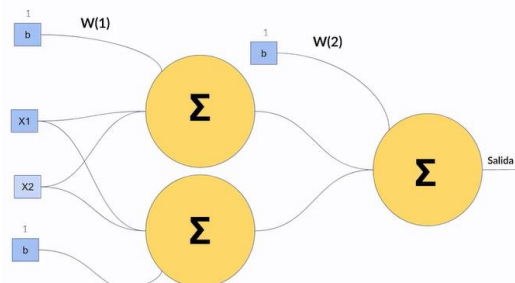
Usualmente, las soluciones no se hallan de forma lineal. Por lo tanto, un solo perceptrón no es suficiente.

La neurona - XOR



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

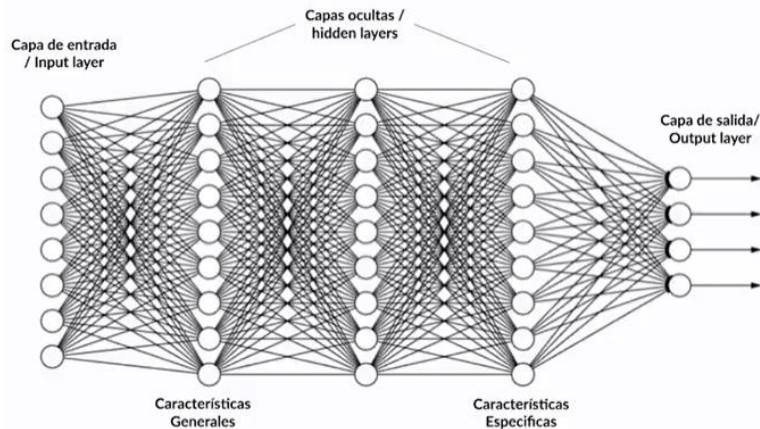
La neurona - XOR



5. Arquitectura de una red neuronal

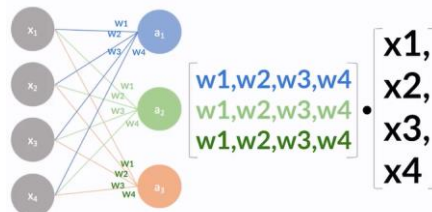
- Una red neuronal se maneja x capas, cada capa pasa información a la siguiente capa.
- Capa inicial: capa de entrada
- Capas ocultas: capas intermedias
- Capa de salida: resultados, predicciones.

Redes neuronales - Capas



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

Redes neuronales - Vectores



Redes neuronales - Vectores

$$\begin{bmatrix} w_1, w_2, w_3, w_4 \\ w_1, w_2, w_3, w_4 \\ w_1, w_2, w_3, w_4 \end{bmatrix} \cdot \begin{bmatrix} x_1, \\ x_2, \\ x_3, \\ x_4 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \end{bmatrix}$$

Redes neuronales - Vectores

$$\begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

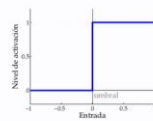
- Las operaciones que se hacen en una red neuronal no son 100% lineales, pues si se maneja así el resultado también será lineal y se pierde la esencia de esta metodología. Por ello la función de activación cumple un importante papel.

6. Funciones de activación

- Hay funciones de activación discretas y continuas.

Tipos de funciones:

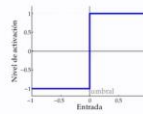
■ Función escalón/escalonada/threshold



$$y = f_{\text{th}}(z) = u(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

Se usa cuando el resultado solo toma 2 valores.

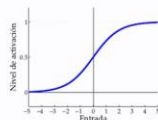
■ Función signo/signum



$$y = f_{\text{sgn}}(z) = \text{sgn}(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ -1 & \text{si } z < 0 \end{cases}$$

Esta admite valores negativos.

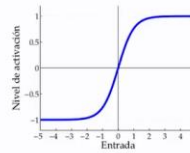
■ Función sigmoideal/sigmoid



$$y = f_{\text{sgate}}(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad \frac{d}{dx} \left(\frac{1}{1 + \exp(-x)} \right) = \frac{e^{-x}}{(e^{-x} + 1)^2}$$

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.

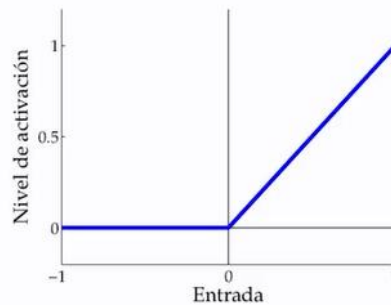
■ Función tangente hiperbólica/tanh



$$y = f_{\tanh}(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad \frac{dy}{dx} \left(\frac{1}{1 + \exp(-x)} \right) = \frac{e^{-x}}{(e^{-x} + 1)^2}$$

Similar a la sigmoide, pero admite valores negativos.

■ Función lineal rectificada/ReLU



$$y = f_{relu}(z) = \begin{cases} z & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases} \quad \frac{d}{dx} f_{relu}(z) = u(z) = \mathbf{1}_{z \geq 0} = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

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.

■ Softmax

$$z \equiv \begin{pmatrix} 1.6 \\ 0.55 \\ 0.98 \end{pmatrix} \equiv \begin{pmatrix} 0.51 \\ 0.18 \\ 0.31 \end{pmatrix}$$

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

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

7. Función de pérdida

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 pérdida.

Algunos tipos de función de pérdida:

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

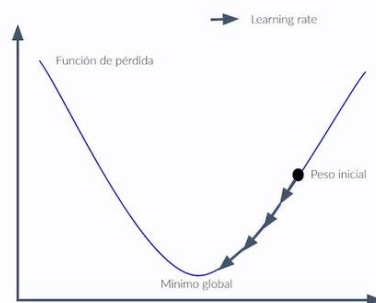
Las redes neuronales actualizan los pesos a partir de las derivadas de las funciones de activación y las funciones de pérdida. La red toma los resultados de la función de pérdida, los deriva y a partir de eso ajusta los pesos.

Ver script Funciones

8. Descenso del gradiente

- Al derivar, asumiendo que se quiere optimizar la función de pérdida, 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) tardará mucho en encontrar el mejor punto mínimo y requerirá potencia de cómputo.

■ Descenso del gradiente

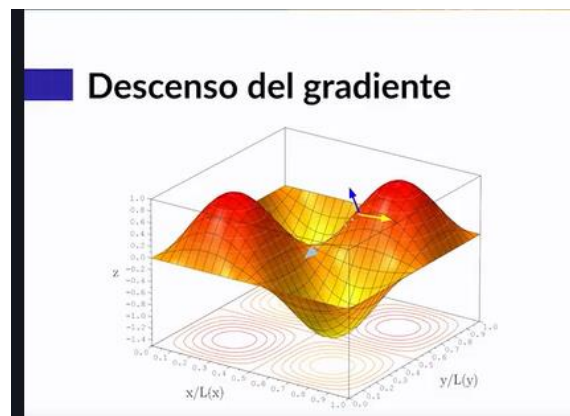


Las funciones a veces tienen más de un mínimo, como mínimos locales y el mínimo global. Lo ideal es siempre llegar al mínimo global. Para alcanzarlo, se hace uso del concepto físico **Momentum**.

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

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

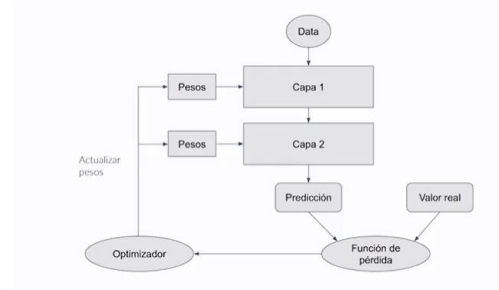
Para llevar la función de pérdida a los valores más bajos, se hace uso de las derivadas parciales, las cuales dan el gradiente, el cual debo multiplicar por (-1) .



9. Backpropagation

- Como se distribuye el error que se obtuvo en la función de pérdida en la red neuronal.

Backpropagation

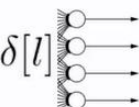


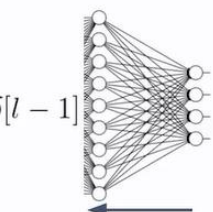
Los pesos se van actualizando de acuerdo a la función de pérdida y el descenso de gradiente.

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)

■ Backpropagation - Derivadas parciales



$$\delta[l] \equiv \text{Costo}(\text{Activación}(z)) \equiv \frac{\partial C}{\partial a[l]} * \frac{\partial a[l]}{\partial z[l]}$$


$$\delta[l-1] \equiv W[l] \delta[l] * \frac{\partial a[l-1]}{\partial z[l-1]}$$

Error

Playground TensorFlow:

<https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle®Dataset=reg-plane&learningRate=0.03®ularizationRate=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>

10. 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.

11. 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 peso 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.

■ Regularización

$$\begin{aligned} \text{L1 Regularization} \\ \text{Cost} &= \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j| \\ \text{L2 Regularization} \\ \text{Cost} &= \underbrace{\sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2}_{\text{Loss function}} + \lambda \underbrace{\sum_{j=0}^M W_j^2}_{\text{Regularization Term}} \end{aligned}$$

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

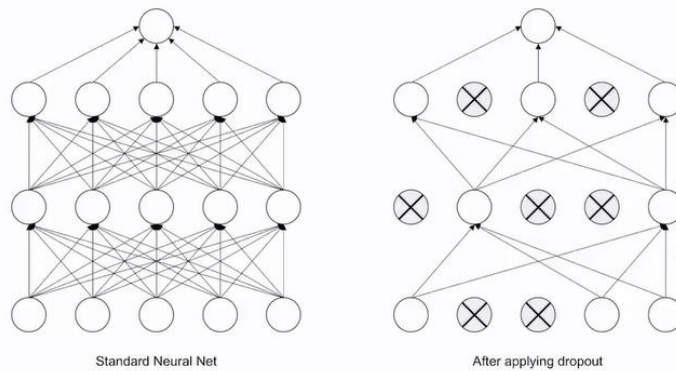
Dropout

0.5	2.1	0.6	1.5
0.6	0.2	1.1	2.3
3.0	0.8	1.5	1.2
2.2	0.7	2.2	0.9

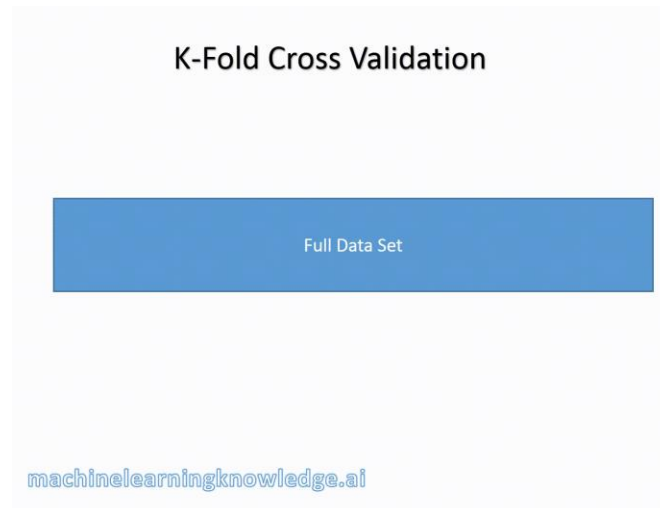
Dropout 50%

0	2.1	0.6	0
0	0.2	0	2.3
3.0	0	1.5	0
2.2	0	0	0.9

Dropout



12. Validación cruzada (K-Fold Cross Validation)



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

¿Qué funciones utilizar?

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy