Guía de estudio - Redes Neuronales (parte I)

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

Las redes neuronales, inspiradas en el complejo sistema neuronal del cerebro humano, representan una fascinante área de estudio en el campo del aprendizaje automático y la inteligencia artificial. En esta guía revisaremos los conceptos asociados a las redes neuronales desde los fundamentos biológicos hasta los conceptos claves en el entrenamiento de redes neuronales, abordando la evolución desde la neurona biológica y la simplificación de esta en el Perceptrón. Exploramos el funcionamiento del Perceptrón (neurona artificial), sus operaciones, función de activación y salida, permitiendo la clasificación lineal de datos. Veremos además de qué forma el Perceptrón logra aprender, lo que nos lleva a revisar el algoritmo del descenso del gradiente y sus variantes en el contexto de redes neuronales.

Posteriormente, veremos el funcionamiento del perceptrón multicapa, que representa la conexión de más de un Perceptrón, dando origen a una arquitectura de red neuronal. Esto lo podremos aplicar en ejemplos prácticos usando las librerías Tensorflow y Keras, que nos permitirán resolver problemas cada vez más apasionantes.

¡Prepárate para continuar profundizando en este apasionante mundo!

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Neurona biológica vs artificial

Las redes neuronales artificiales se inspiran en el complejo sistema neuronal biológico del cerebro humano para realizar tareas inteligentes. Para comprender la forma en que opera es fundamental conocer la neurona biológica, y comparar esta con su contraparte artificial.

La neurona biológica es una célula especializada que procesa y transmite información a través de señales eléctricas y químicas. Esta se compone de dendritas para recibir señales, un núcleo celular que procesa la información y un axón para transmitir los impulsos a otras neuronas. La comunicación entre neuronas se lleva a cabo mediante sinapsis, conexiones donde se intercambian neurotransmisores, como se muestra en la figura 1.

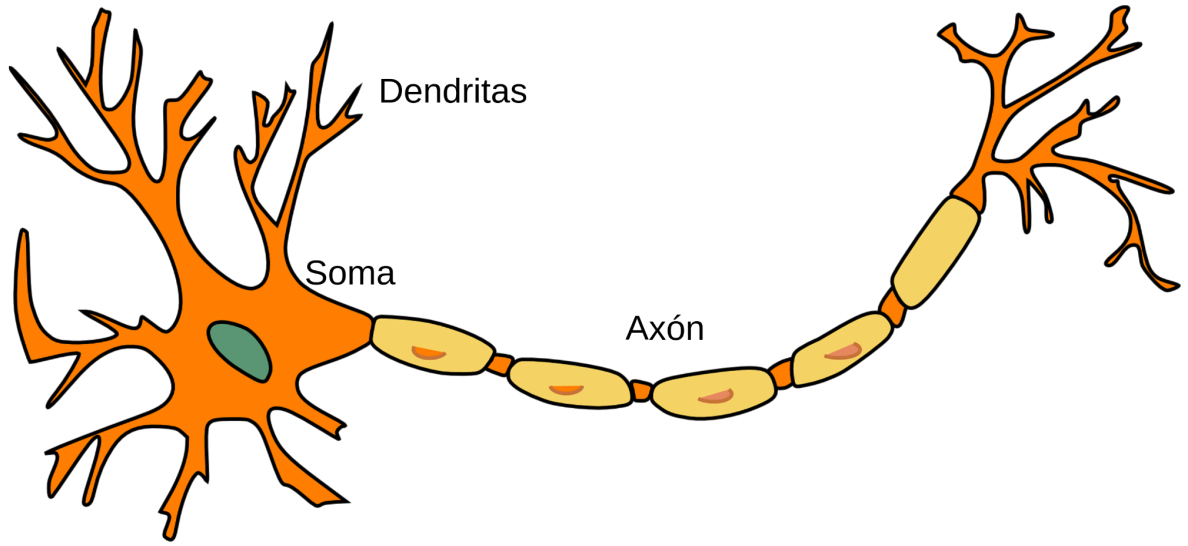


Figura 1. Neurona biológica
Fuente: PixaBay

Las redes neuronales artificiales intentan replicar este modelo biológico en forma simplificada mediante unidades básicas llamadas neuronas artificiales o nodos. Cada neurona artificial recibe múltiples entradas, a las cuales asigna pesos o ponderadores y sumando todo esto, luego aplica una función de activación al cálculo anterior, para producir una salida. Las conexiones entre estas neuronas se establecen mediante capas, donde la información fluye desde la capa de entrada hasta la capa de salida, pasando por capas ocultas (aquellas que se encuentran entre la capa de entrada y la de salida) donde se procesa y extrae información.

Aunque las neuronas biológicas y artificiales difieren en su estructura física, ambas comparten el concepto fundamental de recibir, procesar y transmitir información. La plasticidad sináptica en el cerebro humano, que permite el aprendizaje y la adaptación, tiene

un reflejo en las redes neuronales artificiales a través del ajuste de los pesos asociados a las conexiones entre las intensidades de las neuronas de entrada y las neuronas con las que se conectan.

A pesar de estas similitudes, las redes neuronales artificiales simplifican en gran medida la complejidad del sistema biológico, lo que les permite ser aplicadas en una variedad de campos entre ellos: el reconocimiento de patrones, procesamiento del lenguaje natural, visión por computador y otras. La evolución contínua en la investigación de las redes neuronales busca imitar cada vez más aquella plasticidad y eficiencia que posee el cerebro humano para impulsar avances en el área de la inteligencia artificial.

El Perceptrón

El perceptrón, desarrollado por Frank Rosenblatt en el año 1957, es una de las primeras formas de red neuronal de aprendizaje supervisado. Consiste en un un algoritmo de clasificación binario que toma un conjunto de entradas, las cuales multiplica por sus pesos o ponderadores y suma todo esto, luego este resultado es sometido a una función de activación que originalmente es una función escalón unitario. Esta función asigna un valor de salida de 1 si la suma ponderada entre pesos y entradas supera un umbral específico y 0 en otro caso.

El proceso de aprendizaje del perceptrón se basa en el ajuste iterativo de los pesos para minimizar el error entre la salida predicha y la salida real. Para ir ajustando estos pesos se usa una regla de aprendizaje que actualiza los pesos en función de la diferencia entre la salida esperada y la estimada, moviendo gradualmente el hiperplano de decisión para clasificar correctamente los patrones de entrada.

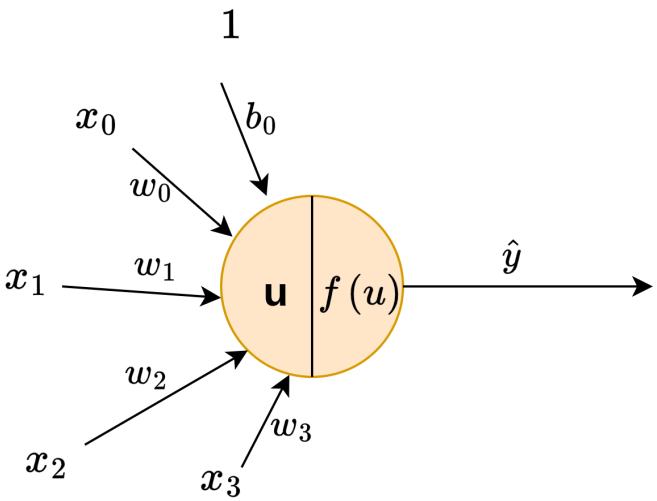


Figura 2. Perceptrón
Fuente: Desafío Latam.

- **x's:** corresponden a las características de un sujeto u observación.
- **w's:** ponderaciones asociadas a cada entrada, se conocen como pesos.
- **u:** es una operación matemática que realiza la suma ponderada de todas las entradas por los pesos.

$$u = \left(\sum_{i=0}^{N} x_i w_i \right) + b_0$$

- **b**: es un término que se utiliza para representar el sesgo (bias), igual como se realiza en una regresión lineal para mover la recta. Para modelar esto se emplea un uno como entrada.
- **y estimado**: se calcula al aplicar una función de activación sobre u, que en el caso del perceptrón de Rosenblatt es la función escalón que asigna un 1 si **u** es mayor o igual que el umbral, y 0 en caso contrario.

El proceso consiste en ir pasando las observaciones a la neurona, la que realiza los cálculos mencionados. Con la salida se emplea una regla de aprendizaje que es una variante de la ley de Hebb.

$$\begin{aligned} w_{i+1} &= w_i + \Delta w_i \\ &= w_i + \eta(y - \hat{y})x_i \end{aligned}$$

En el caso de una clasificación equivocada -es decir, el valor esperado es un 1 y obtenemos un 0 o viceversa- es estos casos la actualización de pesos será 1 vez ***eta***, o -1 vez **eta**, en que ***eta*** es una constante que representa la ponderación con la que se actualizarán los pesos, comúnmente llamado **tasa de aprendizaje**.

El perceptrón funciona bien cuando lo que deseamos separar es linealmente separable. Por ejemplo, las compuertas lógicas AND y OR.

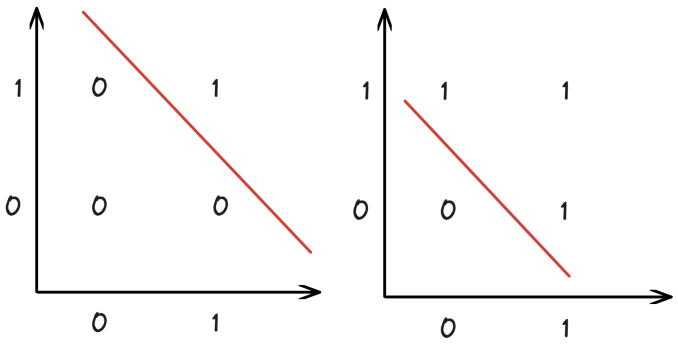


Figura 3: representación gráfica de AND y OR
Fuente: Desafío Latam

Pero en el caso de un OR, el perceptrón no es capaz de aprender esta función.

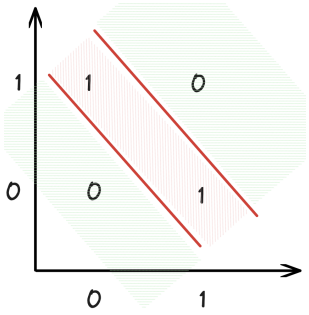


Figura 4: Representación gráfica de OR
Fuente: Desafío Latam

Una forma de resolver esto es pensar en más de una neurona, de forma que cada una de ella resuelva una parte. Esto parece simple ahora que ya sabemos todo esto, sin embargo, tuvieron que pasar varios años para llegar a solucionar este tipo de problemas.

En una investigación paralela a la de Rosenblatt, en 1960, Bernard Widrow y Tedd Hoff publican **El Adaline** (Adaptive Linear Neuron), que es similar al perceptrón en términos de arquitectura pero difiere en el proceso de aprendizaje. A diferencia del perceptrón, el Adaline usa una función de activación lineal en vez de la función escalón usada en el perceptrón. Su proceso de aprendizaje implica ajustar los pesos para minimizar una función de error cuadrático medio. Debido a las salidas con función lineal, Adaline permite una clasificación más suave y continua.

Las limitaciones del perceptrón incluyen su incapacidad para resolver problemas no lineales, y su propensión a la convergencia sólo si los datos son linealmente separables. Puede quedarse atascado en un límite de decisión lineal si los datos no son lineales. Esto llevó al perceptrón a ser considerado limitado en aplicaciones del mundo real antes del desarrollo de redes neuronales más complejas y capaces de resolver problemas no lineales, como las redes neuronales multicapa.

El perceptrón y el Adaline son precursores importantes en el desarrollo de redes neuronales, pero sus limitantes en la resolución de problemas complejos y no lineales impulsaron la investigación hacia arquitecturas más sofisticadas y flexibles.

Procedimiento de aprendizaje perceptrón

1. Inicializar los pesos de las entradas con algún valor pequeño, típicamente se hace en forma aleatoria.

2. Para cada observación, iterar una cantidad limitada de veces:
    a. Calcular la suma ponderada y aplicar la función de activación
    b. Evaluar el valor de salida esperado (real) de la observación con el valor de salida de la neurona, y por medio de la regla de aprendizaje cambiar los pesos.
3. Evaluar el perceptrón entrenado en los datos de test

Perceptrón Multicapa

Hemos visto que el perceptrón presenta algunos problemas, ya que no podemos resolver problemas que no sean linealmente separables. En este camino se propone el Perceptrón multicapa.

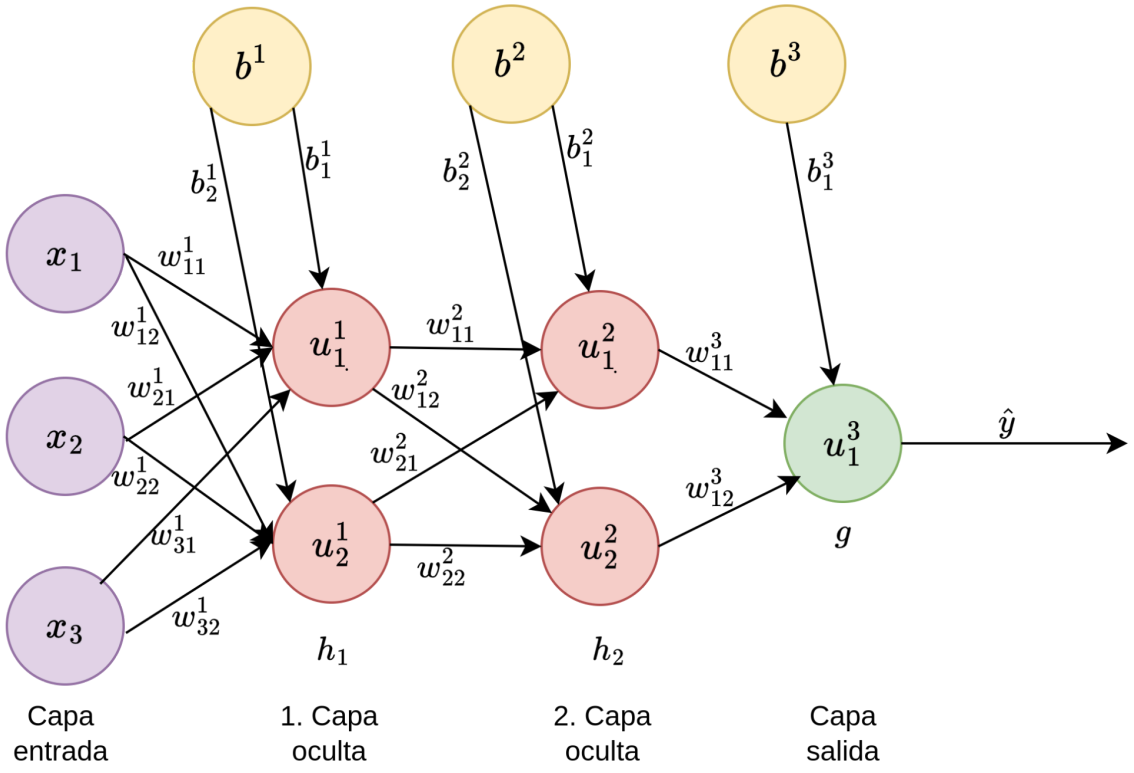


Figura 5. Perceptrón multicapa
Fuente: Desafío Latam

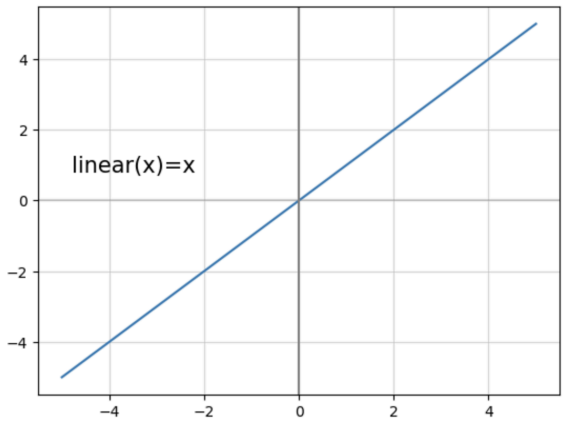
Por motivos de simpleza de los cálculos es común aplicar la misma función de activación en todas las capas ocultas y en la de salida una diferente de acuerdo a lo que se desea aprender.

Funciones de activación

Si conectamos dos o más perceptrones variando la cantidad de capas y/o perceptrones, aún tendremos el problema de que el algoritmo solo puede separar datos linealmente separables. La solución es usar una función de activación que no es lineal ni escalón, dentro de estas funciones están:

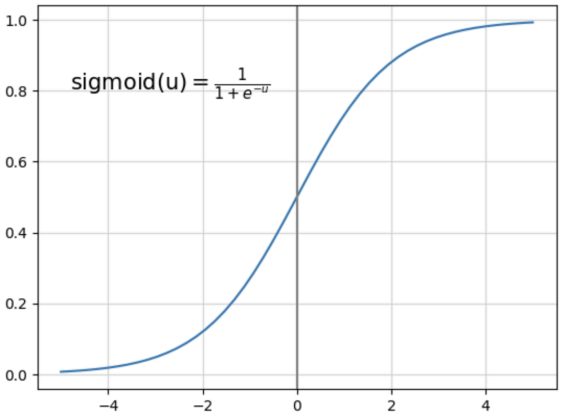
Función lineal

$$y = x$$



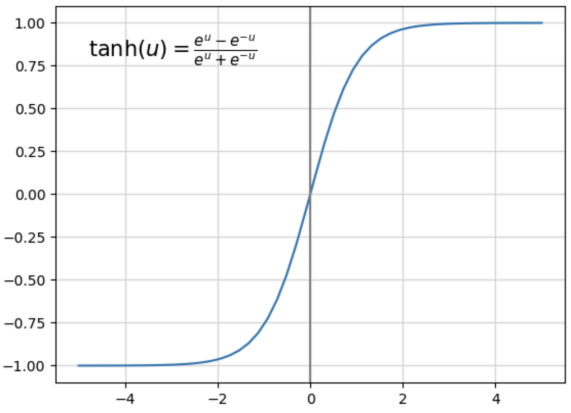
Función sigmoidal

$$y = \frac{1}{1 + e^{-u}}$$

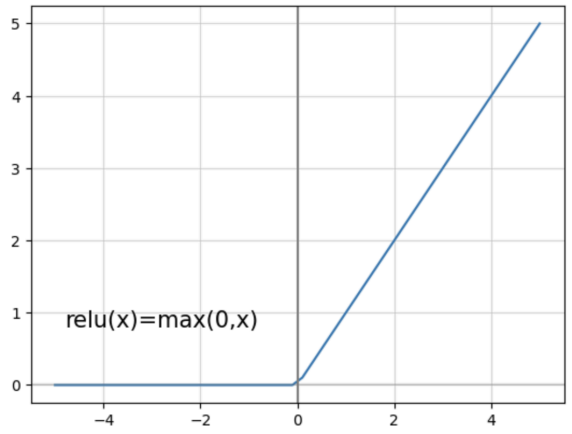


Función Tangente hiperbólica

$$y = tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$



Función Relu



Adicionalmente, incluimos la generalización de la función logística multiclase, conocida como función **softmax**

$$\sigma : \mathbb{R}^K \longrightarrow [0,1]^K$$

$$\sigma(z)_j = \frac{e^{k_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, 2, \ldots, k$$

Los mejores resultados se encuentran generalmente usando funciones Relu en las capas ocultas, y sigmoidal o softmax en la capa de salida.

La función sigmoidal se usa principalmente para problemas de clasificación binaria. Esta función convierte cualquier valor en el rango de 0 a 1, lo que puede ser interpretado como probabilidad de pertenencia a una clase. En cambio, la función Softmax se emplea para problemas de clasificación multiclase. Esta función convierte un vector de números reales en una distribución de probabilidades donde la suma de todas las salidas es igual a 1, en este caso se interpreta la salida como la probabilidad de pertenecer a las clases.

Teorema de aproximación universal

Supongamos la siguiente arquitectura de red neuronal, en la que tenemos:

- F características
- una capa oculta (h) con r neuronas
- una función de activación f, en que cada neurona posee un sesgo b. Los pesos son representados por W
- una capa de salida con pesos U, bias (sesgo) c y función de activación g.

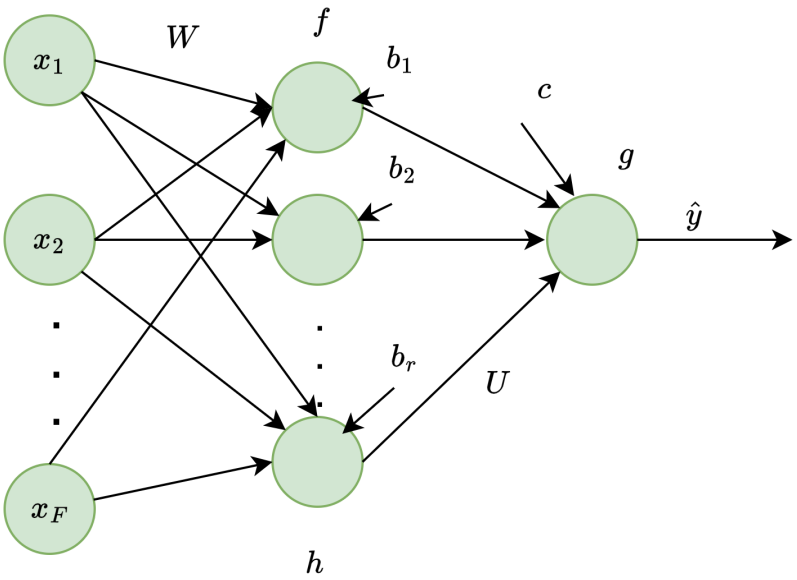


Figura 6. Teorema de aproximación universal
Fuente: Desafío Latam

Los cálculos de salida se resumen en::

$$\mathbf{h} = f(\mathbf{XW} + \mathbf{b})$$

$$\hat{\mathbf{y}} = g(\mathbf{hU} + \mathbf{c})$$

El teorema de aproximación universal plantea que **una arquitectura con una capa oculta es suficiente para aproximar cualquier función.**

Sea f una función continua $[0,1]^k \longrightarrow [0,1]$ para todo $\epsilon > 0$ existen W, b, U tal que:

$$F(x) = \text{sigmoid}(XW + b)U$$

$$|f(x) - F(x)| < \epsilon \; x \in [0,1]^k$$

El teorema de aproximación universal es un resultado fundamental, que subraya la potencia de las redes neuronales. Sin embargo, su aplicación efectiva requiere considerar aspectos como el equilibrio entre la capacidad del modelo y el riesgo de sobreajuste, así como la optimización de la arquitectura de la red para la tarea específica que se esté abordando.

Función de pérdida

La función de pérdida en una red neuronal es una métrica que evalúa qué tan bien está realizando la predicción el modelo en comparación con los valores reales de los datos de entrenamiento. Se usa para ajustar los pesos de la red con el fin de minimizar esta pérdida. La elección de la función de pérdida depende del tipo de problema que se aborde. Entre las funciones de pérdidas más comunes están:

- **Error cuadrático Medio**: se usa para problemas de regresión, en la predicción de un valor numérico. Para ello se calcula la diferencia cuadrática entre las predicciones y los valores reales, promediando todas estas.
- **Entropía cruzada binaria**: se emplea en problemas de clasificación binaria. Esta mide la discrepancia entre las distribuciones de probabilidad de las predicciones y los valores reales.
- **Entropía cruzada categórica**: utilizada para problemas de clasificación multiclase.

Nuestra búsqueda es que la función de pérdida sea cero, caso en el que tendremos los parámetros óptimos.

Uno de los problemas importantes en redes neuronales multicapa es la cantidad de parámetros a estimar. Ejemplo para una red cuya capa con menor cantidad de neuronas es de 100 y una profundidad de 10, tendremos 100.000 parámetros a estimar para la función de pérdida.

Descenso del gradiente

El método que se utiliza para ir ajustando la función pérdida en cada iteración y con ello ir acercándonos a mínimos locales -y en el mejor de los casos a un mínimo global- es por medio del algoritmo del descenso del gradiente.

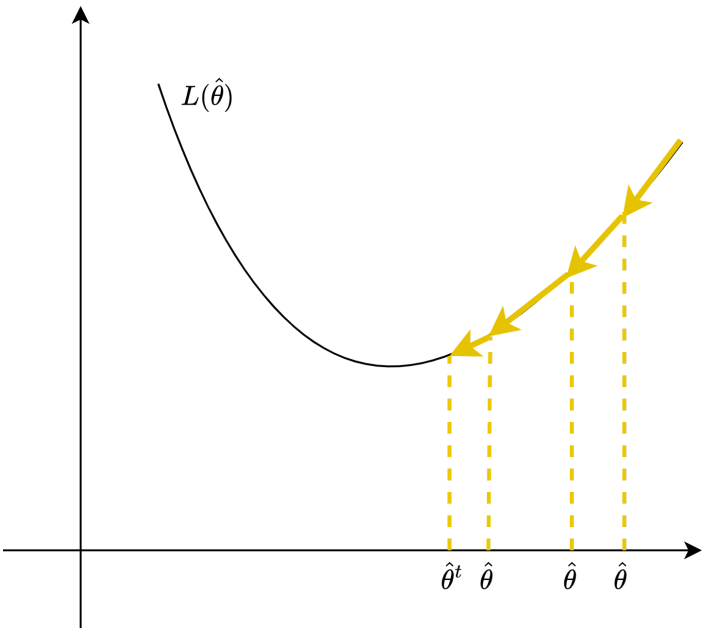


Figura 7. Función de pérdida, obtención del mínimo.
Fuente: Desafío Latam

El procedimiento para encontrar un mínimo se puede resumir como:

1. Nos ubicamos en un punto cualquiera (aleatorio), es decir, para algún valor para theta.
2. Iteramos los siguientes pasos una determinada cantidad de veces:
    a. Calculamos la función de pérdida para el theta escogido
    b. Determinamos la dirección en la que la función de pérdida crece (gradiente) y usamos exactamente la dirección contraria para dirigirnos a un valor mínimo.
    c. Nos movemos un poco en la dirección encontrada

$$\hat{\theta}_{i+1} := \hat{\theta}_i - \lambda \frac{dL}{d\theta}_{\theta_i}$$

Lambda representa cuánto nos moveremos en cada iteración, en la dirección del máximo decrecimiento.

El Descenso del gradiente realiza actualizaciones una vez han pasado todos los ejemplos por la red. A este proceso se le conoce como **Batch Gradient Descent,** el cual puede ser lento para grandes conjuntos de datos y es propenso a quedar atrapado en mínimos locales. Es posible utilizar, alternativamente, los siguientes procesos:

- **Stochastic Gradient Descent:** En este método el gradiente se calcula para cada ejemplo del conjunto de entrenamiento y se actualiza el parámetro. Esto tiene una gran ventaja en cuanto a que es rápido y se adapta bien a conjuntos de datos grandes. Sin embargo, la actualización frecuente puede causar una trayectoria de convergencia ruidosa y menos estable.

- **Mini-Batch Gradient Descent:** Este método divide el conjunto de datos en grupos más pequeños, y calcula el gradiente utilizando un grupo a la vez. Una vez que ha pasado un grupo y se calcula su gradiente, entonces se actualizan los parámetros. Combina ventajas de eficiencia de Batch Gradient Descent y la capacidad de generalización del Stochastic Gradient Descent.

Actividad guiada: Entrenando un perceptrón

Puedes revisar lo realizado en la clase, en el archivo 01 - Redes neuronales (parte I)

Algoritmos de optimización usados actualmente

Momentum

Este algoritmo es utilizado para acelerar la convergencia y mejorar la estabilidad. Incorpora información adicional al gradiente instantáneo del Descenso del Gradiente Estocástico, acumulando una fracción del gradiente anterior combinándolo con el actual, lo que permite mantener un impulso en la dirección de los cambios. Gracias a esto se suavizan las oscilaciones en la convergencia, y se acelera el proceso. Los pesos son actualizados usando la tasa aprendizaje y la fracción acumulada del gradiente anterior, lo que ayuda a trabajar mejor en superficies con pequeñas pendientes en la función de pérdida.

Momentum de Nesterov

Es una mejora del optimizador de Momentum, que resuelve la problemática que afecta al algoritmo de Momentum clásico en que la acumulación del "momentum" puede generar oscilaciones alrededor del mínimo óptimo. Este algoritmo calcula el gradiente estimado en una posición futura de acuerdo a la dirección del "momentum", y usando esta estimación se actualizan los pesos. Esta modificación es más efectiva para la convergencia en redes neuronales que la clásica.

Adagrad

Este algoritmo plantea la adaptación de la tasa de aprendizaje para cada parámetro de la red de acuerdo a la frecuencia con la que aparece en las actualizaciones de los gradientes. Esto conlleva que no tendremos que especificar o sintonizar una tasa de aprendizaje, sin embargo puede sufrir de un problema conocido como "olvido de la tasa de aprendizaje" ya que puede generar disminuciones excesivas de la tasa, con lo que se ralentiza la convergencia.

Adadelta

Adadelta es una mejora a Adagrad, en la que en vez de acumular todos los gradientes históricos para adaptar el learning rate se utiliza un promedio ponderado exponencial de los cuadrados de los gradientes anteriores. Ahora RMSprop es un Adadelta con una combinación específica de hiper parámetros.

Incidencia de learning rate en Descenso del Gradiente

El hiper parámetro de la tasa de aprendizaje es importante para alcanzar la convergencia o acercarnos a estas, y determina la velocidad con la que esto se logra. Para utilizarla debemos tener en cuenta algunas consideraciones:

- Si la tasa de aprendizaje es demasiado pequeña nos tardaremos más en alcanzar un óptimo, y podemos quedar atrapados en mínimos locales.
- Con una tasa de aprendizaje alta el algoritmo puede quedarse saltando una y otra vez sin alcanzar un óptimo, a pesar de estar eventualmente cerca de este.

En la Figura 8 podemos ver el efecto del gradiente descendente para una función de pérdida del error cuadrático medio. Una tasa de aprendizaje baja (figura de la izquierda) requiere una cantidad mayor de cálculos para alcanzar convergencia, y una tasa de aprendizaje alta puede ocasionar alejarse del óptimo (Figura de la derecha). En la Figura del centro vemos cómo se alcanza rápidamente un óptimo.

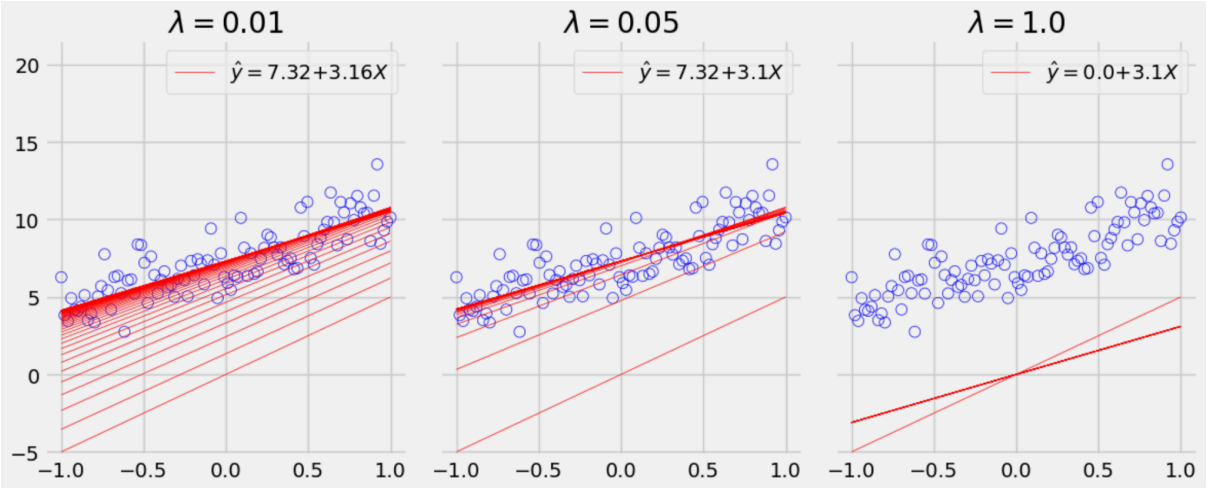


Figura 8. Descenso del gradiente con diferentes tasas de aprendizaje.
Fuente: Desafío Latam

Implementación de una red neuronal feedforward

Con lo visto hasta ahora desarrollaremos un ejemplo utilizando una arquitectura con sólo una neurona (perceptrón) y una con una capa de entrada, una capa oculta y una capa de salida. Para este desarrollo se requiere instalar el paquete TensorFlow y Keras, de los cuales hablaremos en detalle la siguiente semana. Para instalarlos, ejecuta los siguientes comandos:

```
pip install tensorflow
pip install keras
```

Actividad guiada: Implementación de redes neuronales

Puedes revisar lo realizado en la clase, en el archivo 01 - Redes Neuronales (parte I)

Preguntas de proceso

Reflexiona:

- ¿Cómo se diferencia un perceptrón de una red neuronal?
- ¿Cual es el propósito principal de la función sigmoid en redes neuronales?
- ¿Cúal es la diferencia entre SGD y Descenso de gradiente por lotes?
- ¿Cuántas capas ocultas son suficientes para aproximar cualquier función?

Referencias bibliográficas

Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville 2016.

Dive into Deep Learning, Aston Zhang (Author), Zachary C. Lipton (Author), Mu Li (Author), Alexander J. Smola (Author)

Chollet, François. Deep Learning with Python . : Manning, 2017.

Patterson, J; Gibson, A. 2017. Deep Learning: a Practitioner's Approach. Ch1: A Review of Machine Learning. Stochastic Gradient Descent. O'Reilly.