Redes Neuronales (Parte I)

Clase sincrónica

{desafío} latam_

Diseñar e implementar modelos de redes neuronales, entendiendo el funcionamiento, los parámetros e hiper parámetros de esta.

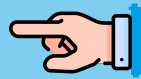
- **Unidad 1: Modelos de ensamble**
  (Parte I)

  (Parte II)

  (Parte III)
- **Unidad 2: Redes neuronales**
  (Parte I)

  (Parte II)
- **Unidad 3: Procesamiento y Redes recurrentes**
  (Parte I)

  (Parte II)



Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

Entenderás los componentes de una red neuronal sencilla, y generarás el ambiente para desarrollar modelos de redes neuronales.

{desafío} latam_

¿ Qué se nos viene a la cabeza, al escuchar Red Neuronal y por qué es importante hoy en día?

{desafío} latam_

/* Neurona - *Perceptrón* */

Neurona Biológica

Descripción

Unidad fundamental del cerebro humano Se compone básicamente de:

- **Dendritas**, que reciben señales
- **Soma** o Cuerpo Neuronal, que procesa la información
- **Axón**, que transmite información a otras neuronas

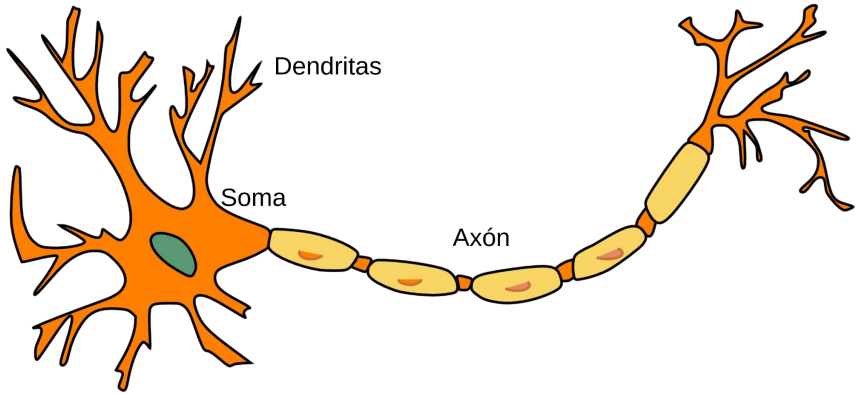


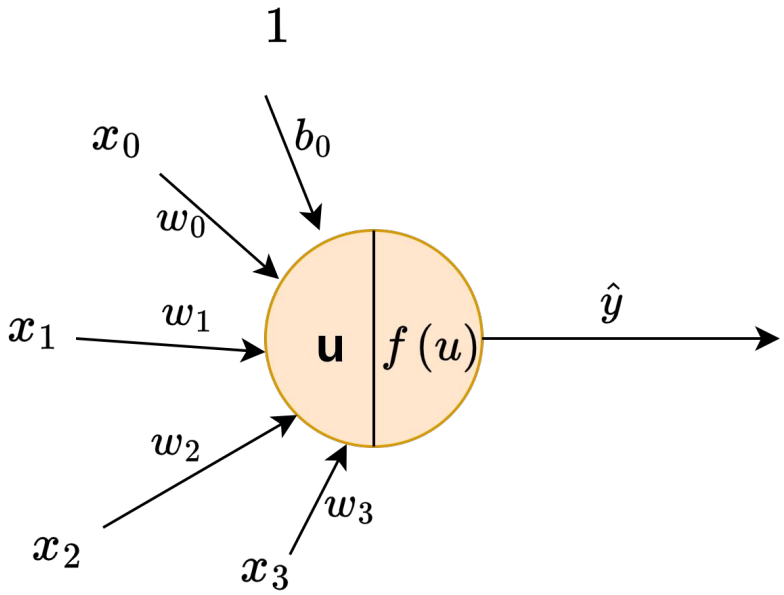
Figura 1: Neurona biológica
Fuente: PixaBay

Perceptrón

¿Qué es?

Corresponde a un Modelo básico de Neurona Artificial. Se compone básicamente de:

- Entradas (X's)
- Pesos (W's)
- Función de activación (f(u))

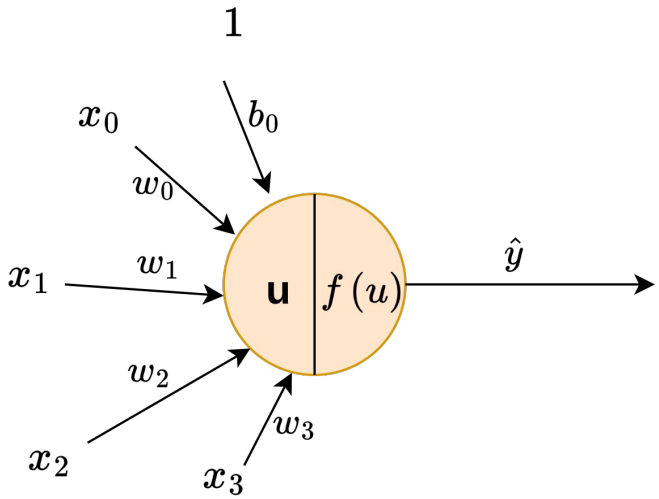


Perceptrón

Entradas

Corresponden a las características asociadas a cada observación

	Es hombre	Edad	Realiza ejercicio	Consumo de proteínas	Promedio notas
0	Sí	25	Sí	Alto	8.5
1	No	30	No	Medio	7.2
2	Sí	22	Sí	Bajo	8.9
3	Sí	28	Sí	Alto	7.8
4	No	35	No	Bajo	6.5
5	Sí	20	Sí	Medio	8.2
6	No	26	No	Medio	7.0



En este caso las entradas son:

- Es hombre
- Edad
- Realiza ejercicio
- Consumo de proteínas
- Promedio de notas

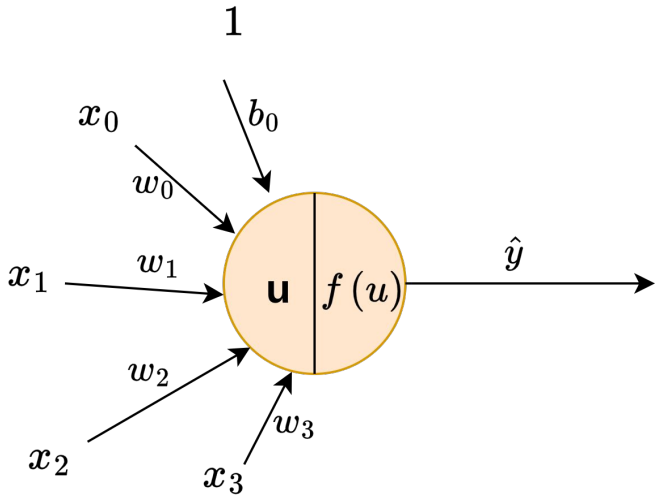
{desafío} latam_

Perceptrón

Pesos y bias

- Los pesos (W's) son ponderadores de las entradas y representan los parámetros que gobiernan la función de pérdida asociada al perceptrón.

- Los valores de los pesos inicialmente son asignados en forma aleatoria.



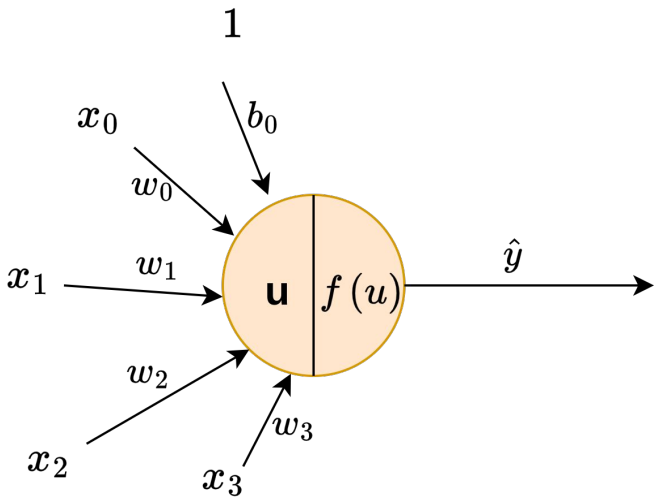
Perceptrón

El valor u

- **u** corresponde a la suma ponderada de entradas y pesos, más el bias.

- **b (Bias)** es un parámetro libre que permite ajustar el sesgo. En el caso de la regresión corresponde al intercepto

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



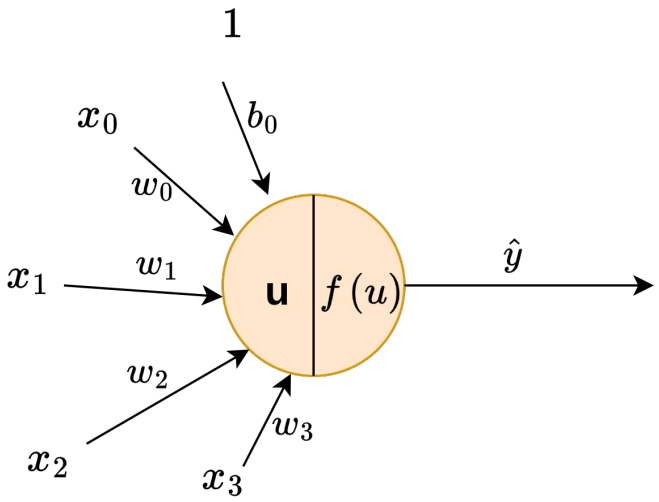
Perceptrón

Función de activación f(u)

f(u) se llama **función de activación,** y se aplica a la suma ponderada u. Inicialmente se usó la función escalón.

$$\mathrm{f(u)} = \begin{cases} 0 & \text{si } u < 0 \\ 1 & \text{si } u \geq 0 \end{cases}$$

Luego la salida del perceptrón (valor de y estimado) corresponde al f(u)



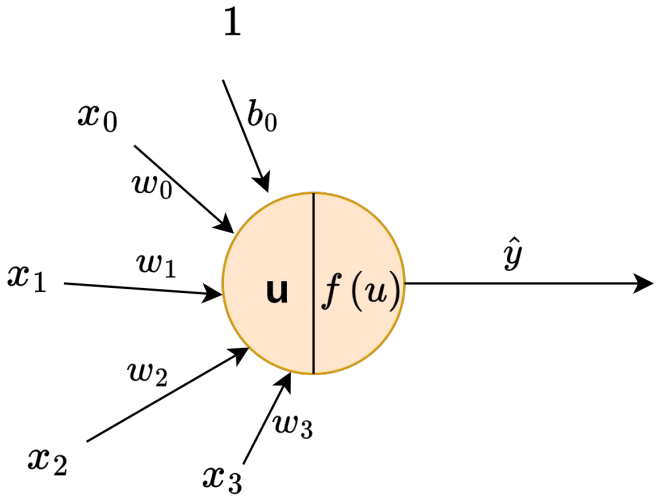
Perceptrón

Función de pérdida

La **función de pérdida** representa el nivel de error que se está cometiendo con los parámetros actuales (w's)

Cada vez que se transfieran valores por la neurona, se evalúa la discrepancia de la predicción con el verdadero valor. Con esta diferencia se ajustan los pesos.

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



η : representa la tasa con la cual se ajustará cada vez los parámetros (pesos).

Ejemplo
Entrenando un perceptrón

Ejercicio

Entrenando al perceptrón

Consideremos el caso de un atleta que puede observar o no dos conductas:

- x_0: entrena todos los días
- x_1: Consume adecuadamente proteínas

En base a esto, su resultado puede ser ganar (win) de acuerdo con la siguiente tabla:

x_0	x_1	win
0	0	0
0	1	0
1	0	0
1	1	1

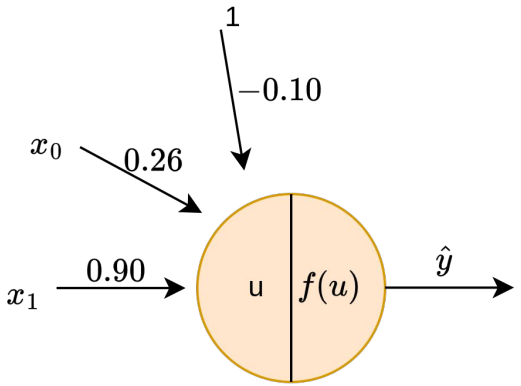
Ejercicio

Entrenando al perceptrón

Comenzaremos agregando la columna x2, correspondiente al bias, e inicializaremos los pesos aleatoriamente.

x_0	x_1	x_2	win
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

$$w_0^0 = 0.26$$
$$w_1^0 = 0.90$$
$$b^0 = -0.10$$



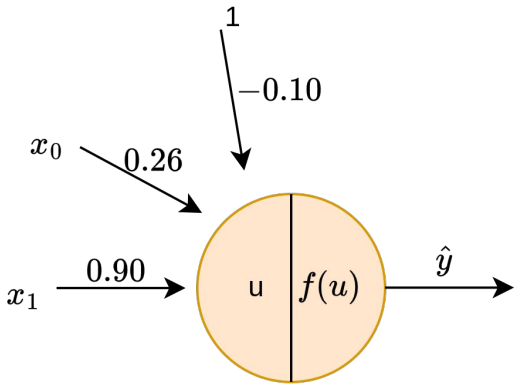
Ejercicio

Entrenando al perceptrón

Para la primera fila calcularemos el valor de u, con la fórmula vista.

x_0	x_1	x_2	win
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

$$u = w_0^0 \cdot x_0 + w_1^0 \cdot x_1 + b^0 \cdot 1$$
$$u = 0.26 \cdot 0 + 0.90 \cdot 0 + -0.10 \cdot 1$$
$$u = 0 + 0 - 0.10$$
$$u = -0.10$$



{desafío} latam_

Ejercicio

Entrenando al perceptrón

Para calcular f(u), utilizamos la función escalón. En nuestro caso, ya que el valor de u es negativo, f(u) = 0.

Podemos observar que el valor de f(u) es igual al valor esperado en la tabla (win = 0). Por lo tanto, utilizamos los mismos pesos para calcular en la siguiente fila

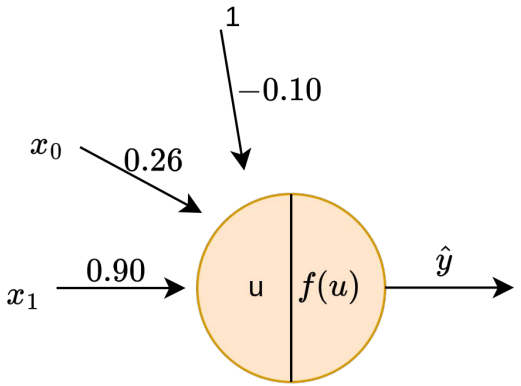
Ejercicio

Entrenando al perceptrón

Para la segunda fila tenemos:

x_0	x_1	x_2	win
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

$$u = w_0^0 \cdot x_0 + w_1^0 \cdot x_1 + b^0 \cdot 1$$
$$u = 0.26 \cdot 0 + 0.90 \cdot 1 + -0.10 \cdot 1$$
$$u = 0 + 0.90 - 0.10$$
$$u = 0.80$$



Ejercicio

Entrenando al perceptrón

Tenemos así que f(u = 0.8) = 1, que es distinto del valor win = 0 de la tabla. Por lo tanto, el error corresponde a 0 - 1 = -1.

Debemos recalcular los pesos, con la fórmula

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

Donde η corresponde a la tasa de ajuste del error, que en nuestro caso será 0.5

Ejercicio

Entrenando al perceptrón

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

$$w_0^1 = w_0^0 + 0.5 \cdot (y - \hat{y}) \cdot x_0$$
$$w_0^1 = 0.26 + 0.5 \cdot (0 - 0) \cdot 0$$
$$w_0^1 = 0.26$$

$$w_1^1 = w_1^0 + 0.5 \cdot (y - \hat{y}) \cdot x_1$$
$$w_1^1 = 0.90 + 0.5 \cdot (0 - 1) \cdot 1$$
$$w_1^1 = 0.9 - 0.5$$
$$w_1^1 = 0.4$$

$$b^1 = b^0 + 0.5 \cdot (y - \hat{y}) \cdot x_2$$
$$b^1 = -0.1 + 0.4 \cdot (0 - 1) \cdot 1$$
$$b^1 = -0.1 - 0.4$$
$$b^1 = -0.5$$

Ejercicio

Entrenando al perceptrón

Verificamos los pesos recalculados

x_0	x_1	x_2	win
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

$$w_0^1 = 0.26$$
$$w_1^1 = 0.4$$
$$b^1 = -0.5$$

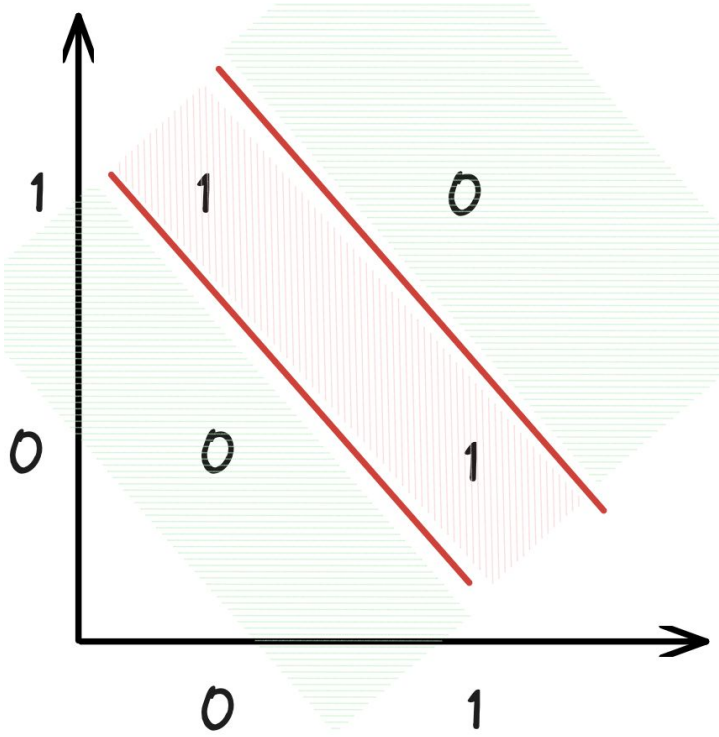
$$u_1 = 0.26 \cdot 0 + 0.4 \cdot 0 + -0.5 \cdot 1 = -0.5 \longrightarrow f(u) = 0$$
$$u_2 = 0.26 \cdot 0 + 0.4 \cdot 1 + -0.5 \cdot 1 = -0.1 \longrightarrow f(u) = 0$$
$$u_3 = 0.26 \cdot 1 + 0.4 \cdot 0 + -0.5 \cdot 1 = -0.24 \longrightarrow f(u) = 0$$
$$u_4 = 0.26 \cdot 1 + 0.4 \cdot 1 + -0.5 \cdot 1 = 0.16 \longrightarrow f(u) = 1$$

Perceptrón

Limitantes

No puede encontrar una frontera de separación cuando los datos no son linealmente separables.

El perceptrón no puede aprender la compuerta lógica OR, por ejemplo.

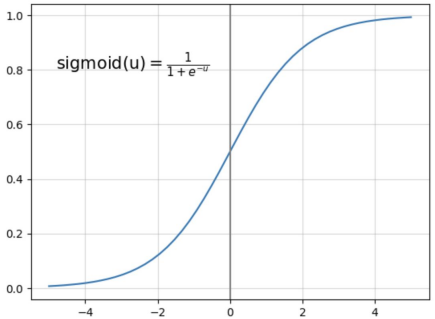


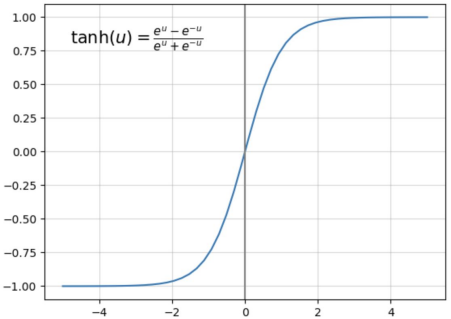
/* Función de activación */

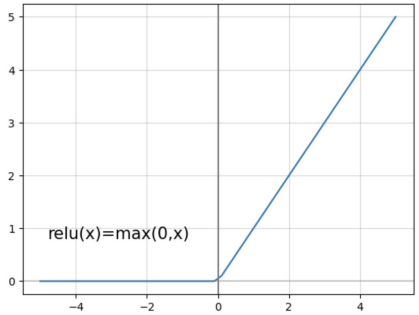
Funciones de activación

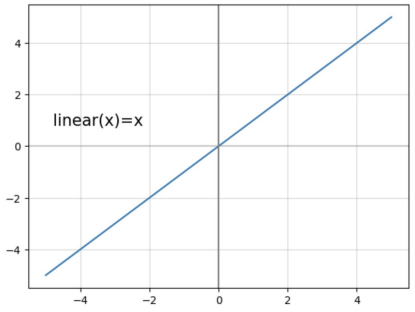
Superando la limitación del Perceptrón

Para modelar situaciones no lineales, podemos utilizar otras funciones de activación









{desafío} latam_

Funciones de activación

Función softmax

Otra alternativa es la función **softmax**, que corresponde a una generalización de la función logística.

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

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

/* Descenso del Gradiente */

Descenso del gradiente

Descripción

- Algoritmo de optimización que se utiliza para minimizar una función objetivo (función de pérdida)
- El procedimiento contempla actualizar en forma iterativa los parámetros de un modelo (pesos) siguiendo la dirección de máximo decrecimiento.
- Para un problema de regresión queremos encontrar

$$\hat{y} = \theta_0 + \theta_1 x$$

- Una función de pérdida para encontrar los mejores parámetros son los mínimos cuadrados ordinarios (OLS)

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \underset{\theta \in \theta}{argmin} L(\theta)$$

Descenso del gradiente

Descripción

Para encontrar el mínimo de la superficie que define la función de pérdida, nos moveremos por esta función en la dirección de máximo decrecimiento. Para esto calculamos el gradiente de la función.

1. Escogemos en forma aleatoria valores para los parámetros
2. Iteramos los siguientes pasos n veces, o hasta que haya convergencia:
   2.1. Calculamos el gradiente de la función de pérdida (derivadas parciales con respecto a cada parámetro) -> indica la dirección de máximo crecimiento (tomamos la dirección contraria)
   2.2. Actualizamos los parámetros.

Descenso del gradiente

Actualización de los parámetros

Para dirigirnos en la dirección contraria del gradiente. Máximo decrecimiento

Tasa de aprendizaje

$$\theta_{t+1} = \theta_t - \alpha \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t) \right)$$

Primer candidato

Gradiente de la función de pérdida con respecto a los parámetros, evaluada en una observación (i)

Descenso del gradiente

En acción

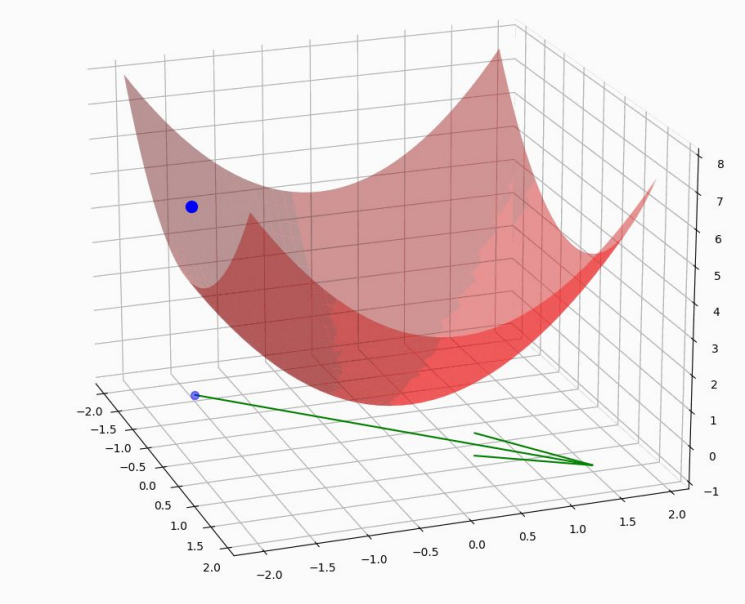


Figura 2: Descenso del Gradiente
Fuente: Desafío Latam

/* Variantes
Descenso del Gradiente */

Variantes Descenso del gradiente

Alternativas

Batch Gradient Descent: la actualización de los nuevos valores para los parámetros se realiza luego de haber visitado todos los ejemplos. Es el caso que describimos anteriormente.

Mini Batch Gradient Descent: la actualización de pesos (parámetro) se realiza usando sólo un subconjunto de ejemplos.

Stochastic Gradient Descent: los parámetros son actualizados utilizando solamente una observación, elegida aleatoriamente.

Mini Batch Gradient Descent

Descripción

- Se usan pequeños lotes de ejemplos cada vez para calcular el gradiente.
- Menos estable que Batch Gradient Descent, pero tiene más posibilidad de escapar de mínimos locales
- Presenta rápida convergencia

$$\theta_{t+1} = \theta_t - \alpha \left(\frac{1}{b} \sum_{i=j}^{j+b} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t) \right)$$

Descenso del gradiente estocástico

Descripción

- Posee una convergencia rápida, debido a que los pesos se actualizan con mayor frecuencia.
- Presenta mayor inestabilidad ya que los gradientes son calculados con sólo una observación.
- Baja propensión a quedar atrapado en mínimos locales.

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t)$$

Manos a la obra - Entrenando un perceptrón

Manos a la obra

Implementación de una red neuronal

A continuación, veremos cómo implementar lo aprendido con Python. Para ello utilizaremos dos herramientas fundamentales, **Tensorflow** y **Keras.**

Utiliza el archivo 01 - Clases - Redes neuronales (parte I) y sigue paso a paso lo que vamos a realizar.

/* Variantes Actuales Descenso del Gradiente */

Momentum Based Gradient Descent

Descripción

- **Idea:** a medida que una pelota desciende por una pendiente adquiere aceleración.
- En este método no sólo consideramos el gradiente sino también el **momentum** anterior, lo que ayuda a moverse más rápido en la búsqueda del mínimo.
- Asignamos así un nuevo hiper parámetro asociado al momentum

$$\theta_{t+1} = \theta_t - \nu_t \qquad \nu_t = \alpha \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t) \right)$$

$$\nu_{t+1} = \gamma \nu_t + \alpha \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t) \right)$$

Adagrad

Descripción

- Es un algoritmo adaptativo que va ajustando la tasa de aprendizaje, de manera que los parámetros que hayan recibido más actualizaciones reciban cada vez menos.
- A medida que avanzan las iteraciones, la tasa de aprendizaje va disminuyendo, lo que provoca que la red deje de aprender

$$\theta_{t+1} = \theta_t - \alpha' g_t$$

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t)$$

$$\alpha' = \frac{\alpha}{\sqrt{\epsilon + G_t}}$$

$$G_t = G_{t-1} + g_t^2$$

/* Perceptrón Multicapa (MLP) */

Perceptrón Multicapa

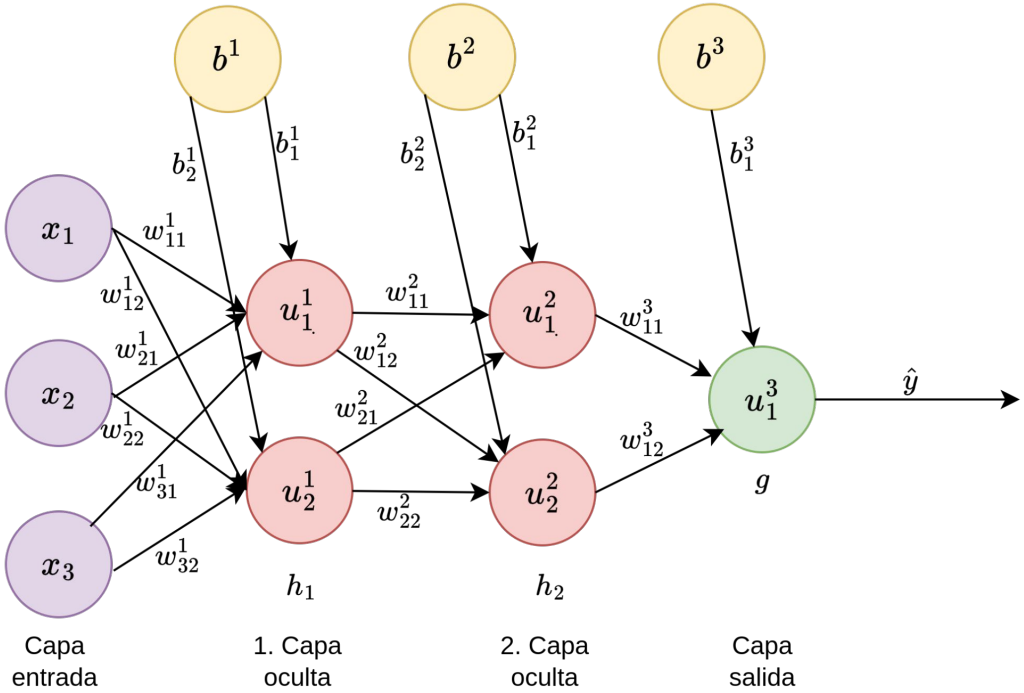
¿Qué es?

Se llama así a una red neuronal de múltiples capas (mínimo 3)

- **Capa de entrada**: Corresponde a los valores de las características del conjunto de entrenamiento. Para una conjunto de entrenamiento con cinco características tendremos cinco entradas.
- **Capas ocultas**: Estas se encuentran entre la capa de entrada y la capa de salida y son las encargadas de procesar la información por medio de transformaciones no lineales (funciones de activación)
- **Capa de salida:** entrega la salida final de la red. La función de activación y la cantidad de neuronas en esta capa depende del tipo de problema que se desea resolver.

Perceptrón Multicapa

Esquema



{desafío} latam_

Red Neuronal

Características

El proceso de aprendizaje en una red neuronal se realiza por medio del algoritmo **Backpropagation** que veremos durante la siguiente sesión. Dependiendo de las características del problema conviene considerar algunos aspectos al modelar

	Regresión	Clasificación binaria	Clasificación multiclase
Función de activación	lineal	binary crossentropy	softmax
Función de pérdida	mean squared error	sigmoid	categorical crossentropy

Manos a la obra - Implementación de una red neuronal

Manos a la obra

Implementación de una red neuronal

A continuación, veremos cómo implementar lo aprendido con Python. Para ello utilizaremos el mismo archivo anterior, en la siguiente sección.

¡Sigue atentamente los pasos propuestos!

Desafío - Predicción renuncia de clientes usando una red neuronal

{desafío} latam_

Desafío

"Predicción de renuncia de clientes usando una red neuronal"

- Descarga el archivo "Desafío".
- Tiempo de desarrollo asincrónico: desde 3 horas.
- Tipo de desafío: individual.

¡AHORA TE TOCA A TI! 💪

{desafío} latam_

Ideas fuerza

La ***Perceptrón*** *neurona artificial que aprende con entradas y pesos produciendo salidas basadas en umbral (* ***función de activación)***

Las ***función de activación*** *regulan la salida de neuronas artificial permitiendo no linealidad y aprendizaje, entre ellas están:* ***sigmoid, tanh, relu, linear.***

Red neuronal *Estructuras interconectadas de neuronas que se entrenan para aproximar funciones de diversa complejidad.*

“La capacidad adaptativa de las redes neuronales devela un universo de posibilidades en la comprensión y solución de problemas complejos”.

Ya tenemos los datos y máquinas “poderosas”

{desafío} latam_

Próxima sesión...

Se profundizará en redes neuronales multicapa explicando en detalle el algoritmo backpropagation y mecanismos de regularización en redes neuronales

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales