

ACÀMICA

¡Bienvenidos/as a Data Science!



Agenda

¿Cómo anduvieron?

Repaso: Redes Neuronales, Preliminares

Explicación: Perceptrón

- Función logística sigmoide

- Perceptrón con variable

- Entropía cruzada

- Propagation

Break

Recursos : Pair programming

Hands-On

Cierre



¿Cómo anduvieron?



Repaso: Descenso por Gradiente

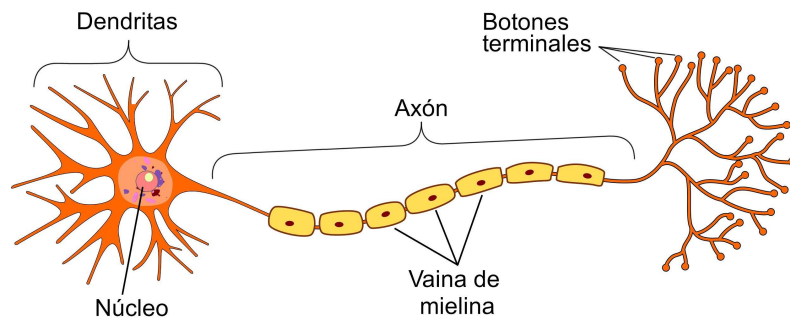


Redes neuronales

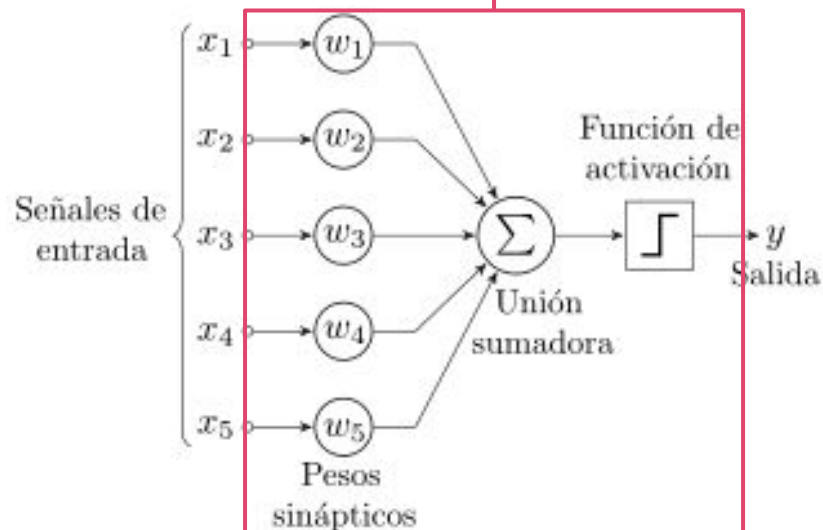
Esperamos que aprendan los siguientes conceptos:

- Perceptrón, Funciones de Activación
- Forward Propagation
- Backpropagation
- Descenso por gradiente (Gradient Descent)
- Redes Neuronales Profundas
- Regularización
- Redes Neuronales Convolucionales (CNN, si hay tiempo)
- Entornos de desarrollo: Keras, Tensor Flow
- Y muchos que probablemente nos estemos olvidando.

Redes neuronales



Neurona básica:
Perceptrón Simple



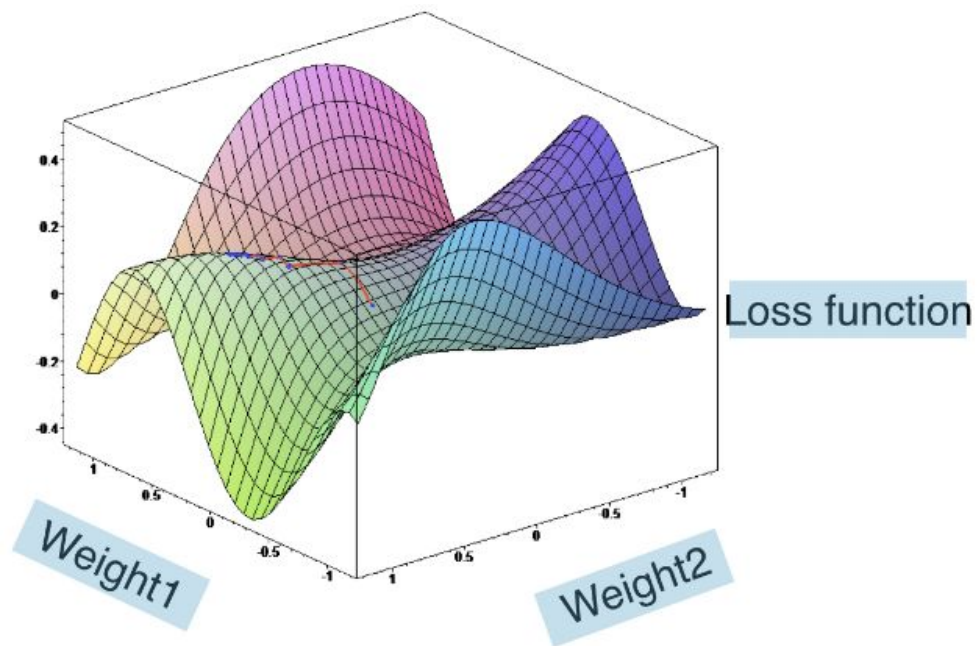
Funciones de costo

Obtenemos un número que representa la performance del modelo (errores en las predicciones de muchos data points)

Predicción	Objetivo	Error	Error**2
10	20	-10	100
8	3	5	25
6	1	5	25

- Error cuadrado total: 150
- Mean squared error: 50

Funciones de costo

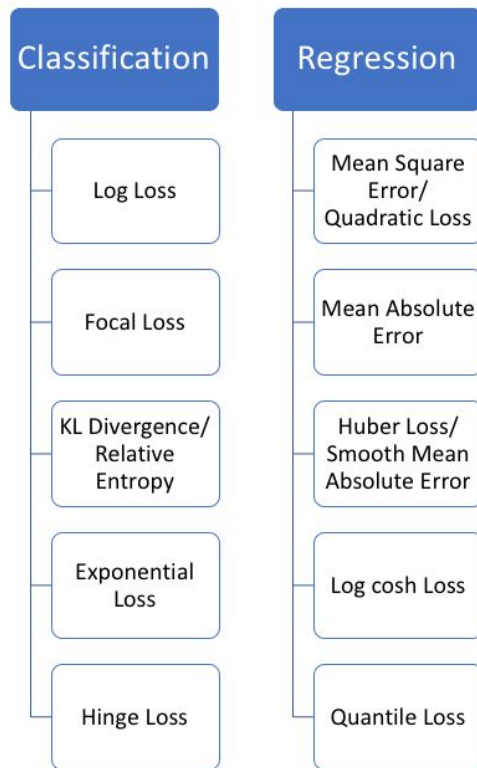


Funciones de costo

Mientras menor sea el MSE, mejor funciona el modelo.

El objetivo es **encontrar los pesos** que proporcionan el menor valor para la función de costo. Esto lo logramos con el algoritmo de **Descenso por Gradiente (Gradient descent)**.

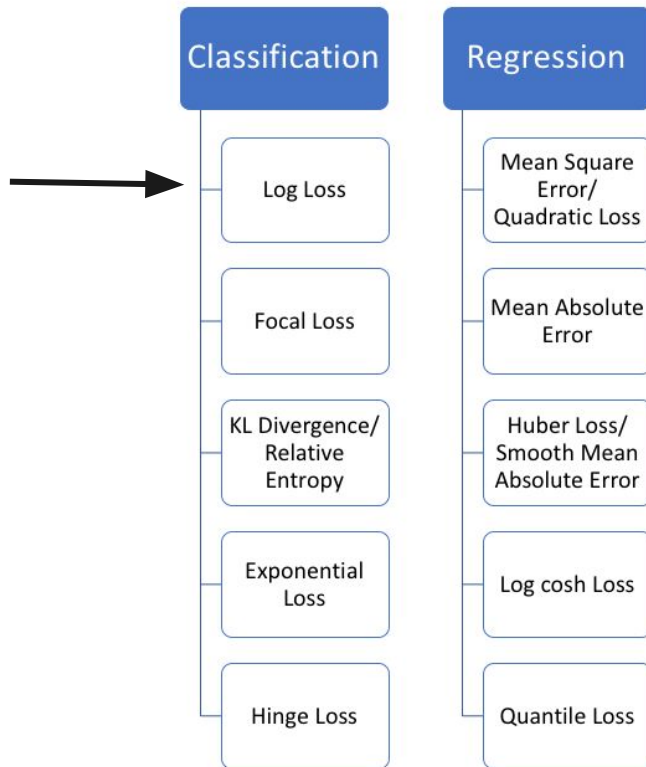
Funciones de costo - Tipos



Funciones de costo - Tipos

Entropía Cruzada (Cross Entropy)

Lo vamos a ver en un rato.



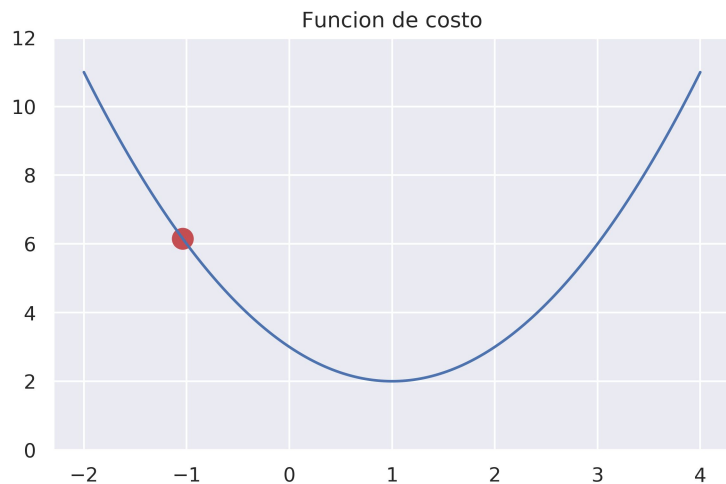
Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:



Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

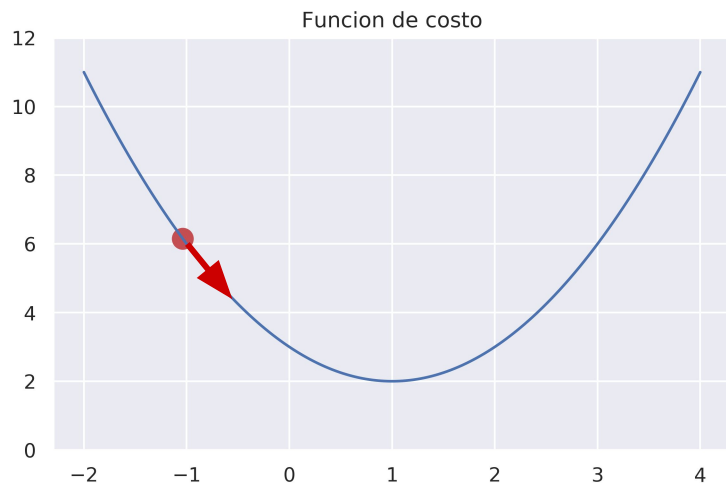


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

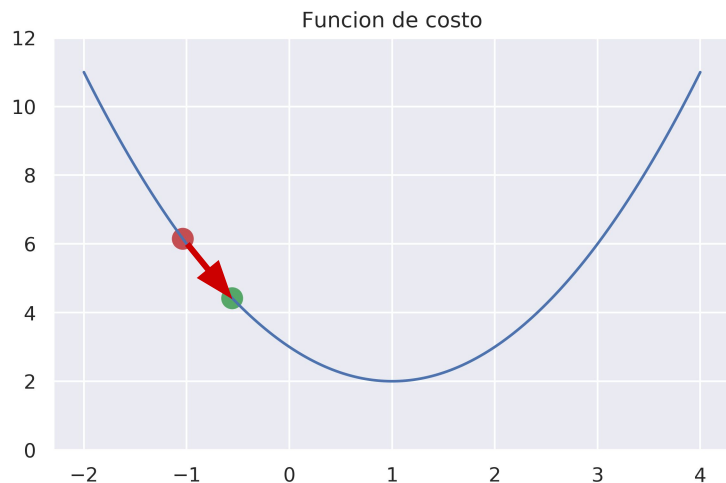


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto. Técnicamente, derivamos o calculamos el gradiente.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

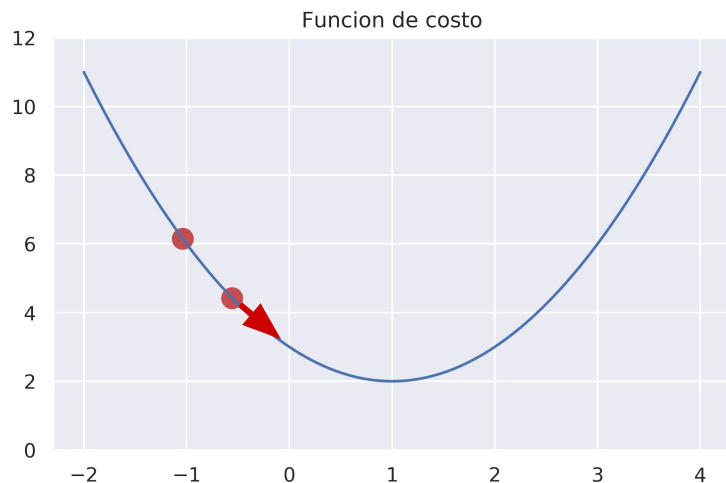


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto. Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

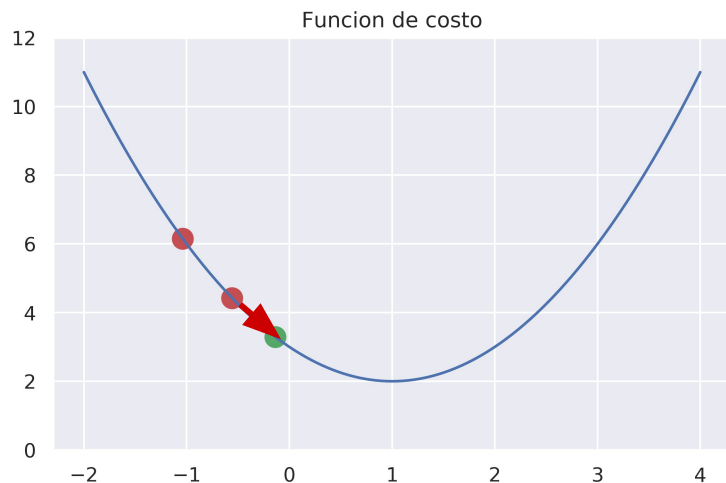


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto.
Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

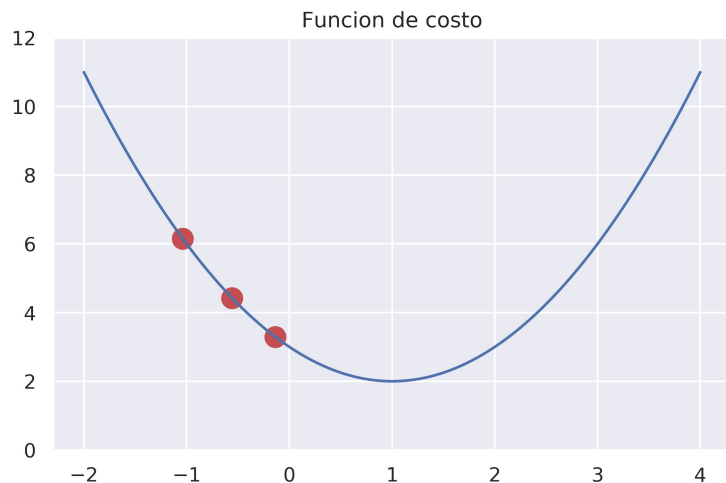


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto.
Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

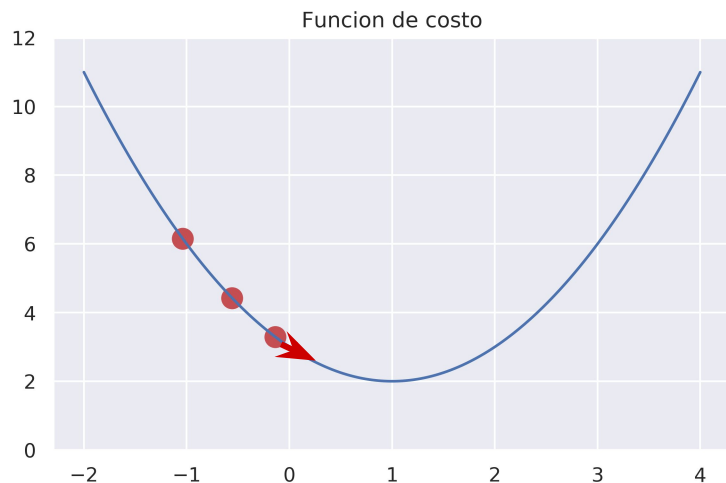


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto.
Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

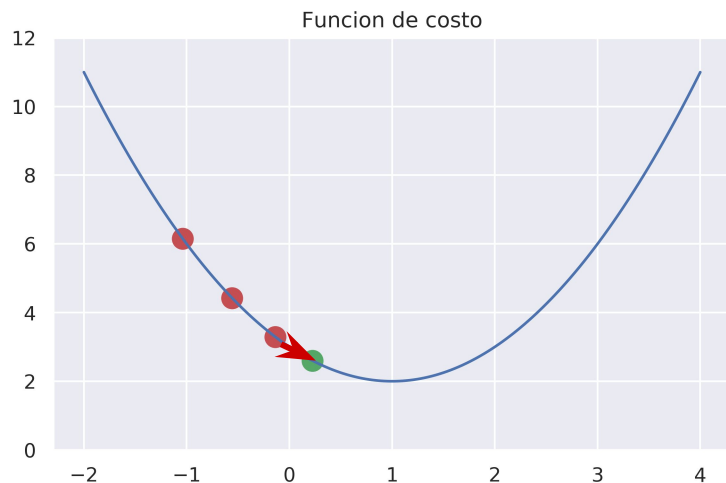


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto.
Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

Descenso por gradiente

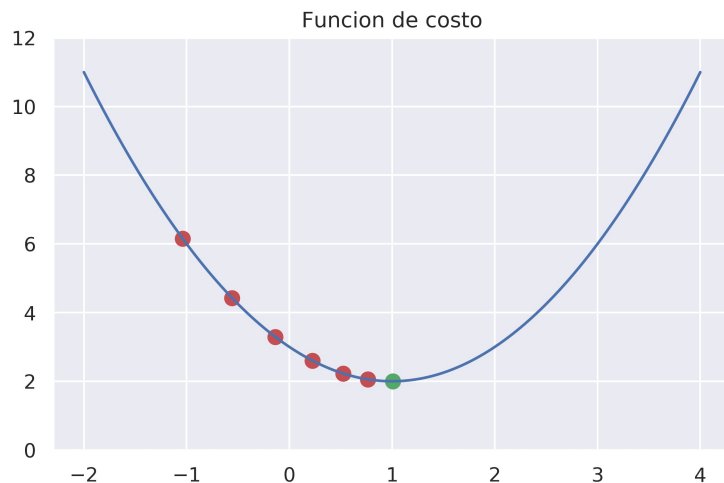
Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:



Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto.
Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

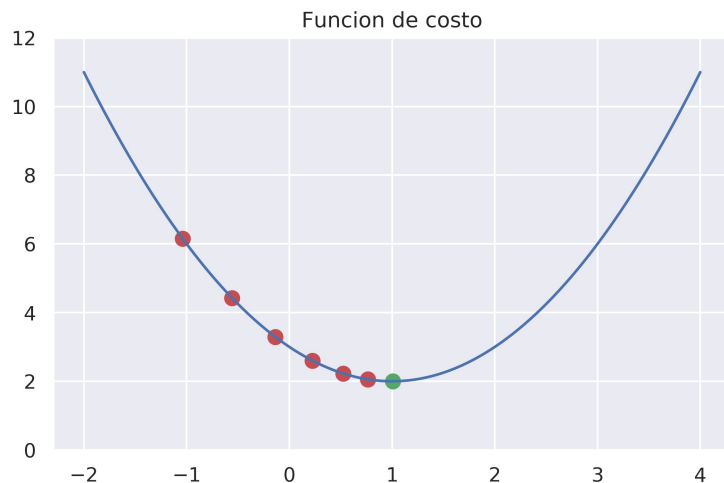
Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:



Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto.
Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:



Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
 - a. Nos fijamos la dirección de decrecimiento en ese punto.
Técnicamente, derivamos o calculamos el gradiente.
 - b. Actualizamos los valores de los parámetros.

Perceptrón

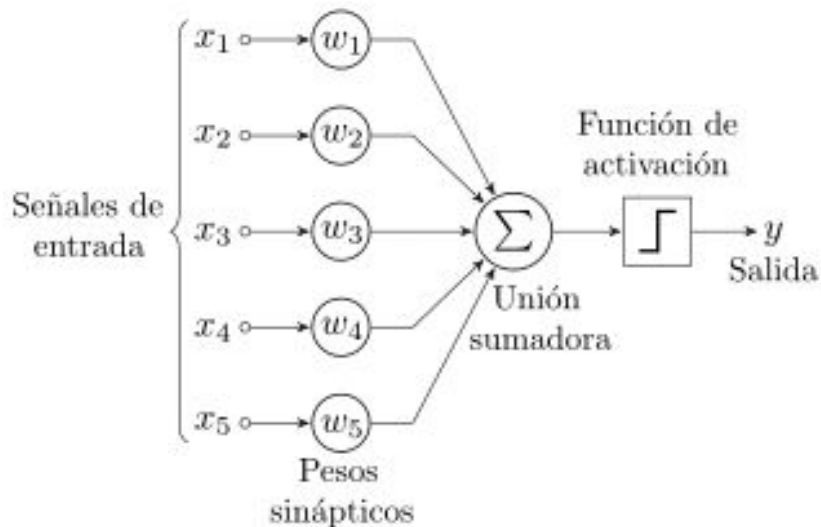


Objetivos de este encuentro

- Presentar el Perceptrón con un grado de detalle que permita entender lo que viene después. Para eso, vamos a trabajar sobre los conceptos más difíciles que suelen aparecer.
 - La idea es que entiendan “la cocina” del asunto – lo que hay detrás.
 - No se preocupen si se pierden en los detalles matemáticos.
- Presentar Keras como entorno de desarrollo para Redes Neuronales

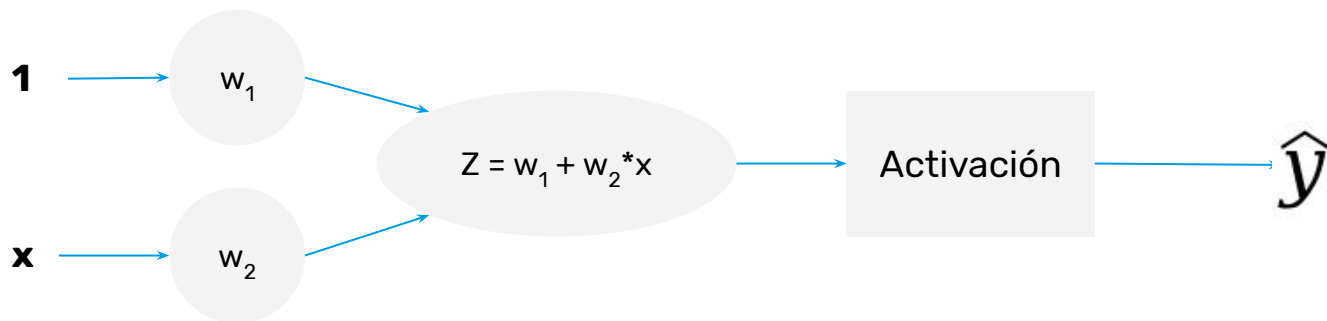
Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



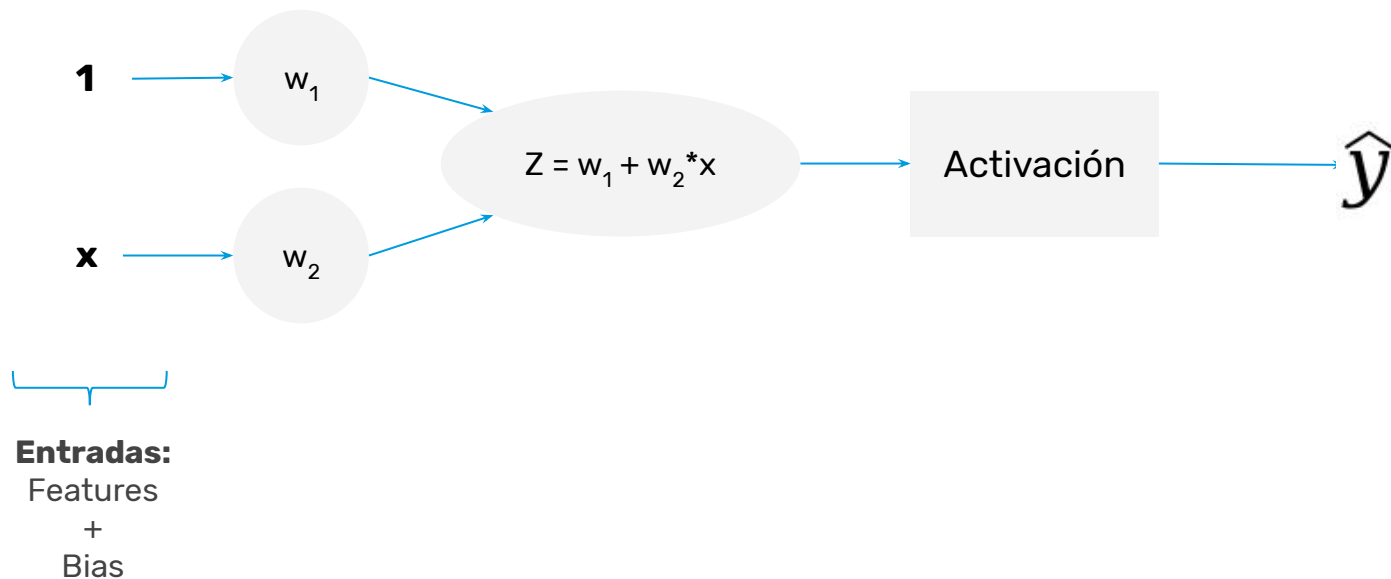
Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



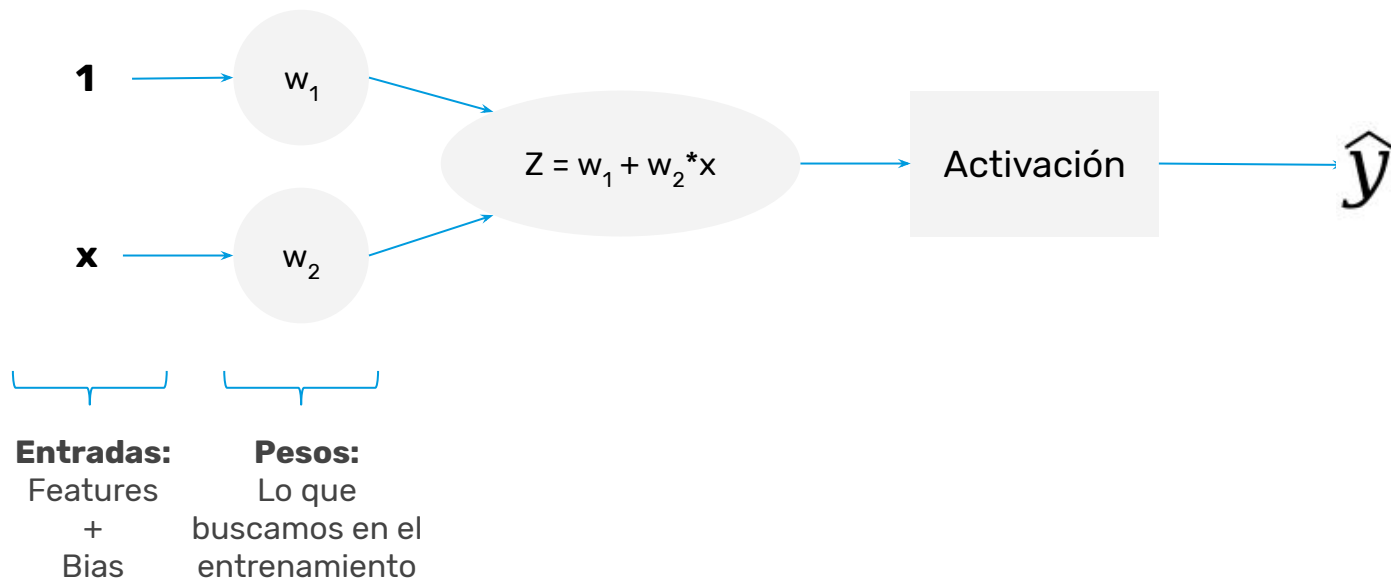
Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



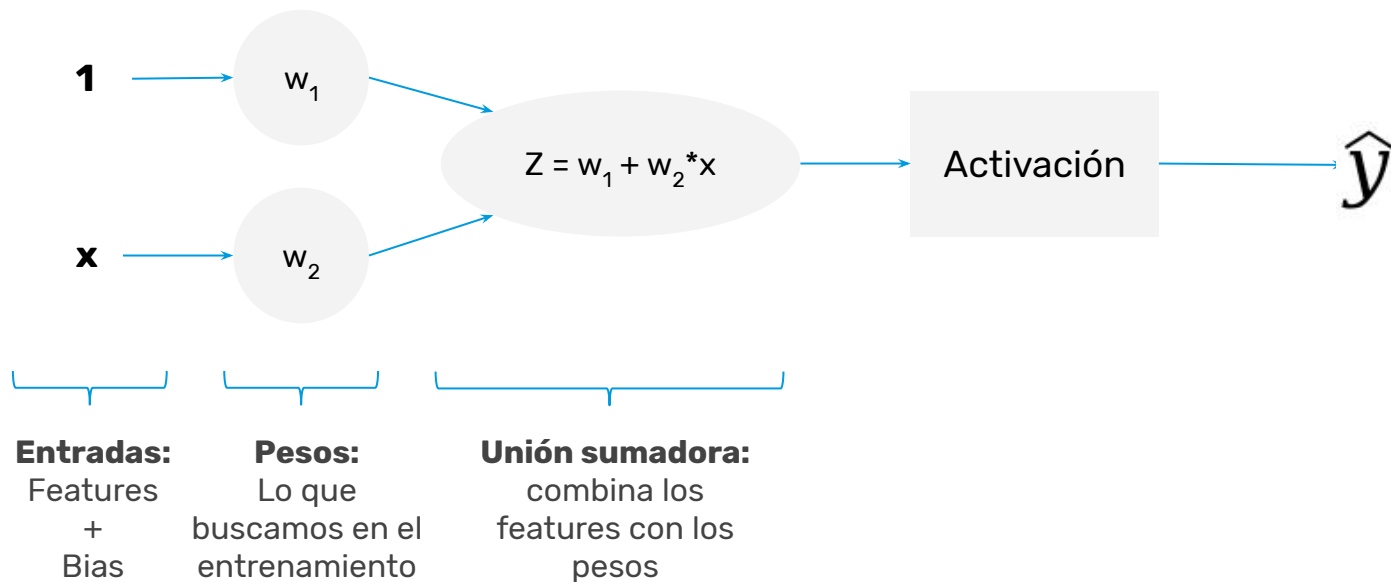
Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



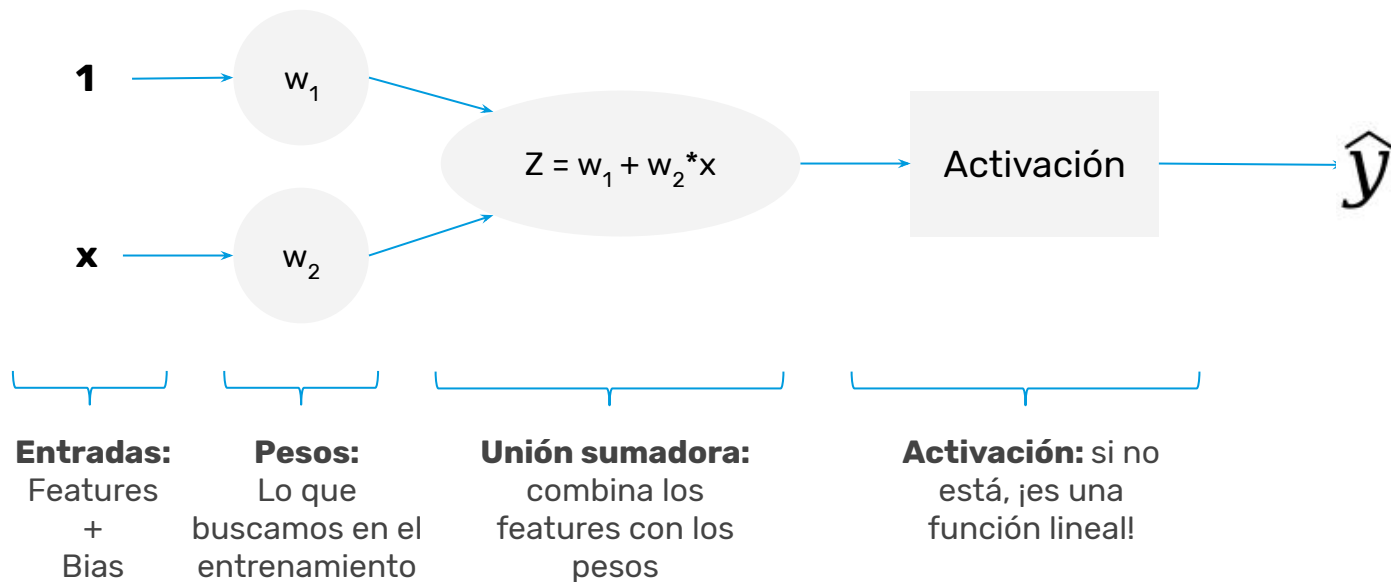
Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



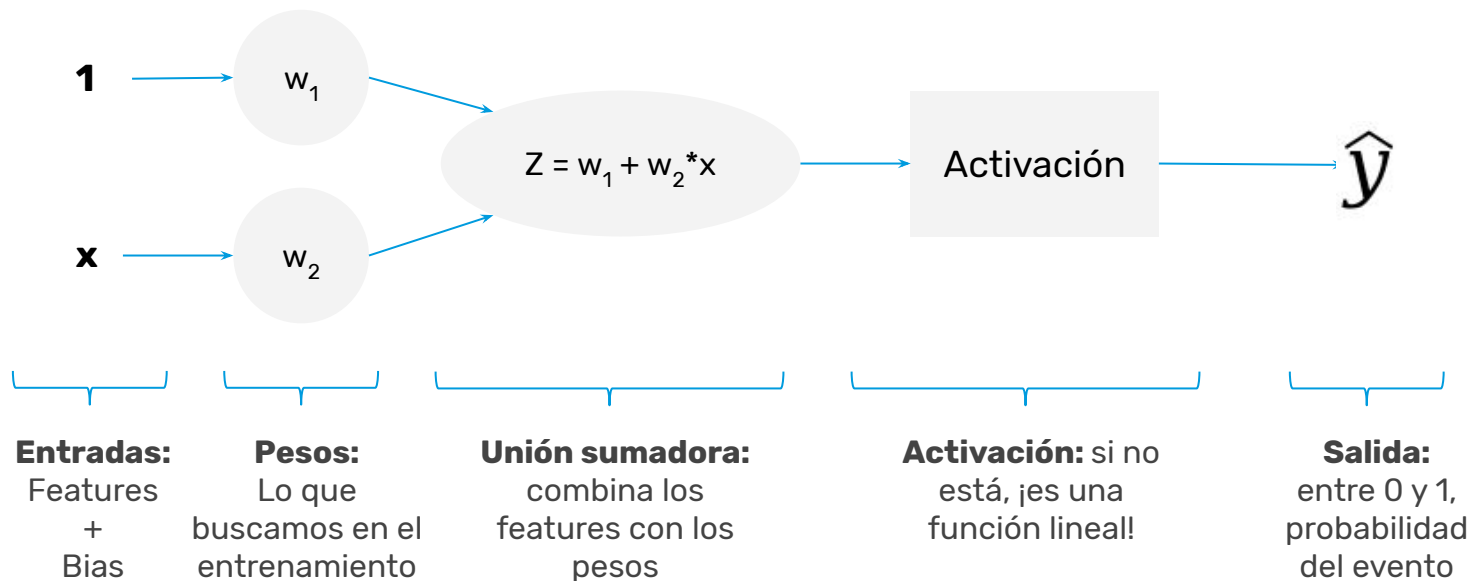
Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



Perceptrón con una variable

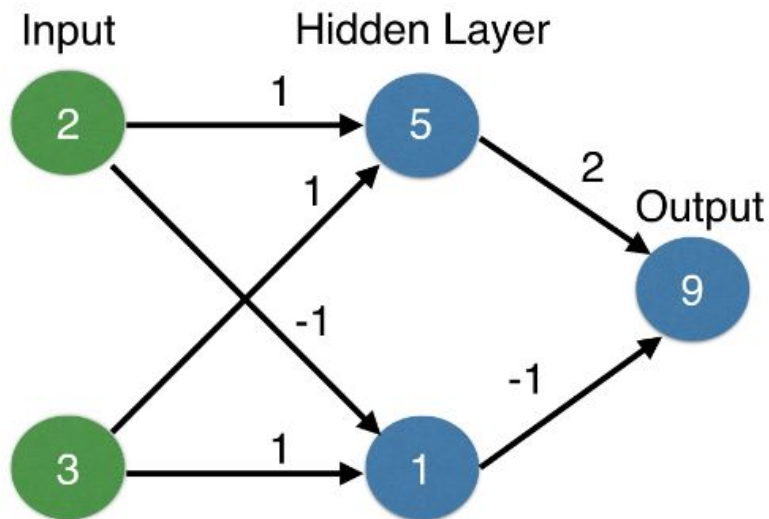
Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1

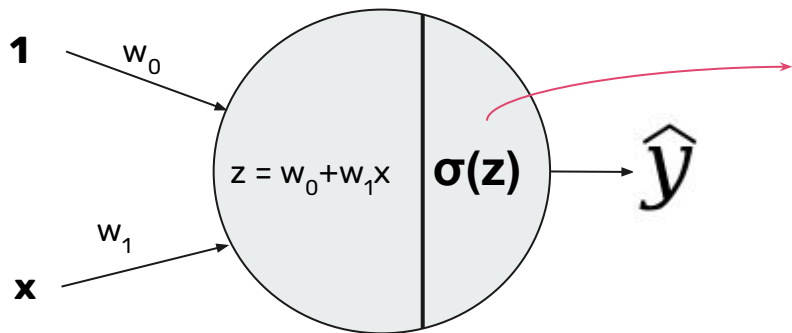
Otra representación



Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1

Otra representación



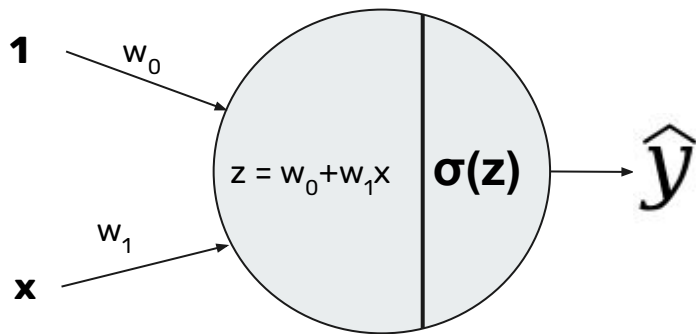
Activación:

- Sin la activación, es una función lineal
- Necesitamos introducir algo que *sature* la entrada en 0 o en 1 dependiendo del resultado de la unión sumadora

Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1

Otra representación



¿Qué falta?

¡Falta encontrar los pesos w_0 y w_1 apropiados para nuestros datos!

Para eso necesitamos una **función de costo**

Funciones de activación

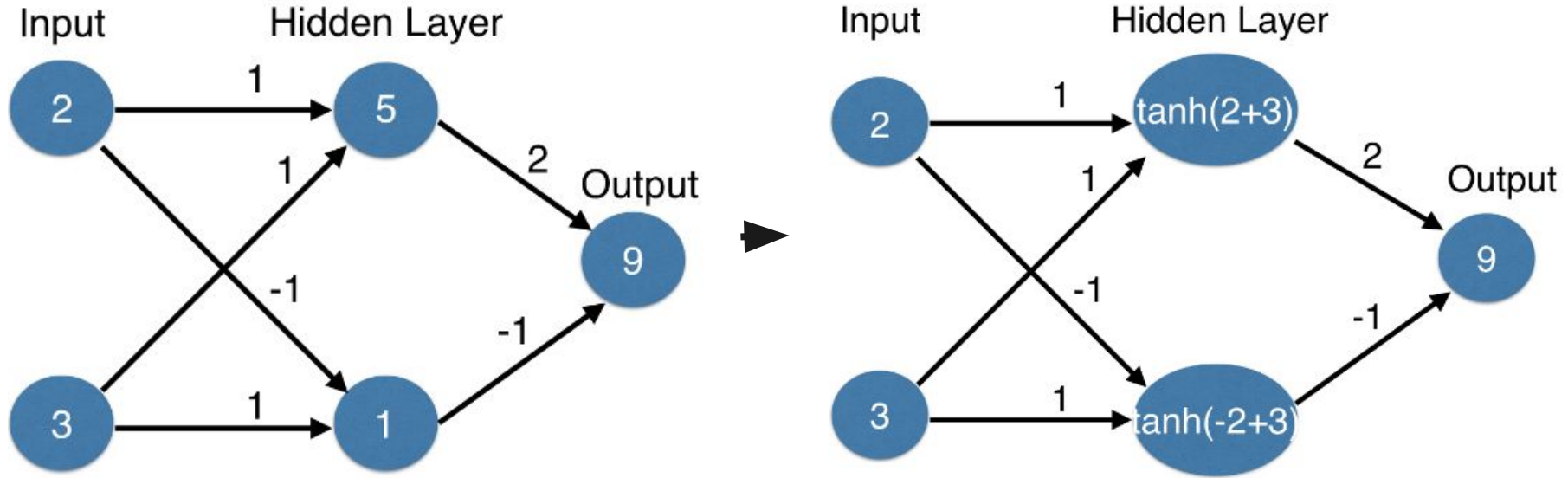


Función de activación

Una función de activación permite capturar funciones no lineales. Esto es especialmente útil cuando nuestros datos no son linealmente separables.

Se aplica sobre los inputs de un nodo para producir un output.

Función de activación



En este caso, se usa la función **TanH (tangente hiperbólica)**.

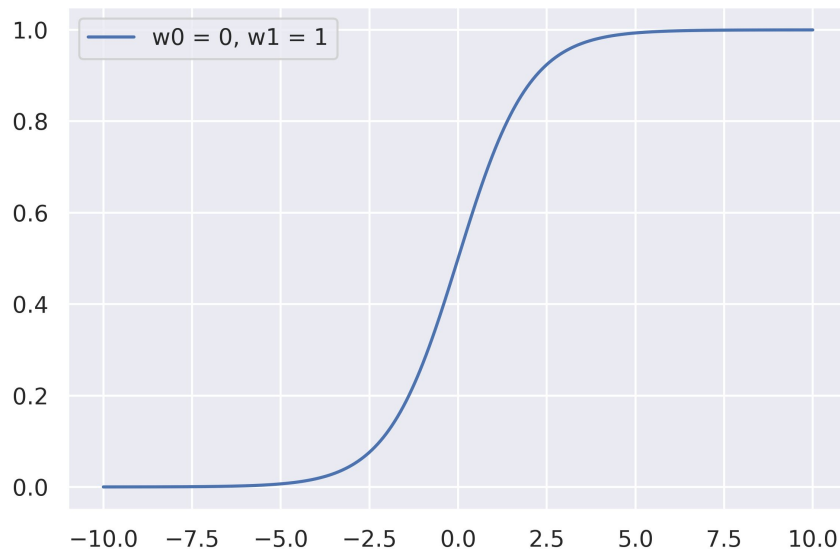
Función Logística / Sigmoide

$$y(z) = \frac{1}{1 + e^{-z}}$$

↓

$$z = w_0 + w_1 x$$

$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$

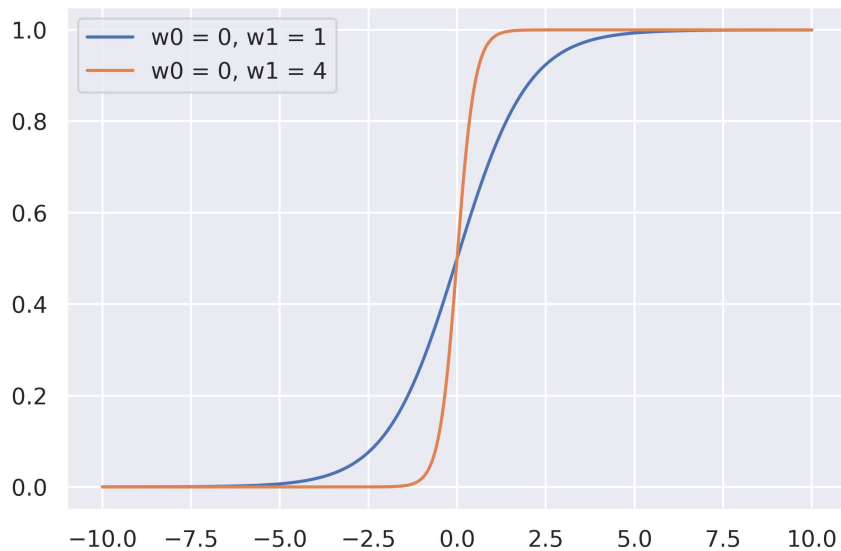


Función Logística / Sigmoides

$$y(z) = \frac{1}{1 + e^{-z}}$$

↓
 $z = w_0 + w_1 x$

$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$

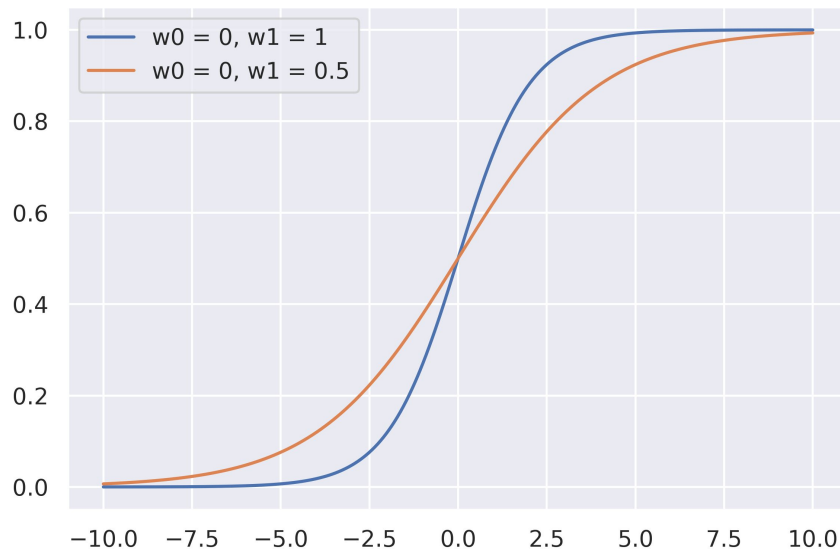


Función Logística / Sigmoides

$$y(z) = \frac{1}{1 + e^{-z}}$$

↓
 $z = w_0 + w_1 x$

$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$

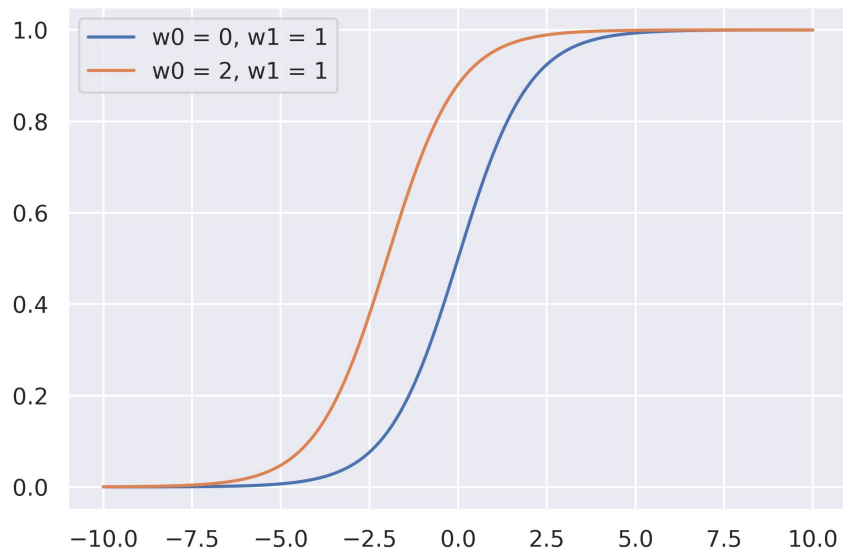


Función Logística / Sigmoides

$$y(z) = \frac{1}{1 + e^{-z}}$$

↓
 $z = w_0 + w_1 x$

$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$



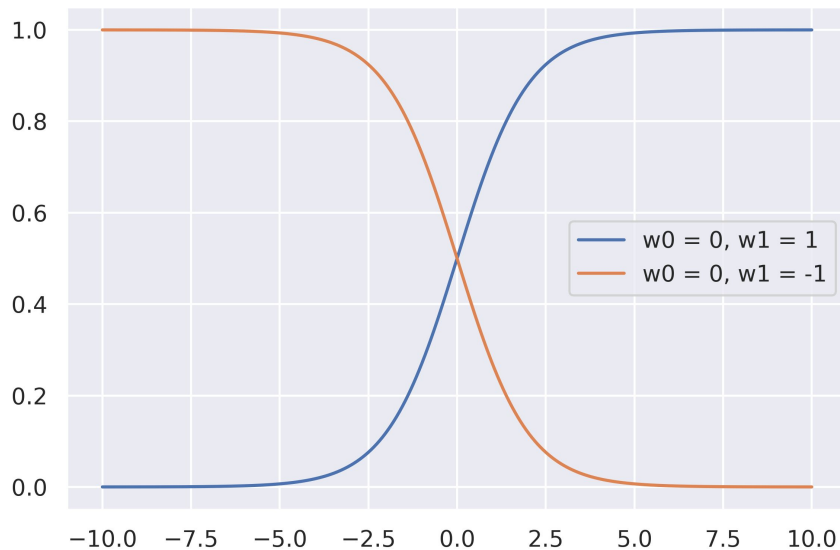
Función Logística / Sigmoide

$$y(z) = \frac{1}{1 + e^{-z}}$$

↓

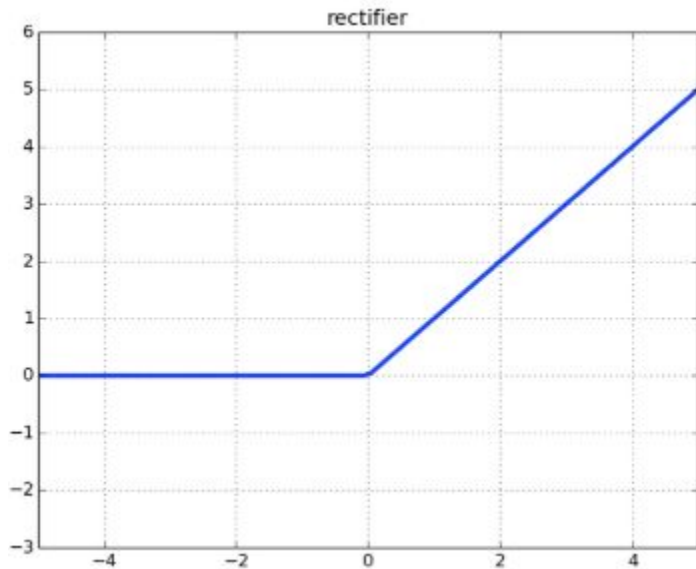
$$z = w_0 + w_1 x$$

$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$



Función de activación

Sin embargo, la más común hoy es **ReLU (rectified linear activation)**



$$ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

A close-up photograph of a white ceramic cup filled with a latte. The surface of the milk is decorated with intricate latte art, featuring a central heart shape surrounded by concentric, wavy lines. The cup is placed on a matching white saucer. In the background, a white napkin and a silver fork are visible, though they are out of focus. The overall lighting is soft and even, highlighting the textures of the coffee and the smooth surface of the cup.

¡BREAK!



Propagation



1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.

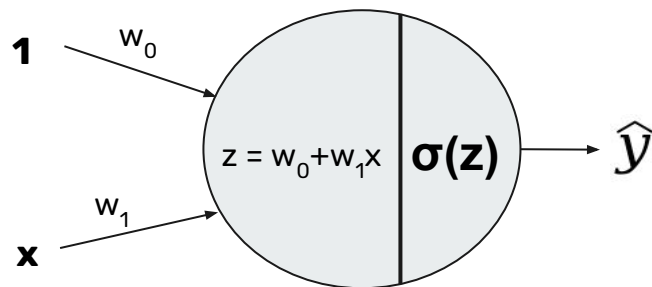
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation.**

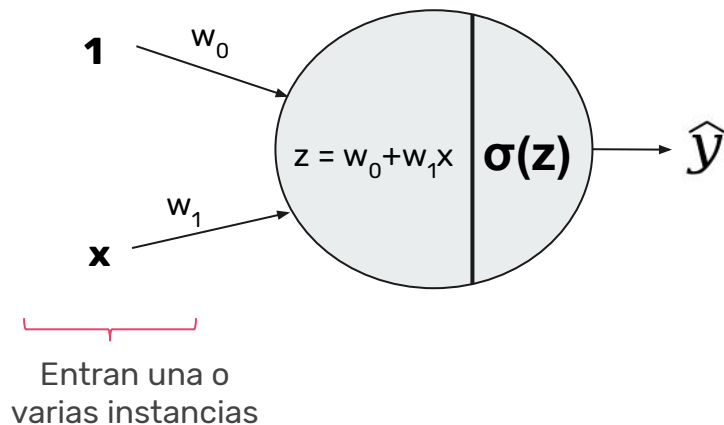
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation**.



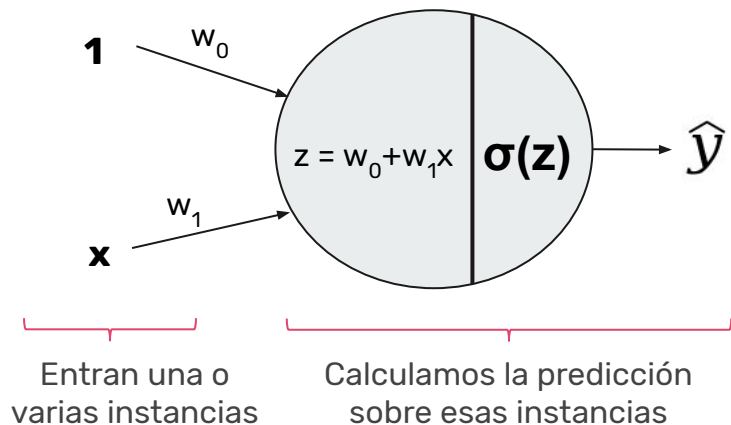
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation**.



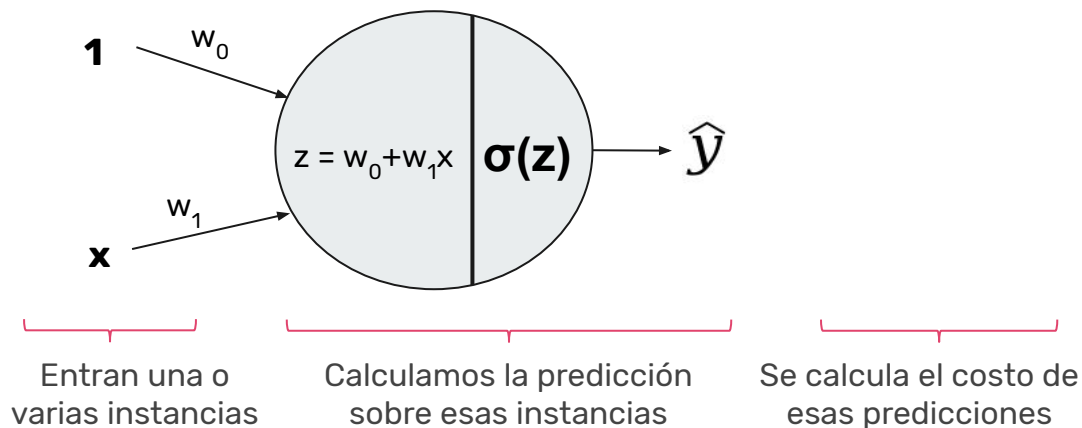
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation**.



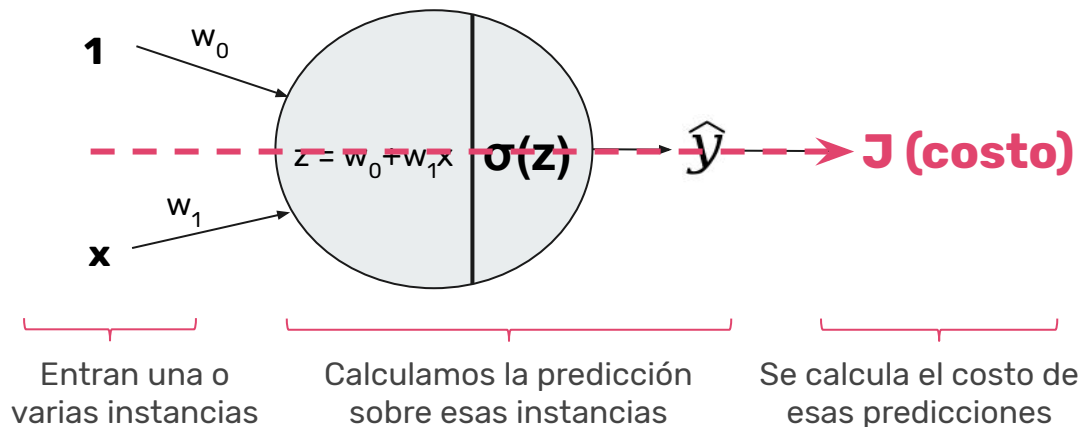
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation**.



1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation**.



1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1 .

$$\begin{aligned}w_0^{nuevo} &= w_0^{viejo} - \alpha * \frac{dJ}{dw_0} \\w_1^{nuevo} &= w_1^{viejo} - \alpha * \frac{dJ}{dw_1}\end{aligned}$$

1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1 .

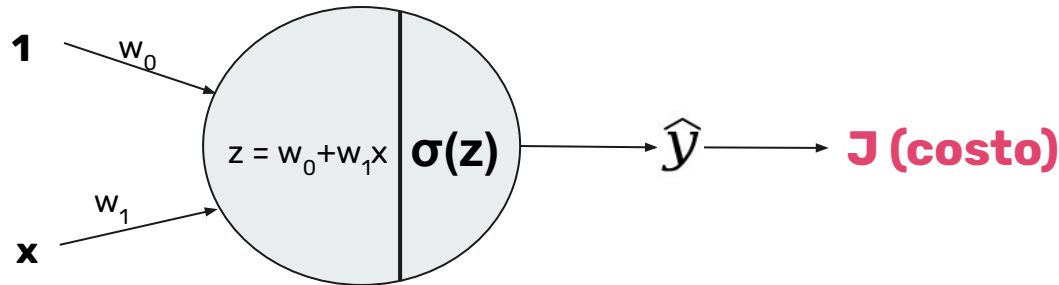
$$\begin{aligned}w_0^{nuevo} &= w_0^{viejo} - \alpha * \frac{dJ}{dw_0} \\w_1^{nuevo} &= w_1^{viejo} - \alpha * \frac{dJ}{dw_1}\end{aligned}$$

Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation**.

1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1 .

$$\begin{aligned}w_0^{nuevo} &= w_0^{viejo} - \alpha * \frac{dJ}{dw_0} \\w_1^{nuevo} &= w_1^{viejo} - \alpha * \frac{dJ}{dw_1}\end{aligned}$$

Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation**.

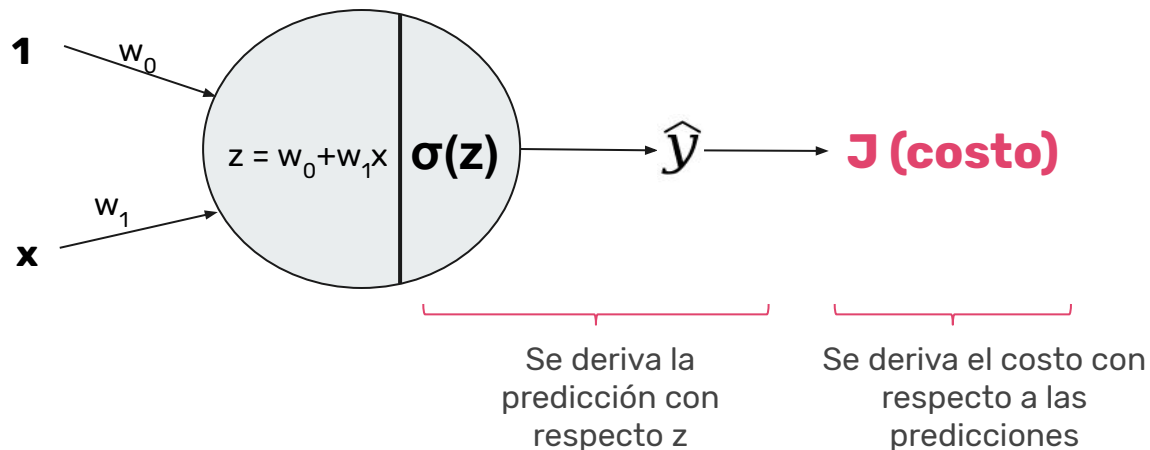


Se deriva el costo con respecto a las predicciones

1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1 .

$$\begin{aligned}w_0^{nuevo} &= w_0^{viejo} - \alpha * \frac{dJ}{dw_0} \\w_1^{nuevo} &= w_1^{viejo} - \alpha * \frac{dJ}{dw_1}\end{aligned}$$

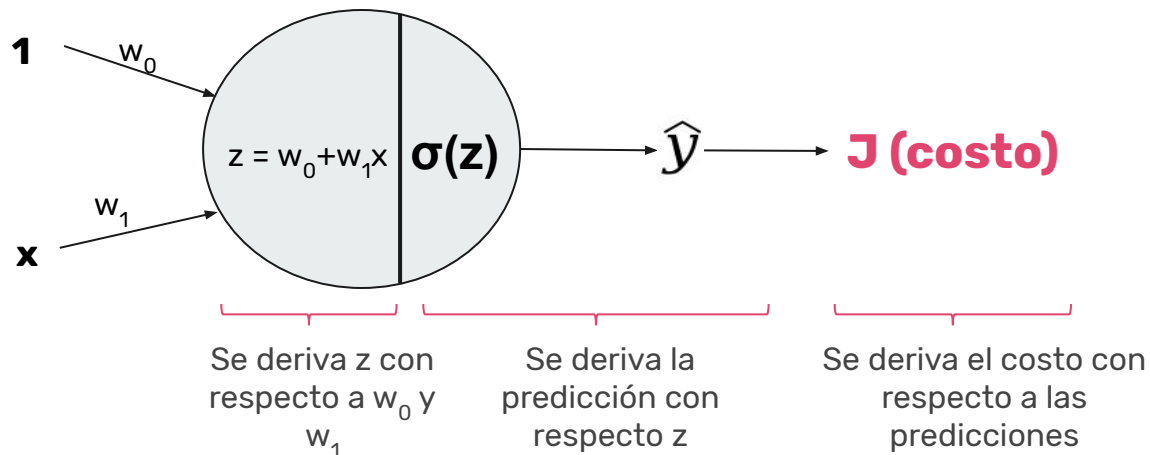
Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation**.



1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1 .

$$\begin{aligned}w_0^{nuevo} &= w_0^{viejo} - \alpha * \frac{dJ}{dw_0} \\w_1^{nuevo} &= w_1^{viejo} - \alpha * \frac{dJ}{dw_1}\end{aligned}$$

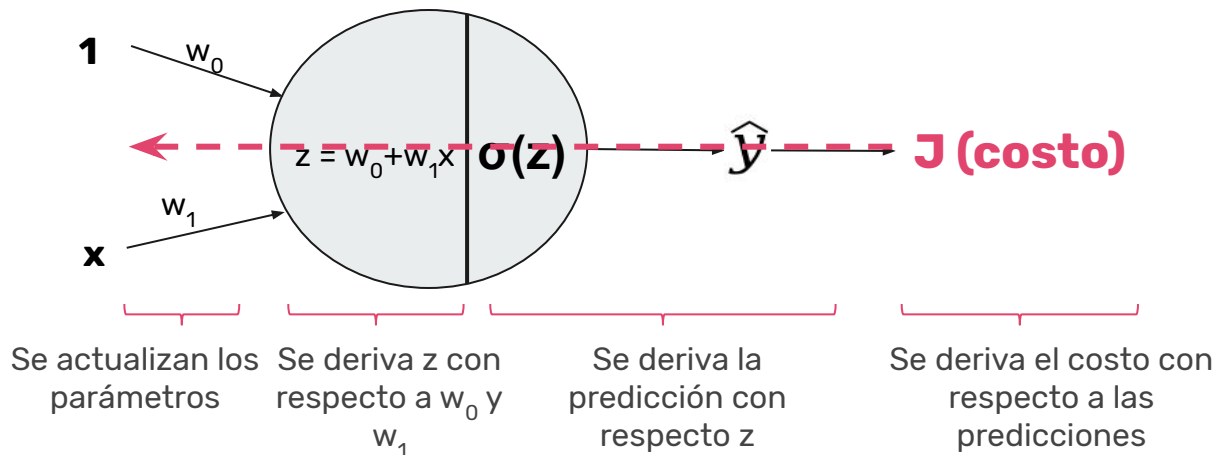
Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation**.



1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1 .

$$\begin{aligned}w_0^{nuevo} &= w_0^{viejo} - \alpha * \frac{dJ}{dw_0} \\w_1^{nuevo} &= w_1^{viejo} - \alpha * \frac{dJ}{dw_1}\end{aligned}$$

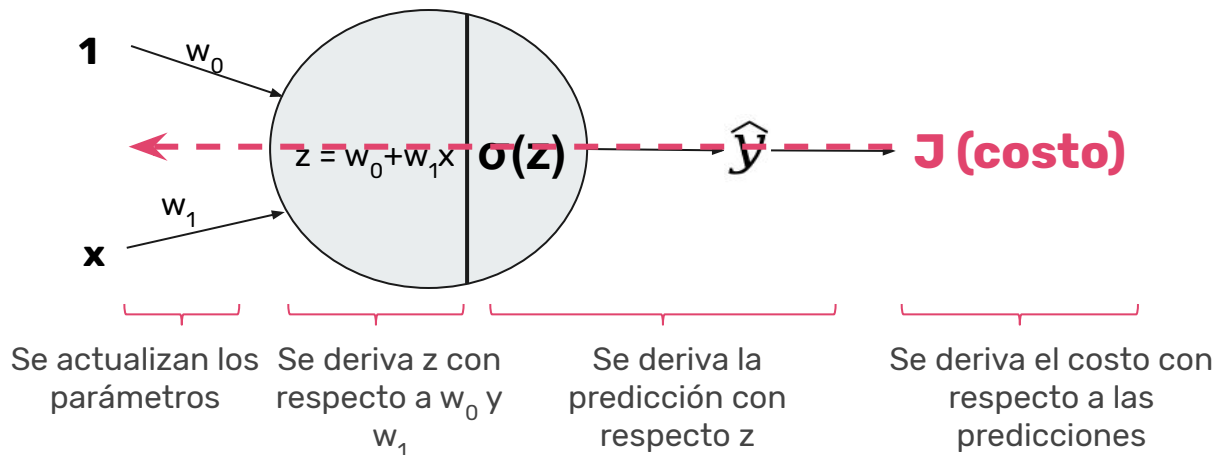
Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation**.



1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1 .

$$\begin{aligned}w_0^{nuevo} &= w_0^{viejo} - \alpha * \frac{dJ}{dw_0} \\w_1^{nuevo} &= w_1^{viejo} - \alpha * \frac{dJ}{dw_1}\end{aligned}$$

Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation**.



Para los que hicieron análisis matemático, ¡es la vieja y conocida Regla de la Cadena!

A close-up photograph of a white ceramic cup filled with a latte. The surface of the milk is decorated with intricate latte art, featuring a central heart shape surrounded by concentric, wavy lines. The cup is placed on a matching white saucer. In the background, a white napkin and a silver spoon are visible, though they are out of focus. The overall lighting is soft and even, highlighting the textures of the coffee and the smooth surface of the cup.

¡BREAK!



Entropía cruzada



Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. \hat{y}

Caso binario: etiquetas $y = 0$ y 1 .

Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. \hat{y}

Caso binario: etiquetas $y = 0$ y 1 .

$$L(\hat{y}, y) = -y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})$$

Pérdida para una instancia

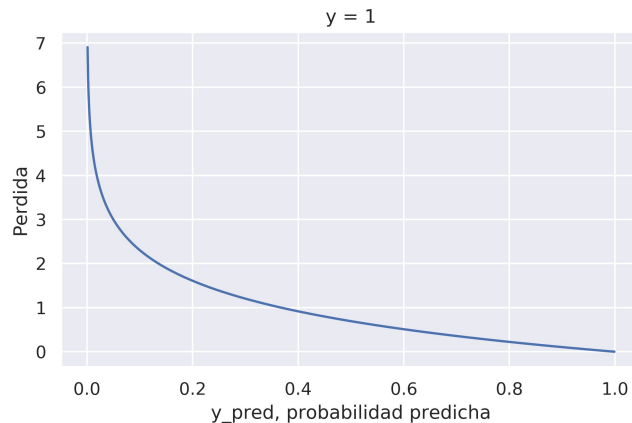
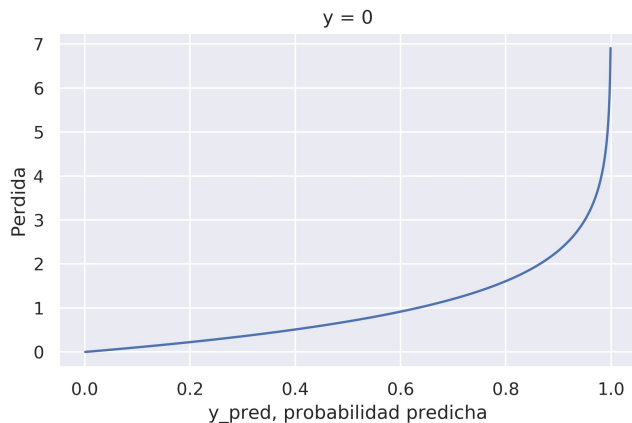
Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. \hat{y}

Caso binario: etiquetas $y = 0$ y 1 .

$$L(\hat{y}, y) = -y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})$$

Pérdida para una instancia



Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. \hat{y}

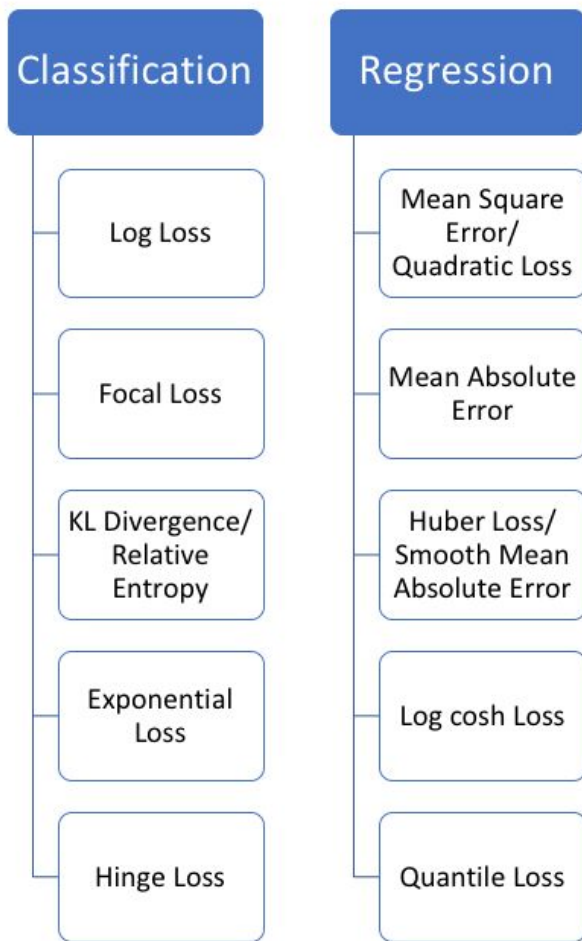
Caso binario: etiquetas $y = 0$ y 1 .

La función requiere que la capa de salida esté configurada con un solo nodo y una activación ' *sigmoide* ' para predecir la probabilidad de la clase 1.

Scikit Learn:

```
sklearn.metrics.log_loss
```

Tenemos muchas funciones de perdida disponibles:



¡Comencemos!



Hands-on training



Hands-on training



DS_Encuentro_30_Perceptron.ipynb

Recursos



Recursos

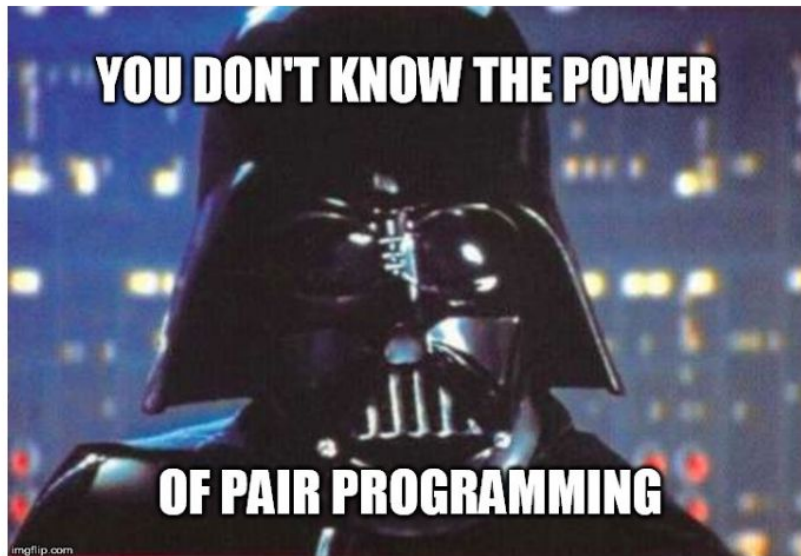


Pair Programming Guide



Weblab Technology [Follow](#)

Jan 25, 2018 · 8 min read



Recursos

Videos de YouTube

- [¿Pero qué "es" una Red neuronal? | Aprendizaje profundo, Parte 1](#)
- [Descenso de gradiente, es como las redes neuronales aprenden | Aprendizaje profundo, capítulo 2](#)



Para la próxima

1. Completar el Notebook de la clase de hoy
2. Ver los siguientes videos (muy cortos):
 - a. <https://www.youtube.com/watch?v=D8iMDH5va9M>
 - b. <https://www.youtube.com/watch?v=fAKwocta2wM>

ACÀMICA