El perceptrón; arquitectura básica de redes neuronales; funciones de activación; backpropagation

Julia Milanese
juliamilanese@gmail.com

Clase 1 Parte 2
Martes 15/04/2025

- **Clase 1: Nociones básicas de redes neuronales para el procesamiento del lenguaje natural**
  - Parte 1: Tokenización, embeddings estáticos
  - **Parte 2: el perceptrón; arquitectura básica de redes neuronales; funciones de activación; backpropagation**.
- Clase 2: Grandes y pequeños modelos de lenguaje.
  - Parte 1: Mecanismo de atención; transformers; embeddings posicionales
  - Parte 2: Similitudes y diferencias entre modelos de lenguaje
- Clase 3: *Prompt engineering*
  - Parte 1: Hiperparámetros de los modelos de lenguaje; prompting: Zero-Shot; Few-Shot
  - Parte 3: Prompting: RAG, Chain of Thought y otros

Presentación

Estructura y temas de la clase de hoy:

1. Introducción
2. Clasificación usando una neurona
3. Problemas lineales y no lineales
   - Problemas no lineales: AND, OR y XOR
   - La función de activación
4. ¿Cómo aprende una red neuronal?
   - La función de pérdida
   - Funciones diferenciables
   - Propiedades de la derivada: regla de la cadena
   - Optimización: descenso del gradiente
5. Recapitulación
6. Bibliografía

Introducción

Neural networks are a fundamental computational tool for language processing, and a very old one. They are called neural because their origins lie in the McCulloch-Pitts neuron (McCulloch and Pitts, 1943), a simplified model of the biological neuron as a kind of computing element that could be described in terms of propositional logic. Jurafsky y Martin (2025)

- Lo modelos de aprendizaje automático previos al uso extendido de las redes neuronales nos permitían asociar un vector de rasgos (features o variables independientes) con la probabilidad de una clase (regresión logística) o una variable continua (regresión lineal).

- Lo modelos de aprendizaje automático previos al uso extendido de las redes neuronales nos permitían asociar un vector de rasgos (features o variables independientes) con la probabilidad de una clase (regresión logística) o una variable continua (regresión lineal).
- La regresión logística, por ejemplo, hace uso de una función lineal para asignar pesos a los rasgos de entrada, y una función de activación para asignar una probabilidad de pertenencia a una clase.

- Lo modelos de aprendizaje automático previos al uso extendido de las redes neuronales nos permitían asociar un vector de rasgos (features o variables independientes) con la probabilidad de una clase (regresión logística) o una variable continua (regresión lineal).
- La regresión logística, por ejemplo, hace uso de una función lineal para asignar pesos a los rasgos de entrada, y una función de activación para asignar una probabilidad de pertenencia a una clase.

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

- Lo modelos de aprendizaje automático previos al uso extendido de las redes neuronales nos permitían asociar un vector de rasgos (features o variables independientes) con la probabilidad de una clase (regresión logística) o una variable continua (regresión lineal).
- La regresión logística, por ejemplo, hace uso de una función lineal para asignar pesos a los rasgos de entrada, y una función de activación para asignar una probabilidad de pertenencia a una clase.

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

- Este tipo de modelo estaba fuertemente asociado a esa selección previa de rasgos que serían finalmente los responsables del éxito del modelo.

- Lo modelos de aprendizaje automático previos al uso extendido de las redes neuronales nos permitían asociar un vector de rasgos (features o variables independientes) con la probabilidad de una clase (regresión logística) o una variable continua (regresión lineal).
- La regresión logística, por ejemplo, hace uso de una función lineal para asignar pesos a los rasgos de entrada, y una función de activación para asignar una probabilidad de pertenencia a una clase.

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

- Este tipo de modelo estaba fuertemente asociado a esa selección previa de rasgos que serían finalmente los responsables del éxito del modelo.
- Las redes neuronales, en cambio, ofrecen la posibilidad de inferir esos rasgos a partir de los casos de entrada (input).

- Lo modelos de aprendizaje automático previos al uso extendido de las redes neuronales nos permitían asociar un vector de rasgos (features o variables independientes) con la probabilidad de una clase (regresión logística) o una variable continua (regresión lineal).
- La regresión logística, por ejemplo, hace uso de una función lineal para asignar pesos a los rasgos de entrada, y una función de activación para asignar una probabilidad de pertenencia a una clase.

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

- Este tipo de modelo estaba fuertemente asociado a esa selección previa de rasgos que serían finalmente los responsables del éxito del modelo.
- Las redes neuronales, en cambio, ofrecen la posibilidad de inferir esos rasgos a partir de los casos de entrada (input).
- A su vez, aprenden funciones no lineales, y ofrecen así la solución a problemas complejos cuyo modelado es no lineal.

- En una red neuronal podremos tener múltiples capas de unidades que asignan pesos (tantas como lo requiera el problema entre manos).

- En una red neuronal podremos tener múltiples capas de unidades que asignan pesos (tantas como lo requiera el problema entre manos).
- De ahí que a la tarea de aprendizaje por redes neuronales apiladas en capas se la denomine aprendizaje profundo.

- En una red neuronal podremos tener múltiples capas de unidades que asignan pesos (tantas como lo requiera el problema entre manos).
- De ahí que a la tarea de aprendizaje por redes neuronales apiladas en capas se la denomine aprendizaje profundo.
- Una función de pérdida será el punto de partida para entender hacia dónde variar los pesos de cada capa hasta alcanzar valores óptimos.

- En una red neuronal podremos tener múltiples capas de unidades que asignan pesos (tantas como lo requiera el problema entre manos).
- De ahí que a la tarea de aprendizaje por redes neuronales apiladas en capas se la denomine aprendizaje profundo.
- Una función de pérdida será el punto de partida para entender hacia dónde variar los pesos de cada capa hasta alcanzar valores óptimos.
- Para lograr este aprendizaje, las redes neuronales haceb uso de la propagación del error y la regla de la cadena.

Partamos de un ejemplo sencillo.
Imaginemos que debemos clasificar dos oraciones.
La única clase disponible es "ser_robot".
Es decir que, si la oración indica pertenencia a esa clase, debemos obtener un valor cercano a 1 (La probabilidad de la clase). En el caso contrario, un valor cercano a 0.

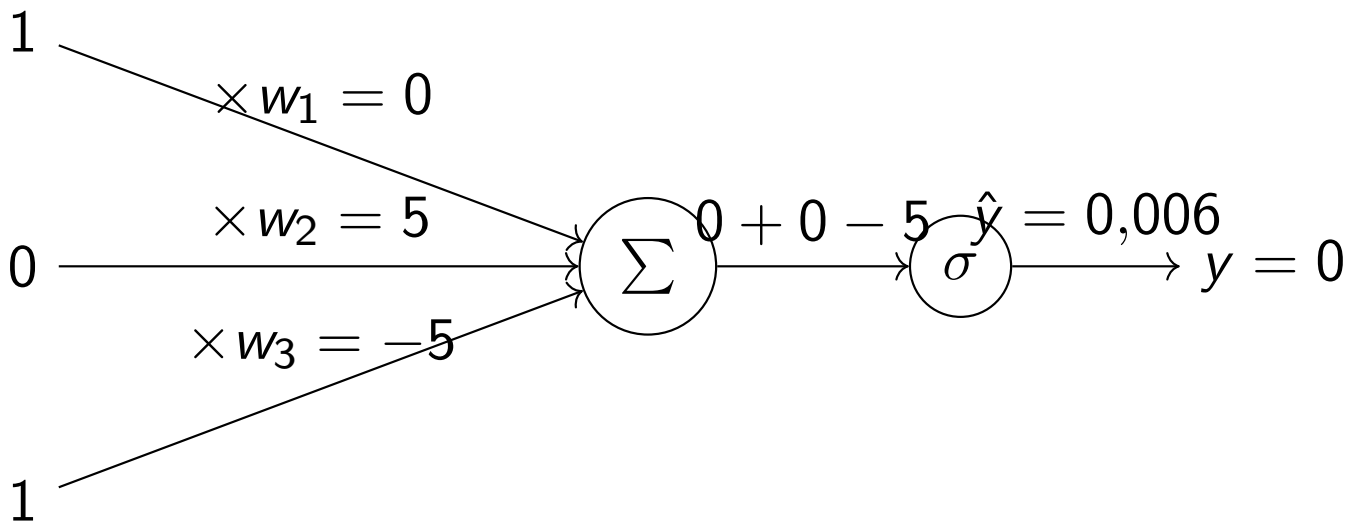
Nuestro vocabulario consta de 3 palabras y nuestro corpus, de dos oraciones:

	Soy	Robot	Humano
Soy Robot	1	1	0
Soy Humano	1	0	1

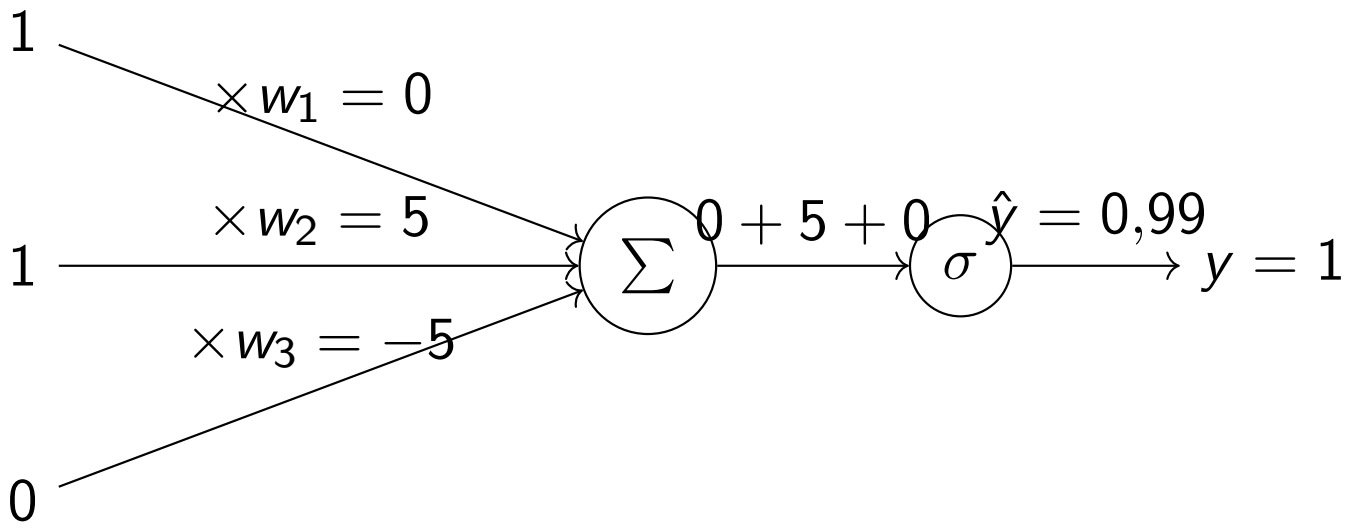
Y los valores de verdad para la clase "ser_robot"de nuestras oraciones son:

	ser_robot
Soy Robot	1
Soy Humano	0

Para este ejemplo de juguete, es bastante sencillo pensar cómo deben estar pesadas las palabras (features) para darnos el output que esperamos.
Si tomamos la oración 'soy humano' y le agregamos pesos, sumamos cada variable por su peso y luego aplicamos la función sigmoidal podemos obtener el resultado esperado.

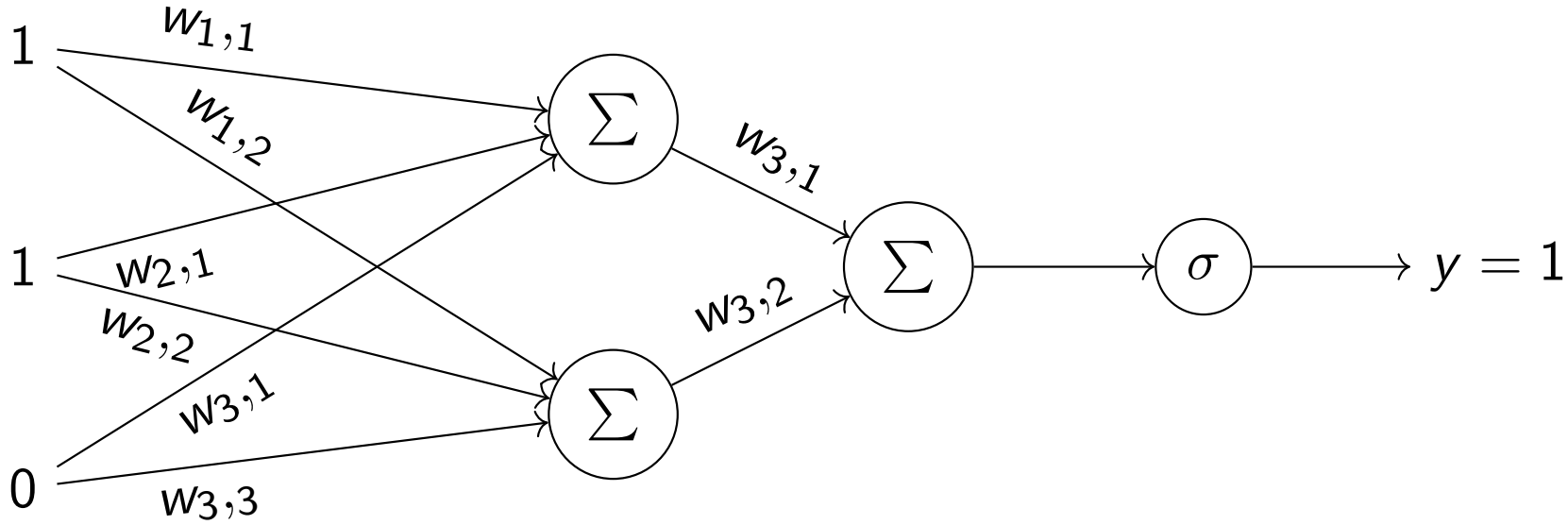


Si tomamos la oración "soy robot", los mismos pesos nos devolverán la clase correcta.



Hasta ahora, nuestro modelo no es más complejo que una regresión logística. Podríamos pensar entonces en sumar una neurona más a nuestra red y esperar que durante el aprendizaje, una de las dos neuronas codifique el aspecto 'robot' y la otra el aspecto 'humano'.

Agregamos una tercera neurona como capa final para sumar los valores de salida de las dos neuronas anteriores, y de forma previa a la función que transformará los valores de salida (logits) en una probabilidad.



La red presentada arriba, sin embargo, no es una red neuronal útil ya que la salida de cada neurona es una transformación lineal.
Es decir,

$$(x_1 \cdot w_{1,1} + x_2 \cdot w_{1,2} + b) \cdot w_{3,1} + (x_1 \cdot w_{2,1} + x_2 \cdot w_{2,2} + b) \cdot w_{3,2} + b$$

es igual a

$$w_{3,1} \cdot x_1 \cdot w_{1,1} + w_{3,1} \cdot x_2 \cdot w_{1,2} + w_{3,2} \cdot x_1 \cdot w_{2,1} + w_{3,2} \cdot x_2 \cdot w_{2,2} + b + b + b$$

que es básicamente

$$X \cdot W + b$$

Observemos las tablas de verdad de Y, O y O-exclusivo

A	B	$A \wedge B$	$A \vee B$	$A \oplus B$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Observemos ahora su representación en el plano:

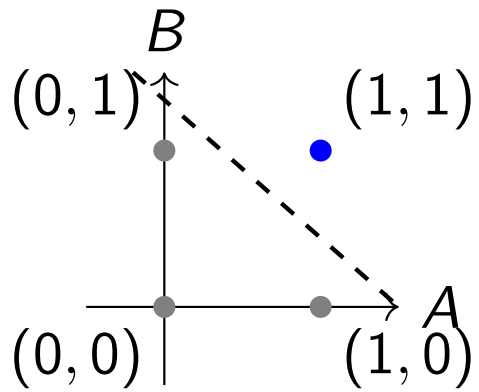


Figura: AND

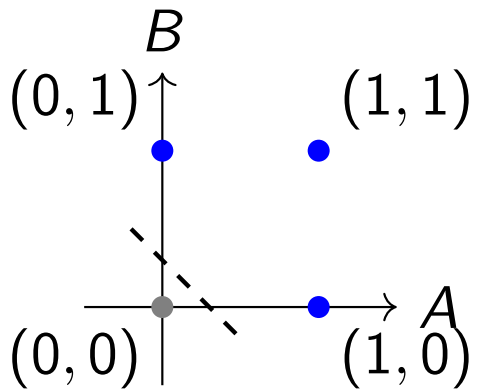


Figura: OR

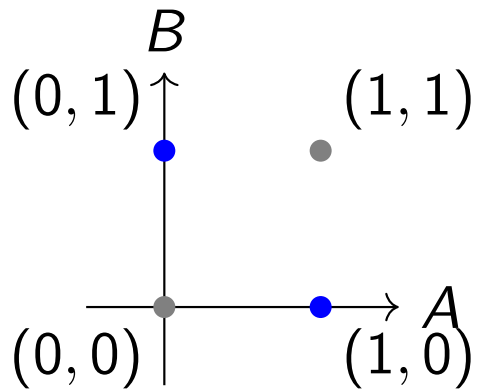


Figura: XOR

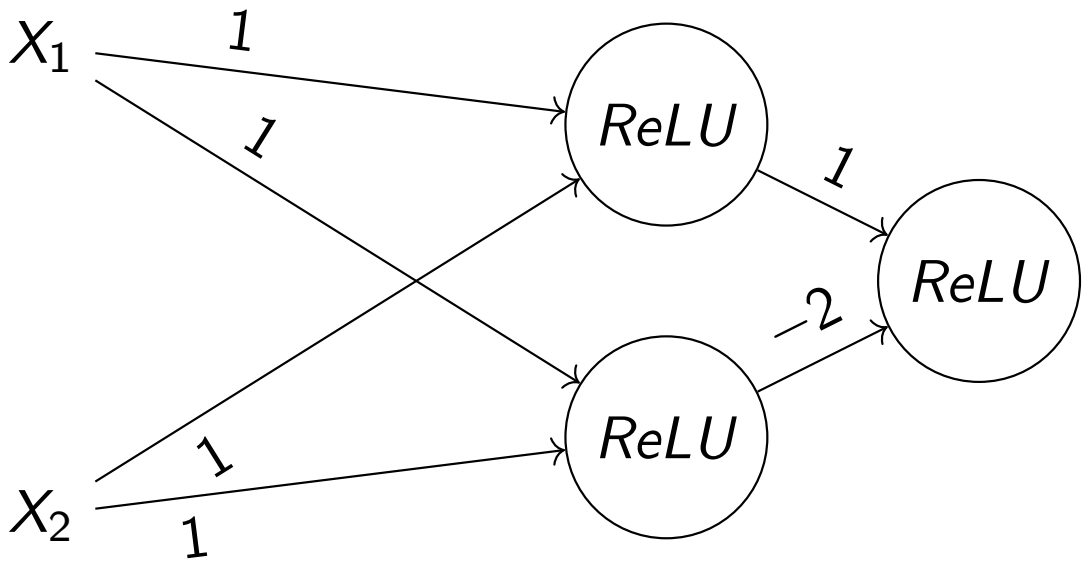
- XOR no es linealmente separable.
- Es decir, no podemos trazar una linea recta que separe los puntos de una clase de los de otra.

Para evitar esta linealidad, las redes neuronales suman una no-linealidad a cada neurona (o parámetro). Esta no linealidad se conoce como función de activación. La función sigmoidal es una función no lineal. Otra función no lineal muy usada en RN es ReLU, función recitlínea uniforme.

$$\text{ReLU}(x) = \text{máx}(0, x)$$

Esto significa que la salida de algunas neuronas va a ser 0 (para números negativos), es decir que van a quedar sin activarse.

Solución de XOR usando Redes Neuronales y ReLU.



Goodfellow et al. (2016). Citado en Jurafsky

Y así obtenemos la definición de la unidad de una red neuronal (la neurona):

$$y = \sigma(b + \sum_i w_i x_i)$$

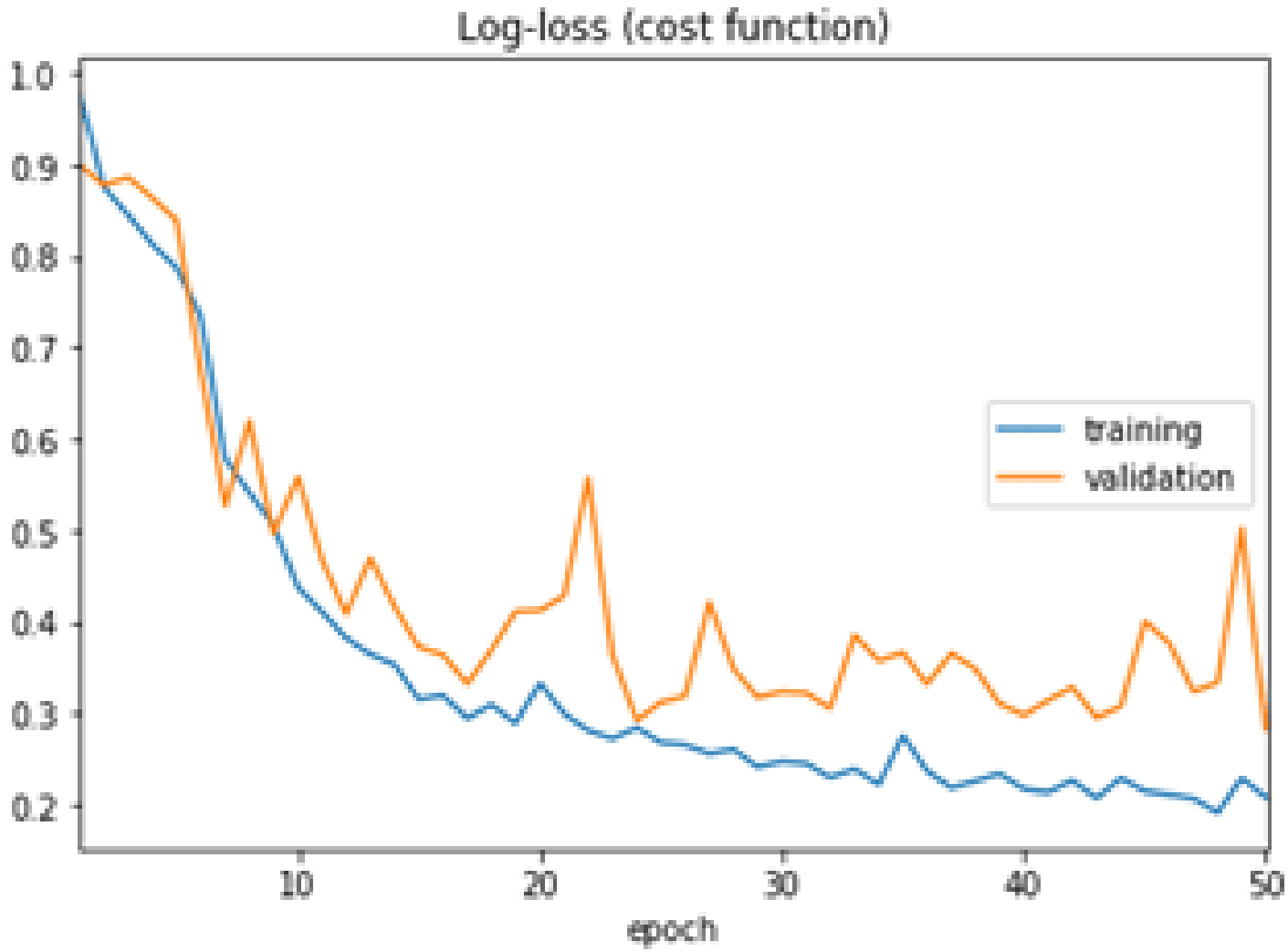
Como vimos en la red completa que dibujamos más arriba, la asignación de pesos para cada una de las conexiones completas (este tipo de capa conectada se conoce como "feedforward") no es trivial como en el primer ejemplo.
Al iniciarse la red, los pesos asociados a cada conexión se seleccionarán de manera aleatoria y es por medio del proceso de entrenamiento que esos pesos (y bias) encontrarán el valor que mejor se adapta a los casos de entrada.

El objetivo de la red neuronal al momento de entrenarse es minimizar la distancia entre sus valores de salida y los valores esperados en el corpus de entrenamiento.
Las redes neuronales van a aprender del error. Y la función de pérdida es la encargada de medir ese error.

La función de pérdida es una función que mide cuán bien la red neuronal está haciendo su trabajo.
La función de pérdida más común para funciones de activación sigmoidales es la función de pérdida de entropía cruzada.

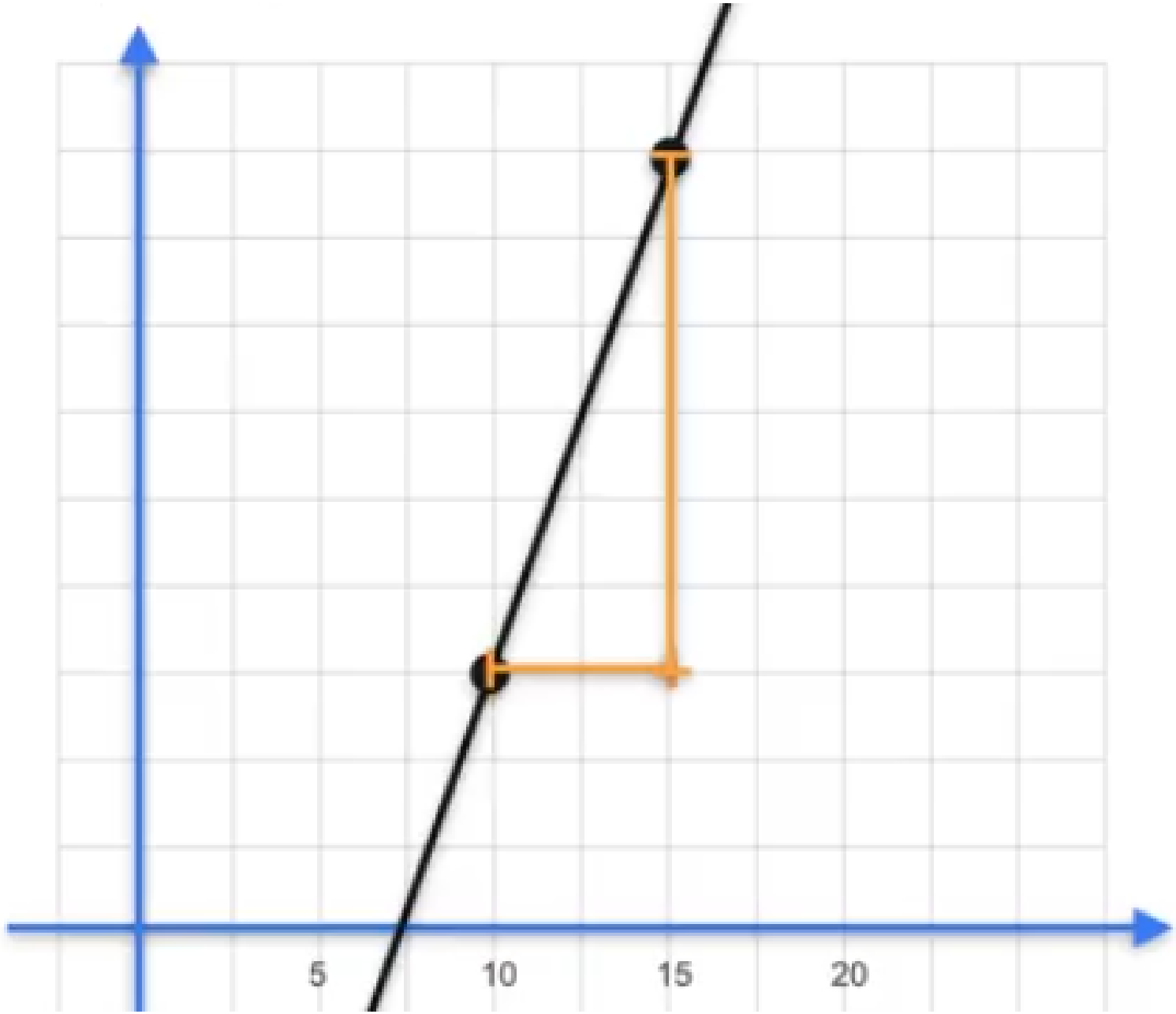
$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

El objectivo, entonces, del entrenamiento de una red de este tipo será minimizar esa función, es decir, que su valor sea lo más cercano a 0 posible.



Para saber cómo debemos modificar los pesos de la red para minimizar la función de pérdida, necesitamos calcular la derivada de la función de pérdida con respecto a los pesos.

La derivada se define como la tasa de cambio instantánea de una función en un punto dado.
Si pensamos en la función de la distancia que recorre un objeto en movimiento, la derivada de esa función nos dirá la velocidad del objeto en un momento dado.
Por ejemplo, si tomamos dos puntos en el tiempo y los metros recorridos, podemos calcular la velocidad promedio del objeto en ese intervalo. La velocidad se calcula como distancia dividida por tiempo. Esta fórmula es la misma que la de la pendiente de una línea recta: elevacion sobre recorrido.

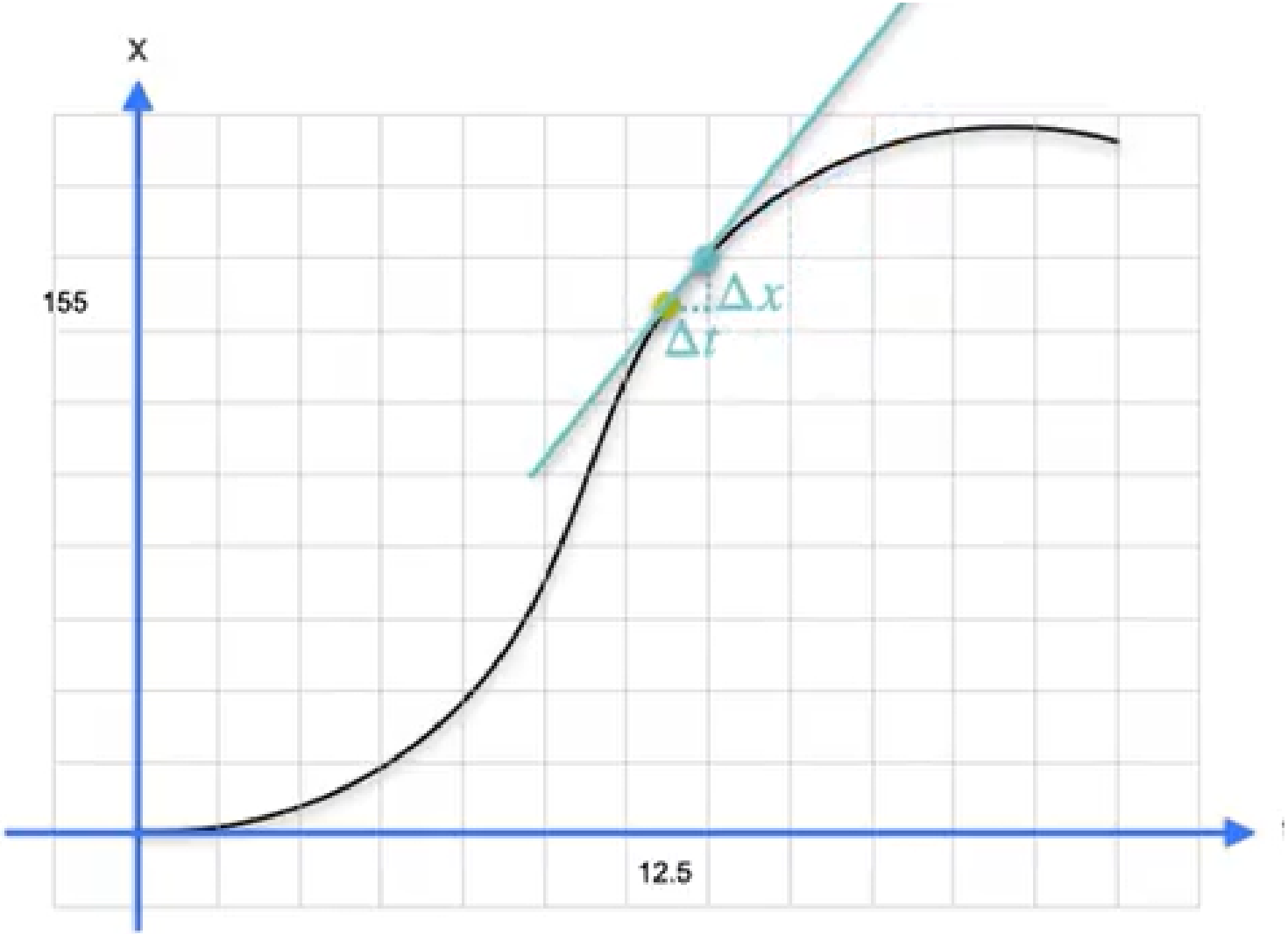

5
10
15
20

Aquí, la elevación es el cambio en la distancia y el recorrido es el cambio en el tiempo. Esto nos muestra el cambio promedio en la velocidad en un intervalo de tiempo.

$$\frac{\Delta x}{\Delta t}$$

¿Pero cuál es la velocidad instantánea en un punto dado? Para calcular la velocidad instantánea, tomamos un intervalo de tiempo infinitesimalmente pequeño, el límite. En este caso, la elevación y el recorrido se vuelven infinitesimales, y la recta que se forma entre esos dos puntos es la tangente a la curva en ese punto.
De un modo más general, la derivada es una medida de cuán rápido cambia la relación entre dos variables en cualquier punto.

La derivada de una función en un punto es la pendiente de la tangente en ese punto.

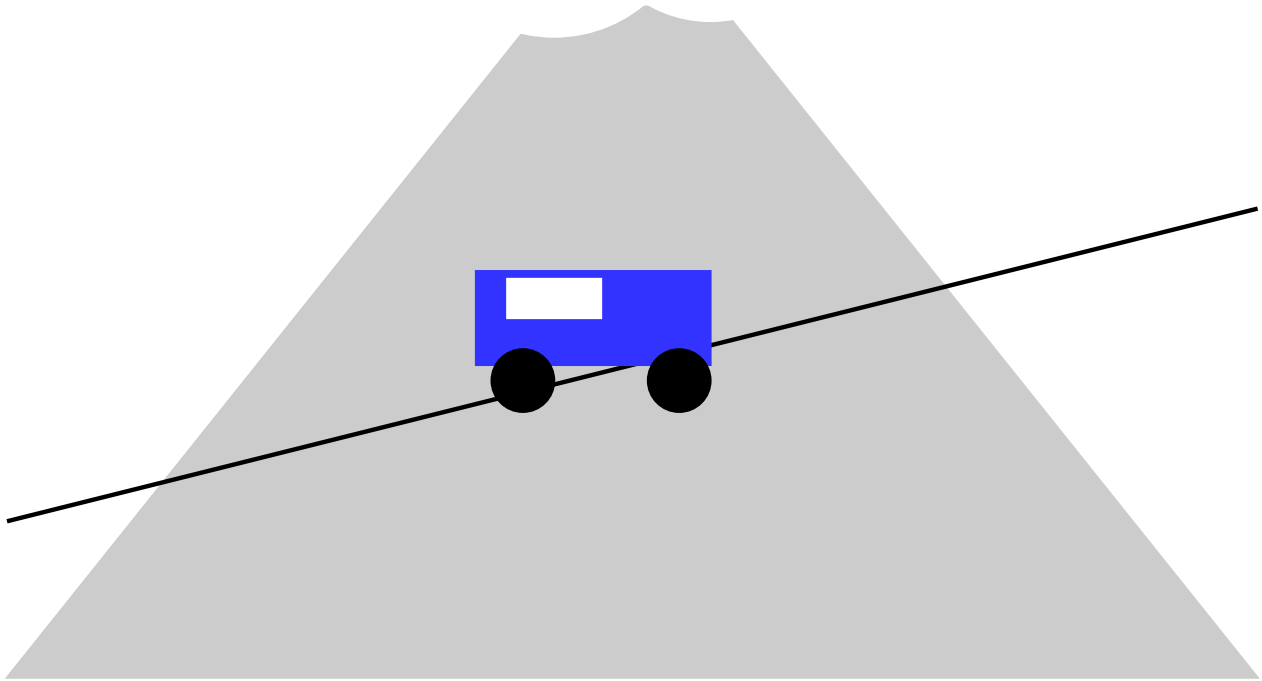


Asumiendo que tenemos una función $f(x)$ que es diferenciable en un punto x_0 y una función $g(x)$ que es diferenciable en $f(x_0)$, entonces la función compuesta $g(f(x))$ es diferenciable en x_0. La derivada de la función compuesta se puede calcular usando la regla de la cadena:

$$\frac{d}{dx}g(f(x)) = g'(f(x)) \cdot f'(x)$$

En otras palabras, la derivada de la función compuesta es el producto de la derivada de la función exterior evaluada en la función interior y la derivada de la función interior.

La rega de la cadena nos permitiría saber, por ejemplo, cómo varía la temperatura con respecto al tiempo de viaje de este autito.



Si sabemos cómo cambia la temperatura con respecto a la altura ,

$$\frac{dT}{da}$$

y como cambia la altura con respecto al tiempo (tiempo que tarda el auto en subir),

$$\frac{da}{dt}$$

podemos saber cómo cambia la temperatura con respecto al tiempo.

$$\frac{dT}{dt}$$

Optimización es el proceso de encontrar el máximo o el mínimo de una función. Los mínimos (y los máximos) de una función son los puntos donde la pendiente de la tangete es cero.
En machine learning, la optimización se refiere a encontrar los valores de los parámetros de un modelo que minimizan la función de pérdida.

Hasta ahora hemos visto ejemplos para funciones de una variable, pero en el caso de una red neuronal, tenemos muchas variables (parámetros), que son los pesos y bias de cada neurona.

Hasta ahora hemos visto ejemplos para funciones de una variable, pero en el caso de una red neuronal, tenemos muchas variables (parámetros), que son los pesos y bias de cada neurona.
La estrategia de optimización será considerar a todas las variables como constantes exceptuando a una. Ahora, la función es una función de una variable de la que se puede elegir en un punto la tangente y obtener así la derivada parcial (con respecto a una de las variables). Por ejemplo:

$$f(x, y) = x^2 + y^2$$

$$f_x = \frac{\partial f}{\partial x} = 2x$$

$$f_y = \frac{\partial f}{\partial y} = 2y$$

El gradiente de una función multivariable es un vector que contiene las derivadas parciales de la función con respecto a cada una de sus variables. En nuestro ejemplo, el gradiente es

$$\begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

La notación para el gradiente es:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{1}$$

El gradiente es útil para encontrar los máximos y mínimos de una función multivariable como la derivada lo era de una función de una variable.
El algoritmo de gradiente descendiente es un método iterativo para encontrar el mínimo de una función multivariable.

El algoritmo funciona de la siguiente manera:

1. Seleccionar un punto inicial aleatorio.
2. Calcular el gradiente de la función en ese punto.
3. Actualizar el punto en la dirección opuesta al gradiente. ¿Cómo? Restando el gradiente al punto actual. Ya que esto puede resultar en un salto muy grande, se multiplica el gradiente por un factor de aprendizaje (learning rate, uno de los hiperparámetros de la red neuronal).
4. Repetir los pasos 2 y 3 hasta que el gradiente sea cero o muy pequeño.

¿Cómo va a aprender el modelo entonces? Calculando la gradiente de cada peso con respecto a la función de pérdida. El gradiente de la función de pérdida con respecto a un peso es la derivada parcial de la función de pérdida con respecto a ese peso.

$$\frac{\partial L}{\partial w}$$

Y la gradiente descendiente es el algoritmo que se utiliza para actualizar los pesos de la red neuronal en cada paso (step) del entrenamiento.

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w}$$

Para terminar, veamos cómo se actualizan los pesos de una red simple utilizando la optimización de descenso del gradiente:

$$\frac{\partial L}{\partial \hat{y}} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial b} = \hat{y}(1-\hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_1} = \hat{y}(1-\hat{y})x_1$$

$$\frac{\partial \hat{y}}{\partial w_2} = \hat{y}(1-\hat{y})x_2$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}\hat{y}(1-\hat{y})$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}\hat{y}(1-\hat{y})x_1$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}\hat{y}(1-\hat{y})x_2$$

$$\frac{\partial L}{\partial b} = -(y - \hat{y})$$

$$\frac{\partial L}{\partial w_1} = -(y - \hat{y})x_1$$

$$\frac{\partial L}{\partial w_2} = -(y - \hat{y})x_2$$

$$w_1 \rightarrow w_1 - \alpha(-x_1(y - \hat{y}))$$

$$w_2 \rightarrow w_2 - \alpha(-x_2(y - \hat{y}))$$

$$b \rightarrow b - \alpha(-(y - \hat{y}))$$

DeepLearning.AI (2023)

En esta clase introdujimos los siguientes temas:

- El perceptrón
- Problemas lineales y no lineales
- La función de activación
- La función de pérdida
- La optimización por descenso del gradiente

Bibliografía I

DeepLearning.AI (2023). Calculus for machine learning and data science.

Jurafsky, D. y Martin, J. H. (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3 edición. Online manuscript released January 12, 2025.