ACÁMICA

¡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?

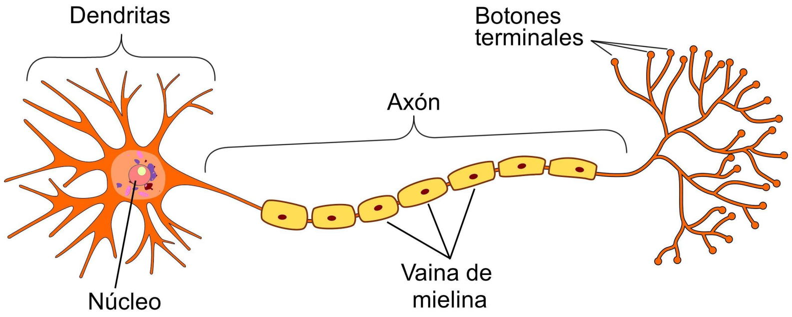
Repaso: Descenso por Gradiente

Redes neuronales

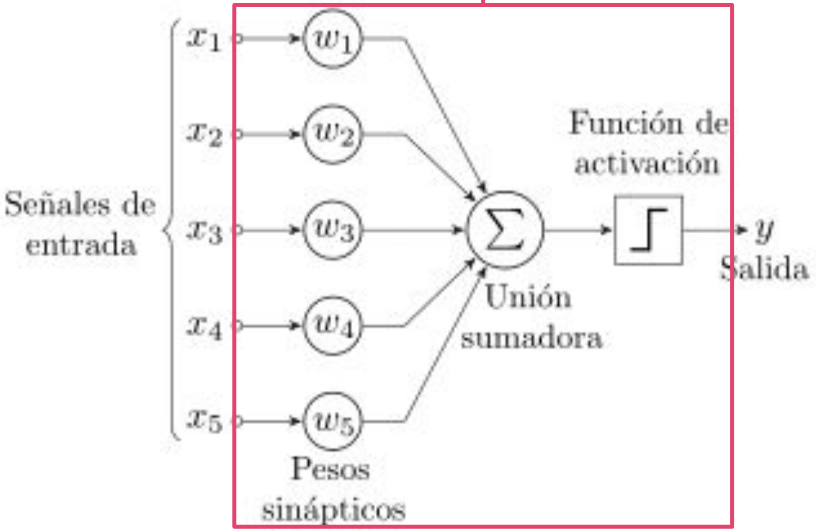
Esperamos que aprendan los siguientes conceptos:

- 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

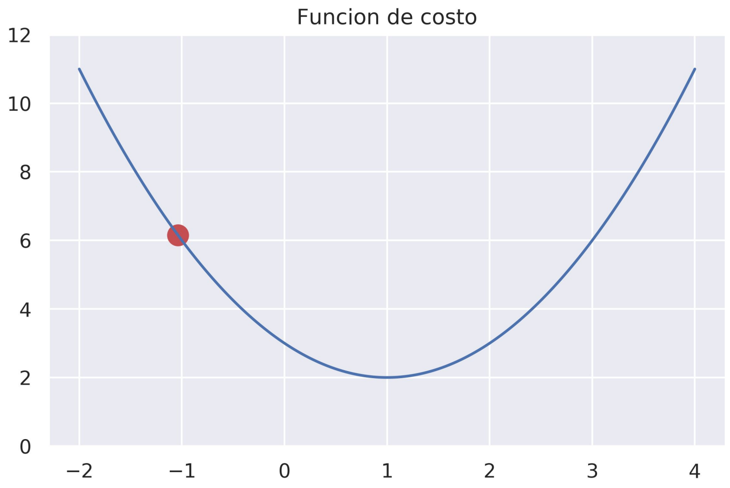


Descenso por gradiente

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:

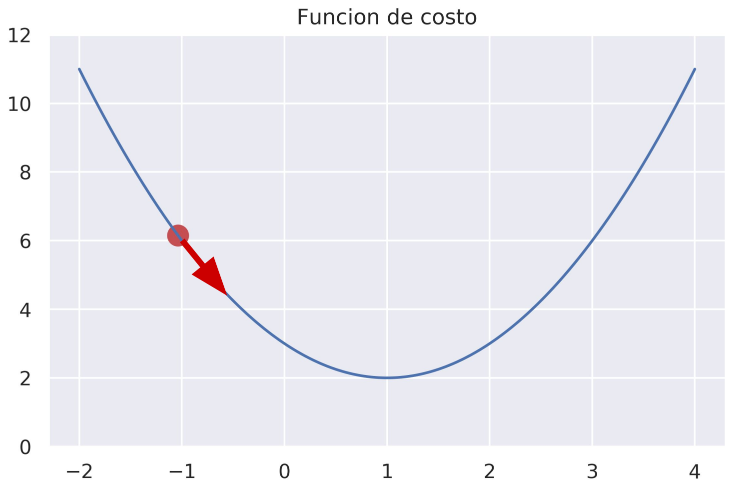


Pasos

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:

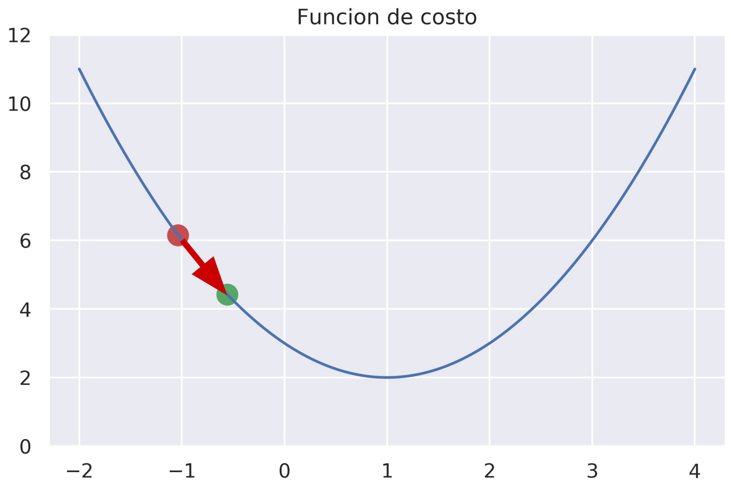


Pasos

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:

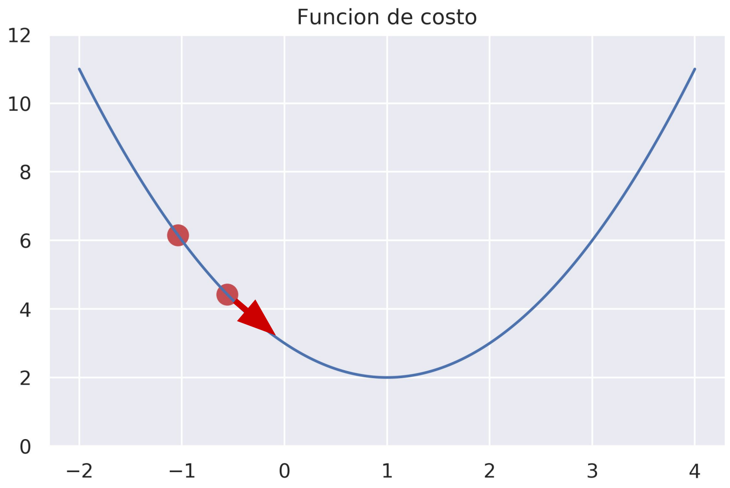


Pasos

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:

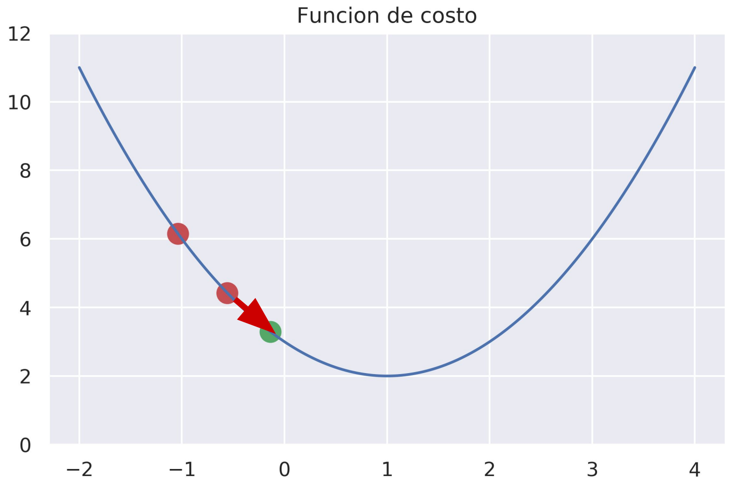


Pasos

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:

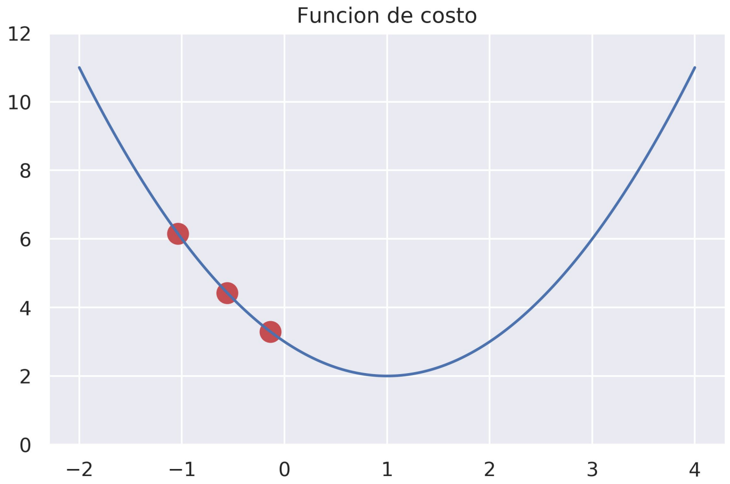


Pasos

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:

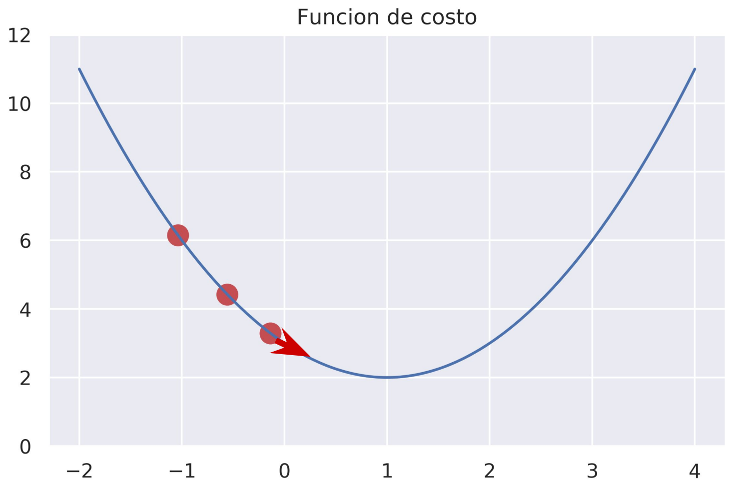


Pasos

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:



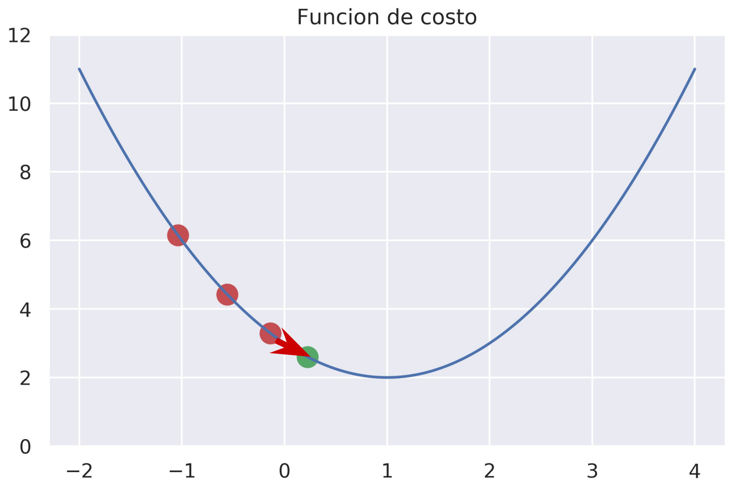
Pasos

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:

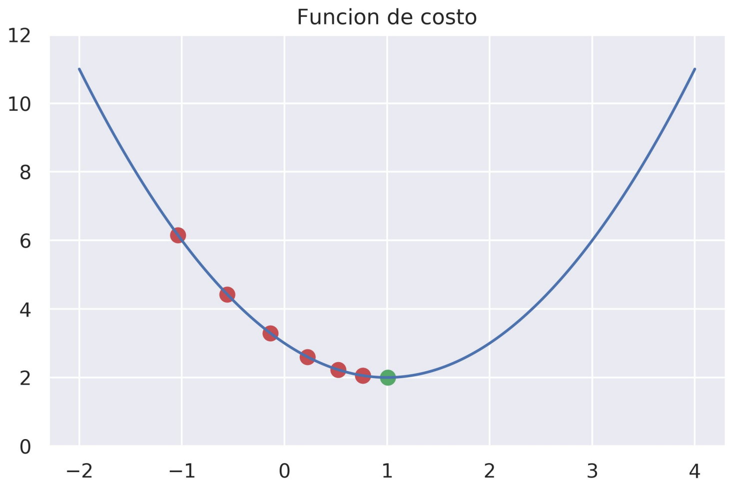


Pasos

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.

Descenso por gradiente

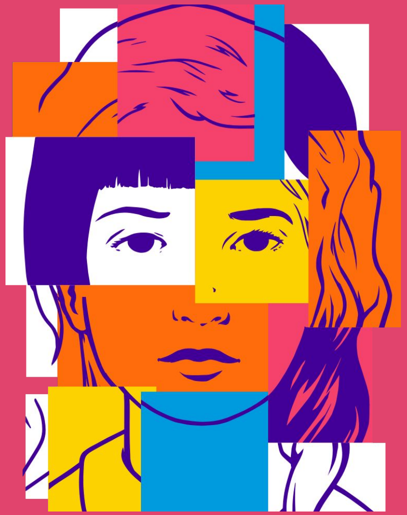
Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:



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.

Perceptrón

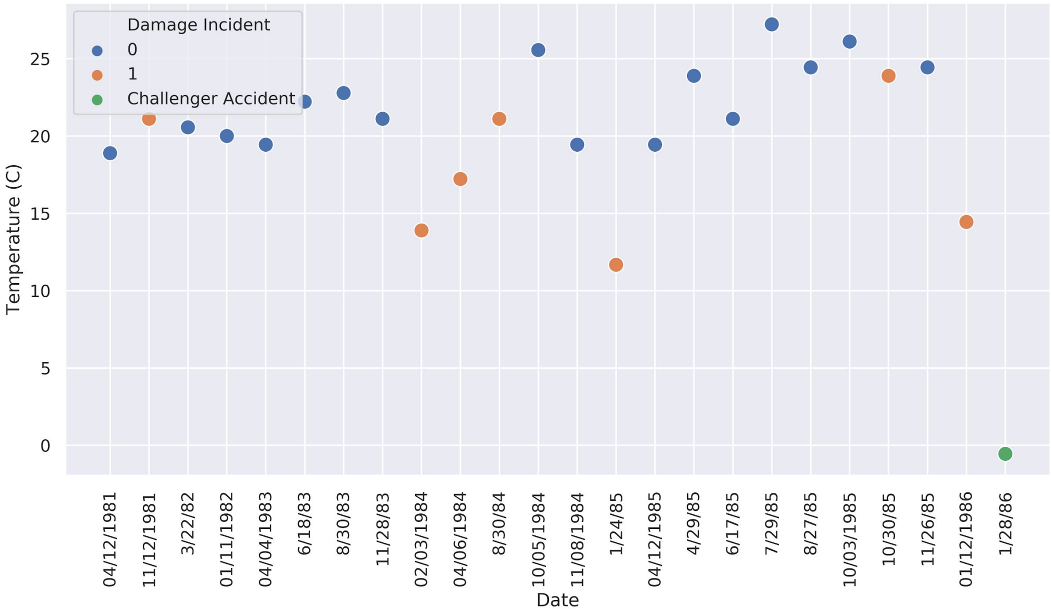


Objetivos de este encuentro

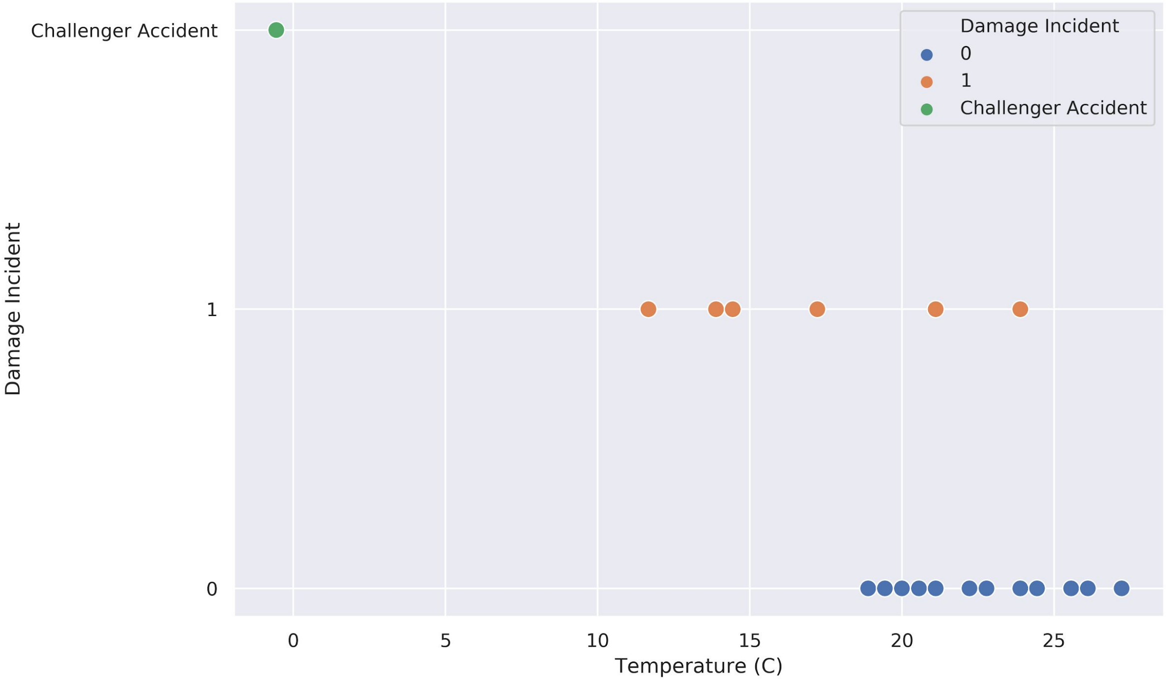
- Presentar el Perceptrón con un grado de detalle que permita entender lo que viene después. Para eso, vamos a trabajar sobre los conceptos más difíciles que suelen aparecer.
  - La idea es que entiendan “la cocina” del asunto - lo que hay detrás.
  - No se preocupen si se pierden en los detalles matemáticos.

- Presentar Keras como entorno de desarrollo para Redes Neuronales

Challenger

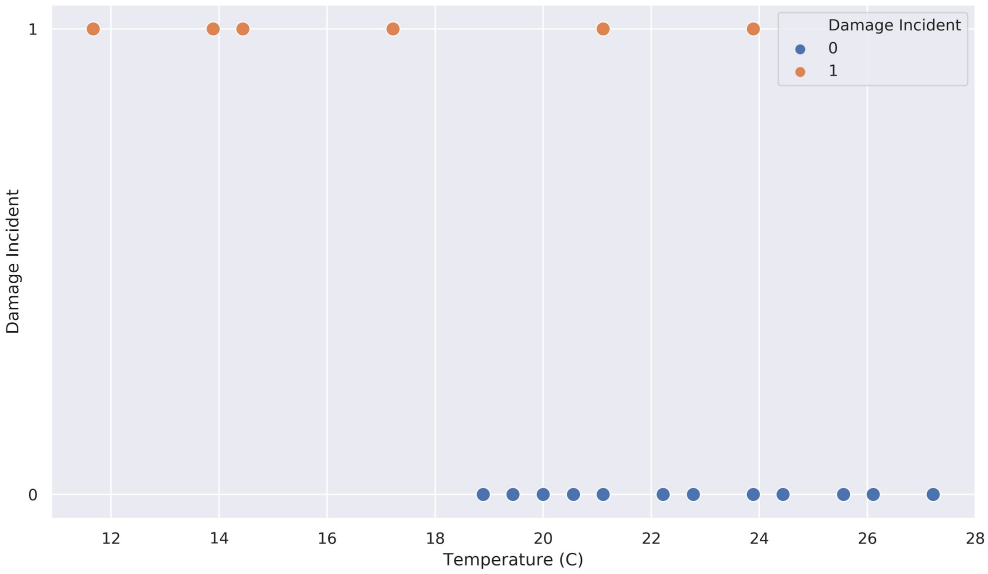

Damage Incident
0
1
Challenger Accident
Temperature (C)
25
20
15
10
5
0
04/12/1981
11/12/1981
3/22/82
01/11/1982
04/04/1983
6/18/83
8/30/83
11/28/83
02/03/1984
04/06/1984
8/30/84
10/05/1984
11/08/1984
1/24/85
04/12/1985
4/29/85
6/17/85
7/29/85
8/27/85
10/03/1985
10/30/85
11/26/85
01/12/1986
1/28/86
Date

Challenger

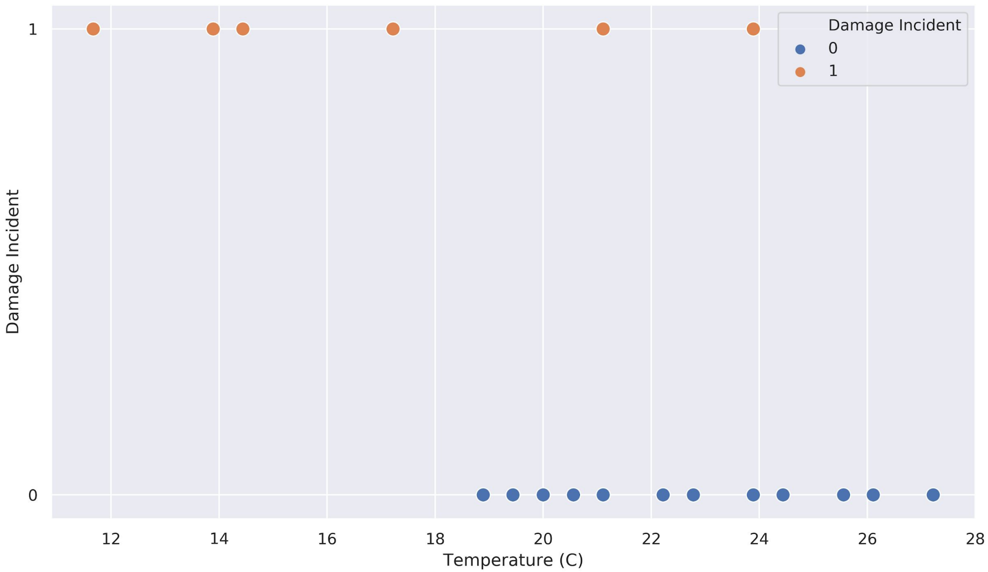


Challenger Accident
1
0
Damage Incident
0
5
10
15
20
25
Temperature (C)
Damage Incident
0
1
Challenger Accident

Challenger


Damage Incident
0
1
1
0
12
14
16
18
20
22
24
26
28
Temperature (C)
Damage Incident

Challenger

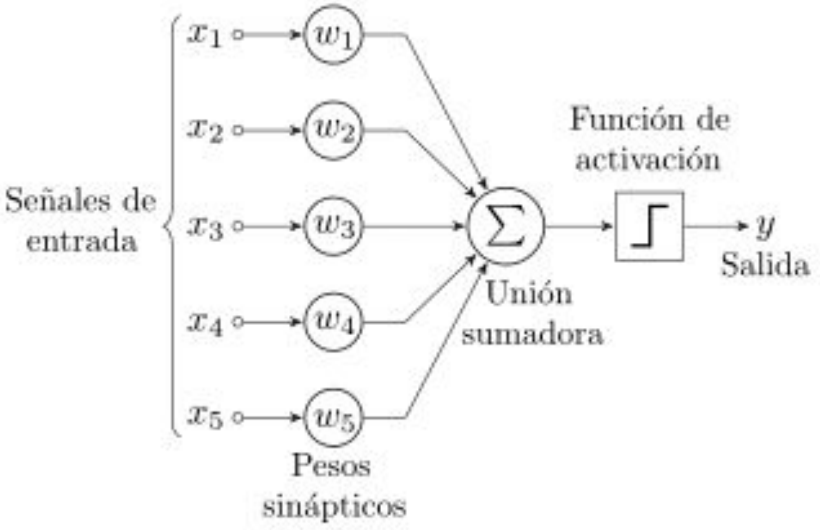


Estamos viendo realizaciones de un fenómeno probabilístico

Perceptrón

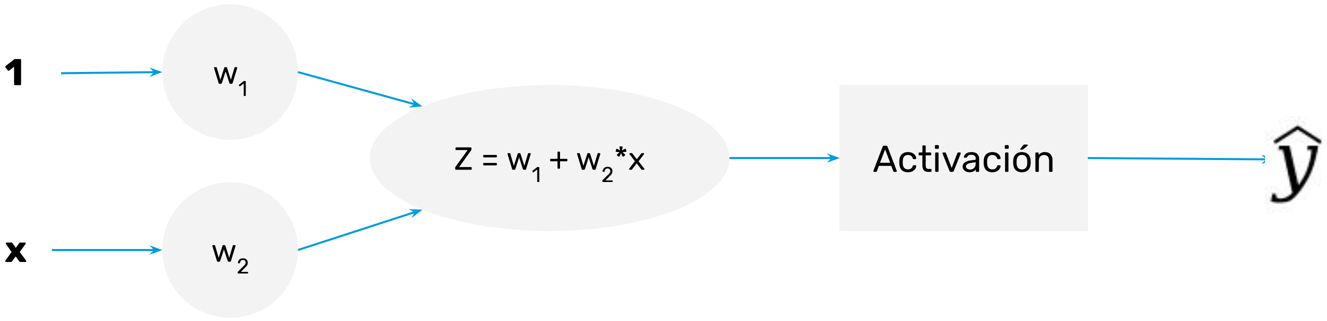
Perceptrón con una variable

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



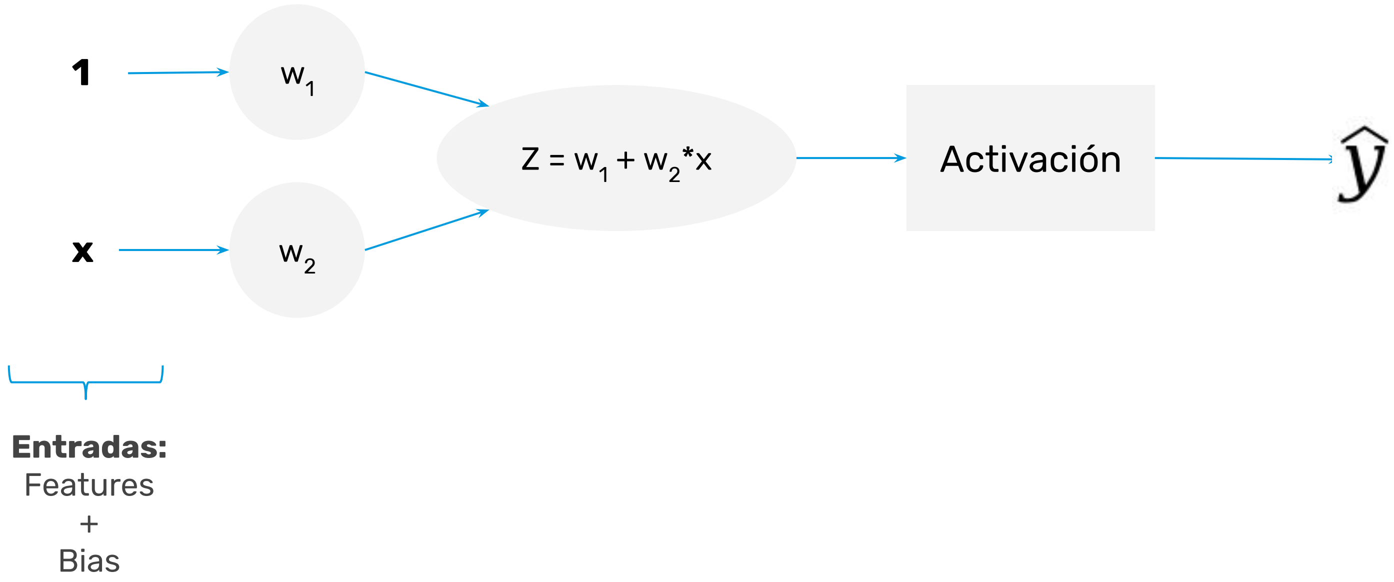
Perceptrón con una variable

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



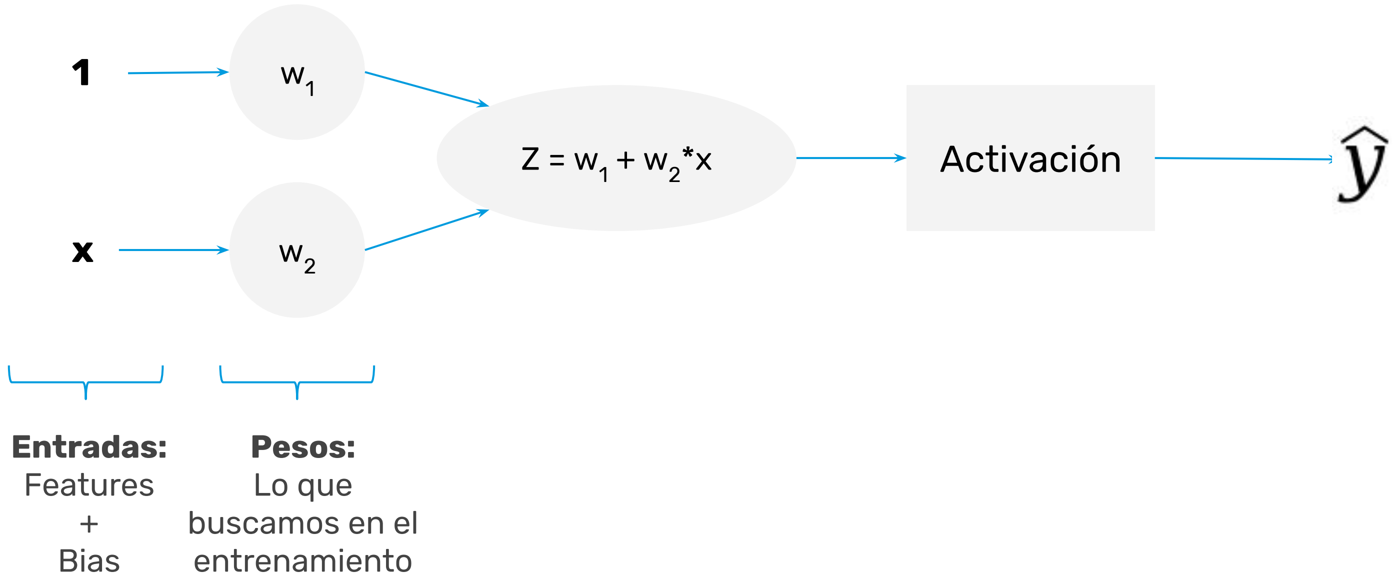
Perceptrón con una variable

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



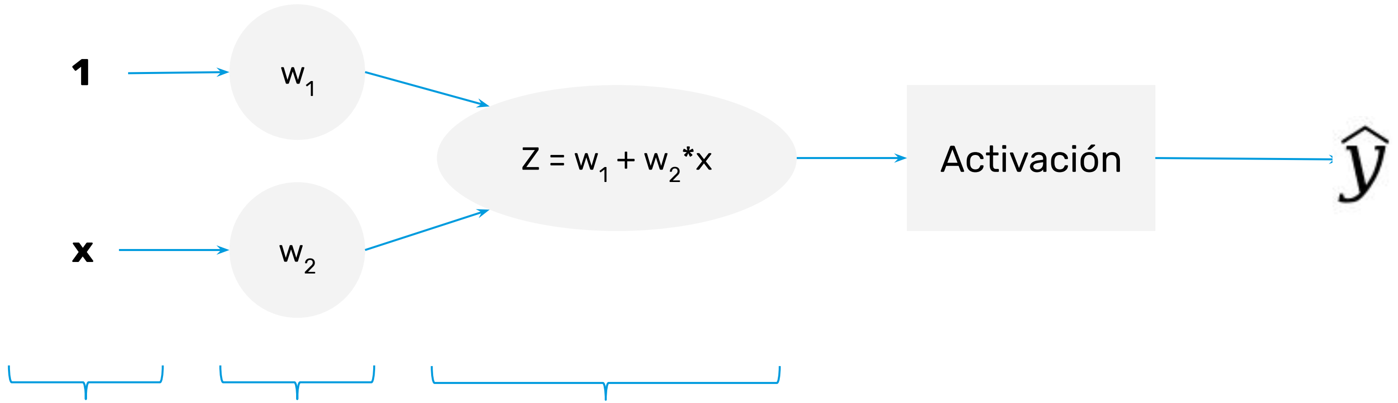
Perceptrón con una variable

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



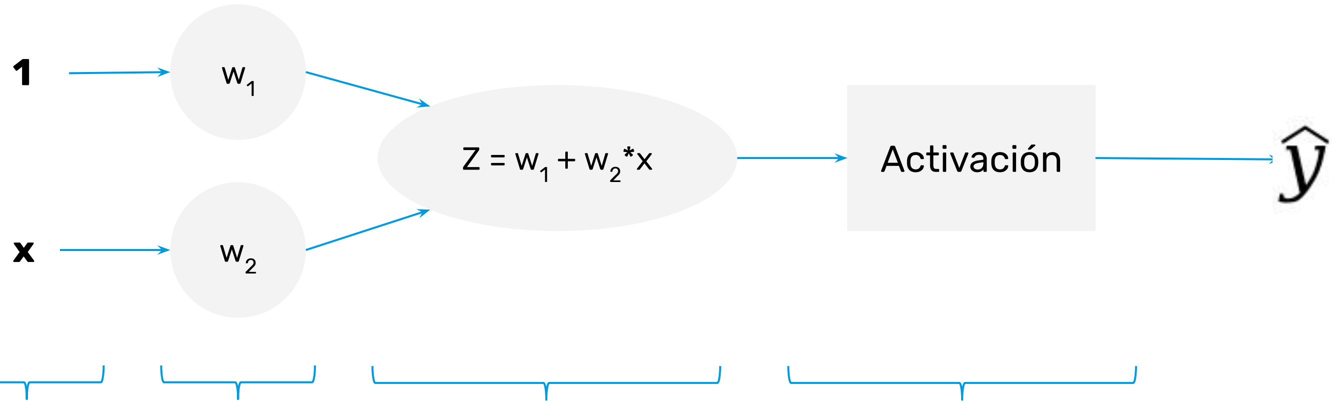
Entradas:
Features
+
Bias

Pesos:
Lo que
buscamos en el
entrenamiento

Unión sumadora:
combina los
features con los
pesos

Perceptrón con una variable

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



Entradas: Features + Bias

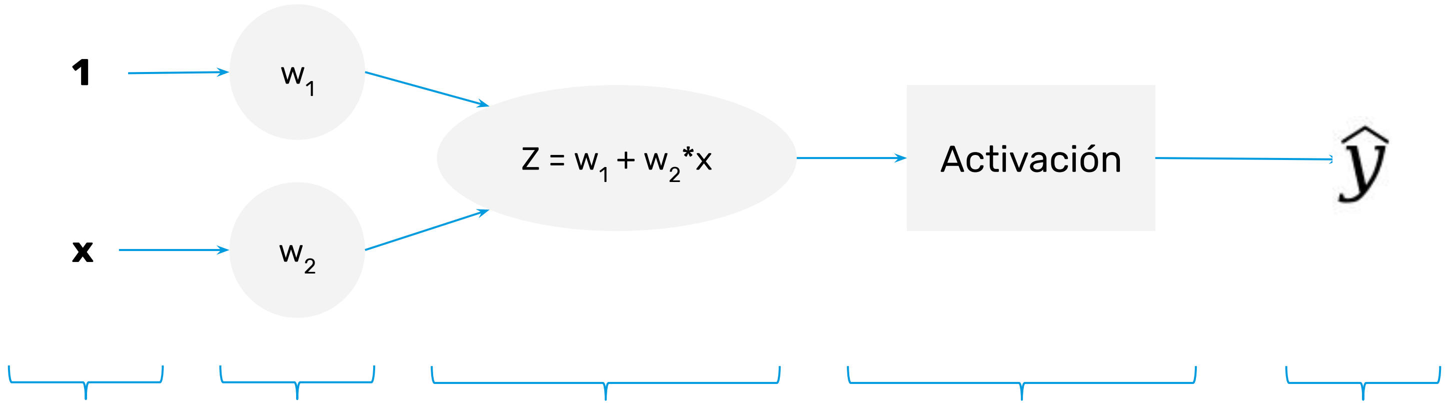
Pesos: Lo que buscamos en el entrenamiento

Unión sumadora: combina los features con los pesos

Activación: si no está, ¡es una función lineal!

Perceptrón con una variable

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



Entradas: Features + Bias

Pesos: Lo que buscamos en el entrenamiento

Unión sumadora: combina los features con los pesos

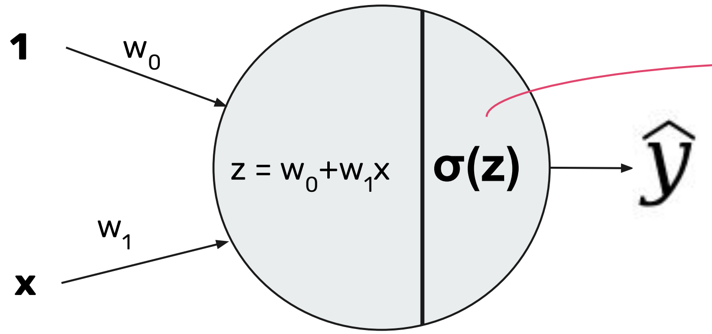
Activación: si no está, ¡es una función lineal!

Salida: entre 0 y 1, probabilidad del evento

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

Función logística sigmoide

Función Logística / Sigmoide

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

Función Logística / Sigmoide

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

$$z = w_0 + w_1 x$$

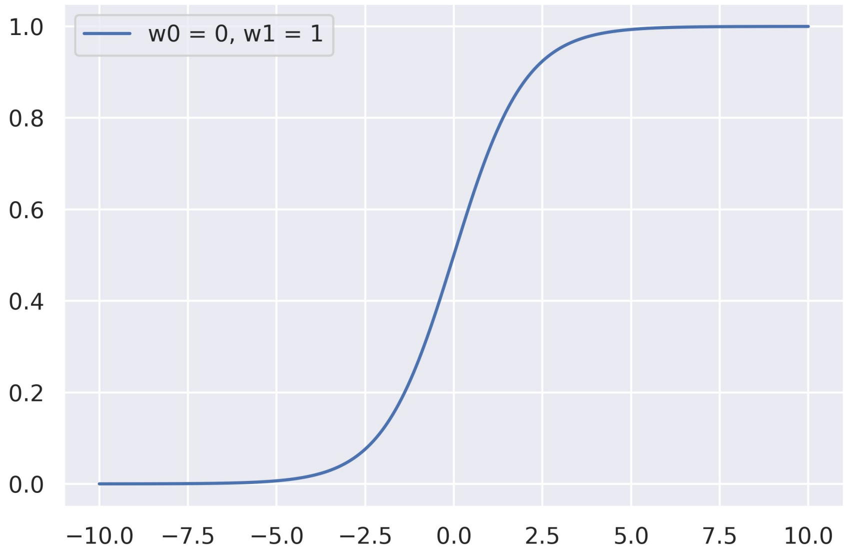
$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$

Función Logística / Sigmoide

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

$$z = w_0 + w_1 x$$

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

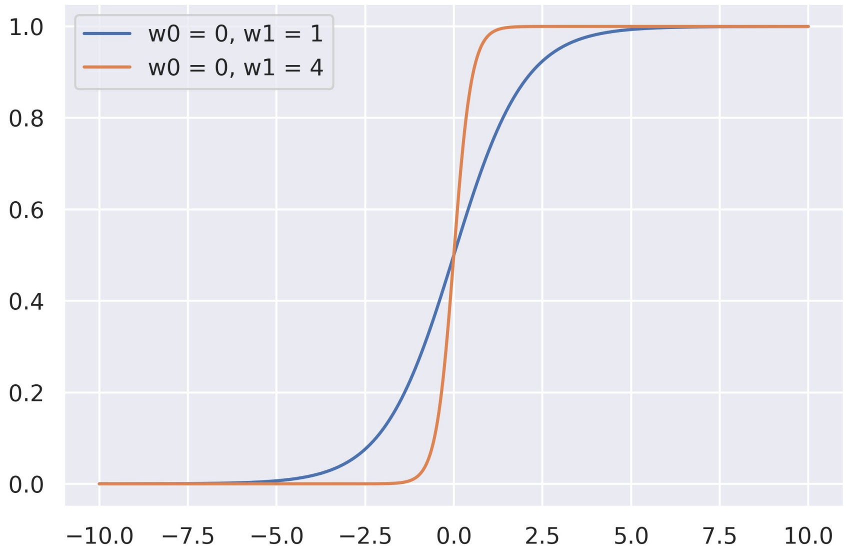


Función Logística / Sigmoide

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

$$z = w_0 + w_1 x$$

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

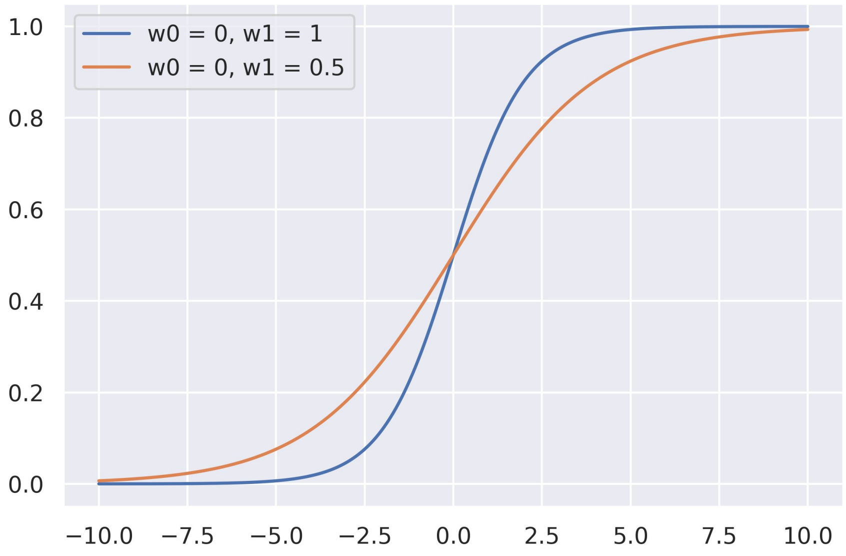


Función Logística / Sigmoide

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

$$z = w_0 + w_1 x$$

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

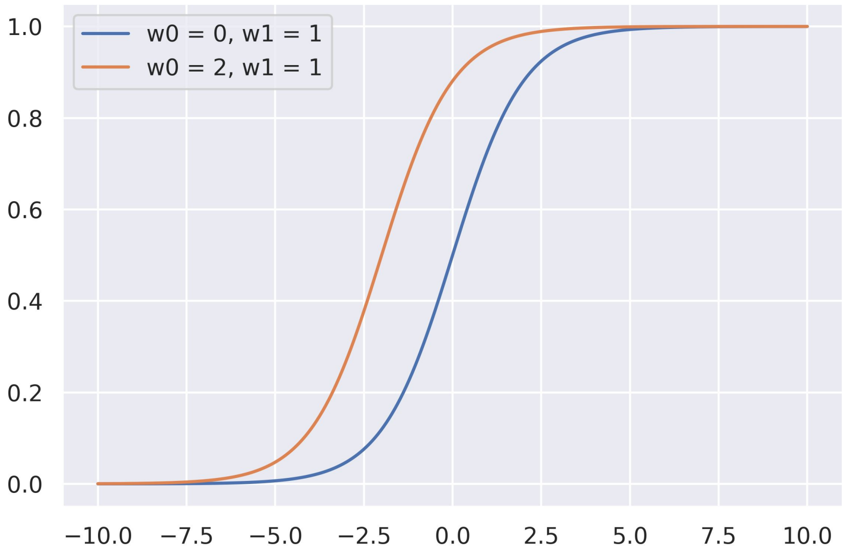


Función Logística / Sigmoide

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

$$z = w_0 + w_1 x$$

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

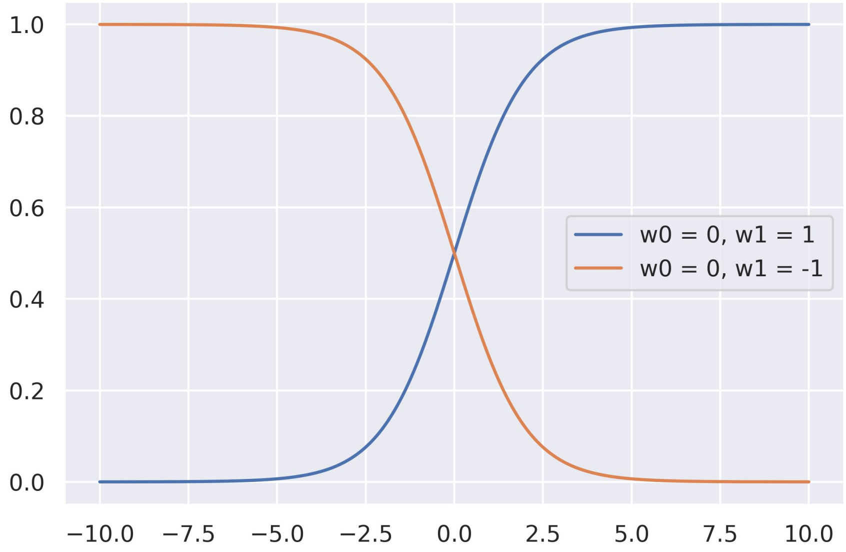


Función Logística / Sigmoide

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

$$z = w_0 + w_1 x$$

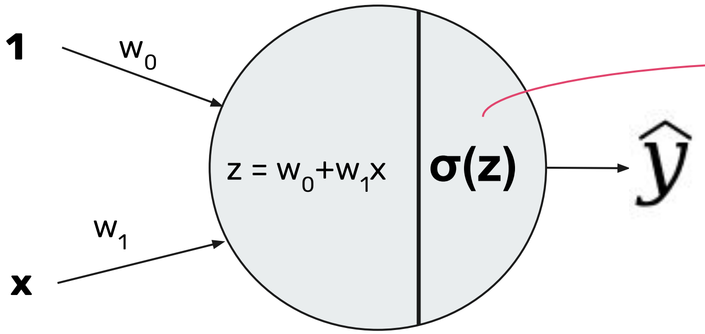
$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$



Perceptrón con una variable

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

Otra representación

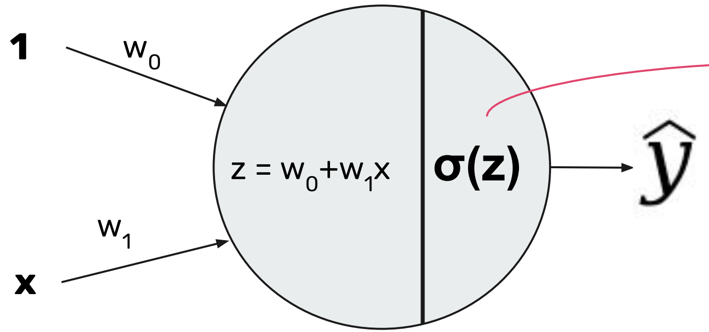


Activación: Función Sigmoidea

Perceptrón con una variable

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

Otra representación



Activación: Función Sigmoidea

¿Qué falta?

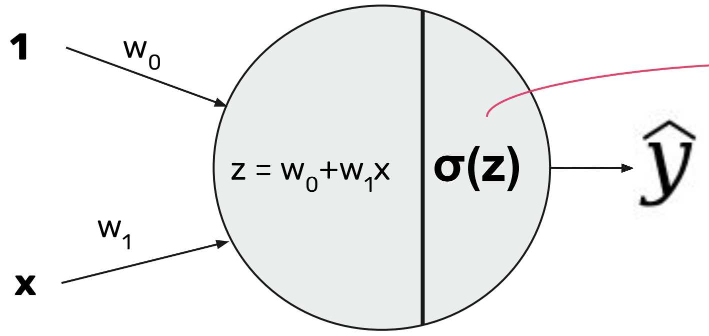
¡Falta encontrar los pesos w_0 y w_1 apropiados para nuestros datos!

Para eso necesitamos una **función de costo**

Perceptrón con una variable

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

Otra representación

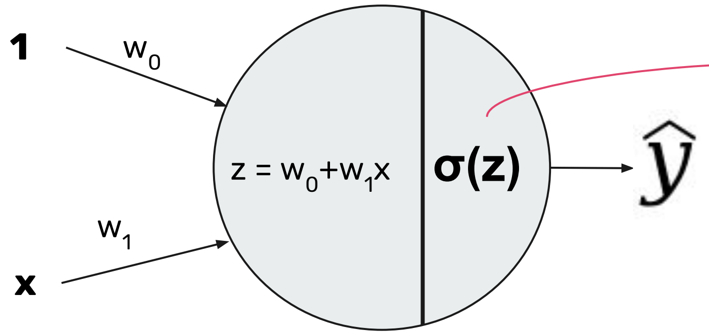


Activación: Función Sigmoidea

Perceptrón con una variable

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

Otra representación



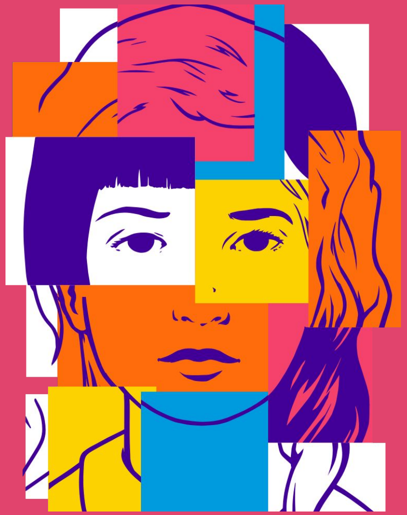
Activación: Función Sigmoidea

¿Qué falta?

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

Para eso necesitamos una **función de costo**

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

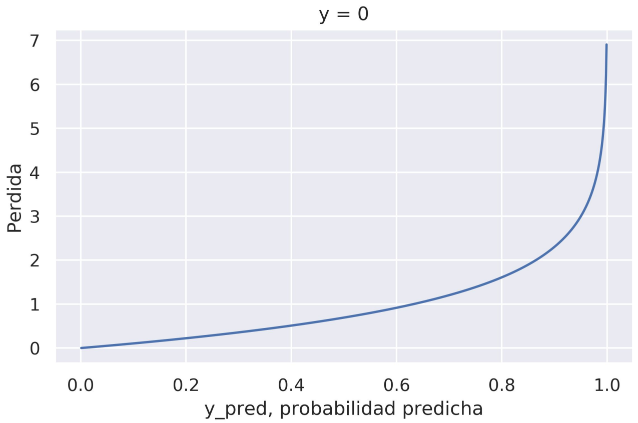
Entropía cruzada (cross-entropy)

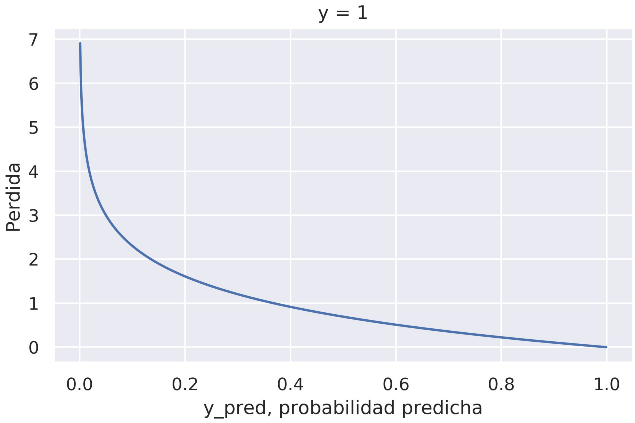
Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

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.

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

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

$$J(\overline{W}) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

Costo para todas las instancias

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

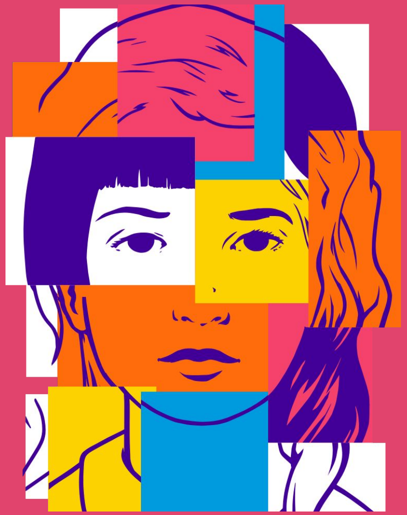
$$J(\overline{W}) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

Costo para todas las instancias

$$J(w_0,\ w_1) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

Costo para todas las instancias, caso 1D

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.

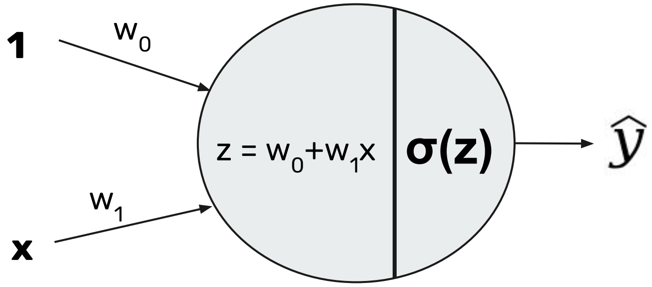
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros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

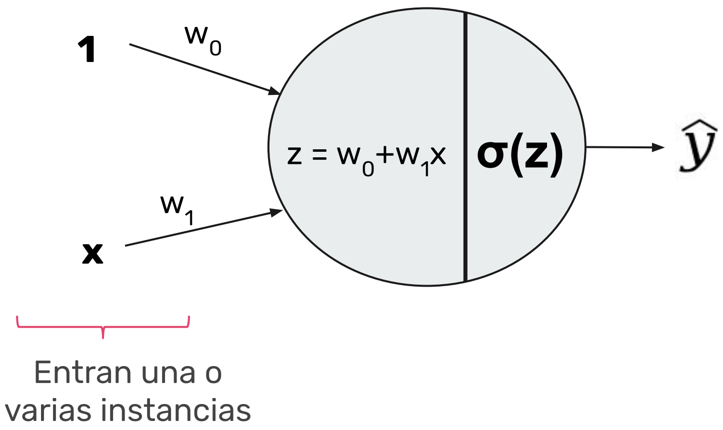
Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation.**



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.

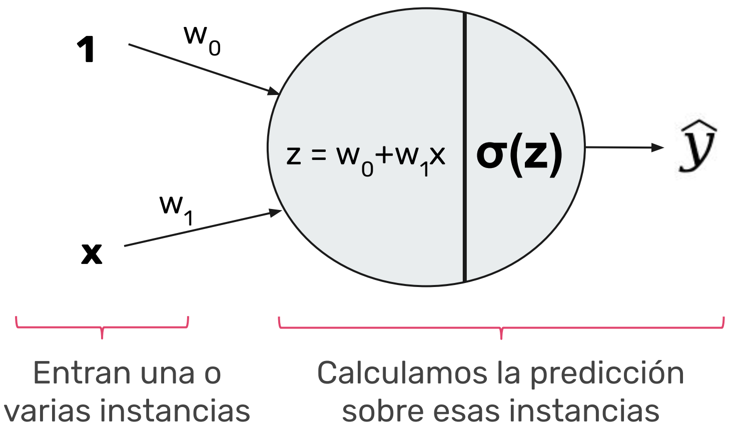
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

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



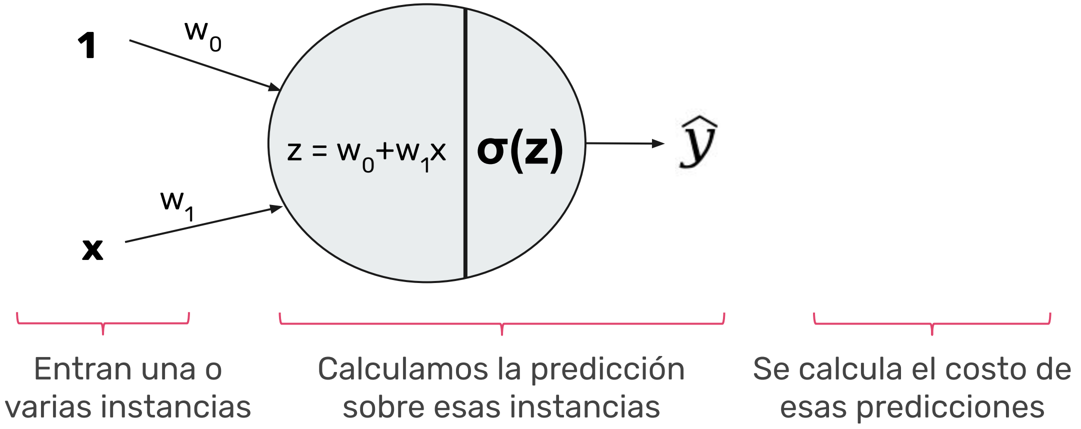
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros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

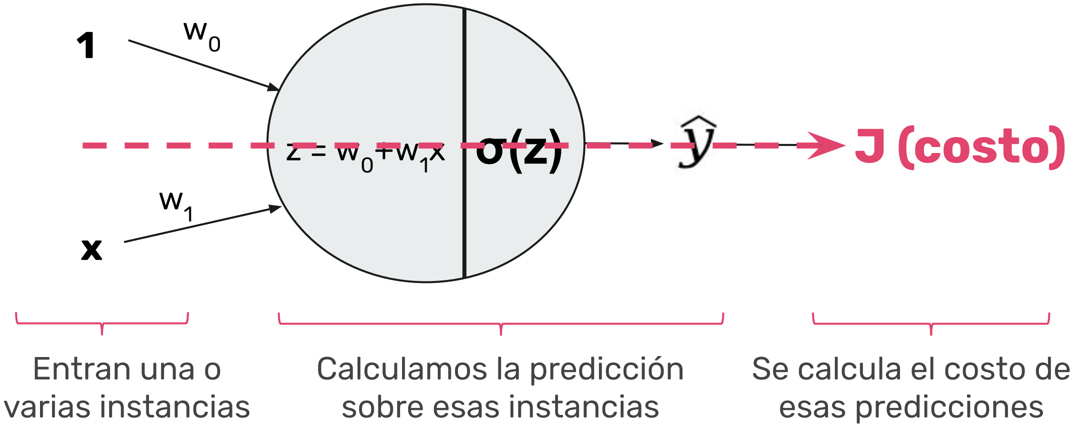
Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation.**



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.

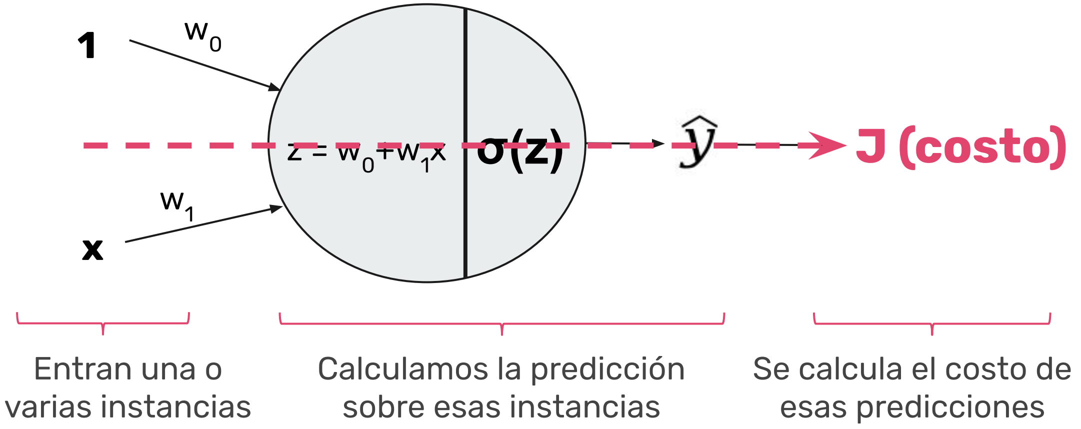
Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation.**



1. 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.**



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

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.

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

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.

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

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.

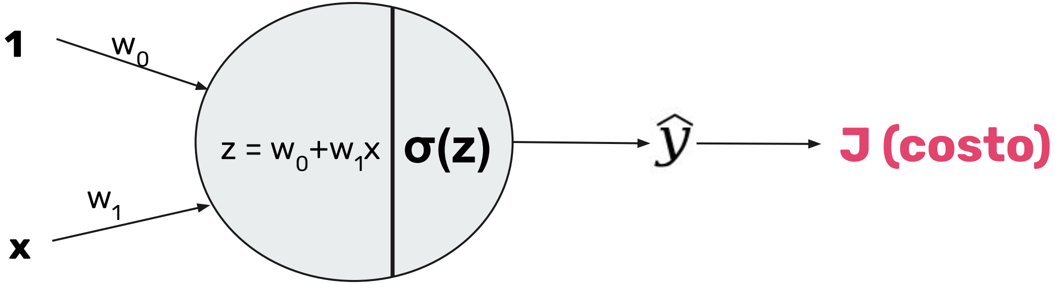
Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation.**

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.

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

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



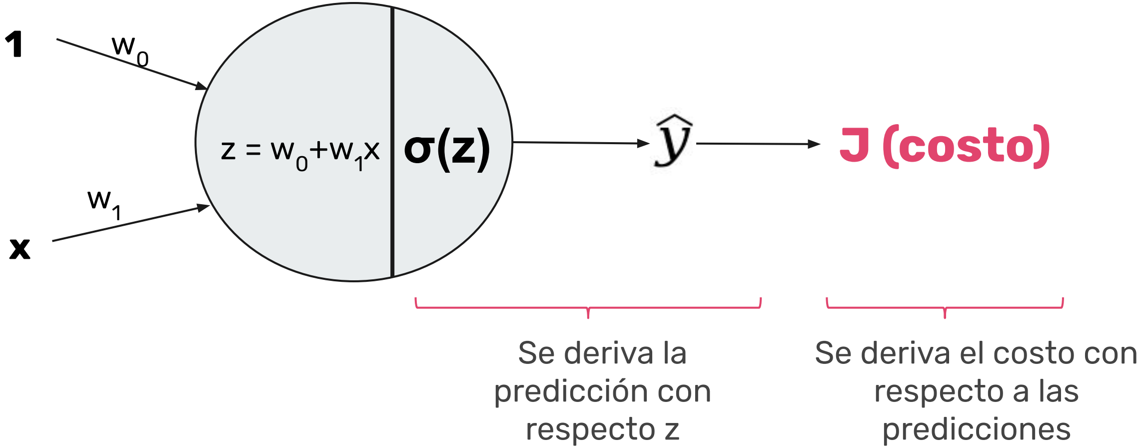
Se deriva el costo con respecto a las predicciones

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.

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

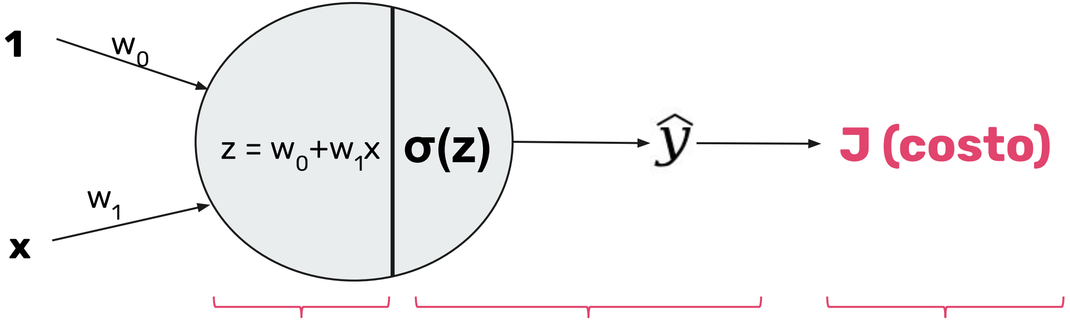
Calcular las derivadas y actualizar los parámetros “hacia atrás” se conoce como **Backpropagation.**



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

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.

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



Se deriva z con respecto a w_0 y w_1

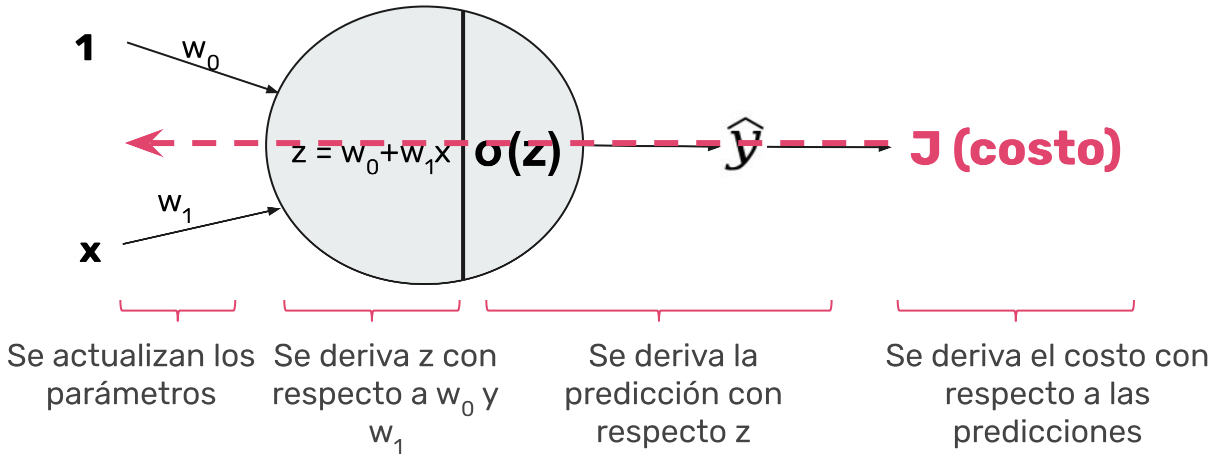
Se deriva la predicción con respecto z

Se deriva el costo con respecto a las predicciones

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

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.

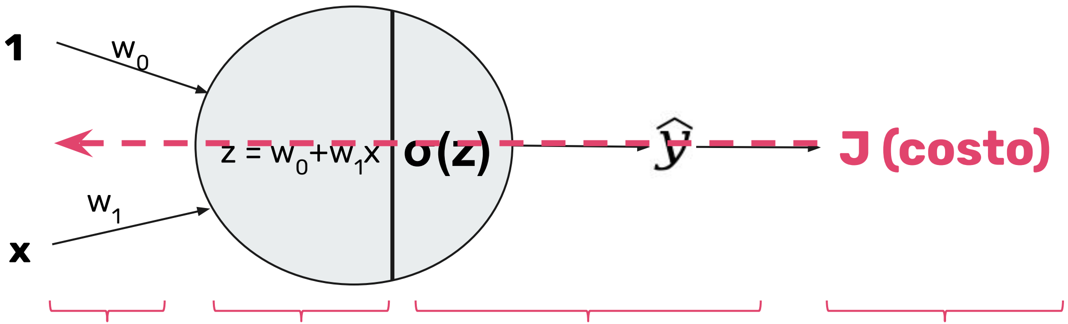
Calcular las derivadas y actualizar los parámetros "hacia atrás" se conoce como Backpropagation.



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

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.

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



Se actualizan los parámetros

Se deriva z con respecto a w_0 y w_1

Se deriva la predicción con respecto z

Se deriva el costo con respecto a las predicciones

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

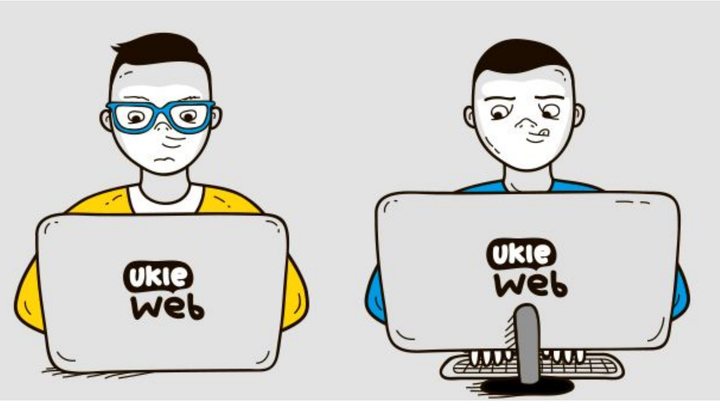
Ph. Credit: Drew Coffmann

Pair programming

(Programación en duplas)

Pair programming

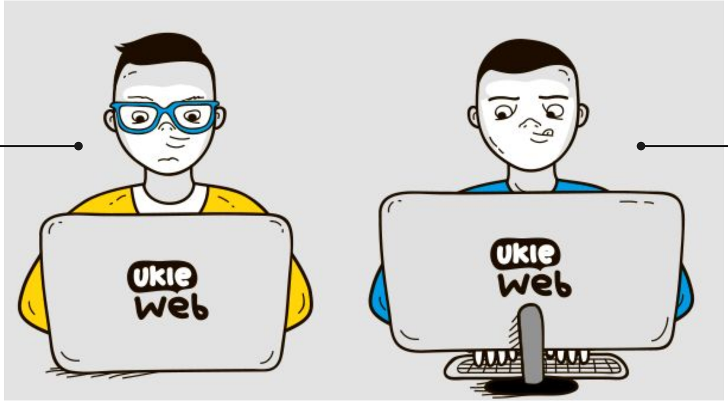
Es una técnica de desarrollo de software en la que dos personas trabajan en un solo bloque de código.



Pair programming

Es una técnica de desarrollo de software en la que dos personas trabajan en un solo bloque de código.

Navegador. Responsable de revisar y seguir el plan de acción.

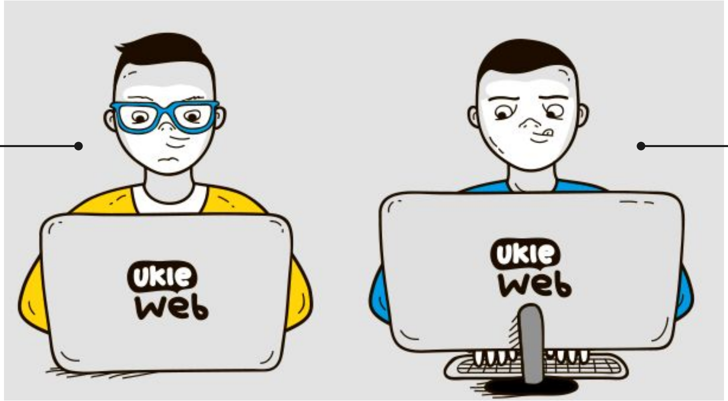


Controlador. Responsable de redactar el código.

Pair programming

¡Estos roles no son fijos!

Navegador. Responsable de revisar y seguir el plan de acción.



Controlador. Responsable de redactar el código.

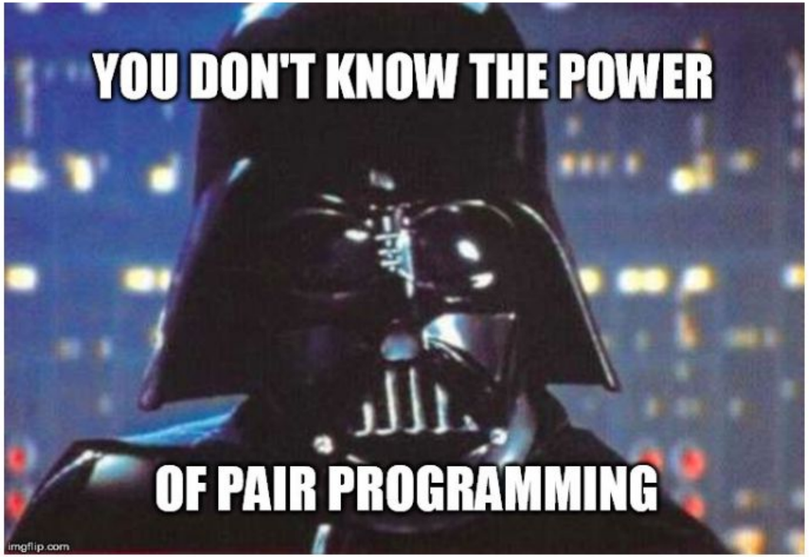
Pair programming • Ventajas

- Hacer foco
- Mejorar el flujo de trabajo
- Reducir la frustración
- Conocimiento compartido
- Mejores soluciones
- Colaboración y trabajo en equipo

Recursos

Pair Programming Guide

Weblab Technology Follow
Jan 25, 2018 · 8 min read



¡Comencemos!

Hands-on training

Hands-on training

DS_Encuentro_30_Perceptron.ipynb

Recursos

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

ACÁMICA