Introducción al Deep Learning

Dr. Ing. Gabriel Hermosilla Vigneau

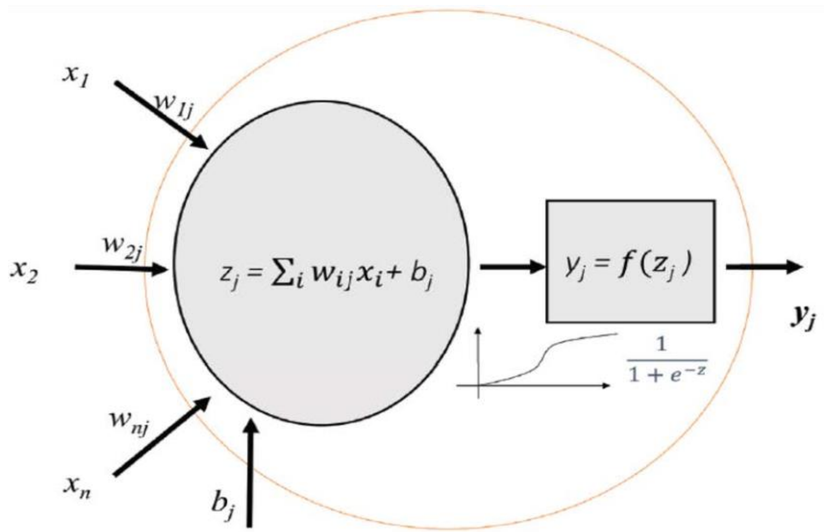
Redes Neuronales Artificiales
Parte 3

Proceso de aprendizaje

- Entrenar nuestra red neuronal, es decir, aprender los valores de nuestros parámetros (pesos wij y sesgos bj) es la parte más genuina de Deep Learning y podemos ver este proceso de aprendizaje en una red neuronal como un proceso iterativo de “ir y venir” por las capas de neuronas.

- El “ir” propagando hacia delante lo llamaremos forwardpropagation y el “venir” retropropagando información en la red lo llamamos **backpropagation**.

- La propagación hacia atrás o **backpropagation** es un algoritmo que funciona mediante la determinación de la pérdida (error) en la salida, propagándolo de nuevo hacia atrás en la red. De esta forma los pesos se van actualizando para minimizar el error resultante de cada neurona. Este algoritmo es lo que les permite a las redes neuronales aprender.

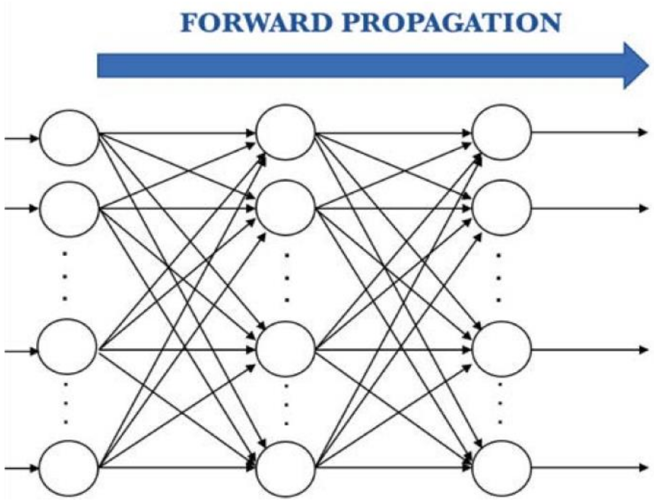


Proceso de aprendizaje: Forwardpropagation

- La primera fase forwardpropagation se da cuando se expone la red a los datos de entrenamiento y estos cruzan toda la red neuronal para ser calculadas sus predicciones (labels).

- Es decir, se pasan los datos de entrada a través de la red, de tal manera que todas las neuronas apliquen su transformación a la información que reciben de las neuronas de la capa anterior, y la envíen a las neuronas de la capa siguiente.

- Cuando los datos hayan cruzado todas las capas, y todas sus neuronas han realizado sus cálculos, se llegará a la capa final con un resultado de predicción de la etiqueta (label) para aquellos ejemplos de entrada.

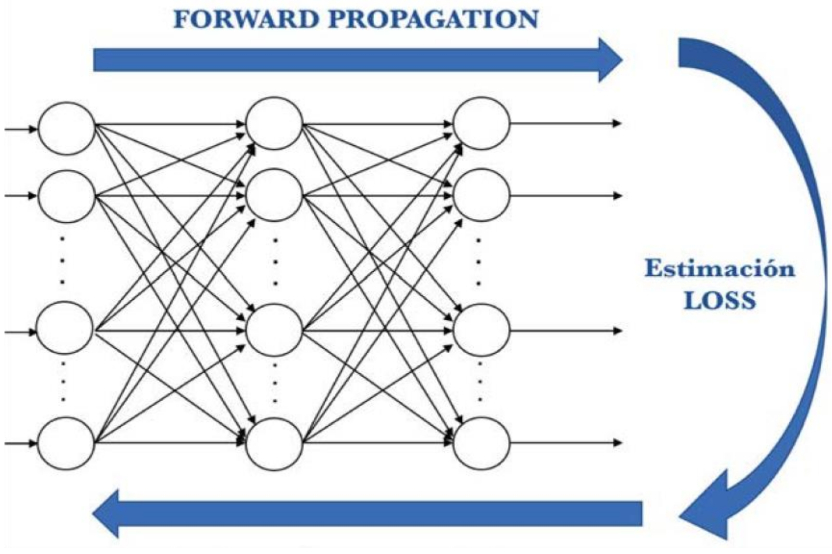


Proceso de aprendizaje: Loss function

- A continuación usaremos una función de pérdida (loss) para estimar la loss (o error) y para comparar y medir cuán bueno/malo fue nuestro resultado de la predicción en relación con el resultado correcto.

- Recordemos que estamos en un entorno de aprendizaje supervisado y disponemos de la etiqueta que nos indica el valor esperado.

- Idealmente, queremos que nuestro coste sea cero, es decir, sin divergencia entre valor estimado y el esperado. Por eso a medida que se entrena el modelo se irán ajustando los pesos de las interconexiones de las neuronas de manera automática hasta obtener buenas predicciones.

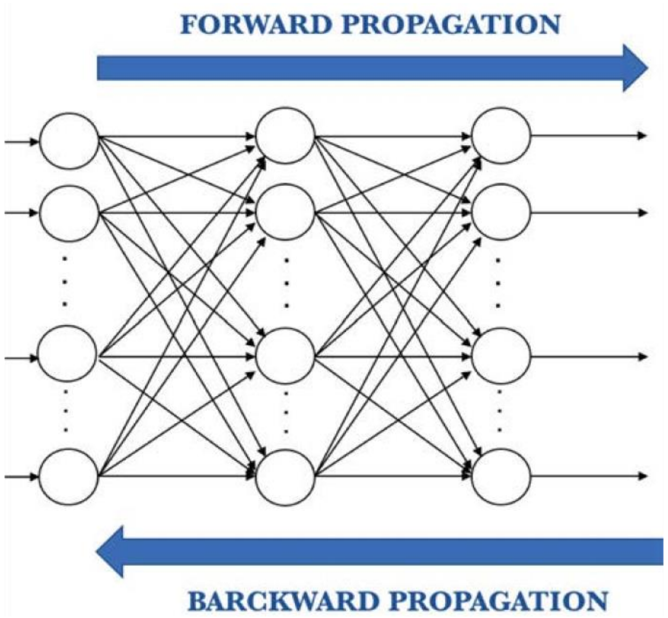


Proceso de aprendizaje: Backpropagation

- Partiendo de la capa de salida, esa información de loss se propaga hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida.

- Sin embargo, las neuronas de la capa oculta solo reciben una fracción de la señal total de la loss, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original.

- Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de loss que describa su contribución relativa a la loss total.

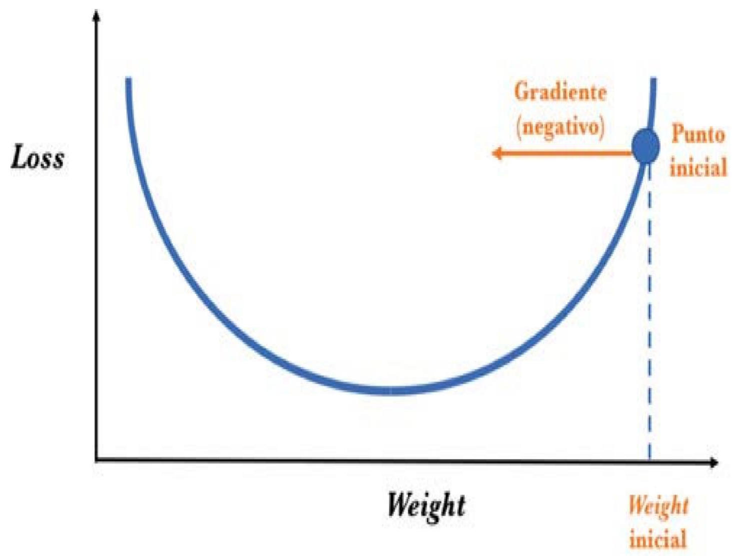


Ajustando los pesos: Gradient descent

- Ahora que ya hemos propagado hacia atrás esta información, podemos ajustar los pesos de las conexiones entre neuronas.

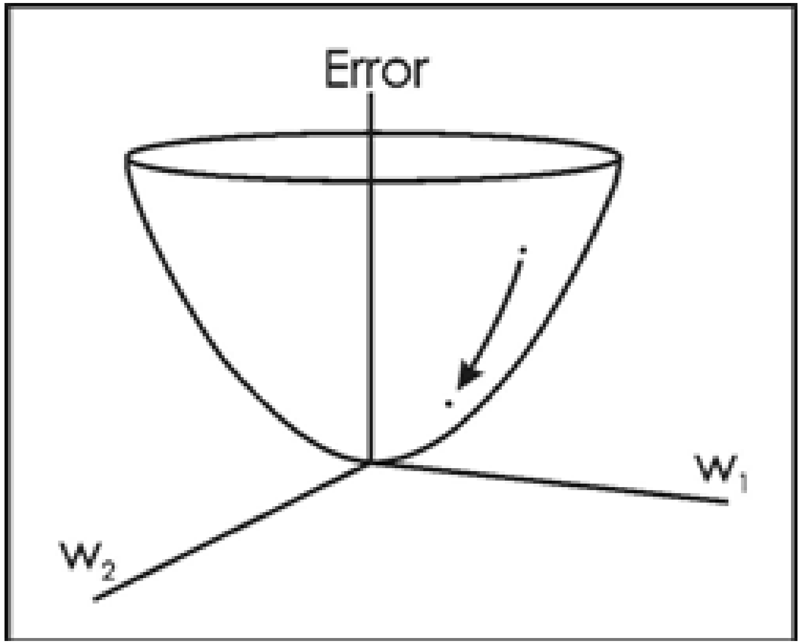
- El gradiente descendente va cambiando los pesos en pequeños incrementos con la ayuda del cálculo de la derivada (o gradiente) de la función de loss, que nos permite ver en qué dirección “descender” hacia el mínimo global.

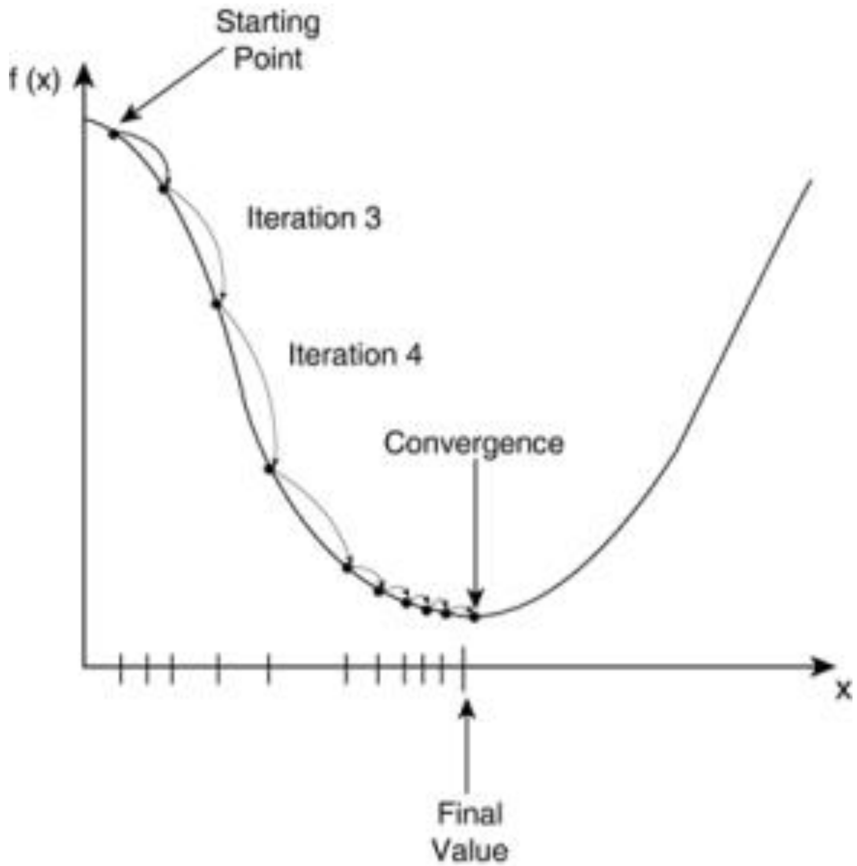
- Esto lo va haciendo en general en lotes de datos (batches) en las sucesivas iteraciones (epochs) del conjunto de todos los datos que le pasamos a la red en cada iteración.



Ajustando los pesos: Gradient descent

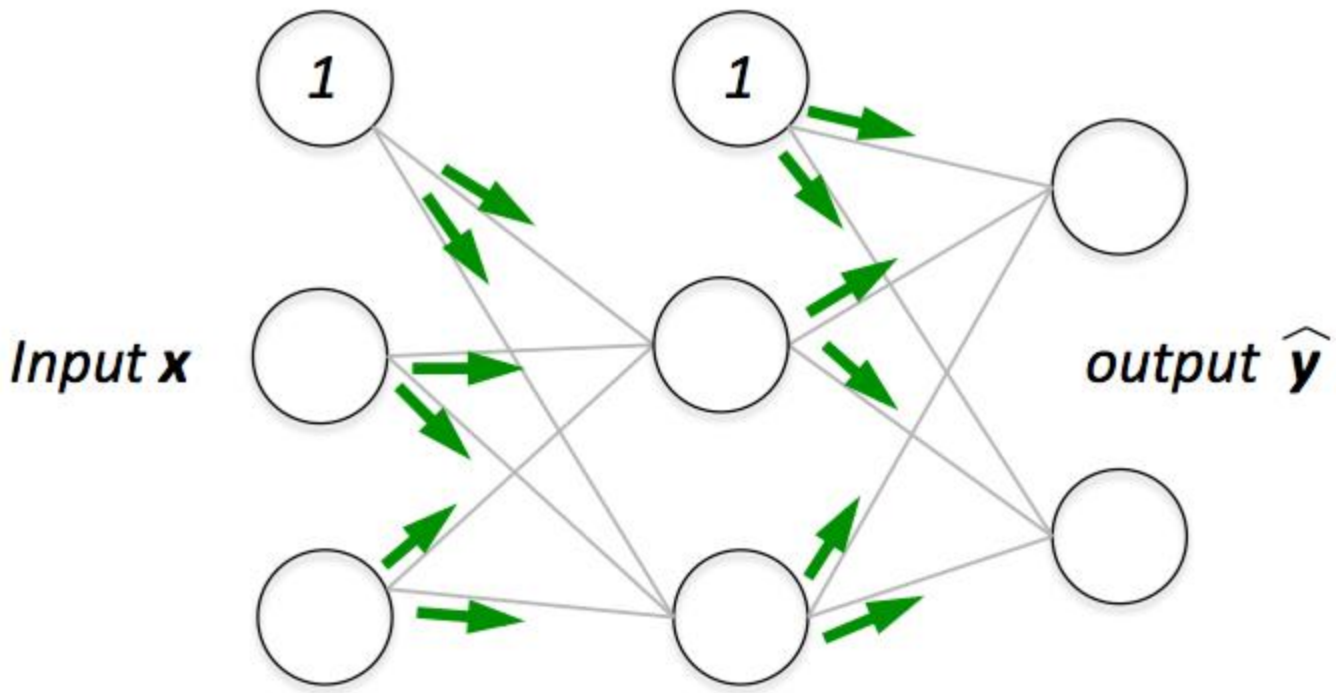
- Observe que a medida que avanzamos en la dirección del gradiente podemos obtener el valor de pesos óptimo, que minimizan la curva de error (loss).





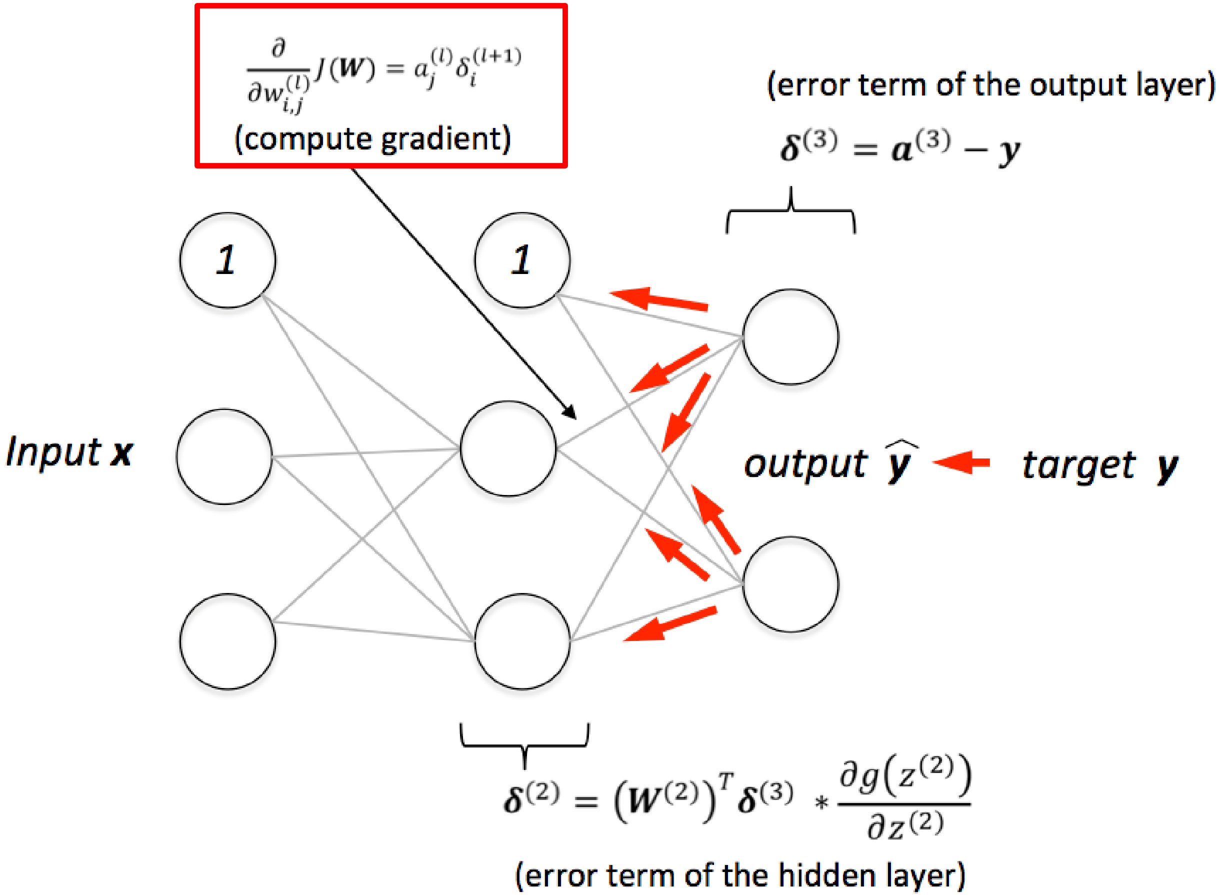
Ajustando los pesos: Backpropagation

- Etapa 1: Forwardpropagation



Ajustando los pesos: Backpropagation

- Etapa 2: Backpropagation



Ajustando los pesos:

Resumen

1. Empezar con unos valores (a menudo aleatorios) para los parámetros de la red (pesos wij y sesgos bj)
2. Tomar un conjunto de ejemplos de datos de entrada y pasarlos por la red para obtener su predicción.
3. Comparar estas predicciones obtenidas con los valores de etiquetas esperadas y con ellas calcular la loss.
4. Realizar el **backpropagation** para propagar esta **loss** a todos y cada uno de los parámetros que conforman el modelo de la red neuronal.
5. Usar esta información propagada para actualizar con el **gradient descent** los parámetros de la red neuronal de manera que reduzca la **loss** total y obtener un mejor modelo.
6. Continuar iterando en los anteriores pasos hasta que consideremos que tenemos un buen modelo (más adelante veremos cuando debemos parar).

Funciones objetivo: Loss function

- Una función objetivo, comúnmente llamada loss o función de pérdida es usada por un optimizador que se encarga de navegar en el espacio de los pesos de la red neuronal para realizar un proceso de minimización de la pérdida.

- Algunas funciones objetivo se presentan a continuación:

- **MSE**: Este es el error cuadrático medio entre las predicciones y los valores verdaderos. Matemáticamente, si Υ es un vector de n predicciones, e Y es el vector de n valores observados, entonces satisfacen la siguiente ecuación:

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(\Upsilon - Y)^2$$

- Estas funciones objetivos promedian todos los errores cometidos para cada predicción, y si la predicción está lejos del valor real, entonces esta distancia se hace más evidente.

Funciones objetivo: Loss function

- Entropía cruzada binaria (Binary Cross-entropy): esta es la pérdida logarítmica binaria. Supongamos que nuestro modelo predice p mientras que el objetivo es t, entonces la entropía cruzada binaria se define de la siguiente manera:

$$-t\log(p) - (1-t)log(1-p)$$

- Esta función objetivo es utilizada para predicciones con etiquetas binarias (2 clases).

Funciones objetivo: Loss function

- Entropía cruzada categórica (Categorical Cross-entropy): esta es la pérdida logarítmica multiclase. Si el objetivo es $t_{i,j}$ y la predicción es $p_{i,j}$, entonces la entropía cruzada categórica es la siguiente:

$$L_i = -\Sigma_j t_{i,j} \log(p_{i,j})$$

- Esta función objetivo es adecuada para predicciones de etiquetas multiclase (>2 clases). También es la opción predeterminada en asociación con la activación de softmax.

Optimización de la Loss function

- Queremos que la red alcance la menor pérdida, para esto se debe realizar un proceso de optimización.

- El proceso de optimización, minimizará la función de costo utilizando el valor que predice la red y la etiqueta real. Mediante el gradiente descendiente podemos encontrar el óptimo.

- Considere θ como los pesos de la red, se desea obtener el valor de θ óptimo.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$$

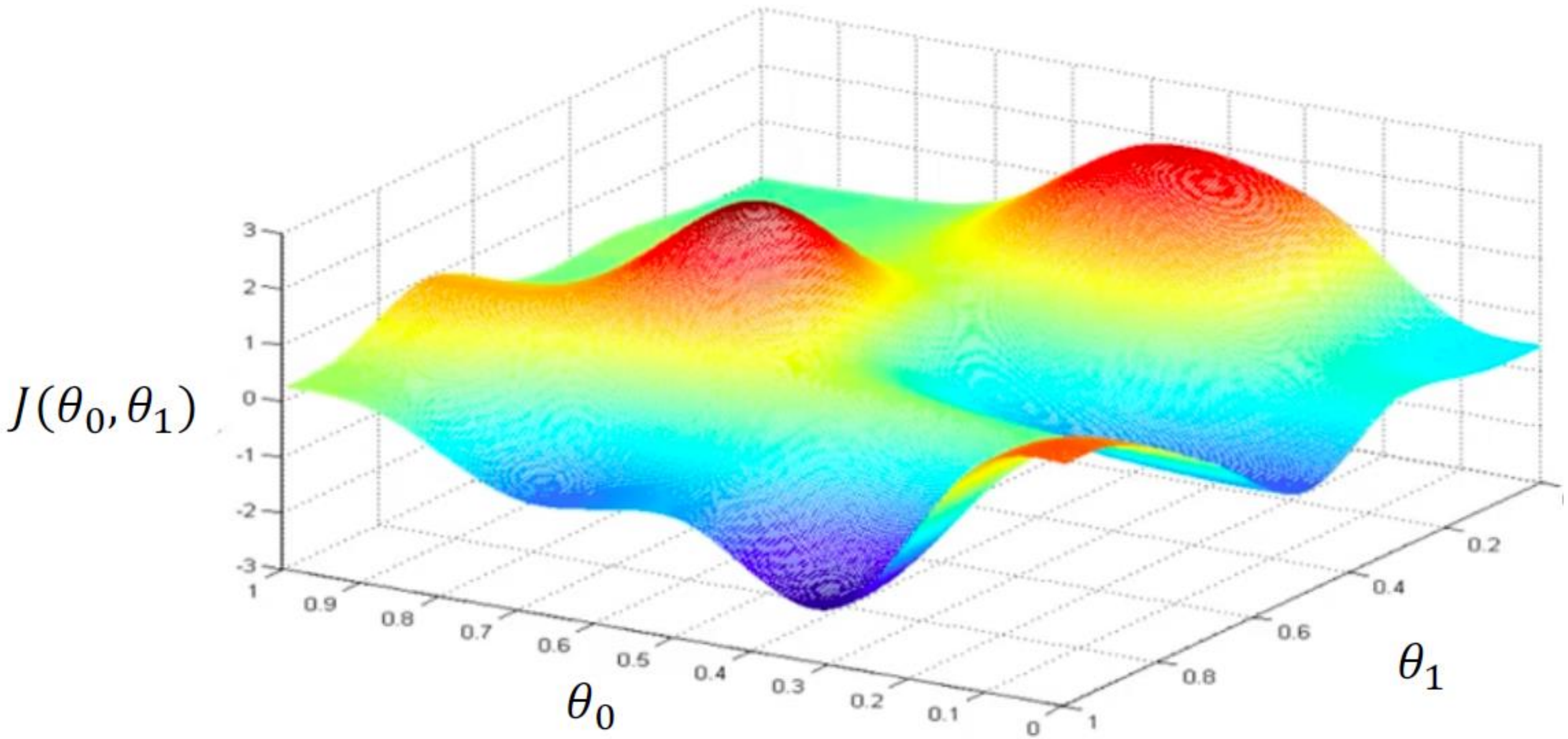
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} = \left\{\theta^{(0)}, \theta^{(1)}, \cdots\right\}$$

Optimización de la Loss function

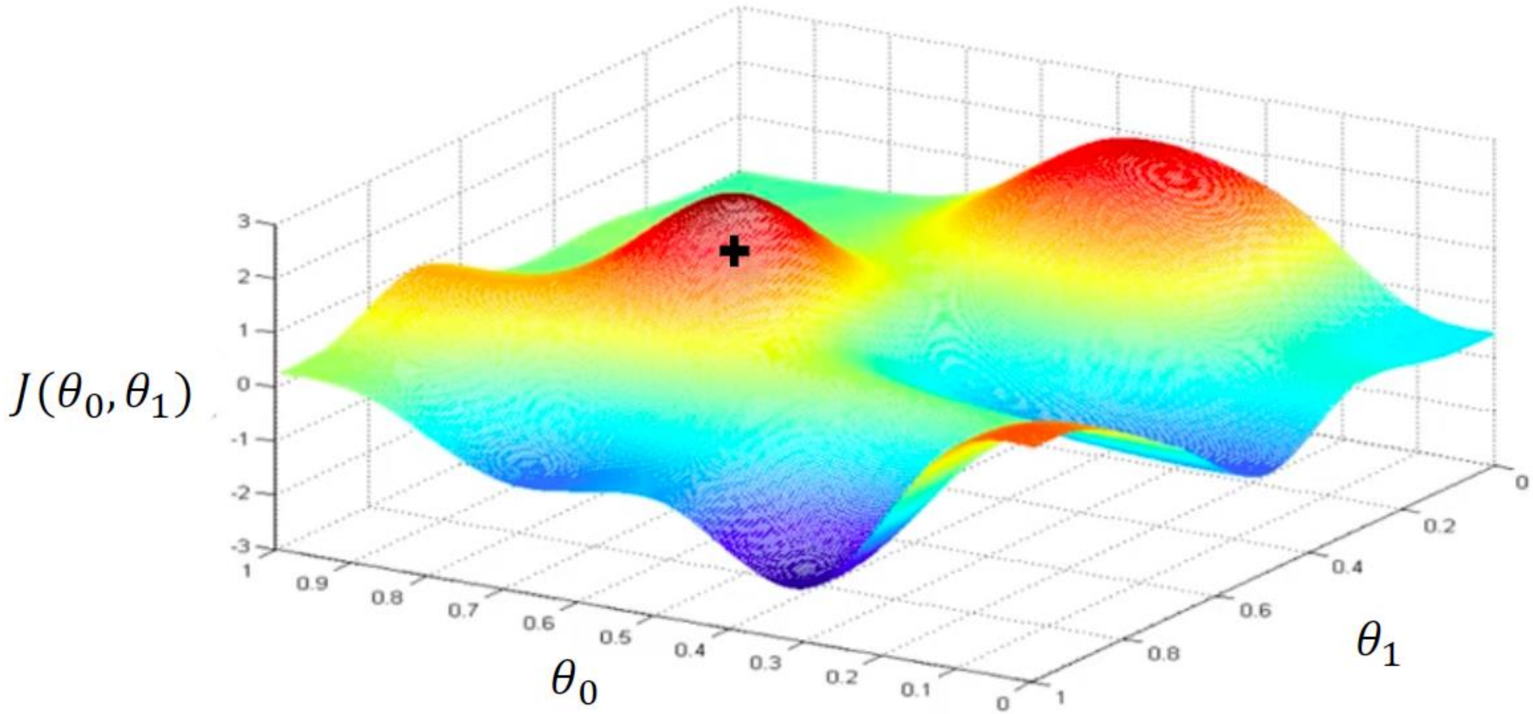
- Supongamos que la forma de nuestra función de pérdida a minimizar es como la que muestra la figura siguiente:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} J(\boldsymbol{\theta})$$



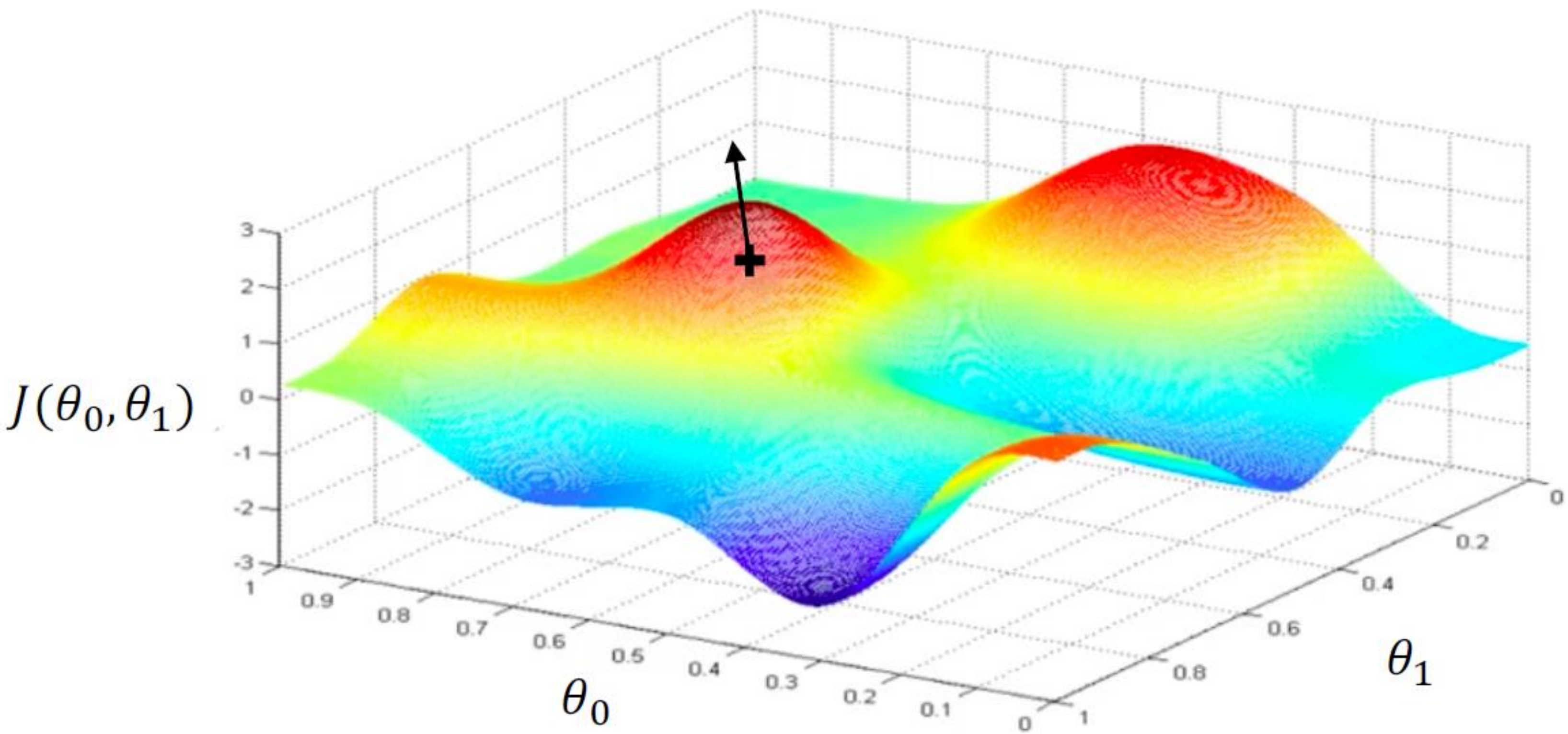
Optimización de la Loss function

- Escogemos un valor inicial al azar (θ_0, θ_1)



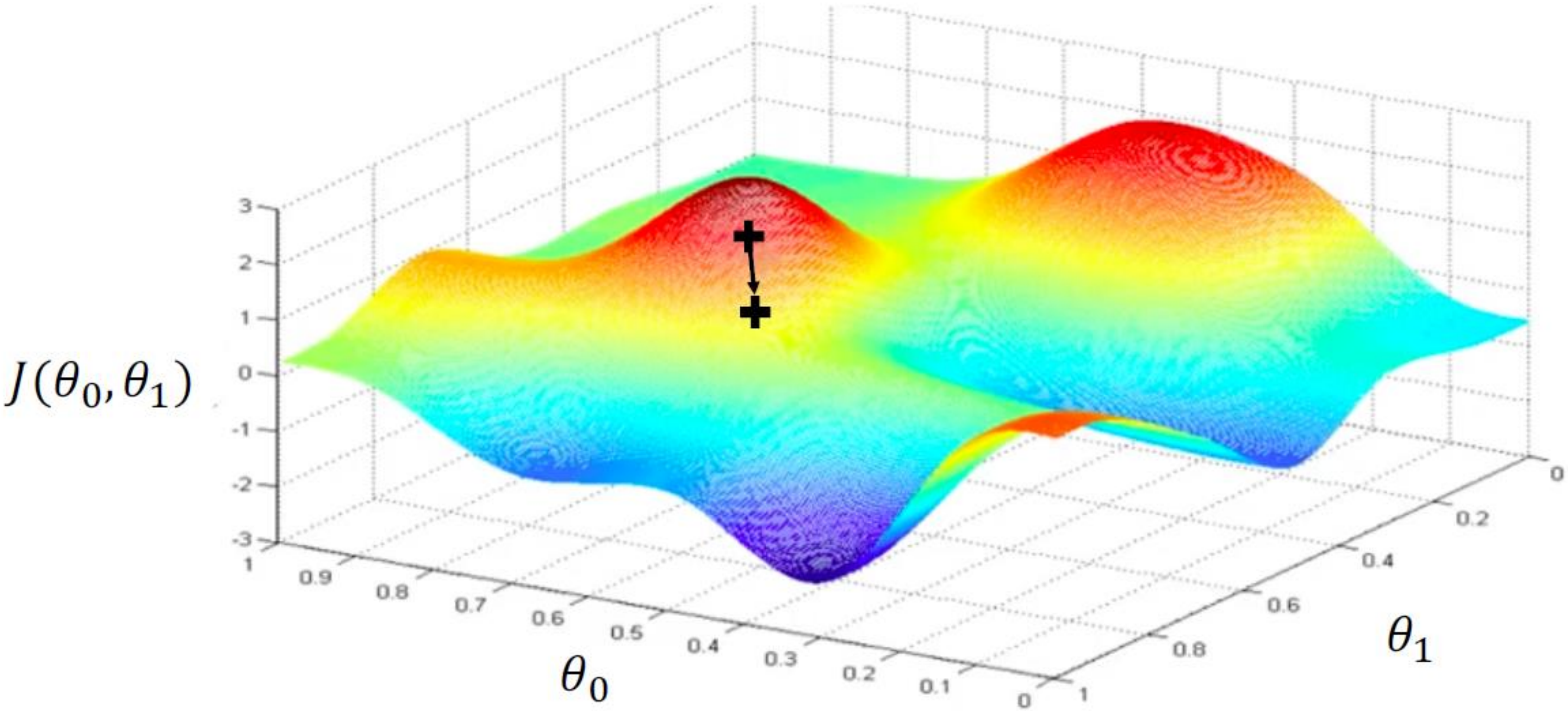
Optimización de la Loss function

- Se calcula el gradiente $\frac{\partial J(\theta)}{\partial \theta}$



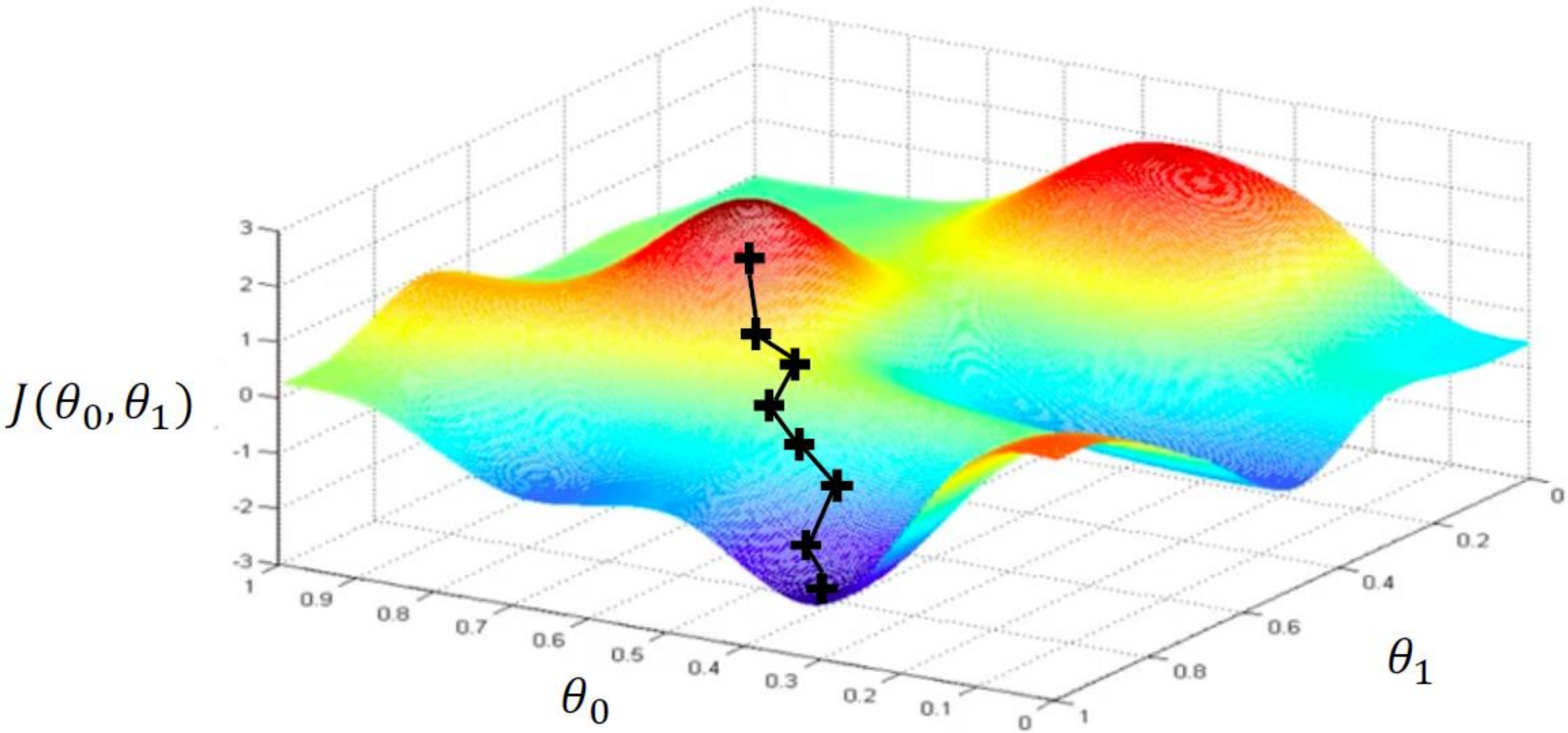
Optimización de la Loss function

- Tomar un pequeño paso en la dirección contraria al gradiente



Optimización de la Loss function

- Se repite hasta que haya convergencia



Gradiente Descendiente

- Algoritmo

1. Inicializar los pesos aleatoriamente $\sim N(0, \sigma^2)$

2. Loop hasta que exista convergencia

3. Calcular el gradiente $\frac{\partial J(\theta)}{\partial \theta}$

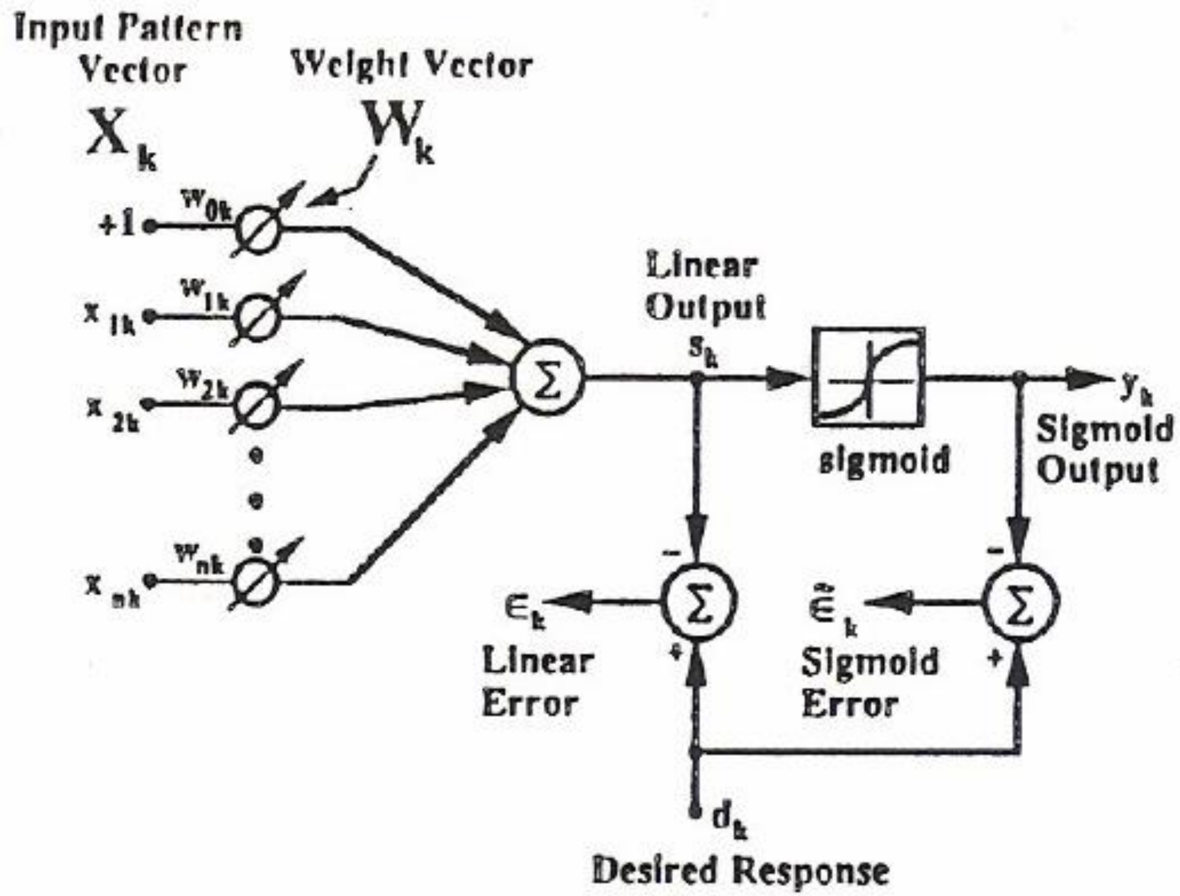
4. Actualizar los pesos $\theta \leftarrow \theta - \boldsymbol{\eta} \frac{\partial J(\theta)}{\partial \theta}$

5. Retornar el valor de pesos

Aplicación del Algoritmo de Backpropagation con MSE

- Consideremos la siguiente red neuronal:

  - Posee entradas X, pesos W, una salida lineal (s_k), una salida no lineal (y_k) y las etiquetas (d_k)
  - Además, se especifica el error lineal a la salida del sumador y el error no lineal de la sigmoide (smg).



Aplicación del Algoritmo de Backpropagation con MSE

- Definamos matemáticamente la salida lineal y no lineal de la siguiente forma:

$$s_k = X_k^T W_k \quad \text{salida lineal}$$

$$y_k = sgm(s_k) \quad \text{salida no-lineal}$$

- Usemos una función de activación sigmoidal, con salida entre 0 y 1

$$sgm(s_k) = \frac{1}{1+e^{-s_k}}, \qquad 0 \leq y_k \leq 1$$

- Sin embargo, también es posible usar una función de activación tangente hiperbólica, con salida entre -1 y 1

$$sgm(s_k) = \tanh(s_k) = \frac{1-e^{-2s_k}}{1+e^{-2s_k}} \quad -1 \leq y_k \leq 1$$

Aplicación del Algoritmo de Backpropagation con MSE

- Como vamos a minimizar el error cuadrático medio (MSE), usamos su función Loss (pág. 12).

- Se debe utilizar el valor de la etiqueta d_k y la salida no lineal que viene después de la sigmoide para calcular el MSE.

- Se minimiza el MSE usando el método del gradiente (instantáneo).

- Error Cuadrático Medio $$\tilde{\varepsilon}_k^{\,2} = (d_k - sgm(s_k))^2$$

- Cálculo del gradiente $$\hat{\nabla}_k = \frac{\partial(\varepsilon_k^{\,2})}{\partial W_k} = 2\varepsilon_k \frac{\partial \varepsilon_k}{\partial W_k}$$

$$\frac{\partial \varepsilon_k}{\partial W_k} = -\frac{\partial sgm(s_k)}{\partial W_k} = -sgm'(s_k)\frac{\partial s_k}{\partial W_k} = -sgm'(s_k)X_k$$

- Resultado del cálculo del gradiente

$$\boxed{\hat{\nabla}_k = -2\varepsilon_k sgm'(s_k)X_k}$$

Aplicación del Algoritmo de Backpropagation con MSE

- Con el resultado obtenido del cálculo del gradiente, podemos usar la regla del algoritmo del gradiente descendiente:

- Usando la regla del gradiente, podemos reemplazar el valor de $\hat{\nabla}_k$.

$$W_{k+1} = W_k - \mu\hat{\nabla}_k = W_k + 2\mu\tilde{\varepsilon}_k sgm'(s_k)X_k$$

- Note que debemos calcular la derivada de la sigmoide:

$$sgm'(s_k) = sgm(s_k)(1 - sgm(s_k)) = y_k(1 - y_k) \quad \text{si} \quad y_k = \frac{1}{1+e^{-s_k}}$$

o

$$sgm'(s_k) = 1 - (tanh(s_k))^2 = 1 - y_k^2 \quad \text{si} \quad y_k = tanh(s_k)$$

Aplicación del Algoritmo de Backpropagation con MSE

- Para simplificar el cálculo, se define delta como la derivada del error cuadrático con respecto a la salida lineal. Este parámetro representa la sensibilidad del error de salida a cambios en la salida lineal del elemento.

$$\delta = -\frac{\partial(\varepsilon^2)}{\partial s} = -2\varepsilon(-sgm'(s)) = 2\varepsilon sgm'(s)$$

- De esta forma, la regla final que permite el aprendizaje usando el error cuadrático medio MSE, regla delta, corresponde a:

$$\boxed{\Delta W_k = \mu \delta_k X_k}$$

Aplicación del Algoritmo de Backpropagation con MSE

- Las figuras muestras ejemplos de MSE para la salida lineal (figura izquierda) y salida no lineal (figura derecha). Note que las curvas presentan un mínimo.

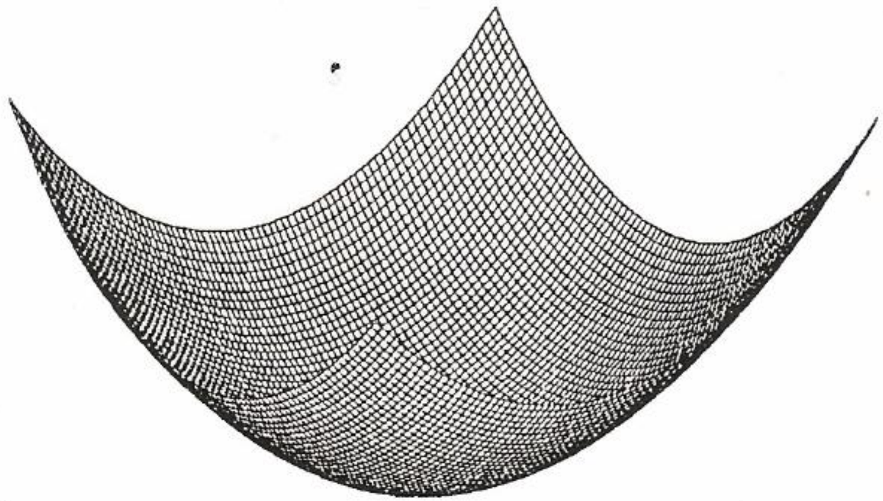


Fig. 22. Example MSE surface of linear error.

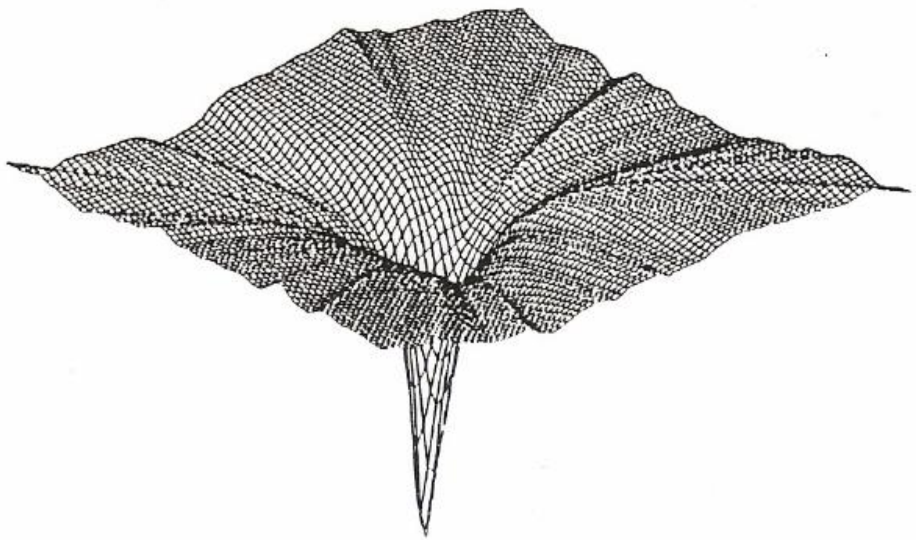
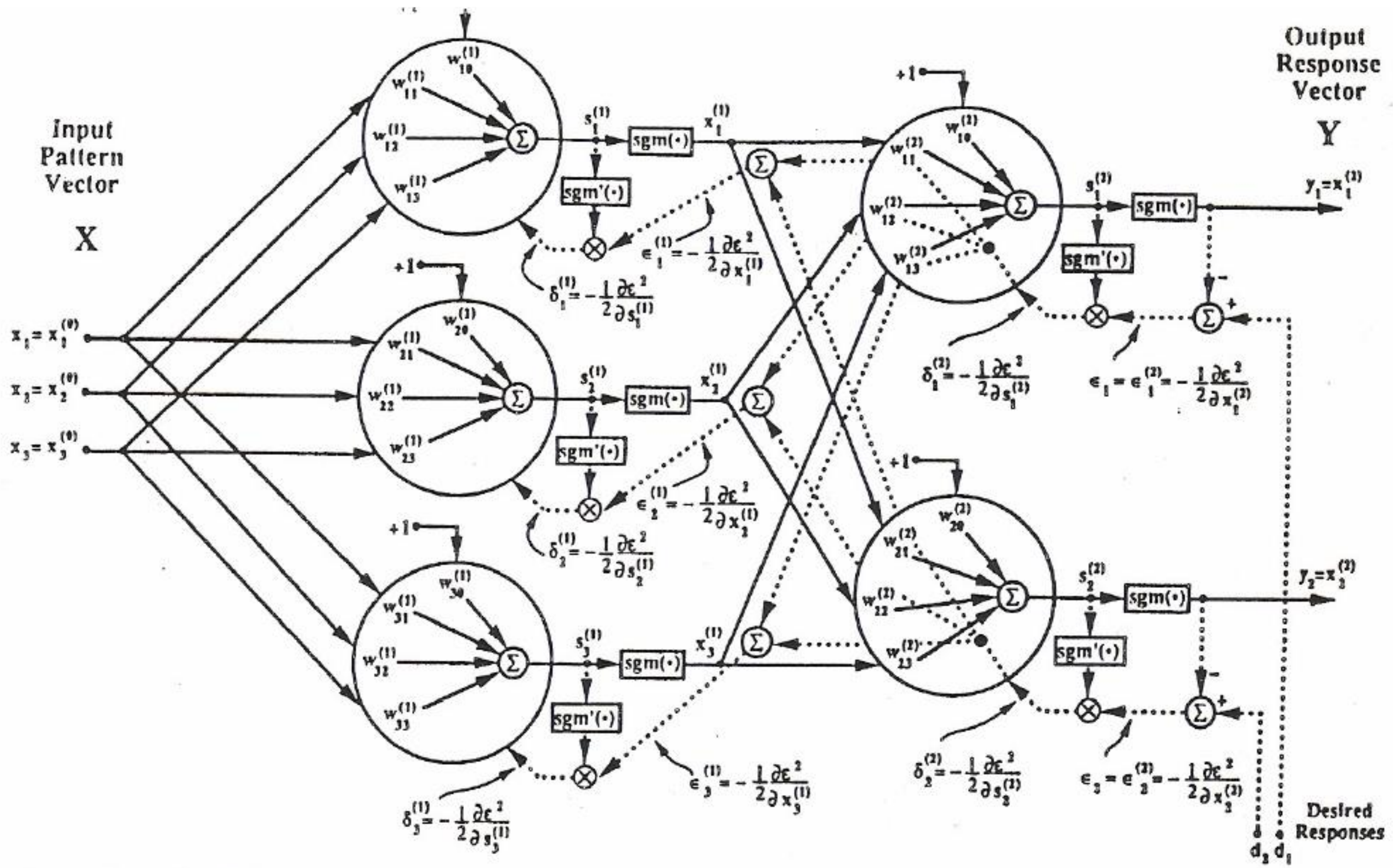


Fig. 23. Example MSE surface of sigmoid error.

Algoritmo de Backpropagation con MSE



Example two-layer backpropagation network architecture.

Resumen del Algoritmo de Backpropagation con MSE

- Método de Retropropagación del error (Backpropagation)

- Funciona en dos pasadas:

  1. Forward: Con los pesos fijos, se calcula la respuesta de las unidades ocultas y de la salida ante las entradas. Se determina el error de la red neuronal.

  2. Backward: La señal de error se propaga hacia atrás usando los pesos de la red. Luego se ajustan los pesos mediante el cálculo del gradiente.

- Dado el error cuadrático medio (MSE) instantáneo en la capa de salida: $\varepsilon_{T_k} = \sum_j \tilde{\varepsilon}_{jk}^2$

- El ajuste de los pesos en la capa de salida se realiza mediante la regla:

Capa de salida :

$$\Delta w_{ji}^k = \mu \delta_j^k z_i^k$$

$$\delta_j^k = -\frac{\partial(\varepsilon_{T_k})}{\partial s_{jk}} = 2\tilde{\varepsilon}_{jk} sgm'(s_{jk})$$

Resumen del Algoritmo de Backpropagation con MSE

- Para las capas ocultas, el ajuste de los pesos se realiza mediante la regla:

$$\boxed{\begin{aligned} \Delta w_{ih}^{k} &= \mu \delta_i^k x_h^k \\ \delta_i^k &= -\frac{\partial(\varepsilon_{T_k})}{\partial s_{ik}} = -\sum_j \frac{\partial \varepsilon_{T_k}}{\partial s_{jk}} \frac{\partial s_{jk}}{\partial s_{ik}} \end{aligned}}$$

- donde:

$$s_{jk} = \sum_i w_{ji} sgm(s_{ik})$$

$$\frac{\partial s_j}{\partial s_i} = w_{ji} sgm'(s_i)$$

Resumen del Algoritmo de Backpropagation con MSE

- Al resolver para la sigmoide, la actualización final de los pesos y deltas para la **capa oculta** se realiza mediante la siguiente regla:

Capa Oculta :

$$\Delta w_{ih}^k = \mu \delta_i^k x_h^k$$

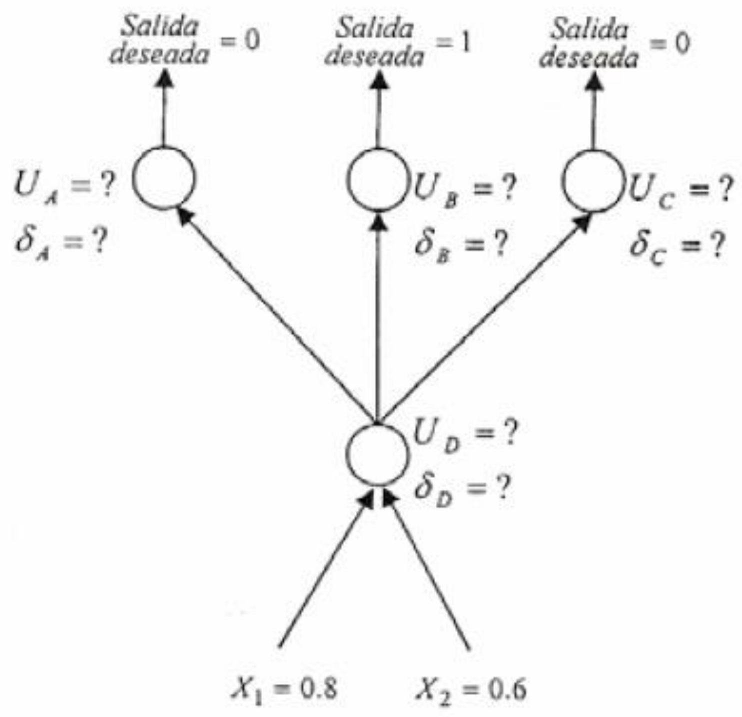
$$\delta_i^k = \left(\sum_j w_{ji} \delta_j \right) sgm'(s_i)$$

Ejemplo: Backpropagation

- Considere el algoritmo Backpropagation para el MSE explicado anteriormente. Observe la red de la figura con una función de activación sigmoidal:

$$sgm(x) = \frac{1}{1 + e^{-x}}$$

a. Encuentre el valor de la salida y de los deltas en A, B, C. Considere los pesos inicialmente nulos.
b. Determine los próximos pesos de la red.



Ejemplo: Backpropagation

Con los pesos (de valor 0), se calcula la respuesta de las unidades ocultas y de la salida ante las entradas.

a) Paso hacia adelante. ($\vec{w} = 0$)

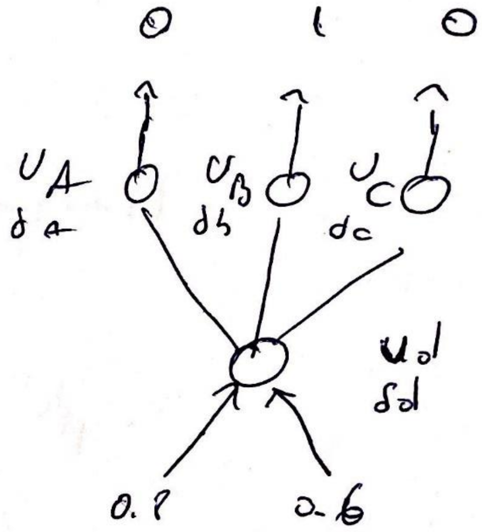
$x_1 = 0.8$, $x_2 = 0.6$

$y = 0.8 \cdot w_1 + 0.6 w_2 = 0$

$$sgm(x) = \frac{1}{1+e^{-x}}$$

$$\therefore \; sgm(0) = \frac{1}{1+1} = \frac{1}{2}$$

$$\therefore \; u_d = \frac{1}{2}$$



continua a la sgte capa, pero $\vec{w} = 0$ $\therefore$ $U_A = U_b = U_c = \frac{1}{2}$

Ejemplo: Backpropagation

La señal de error se propaga hacia atrás usando los pesos de la red.
A continuación se presenta como calcular la derivada de la sigmoide.

b) Paso hacia atrás (backpropagation).

$\delta_a = 2 \cdot \varepsilon \; sgm'(u_a)$

cálculo de sgm' $\Rightarrow$ $sgm(x) = \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1} \quad \cdot / \partial / \partial x$

$= -1 \cdot (1+e^{-x})^{-2} \cdot e^{-x} \cdot -1$

$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2}$

$= \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$

$= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} =$

$sgm'(x) = sgm(x)(1-sgm(x)) = \frac{1}{1+e^{-x}}\left[1 - \frac{1}{1+e^{-x}}\right]$

Ejemplo: Backpropagation

Usando la **regla de la capa de salida**, se calculan las deltas de cada salida.
Se deben considerar las etiquetas para el cálculo del error.
Luego se ajustan los pesos de la capa de salida

$$\therefore \quad \delta_a = 2 \cdot \varepsilon \cdot sgm'(u_a)$$

$$= 2 \cdot \varepsilon \cdot sgm(u_a) \cdot (1 - sgm(u_a))$$

reemplazando

$$\delta_a = 2 \cdot \left(0 - \frac{1}{2}\right) \cdot \frac{1}{2} \cdot \frac{1}{2} = -1/4$$

$$\delta b = 2 \cdot \varepsilon \, sgm(u_b)\,(1 - sgm(u_b))$$

$$= 2 \cdot \left(1 - \frac{1}{2}\right) \cdot \frac{1}{2} \cdot \frac{1}{2} = 1/4$$

$$\delta c = 2 \cdot \left(0 - \frac{1}{2}\right) \frac{1}{2} \cdot \frac{1}{2} = -1/4$$

Ajustar los pesos. en capa de salida.

$$w_a = \mu \cdot \delta a \cdot u_a = \mu \left(-\frac{1}{4}\right) \cdot \frac{1}{2} = -\mu \cdot \frac{1}{8}$$

$$w_b = \mu \cdot \delta b \cdot u_b = \mu \left(\frac{1}{4}\right) \cdot \frac{1}{2} = \mu \frac{1}{8}$$

$$w_c = \mu \cdot \delta c \cdot u_c = -\mu \cdot \frac{1}{8}$$

Ejemplo: Backpropagation

Usando la **regla de la capa oculta**, se calculan primero las deltas de cada capa oculta. Luego se ajustan los nuevos pesos de la red.

Capa oculta.

$$\delta_d = \left(\sum_J w_{iJ}\,\delta_J\right)\left(sgm'(\delta d)\right)$$

$$= \left(w_a\cdot\delta_a + w_b\cdot\delta_b + w_c\cdot\delta_c\right)\cdot sgm(u_d)\cdot(1-sgm(u_d))$$

$$= \left(-\mu\cdot\frac{1}{8}\cdot\left(-\frac{1}{4}\right) + \frac{1}{4}\cdot\mu\frac{1}{8} + -\frac{\mu}{8}\cdot\left(-\frac{1}{4}\right)\right)\cdot\frac{1}{2}\left(1-\frac{1}{2}\right)$$

$$= \left(+\frac{\mu}{32} + \frac{1}{32}\mu + \frac{1}{32}\mu\right)\cdot\frac{1}{4}$$

$$= \frac{3}{32}\mu\cdot\frac{1}{4} = \underline{\frac{3}{128}\mu}$$

b) Nuevos pesos de la RED.

$$\Delta W = \mu\,\delta d\cdot X_h$$

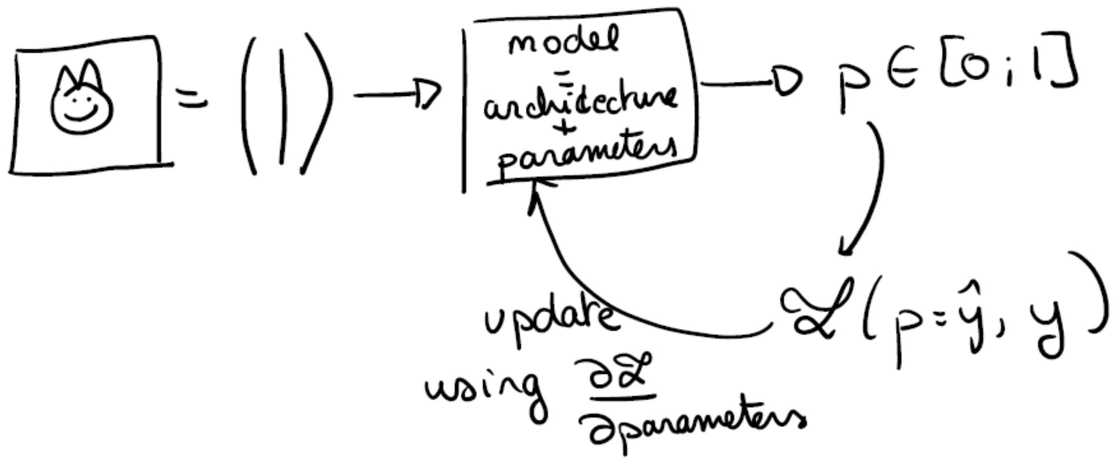
$$W_{h1d} = \mu\cdot\frac{3}{128}\mu\cdot\frac{8}{10} = \mu^2\cdot\frac{3}{160}$$

$$W_{h2d} = \mu\cdot\frac{3}{128}\mu\cdot\frac{6}{10} = \mu^2\cdot\frac{18}{1280}$$

Backpropagation con Binary cross-entropy

- Aplicación de backpropagation con función objetivo: Binary cross-entropy

- Consideremos un ejemplo de clasificación de imágenes de gatos. Primero se convierte la imagen de gatos a un solo vector de características. Este vector se ingresa a la red neuronal la cual nos dirá si la entrada es un gato o no.

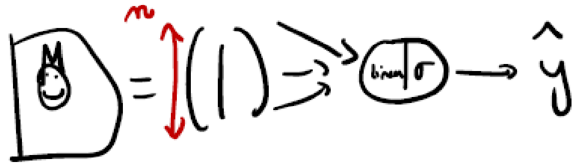


- Regla de ajuste de los pesos

$$
\boxed{\text{GD}: \quad \begin{aligned} w &= w - \alpha \frac{\partial \mathcal{L}}{\partial w} \\ b &= b - \alpha \frac{\partial \mathcal{L}}{\partial b} \end{aligned}}
$$

Backpropagation con Binary cross-entropy

- Forward:



forward propagation:

$$\underset{(1,1)}{z} = \underset{(1,n)}{w}\,\underset{(n,1)}{x} + \underset{(1,1)}{b}$$

$$\hat{y} = a = \sigma(z)$$

- Función de pérdida:

$$\text{Loss}: \mathcal{L} \doteq -\left(y\log\hat{y} + (1-y)\log(1-\hat{y})\right)$$

- Backpropagation:

Backprop: $\frac{\partial \mathcal{L}}{\partial w}$ $\quad \frac{\partial \mathcal{L}}{\partial b}$

$$\frac{\partial z}{\partial w_1} = x_1 \qquad \frac{\partial z}{\partial w_2} = x_2$$

$$z = wx + b = (w_1 \dots w_n)\begin{pmatrix}x_1\\ \vdots \\ x_n\end{pmatrix} + b$$

$$= \sum_{i=1}^{n} w_i x_i$$

Backpropagation con Binary cross-entropy

- Se busca determinar la regla de ajuste mediante backpropagation, es decir : $\frac{\partial \mathcal{L}}{\partial w}$ $\frac{\partial \mathcal{L}}{\partial b}$

- Se deben encontrar las gradientes de la función de costo para los pesos y los sesgos. Es posible aplicar la regla de la cadena para resolver el problema.

- Cálculo de la pérdida respecto los pesos usando Binary cross-entropy.
- Recordar que:

$$\hat{y} = a = \sigma(z)$$

- Derivando la función:

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

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w}$$

- pero

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z}$$

Backpropagation con Binary cross-entropy

- Así, la derivada de la pérdida Binary cross-entropy con respecto a la salida

$$\frac{\partial \mathcal{L}}{\partial a} = -\left(y \frac{\partial \log a}{\partial a} + (1-y) \frac{\partial \log (1-a)}{\partial a} \right)$$

$$= -\left(y \frac{1}{a} + (1-y) \frac{1}{1-a} \cdot (-1) \right)$$

- Respecto a la entrada:

$$\frac{\partial \mathcal{L}}{\partial z} = -\left(y \frac{1}{a} \frac{\partial a}{\partial z} + (1-y) \frac{1}{a-1} \frac{\partial a}{\partial z} \right)$$

$$= -\left(y \frac{1}{\cancel{a}} \cancel{a}(1-a) + (1-y) \frac{-1}{\cancel{a-1}} a(\cancel{1-a}) \right)$$

$$= -y(1-a) - a(1-y) = \boxed{a-y}$$

Backpropagation con Binary cross-entropy

- Así, la derivada respecto a los pesos queda:

$$\overset{(1,n)}{\frac{\partial \mathcal{L}}{\partial w}} = \frac{\partial \mathcal{L}}{\partial z} \cdot \overset{(1,n)}{\frac{\partial z}{\partial w}} = \underbrace{(a-y)}_{(1,1)} \underbrace{x^T}_{(1,n)}$$

- La derivada respecto a los sesgos (bias):

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial b} = (a-y) \cdot 1$$

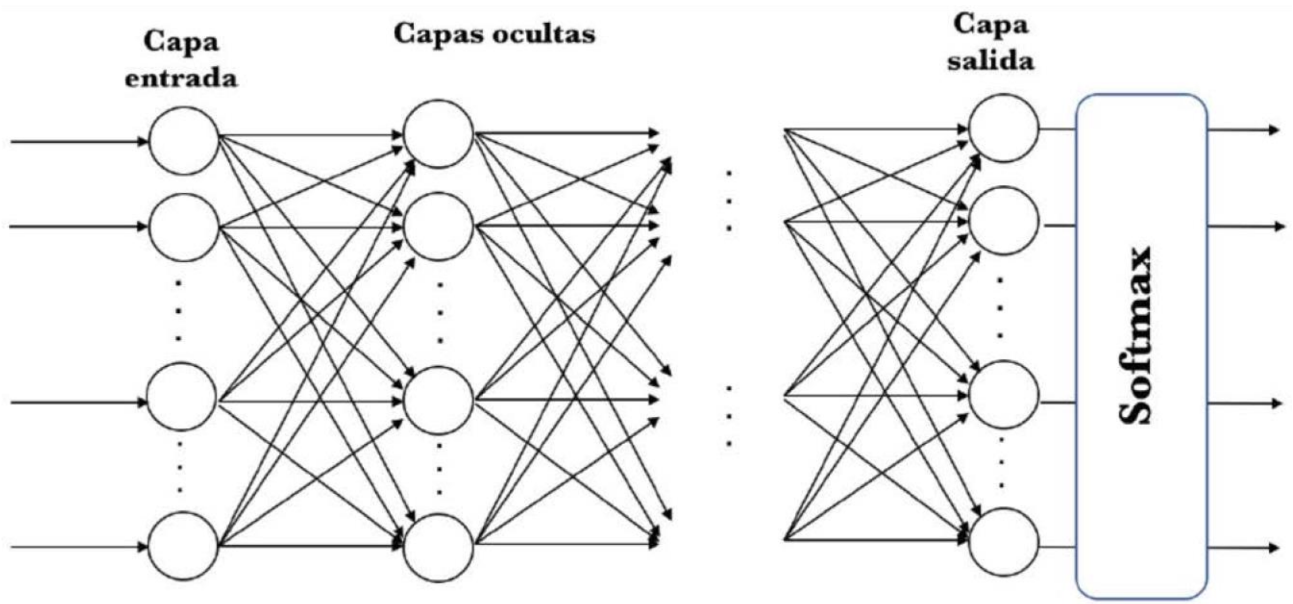
- La regla de actualización de los pesos para el aprendizaje es:

$$w = w - \alpha (a-y) x^T$$

$$b = b - \alpha (a-y) 1$$

Multi-Layer Perceptron para clasificación

- Las Multi-Layer Percetron (MLP) se usan a menudo para clasificar, especialmente cuando hay exclusividad entre etiquetas o clase. En este caso la capa de salida es una función de activación SOFTMAX, en la cual la salida de cada neurona corresponde a la probabilidad estimada para cada etiqueta o clase correspondiente.



Probabilidad de pertenencia

- Por ejemplo, para realizar la clasificación de números 0-9, una vez que se ha calculado la evidencia de pertenencia a cada una de las 10 clases, éstas se deben convertir en probabilidades cuya suma de todos sus componentes sume 1. Para ello softmax usa el valor exponencial de las evidencias calculadas y luego las normaliza de modo que sumen uno, formando una distribución de probabilidad. La probabilidad de pertenencia a la clase i es:

$$Softmax_i = \frac{e^{evidence_i}}{\sum_j e^{evidence_j}}$$

Hora de ensuciar las manos!

A practicar!!