Introducción a "Statistical Machine Learning V2"
Club de Ciencia de Datos La Paz

Introducción a Redes Neuronales

Perceptrón Multicapa

Rafael Villca Poggian

Abril de 2019

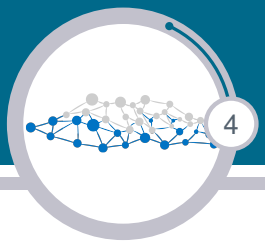
Contenido

Introducción
¿Qué es una Red Neuronal?

Conocido formalmente como "Perceptrón Multicapa", es un modelo matemático que nació bajo la inspiración de modelar la codificación de patrones en la estructura neuronal, como respuesta a impulsos externos.

Introducción
¿Cómo se modela matemáticamente?

- Se puede entender una red neuronal como una regresión logística con capas intermedias, las cuales la convierten en un aproximador universal de funciones, y un selector inteligente de variables.
- Si antes vimos que una Regresión Logística se modelaba como una composición de funciones derivables, la red neuronal mantiene esta misma estructura, sólo un poco más complicada y con más componentes.



- $\mathbf{W_l}$: Matriz de parámetros de transición entre la capa *l* de la red.
- $\mathbf{b_l}$: Vector de Sesgos o bias.
- $\mathbf{A_l}$: Activación en de las neuronas visibles en la capa *l*
- $\mathbf{g_l}(\mathbf{x})$: Función de Activación para la capa *l*.

Introducción
¿Cómo ajustar los parámetros del modelo?

- Debemos dar una pasada por los datos para obtener las activaciones en cada capa de la red neuronal (Forward Propagation).
- Calculamos la derivada de todos los pasos en reversa (Backpropagation).
- Usamos los parámetros ya calculados al navegar por el grafo para evaluar los gradientes al recorrer el grafo de manera inversa.

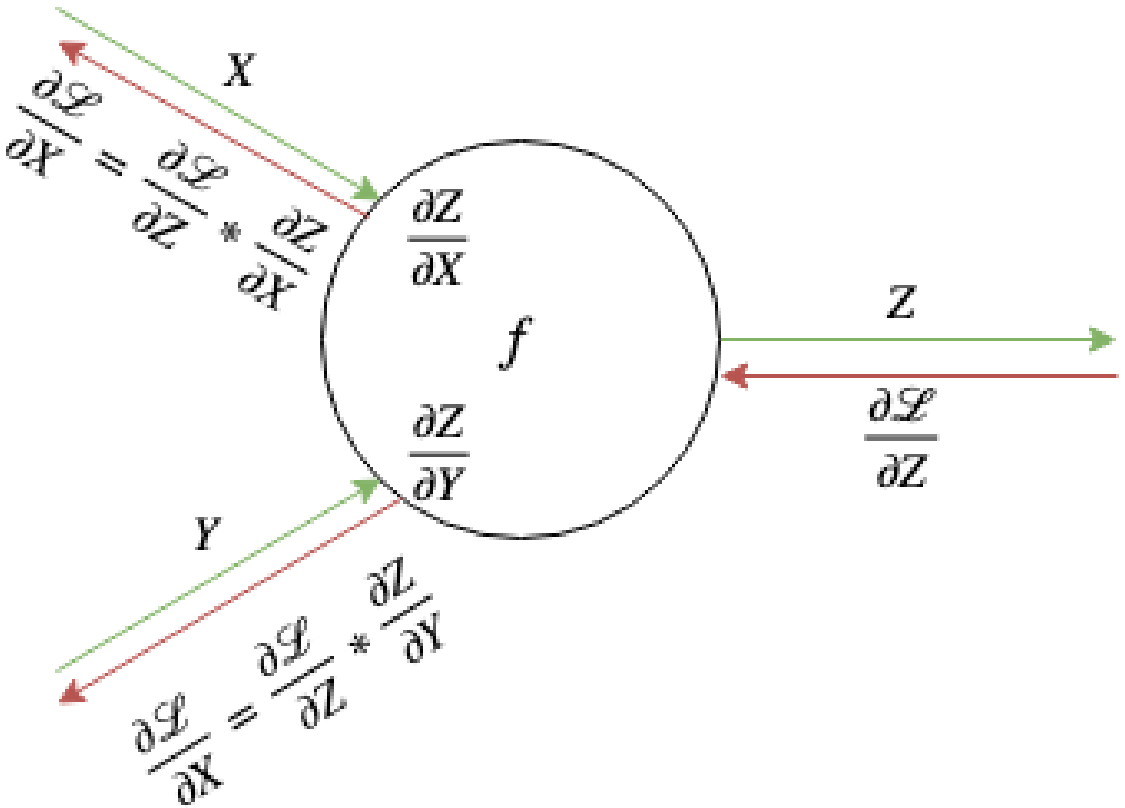
Introducción
¿Cómo ajustar los parámetros del modelo?

- Una vez evaluados los gradientes actualizamos los valores de los parámetros bajo la regla del gradient descent para dar un paso hacia los valores óptimos.
- Para poder almacenar esta información de manera óptima utilizamos un grafo de computación (DAG).

Grafo Computacional
¿Por qué usarlo?

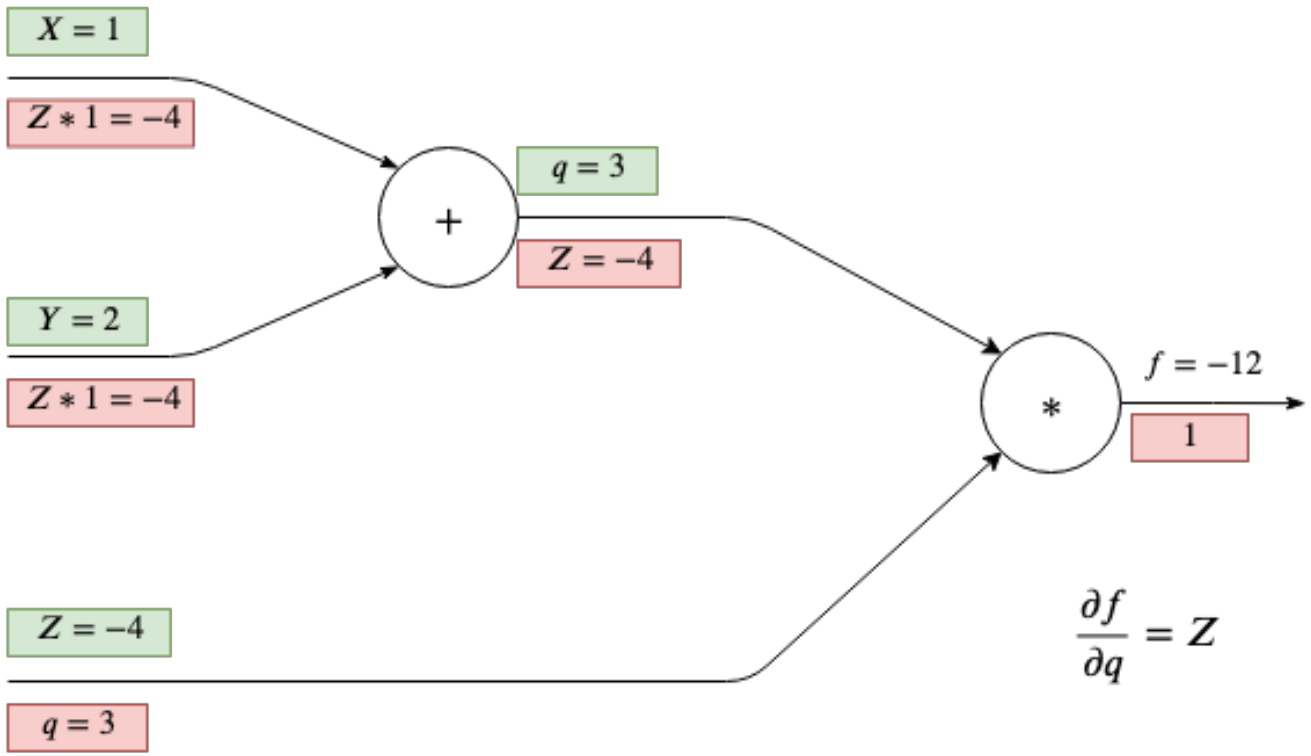
- Podemos descomponer una función compleja en sus distintas operaciones componentes básicas.
- De esta forma es mucho más fácil computar derivadas de funciones compuestas.
- Es gráficamente interpretativo.

Grafo Computacional
Interpretación



Grafo Computacional
Ejemplo

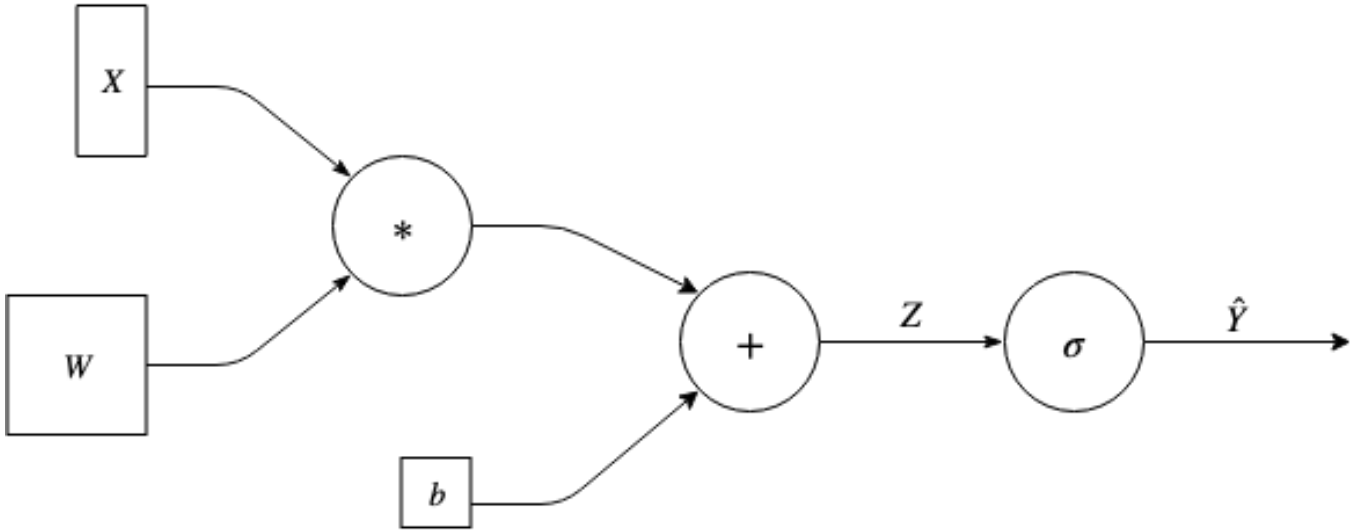
$$f(X, Y, Z) = (X + Y) * Z = q * Z$$

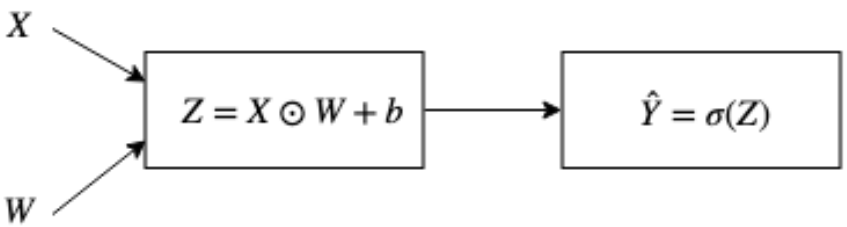


$$\frac{\partial f}{\partial q} = Z \qquad \frac{\partial f}{\partial Z} = q$$

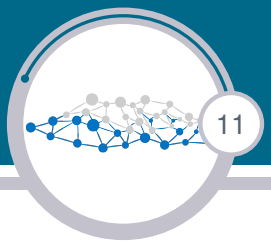
$$\frac{\partial q}{\partial X} = 1 \qquad \frac{\partial f}{\partial X} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial X} \quad , \quad \frac{\partial q}{\partial Y} = 1 \qquad \frac{\partial f}{\partial X} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial Y}$$

Grafo Computacional
Regresión Logística





Funciones de Activación
Definición y Uso

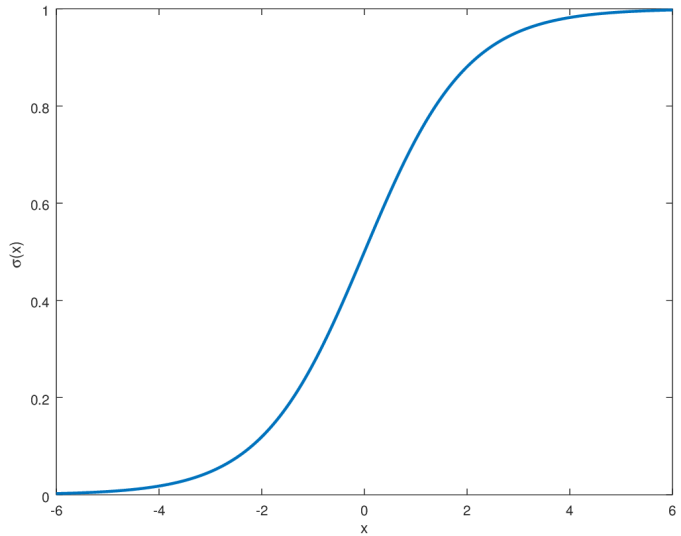


- Es una función no lineal que se utiliza para aumentar la capacidad de ajuste a los datos de la red neuronal.
- Si no se usara, la red sería simplemente un clasificador lineal.
- Es una función

$$g_l(z_l)$$

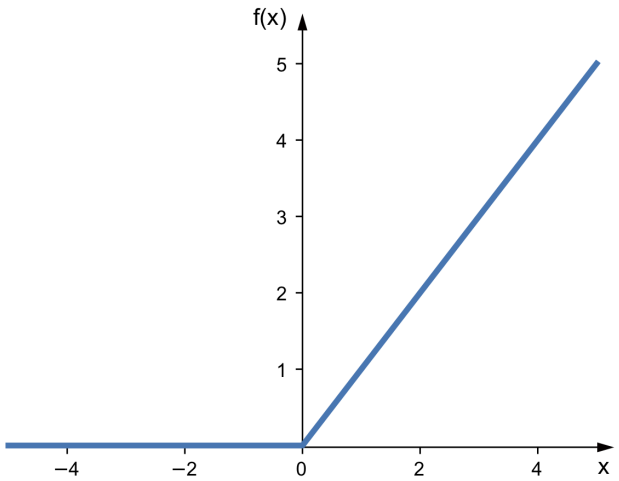
en cada capa oculta.

Funciones de Activación
Sigmoide



- Utilizada bajo inspiración biológica.
- Costosa de computar.
- Los gradientes causan problemas de desvanecimiento.

Funciones de Activación
Relu



- Igualmente no lineal.
- Puede causar desvanecimiento de gradientes en el peor de los casos.
- Fácil de computar.

Funciones de Activación
Relu

Rectied Linear Unit

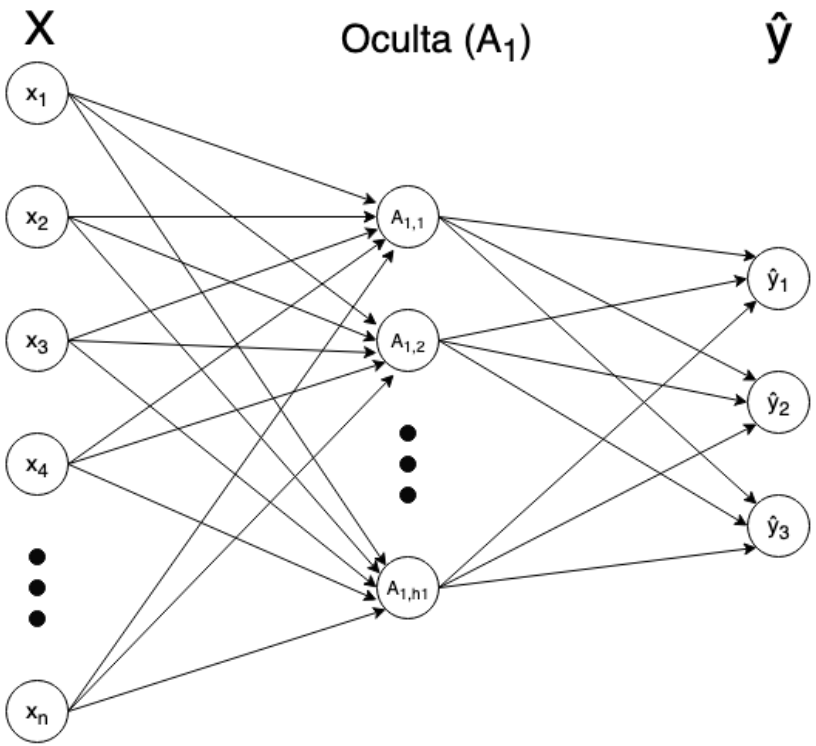
$$Relu(Z) = max(0, Z) \forall Z_j, j : 0, 1, \ldots h_l$$

Dónde h_l es el número de elementos en Z

Derivada

$$\frac{\partial Relu}{\partial Z} = \begin{cases} 1, & \text{si } Z_j > 0 \\ 0, & \text{si } Z_j \leq 0 \end{cases}$$

Perceptrón Multicapa
Modelo



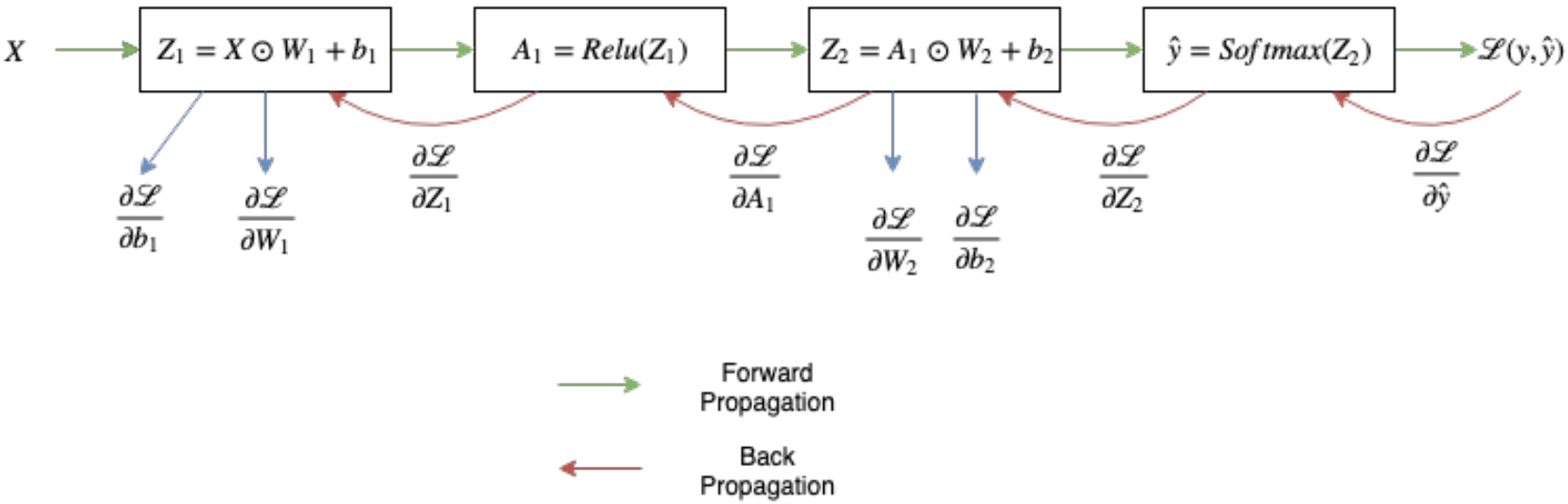
$$Z_1 = X \odot W_1 + b_1$$

$$A_1 = X \odot Relu(Z_1)$$

$$Z_2 = A_1 \odot W_2 + b_2$$

$$\hat{Y} = softmax(Z_2)$$

Perceptrón Multicapa
Forward & Backprop



Algorithm 1 Neural Network Training

procedure GRADIENT DESCENT(X, Y, layer_shapes, n_layers, α)
    $W \leftarrow$ `[...]`, $b \leftarrow$ `[...]`
    for `l = 1 to n_layers` **do**
        `W[l]` $\leftarrow$ `random(layer_shapes[l-1], layer_shapes[l])`
        `b[l]` $\leftarrow$ `zeros( layer_shapes[l])`
    for `i = 1 to iters` **do**
        $\hat{Y}$, *cache* $\leftarrow$ `forward_propagation(X, W, b)`
        $\mathscr{L} \leftarrow$ `cross_entropy(`$\hat{Y}$`, Y)`
        $\nabla W, \nabla b \leftarrow$ `backpropagation(X, W, b, cache)`
        $W \leftarrow W - \alpha * \frac{\partial \mathscr{L}}{\partial W}$
        $b \leftarrow b - \alpha * \frac{\partial \mathscr{L}}{\partial b}$
    return *predicción*