Introducción a Redes Neuronales Artificiales

Mauricio A. Álvarez PhD,
H.F. Garcia C. Guarnizo (TA)

Universidad Tecnológica de Pereira, Pereira, Colombia

Outline

Contenido

Introducción - Redes Neuronales Artificiales

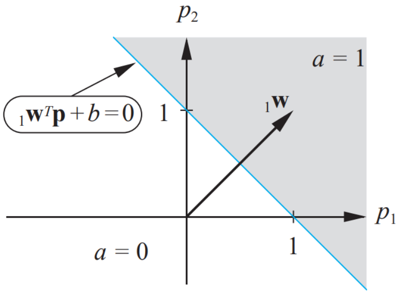
- Esencialmente, es una red de elementos funcionales cada uno con varias entradas y salidas.

$$y(x_1, \ldots, x_n) = f(w_1 x_1 + w_2 x_2 + \ldots, w_n x_n),$$

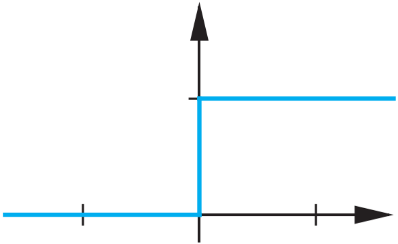
  donde w_i son parámetros, $f(\cdot)$ es una función de activación.
- Cualquier función puede ser aproximada a partir de la composición de funciones básicas

$$y(t) = f_N(f_{N-1}(\ldots f_n(\ldots f_1(t))))$$

Introducción - Perceptron

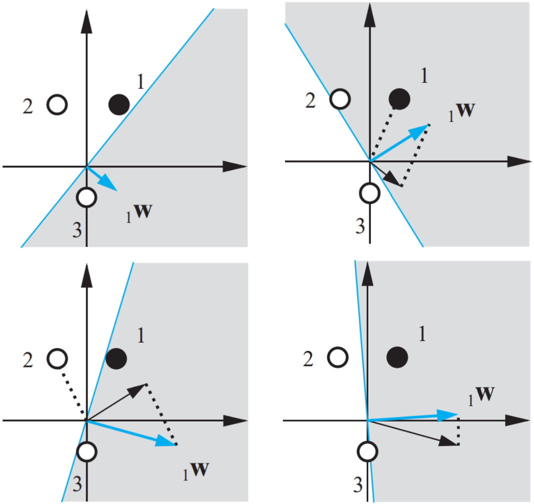


$$n = w_1 p_1 + w_2 p_2 + b$$



$$f(n) = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$

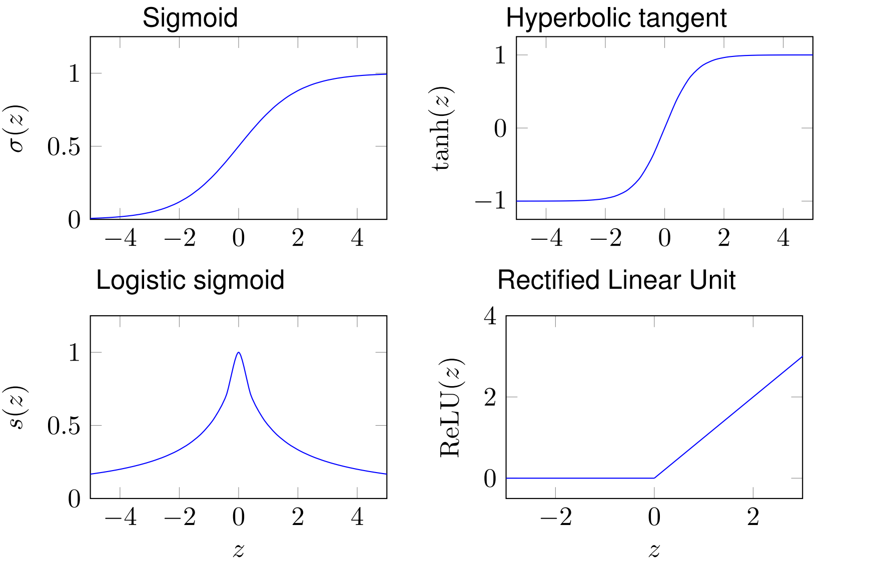
Introducción - Algoritmo Perceptron



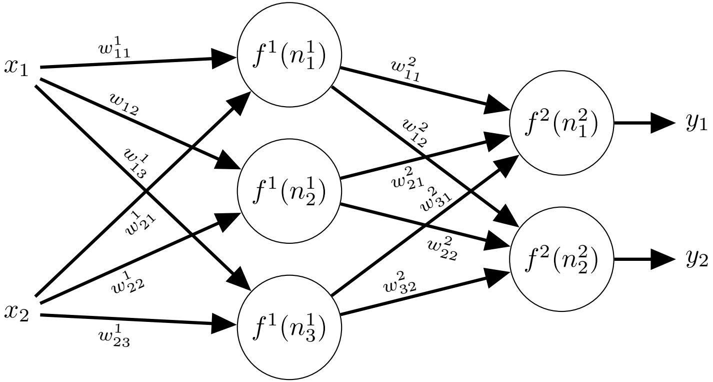
$$a = f(\mathbf{w}^\top \mathbf{p}_i)$$

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + (t_i - a)\mathbf{p}_i$$

Funciones de activación



Introducción - Multi-capas



$$\mathbf{W}_1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \end{bmatrix}, \; \mathbf{W}_2 = \begin{bmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^1_{31} & w^1_{32} \end{bmatrix}, \; \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \; \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\mathbf{y} = f^2\left(\mathbf{W}_2^\top f^1(\mathbf{W}_1^\top \mathbf{x})\right)$$

PyTorch - Example

```python
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
  def __init__(self):
    super().__init__()
    self.hidden = nn.Linear(784, 128)
    self.output = nn.Linear(128, 10)

  def forward(self, x):
    x = self.hidden(x)
    x = F.sigmoid(x)
    x = self.output(x)
    return x
```

Notación - Aprendizaje Supervisado

- Se asume que para poder entrenar la red (aprender los parámetros $\mathbf{w}$), se cuenta con un conjunto de N muestras de entradas y salidas $(\mathbf{X}, \mathbf{t})$, donde: $\mathbf{X} \in \mathbb{R}^{D \times N}$, $\mathbf{x}_n \in \mathbb{R}^D$, $\mathbf{t} \in \mathbb{R}^{K \times N}$ y $\mathbf{t}_n \in \mathbb{R}^K$.

- A partir de los datos anteriores se minimiza la función de error

$$\mathbf{E}(\mathbf{w}) = \mathrm{D}(\mathbf{t}, \mathbf{y}(\mathbf{X}, \mathbf{w}))$$

  donde $\mathbf{y} \in \mathbb{R}^{K \times N}$ es la salida de la red neuronal, y de D es una medida de distancia o error.

- El vector $\mathbf{w} = [\mathbf{W}_1, \ldots, \mathbf{W}_M]$ representa todos los pesos de la red neuronal.

Contenido

Regresión - Consideraciones

- La función de error más usada es el error medio cuadrático (MSE):

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n}\sum_{k}(t_{nk} - y_{nk})^2$$

- Por lo general, la función de activación a la salida es lineal.

Contenido

Clasificación - Consideraciones

- Para problemas de K-clases, se asume que los objetivos $\mathbf{t}_n$ están en forma "1 de K".

$$t_n = 2 \rightarrow \mathbf{t}_n = \begin{array}{c@{}ccccc@{}c} \big[& 0 & 1 & 0 & \dots & 0 & \big] \\ & & 2 & & & & \end{array}$$

- El vector de salida de la red neuronal se transforma usando *Softmax*.
- La función objetivo es la Entropía cruzada.

$$E(\mathbf{w}, \mathbf{t}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln(y_{nk})$$

Clasificación - Softmax

La función *Softmax* se define

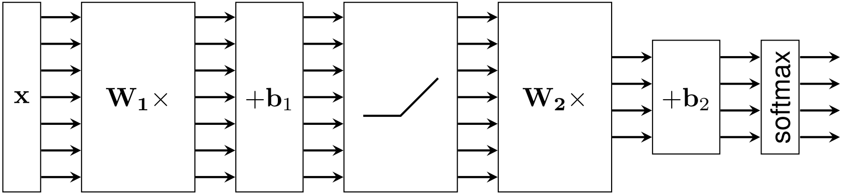
$$\sigma(a_j) = \frac{\exp(a_j)}{\sum_{i=1}^{K} \exp(a_i)}.$$

donde a_j es la componente j del vector salida $\mathbf{a}$. Ejemplo

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \to \sigma\left(\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}\right)$$

$$\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \to \sigma\left(\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}\right) = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} = \mathbf{y}$$

Clasificación - Ejemplo



TensorFlow - Example

```
import tensorflow as tf
from tensorflow import keras

model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

Contenido

Backpropagation - Generalidades

- Evalua de forma eficiente el gradiente de la función de error $E(\mathbf{w})$.
- Se realiza en dos etapas:
  - Primera etapa $\rightarrow$ evaluación de las derivadas de la función error con respecto a $\mathbf{w}$.
  - Segunda etapa $\rightarrow$ las derivadas se usan para realizar los ajustes de los pesos, usando un método de optimización.

Backprop - Derivadas de la función de error

- Asumiendo que los datos son iid, las funciones de error toman la forma

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}),$$

donde $E_n(\mathbf{w})$ es el error calculado sobre cada $(\mathbf{x}_n,\ \mathbf{t}_n)$.

- El gradiente también se distribuye

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial E_n(\mathbf{w})}{\partial \mathbf{w}}$$

- Los pesos de la red se actualizan

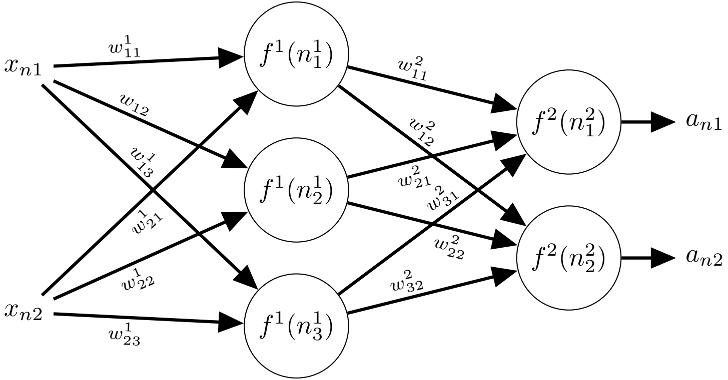
$$\mathbf{w}^{\text{new}} = \mathbf{w} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

Backprop - Derivadas de la función de error

P	$E_n(\mathbf{w})$	$y_n(a_n)$, $y_{nk}(a_{nk})$	$\frac{\partial E_n(\mathbf{w})}{\partial a_n}$, $\frac{\partial E_n(\mathbf{w})}{\partial a_{nk}}$
RS	$\frac{1}{2}(t_n - y_n)^2$	$y_n = a_n$	$y_n - t_n$
RM	$\frac{1}{2}\sum_{k=1}^{K}(t_{nk} - y_{nk})^2$	$y_{nk} = a_{nk}$	$y_{nk} - t_{nk}$
CS	$-\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$	$y_n = \sigma(a_n)$	$y_n - t_n$
CE	$-\sum_{k=1}^{K} t_{nk} \ln y_{nk}$	$y_{nk} = \sigma(a_{nk})$	$y_{nk} - t_{nk}$

P: Problema. RS: Regresión Simple. RM: Regresión Múltiple. CS: Clasificación Simple. CE: CrossEntropy. σ: Softmax.

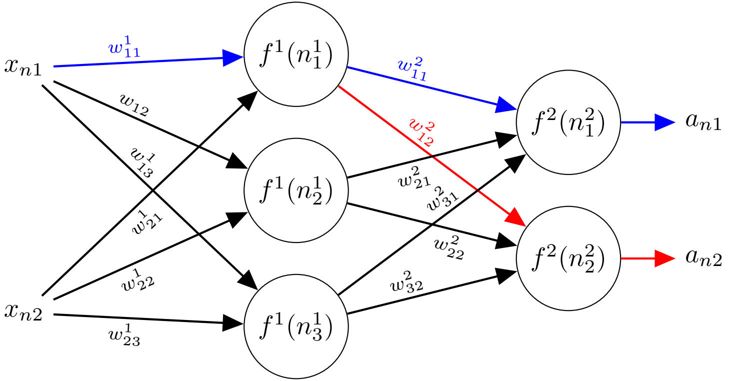
Backprop - Red Neuronal Multi-capa



$$\mathbf{W}_1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{bmatrix}, \mathbf{W}_2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \\ w_{31}^1 & w_{32}^1 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \mathbf{a}_n = \begin{bmatrix} a_{n1} \\ a_{n2} \end{bmatrix}$$

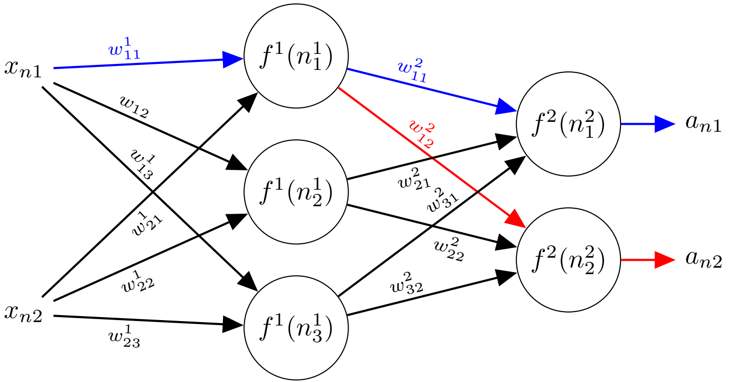
$$\mathbf{a}_n = f^2\left(\mathbf{W}_2^\top f^1(\mathbf{W}_1^\top \mathbf{x}_n)\right)$$

Backprop - Red Neuronal Multi-capa



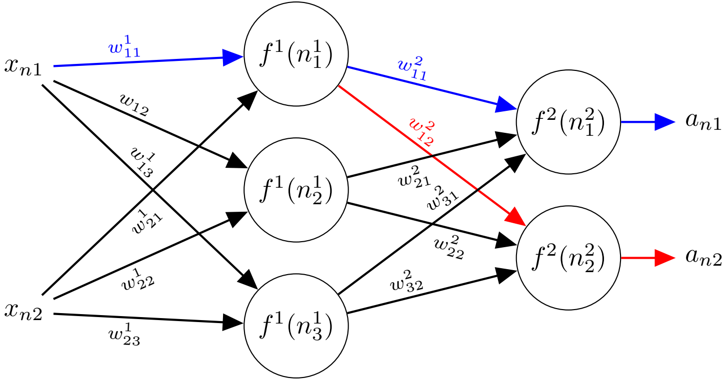
$$
\begin{aligned}
\frac{\partial E_n(\mathbf{w})}{\partial w_{11}^1} = & \frac{\partial E_n(\mathbf{w})}{\partial a_{n1}} \frac{\partial f^2(n_1^2)}{\partial n_1^2} \frac{\partial n_1^2}{\partial f^2(n_1^1)} \frac{\partial f^1(n_1^1)}{\partial n_1^1} \frac{\partial n_1^1}{\partial w_{11}^1} \\
& + \frac{\partial E_n(\mathbf{w})}{\partial a_{n2}} \frac{\partial f^2(n_2^2)}{\partial n_2^2} \frac{\partial f^2(n_2^2)}{\partial n_2^2} \frac{\partial f^1(n_1^1)}{\partial n_1^1} \frac{\partial n_1^1}{\partial w_{11}^1}
\end{aligned}
$$

Backprop - Red Neuronal Multi-capa



$$\frac{\partial E_n(\mathbf{w})}{\partial w_{11}^1} = \frac{\partial E_n}{\partial a_{n1}} f'^2(n_1^2) w_{11}^2 f'^1(n_1^1) x_1 + \frac{\partial E_n}{\partial a_{n2}} f'^2(n_2^2) w_{12}^2 f'^1(n_1^1) x_1$$

Backprop - Red Neuronal Multi-capa



$$\frac{\partial E_n(\mathbf{w})}{\partial w^1_{ij}} = \left(\frac{\partial E_n}{\partial a_{n1}} f'^2(n^2_1) w^2_{j1} + \frac{\partial E_n}{\partial a_{n2}} f'^2(n^2_2) w^2_{j2} \right) f'^1(n^1_j) x_{ni}$$

$$\frac{\partial E_n(\mathbf{w})}{\partial w^2_{ij}} = \frac{\partial E_n}{\partial a_{nj}} f'^2(n^2_j) f^1(n^1_i)$$

Forma general Backpropagation

Las ecuaciones anteriores se pueden generalizar a

$$\mathbf{W}_m^{\text{new}} = \mathbf{W}_m^{\text{old}} - \eta f_{m-1}(\mathbf{n}_{m-1}) \left(f_m' \left(\mathbf{n}_m^\top \right) \circ \boldsymbol{\delta}_m^\top \right), \tag{1}$$

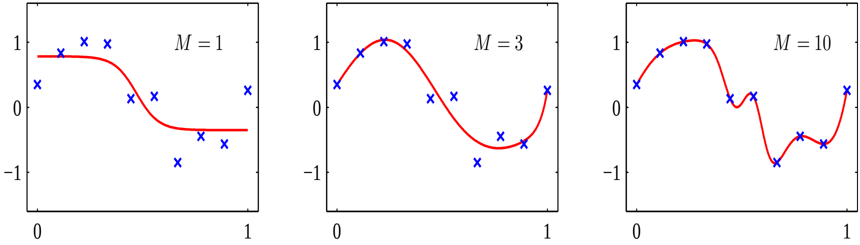
donde $\mathbf{x} \circ \mathbf{y}$ es el producto Hadamard, y

$$\boldsymbol{\delta}_M = \mathbf{y}_n - \mathbf{t}_n, \; \boldsymbol{\delta}_{M-1} = \mathbf{W}_M \boldsymbol{\delta}_M, \; \boldsymbol{\delta}_m = \mathbf{W}_{m+1} \boldsymbol{\delta}_{m+1}.$$

Contenido

Regularización

El numero de capas ocultas M influye en la cantidad de parámetros de la red neuronal.



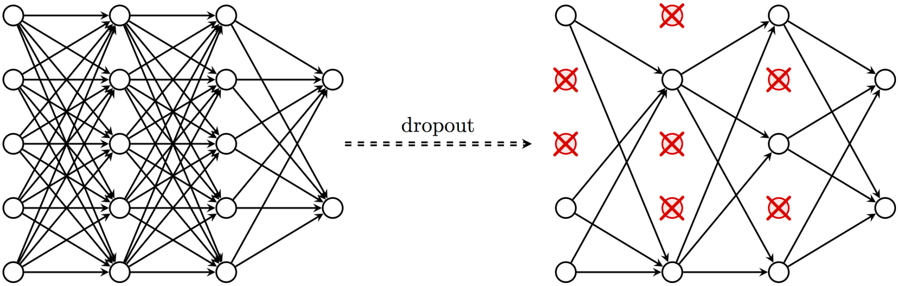
Regularización - Simple

- La regularización más simple tiene la forma

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

  La complejidad del modelo se determina mediante λ.

Regularización - DropOut 0.5



Regularización - Early stopping

El algoritmo de entrenamiento puede detenerse antes de terminar todas las epocas, con el fin de encontrar una solucion con buena capacidad de generalización.