Lecture 02
Artificial Intelligence

Agenda

- Aprendizaje supervisado
- Modelos neuronales
- Perceptron: Single layer NNs
- Álgebra lineal y cálculo para Deep Learning

Aprendizaje supervisado

$$\mathcal{D}_{train} = \left\{ \left(\mathbf{x}^{(\mathbf{i})}, y^{(i)} \right) \right\}$$

$$\min_{\theta} \mathcal{L}(\theta)$$

$$\mathbf{x}^{(i)} \longrightarrow \boxed{f\left(\theta, \mathbf{x}^{(i)}\right)} \longrightarrow \hat{y}^{(i)}$$

```
From:  pliang@cs.stanford.edu
Date:  September 25, 2019
Subject:  CS221 announcement

Hello students,
 Welcome to CS221!  Here's what...
```

```
From:  a9k62n@hotmail.com
Date:  September 25, 2019
Subject:  URGENT

Dear Sir or maDam:
 my friend left sum of 10m dollars...
```

Objetivos:

- Tipo de predictor f
- Función de costo $\mathcal{L}$
- Algoritmo de optimización

Tipos de tareas de predicción

Clasificación

$$\mathbf{x}^{(i)} \longrightarrow \boxed{f} \longrightarrow \begin{array}{l} \hat{y}^{(i)} \in \{+1, -1\} \\ \hat{y}^{(i)} \in \{0, 1, 2, \ldots, K\} \end{array}$$

Regresión (house pricing)

$$\mathbf{x}^{(i)} \longrightarrow \boxed{f} \longrightarrow \hat{y}^{(i)} \in \mathcal{R}$$

Ranking: y es una permutación

$$\boxed{1}\ \boxed{2}\ \boxed{3}\ \boxed{4} \longrightarrow \boxed{f} \longrightarrow 2\ 3\ 4\ 1$$

Predicción estructurada

la casa blu $\longrightarrow \boxed{f} \longrightarrow$ *the blue house*

Metodología

Entrenamiento

$$\mathcal{D}_{train} \longrightarrow \boxed{f\left(\theta, \mathbf{x}^{(i)}\right)} \longrightarrow \theta$$

Despliegue

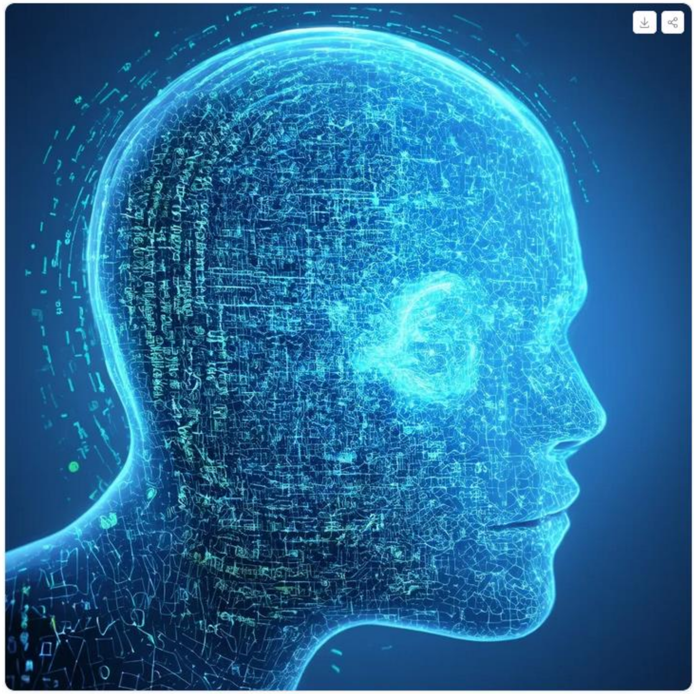
$$\mathbf{x}^{(test)} \longrightarrow \boxed{f\left(\theta, \mathbf{x}^{(test)}\right)} \longrightarrow y^{(test)}$$

Demo [Stable Diffusion 3 Medium]

Learn more about the [Stable Diffusion 3 series]. Try on [Stability AI API], [Stable Assistant], or on Discord via [Stable Artisan]. Run locally with [ComfyUI] or [diffusers]

artificial intelligence math

Run



Advanced Settings

Examples

Astronaut in a jungle, cold color palette, muted colors, detailed, 8k

An astronaut riding a green horse

A delicious ceviche cheesecake slice

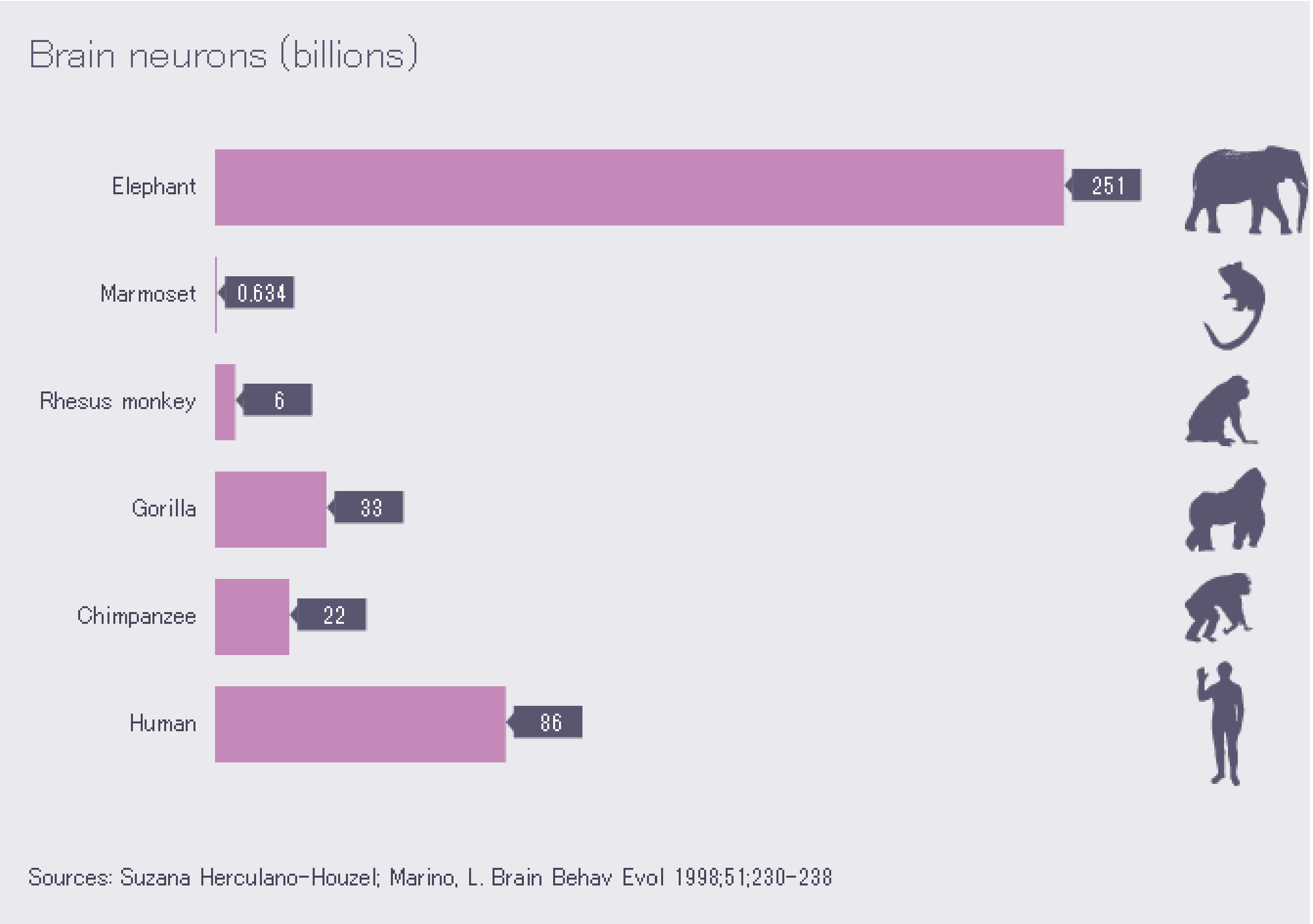
Redes Neuronales: Inspiradas por el cerebro

https://www.verywellmind.com/how-brain-cells-communicate-with-each-other-2584397

Redes Neuronales: Inspiradas por el cerebro

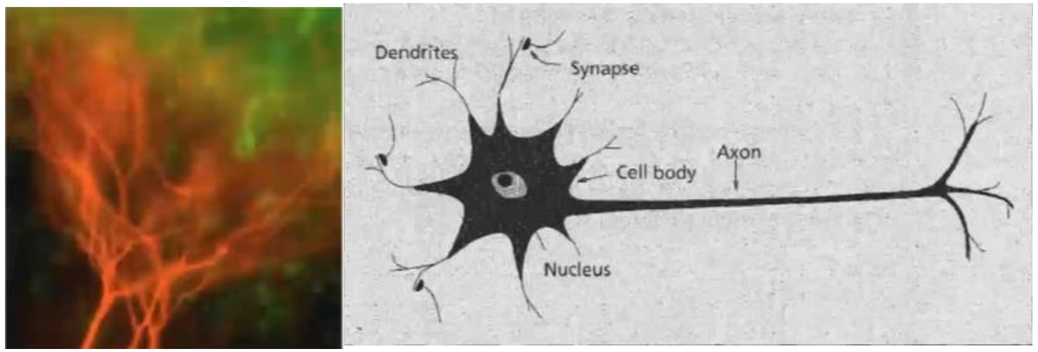
https://medium.com/@adsactly/is-it-a-bird-is-it-a-plane-biomimicry-in-airplanes-9862d331df2e

Número de neuronas en el cerebro



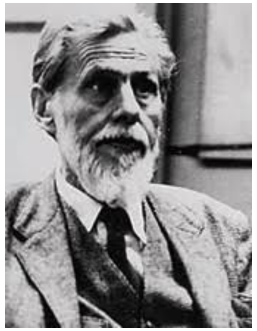
https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons#/media/File:Brain_size_comparison_-_Brain_neurons_(billions).png

Modelo de Neurona de McCulloch Pitts

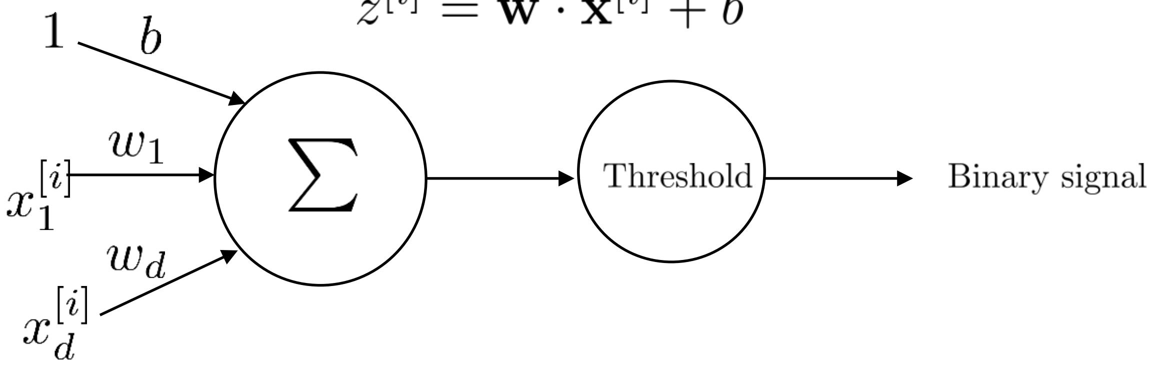


A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. McCULLOCH and WALTER H. PITTS 1943



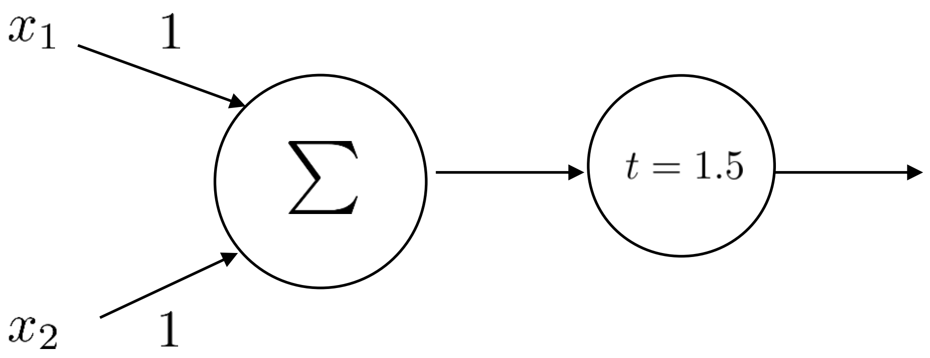
$$z^{[i]} = \mathbf{w} \cdot \mathbf{x}^{[i]} + b$$



Compuertas lógicas

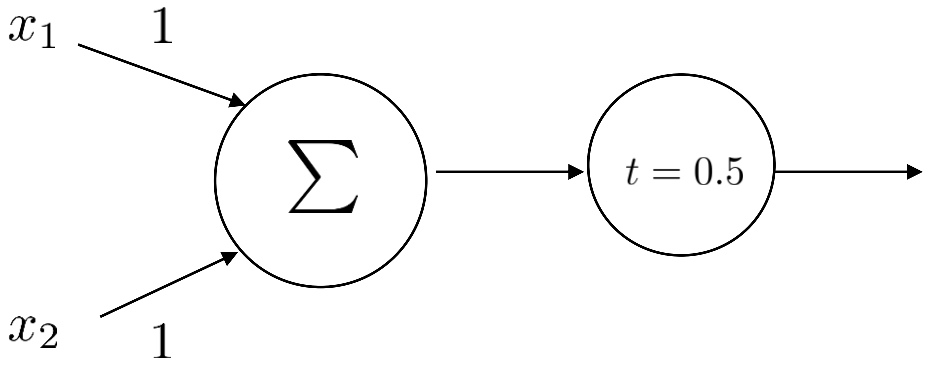
And

x_1	x_2	*Out*
0	0	0
0	1	0
1	0	0
1	1	1



Or

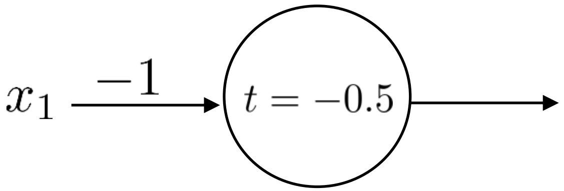
x_1	x_2	*Out*
0	0	0
0	1	1
1	0	1
1	1	1



Compuertas lógicas

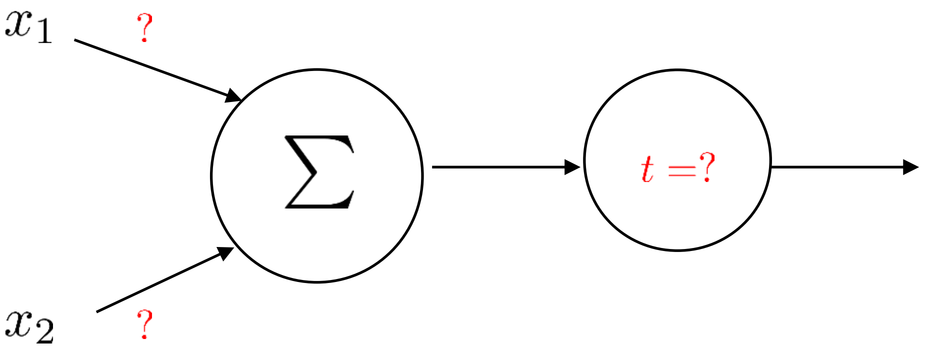
Not

x_1	Out
0	1
1	0



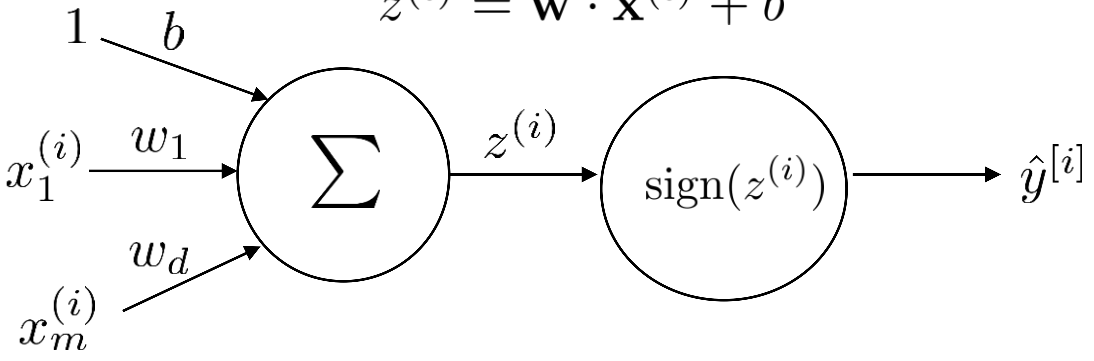
Xor

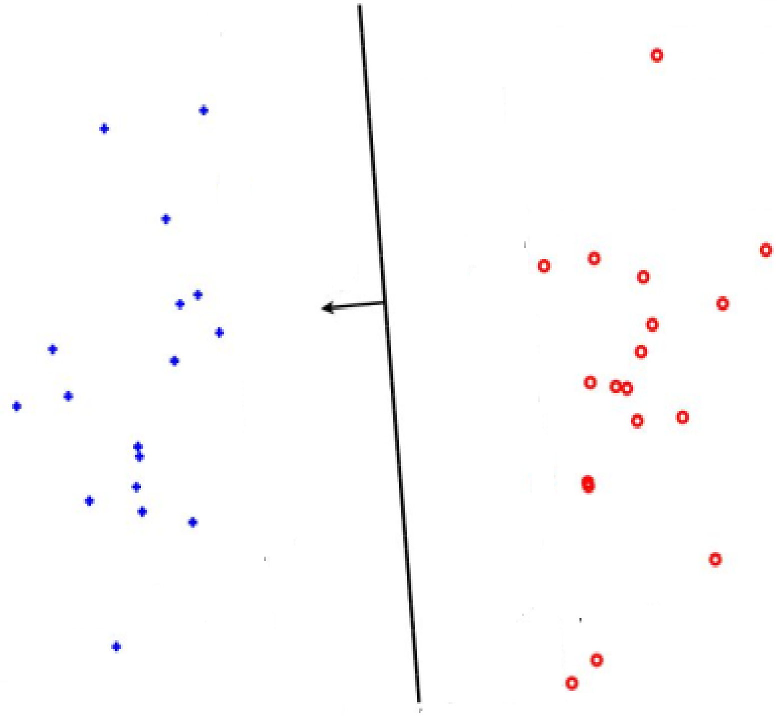
x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	0



Perceptrón

$$z^{(i)} = \mathbf{w} \cdot \mathbf{x}^{(i)} + b$$





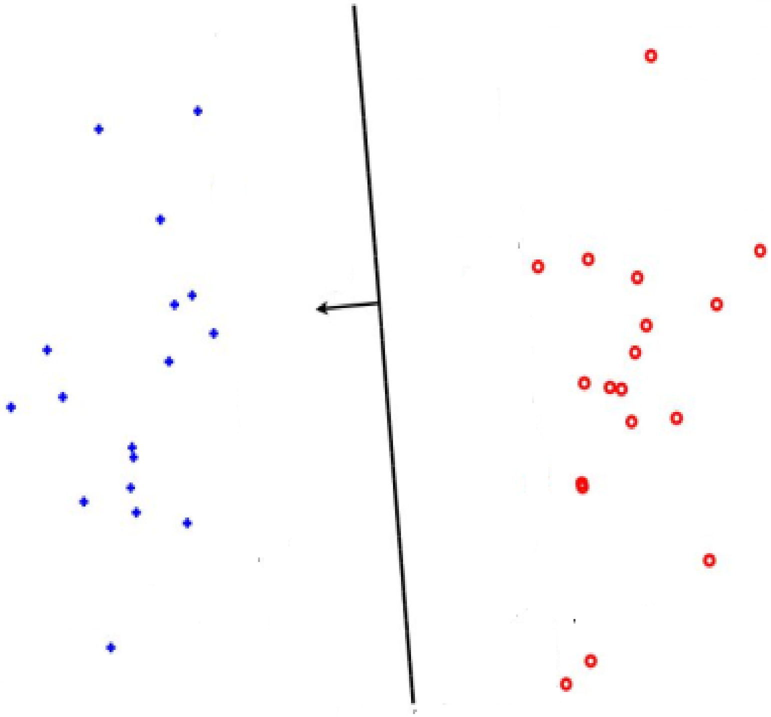
$$f_{\mathbf{w},w_0}(\mathbf{x}^{(i)}) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x}^{(i)} - w_0) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x}^{(i)} - w_0 > 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{x}^{(i)} - w_0 < 0 \\ ? & \text{if } \mathbf{w} \cdot \mathbf{x}^{(i)} - w_0 = 0 \end{cases}$$

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x}^{(i)} + b > 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{x}^{(i)} + b < 0 \\ ? & \text{if } \mathbf{w} \cdot \mathbf{x}^{(i)} + b = 0 \end{cases} \qquad b = -w_0$$

Perceptrón: Algoritmo

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}) | i = 1, \ldots, n\} \quad \mathbf{x}^{(i)} \in \Re^m \quad y^{[i]} \in \{-1, 1\}$$

$$\mathcal{L}(\mathbf{w}, b) = \begin{cases} -y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) & \text{if } y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \leq 0 \\ 0 & \textit{otherwise} \end{cases}$$



Algorithm 1 Perceptron Learning Algorithm

Input: $\mathbf{x}^{(i)} \in \Re^m, y^i \in \{+1, -1\}, i = 1, \ldots, n$
Result: $\mathbf{w}, b$
$\mathbf{w} = \mathbf{0}_m, b = 0$
for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
        instructions;
        if $y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \leq 0$ **then**
            $\mathbf{w} = \mathbf{w} + y^{(i)}\mathbf{x}^{(i)}$
            $b = b + y^{(i)}$
        else
            do nothing
        end if
    end for
end for

Vectorización

In [2]:

```
z = 0.
for i in range(len(x)):
    z += x[i] * w[i]

print(z)
```

```
2.2
```

In [3]:

```
z = sum(x_i*w_i for x_i, w_i in zip(x, w))
print(z)
```

```
2.2
```

In [4]:

```
import numpy as np

x_vec, w_vec = np.array(x), np.array(w)

z = (x_vec.transpose()).dot(w_vec)
print(z)

z = x_vec.dot(w_vec)
print(z)
```

```
2.2
2.2
```

Vectorización

In [6]:
```
%timeit -r 100 -n 10 forloop(x, w)
```

```
38.9 ms ± 1.32 ms per loop (mean ± std. dev. of 100 r
uns, 10 loops each)
```

In [7]:
```
%timeit -r 100 -n 10 listcomprehension(x, w)
```

```
29.7 ms ± 842 µs per loop (mean ± std. dev. of 100 ru
ns, 10 loops each)
```

In [8]:
```
%timeit -r 100 -n 10 vectorized(x_vec, w_vec)
```

```
46.8 µs ± 8.07 µs per loop (mean ± std. dev. of 100 r
uns, 10 loops each)
```

Implementación

```
class Perceptron:
    def __init__(self, num_features, learning_rate=0.01):
        self.num_features = num_features
        self.weights = np.zeros((num_features, 1), dtype=float)
        self.bias = np.zeros(1, dtype=float)
        self.learning_rate = learning_rate

    def forward(self, x):
        linear = np.dot(x,self.weights) + self.bias  # Compute net input
        predictions = np.where(linear > 0., 1, -1)
        return predictions

    def backward(self, x, y):
        linear = np.dot(x,self.weights) + self.bias  # Compute net input
        cost = -y*linear
        if cost<0:
          errors = 0
        else:
          errors = 1
        grad_w = -1*errors*y*x
        grad_b = -1*errors*y
        return grad_w, grad_b

    def train(self, x, y, epochs):
        for e in range(epochs):
            for i in range(y.shape[0]):
                grad_w, grad_b = self.backward(x[i].reshape(1,self.num_features), y[i])
                self.weights -= grad_w.reshape(self.num_features, 1)
                self.bias -= grad_b

    def evaluate(self, x, y):
        predictions = self.forward(x).reshape(-1)
        accuracy = np.sum(predictions == y) / y.shape[0]
        return accuracy
```

Perceptrón: Fun Fact

[...] Where a perceptron had been trained to distinguish between - this was for military purposes - it was looking at a scene of a forest in which there were camouflaged tanks in one picture and no camouflaged tanks in the other. And the perceptron - after a little training - made a 100% correct distinction between these two different sets of photographs. Then they were embarrassed a few hours later to discover that the two rolls of film had been developed differently. And so these pictures were just a little darker than all of these pictures and the perceptron was just measuring the total amount of light in the scene. But it was very clever of the perceptron to find some way of making the distinction.

-- Marvin Minsky, AI researcher & author of the "Perceptrons" book

Source: https://www.webofstories.com/play/marvin.minsky/122

Perceptrón: Fun Fact

https://qph.fs.quoracdn.net/main-qimg-305eb8136c4a20f348bb7ab465bc2e10

http://theconversation.com/want-to-beat-climate-change-protect-our-natural-forests-121491

Principio general de aprendizaje

Como principio general de aprendizaje, aplicable a todos los modelos comunes de neuronas y arquitecturas de redes neuronales (profundas y no profundas):

Sea

$$\mathcal{D} = ((\mathbf{x}^{[1]}, y^{[1]}), (\mathbf{x}^{[2]}, y^{[2]}), \ldots, (\mathbf{x}^{[n]}, y^{[n]})) \in (\Re^m \times \{0, 1\})^n$$

Modo *On-line*

Result: $\mathbf{w}, b$
$\mathbf{w} := \mathbf{0} \in \Re^m, \mathbf{b} := 0;$
for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
        cálculo de la salida;
        cálculo del error ;
        actualización de parámetros $\mathbf{w}, b$;
    end
end

Modo *On-line*

Result: $\mathbf{w}, b$

$\mathbf{w} := \mathbf{0} \in \Re^m, \mathbf{b} := 0$;

for $t = 1, \ldots, T$ **do**

  for $i = 1, \ldots, n$ **do**

    cálculo de la salida;

    cálculo del error ;

    actualización de parámetros $\mathbf{w}, b$;

  end

end

Modo *On-line* II

Result: $\mathbf{w}, b$

$\mathbf{w} := \mathbf{0} \in \Re^m, \mathbf{b} := 0$;

for j *iteraciones* **do**

  Elija un $(\mathbf{x}^{[i]}, y^{[i]}) \in \mathcal{D}$ aleatorio;

    cálculo de la salida;

    cálculo del error ;

    actualización $\mathbf{w}, b$;

end

Modo *On-line*

Result: $\mathbf{w}, b$
$\mathbf{w} := \mathbf{0} \in \Re^m, \mathbf{b} := 0;$
for $t = 1, \ldots, T$ **do**
  for $i = 1, \ldots, n$ **do**
    cálculo de la salida;
    cálculo del error ;
    actualización de parámetros $\mathbf{w}, b$;
  end
end

Modo *Batch*

Result: $\mathbf{w}, b$
$\mathbf{w} := \mathbf{0} \in \Re^m, \mathbf{b} := 0;$
for $t = 1, \ldots, T$ **do**
  $\Delta\mathbf{w} := 0, \Delta b := 0;$
  for $i = 1, \ldots, n$ **do**
    cálculo de la salida;
    cálculo del error ;
    actualización de parámetros $\Delta\mathbf{w}, \Delta b$;
  end
  actualización de parámetros $\mathbf{w}, b$;
  $\mathbf{w} := \mathbf{w} + \Delta\mathbf{w} \;\; b := b + \Delta b;$
end

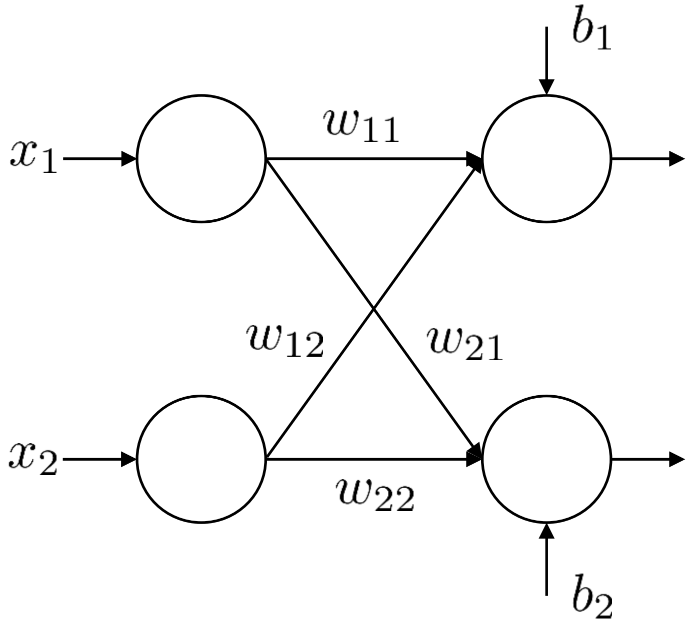
Modo *minibatch*

Modo más común en Deep Learning. Combina *On-line* y *Batch*.

$$\mathcal{D} = ((\mathbf{x}^{[1]}, y^{[1]}), (\mathbf{x}^{[2]}, y^{[2]}), \ldots, (\mathbf{x}^{[n]}, y^{[n]})) \in (\Re^m \times \{0,1\})^n$$

Result: $\mathbf{w}, b$
$\mathbf{w} := \mathbf{0} \in \Re^m, \mathbf{b} := 0;$
for $t = 1, \ldots, T$ **do**
    for $j = 1, \ldots, n/k$ **do**
        $\Delta\mathbf{w} := 0, \Delta b := 0;$
        for $\{(\mathbf{x}^{[i]}, y^{[i]}), \ldots (\mathbf{x}^{[i+k]}, y^{[i+k]})\} \subset D$ **do**
            cálculo de la salida;
            cálculo del error ;
            actualización de $\Delta\mathbf{w}, \Delta b$;
        end
        actualización $\mathbf{w}, b$;
        $\mathbf{w} := \mathbf{w} + \Delta\mathbf{w};$
        $b := b + \Delta w;$
    end
end

Capa Fully Connected



$$z_1 = w_{11}x_1 + w_{12}x_2 + b_1$$

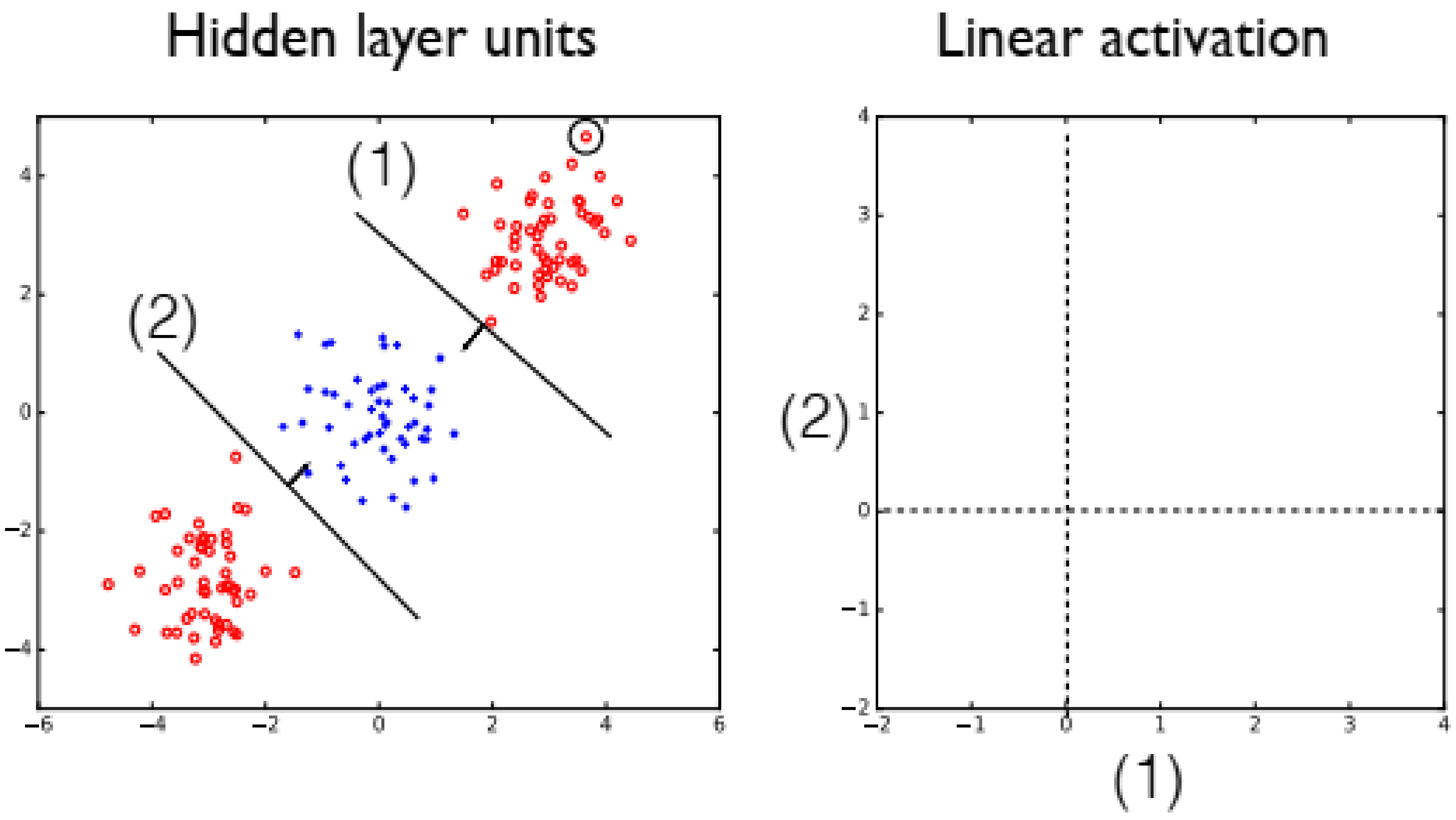
$$z_2 = w_{21}x_1 + w_{22}x_2 + b_2$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

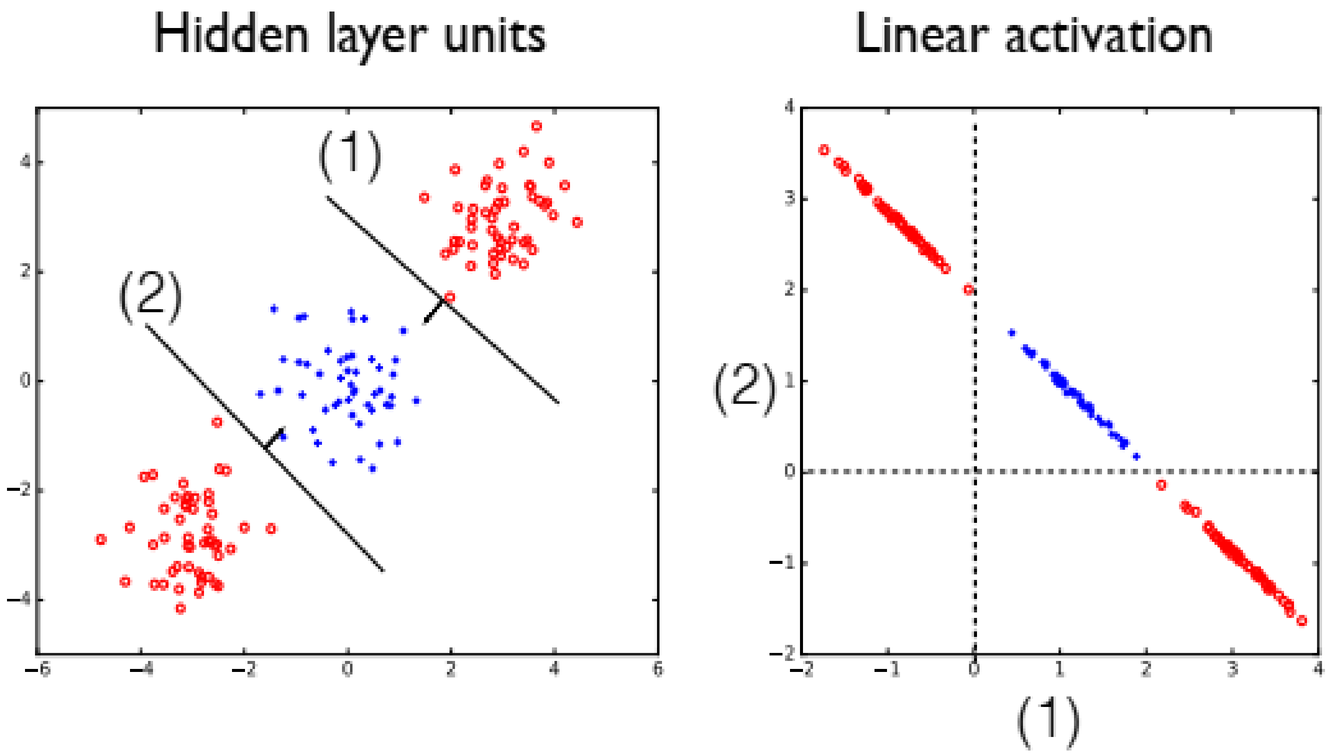
$\mathbf{z} \in \Re^{h_{out} \times 1}$ $\mathbf{W} \in \Re^{h_{out} \times h_{in}}$ $\mathbf{x} \in \Re^{h_{in} \times 1}$ $\mathbf{b} \in \Re^{h_{out} \times 1}$

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

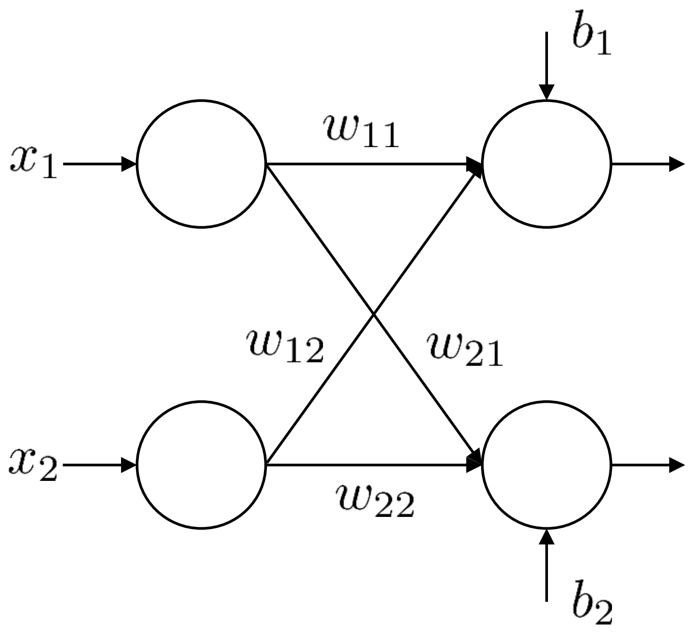
Capa Fully Connected



Capa Fully Connected



Capa Fully Connected



$$\mathbf{X} \in \Re^{n \times h_{in}}$$

$$\mathbf{X} = \begin{bmatrix} x_1^{[1]} & \dots & x_{h_{in}}^{[1]} \\ \vdots & \vdots & \vdots \\ x_1^{[n]} & \dots & x_{h_{in}}^{[n]} \end{bmatrix}$$

$$\mathbf{W} \in \Re^{h_{out} \times h_{in}}$$

$$\mathbf{w}_h \in \Re^{h_{in} \times 1}$$

$$\mathbf{W} = \begin{bmatrix} - - \mathbf{w}_1^\top - - \\ \vdots \\ - - \mathbf{w}_{h_{out}}^\top - - \end{bmatrix}$$

$$\mathbf{b} \in \Re^{h_{out} \times 1}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{h_{out}} \end{bmatrix}$$

$$\mathbf{Z} \in \Re^{n \times h_{out}}$$

$$\mathbf{Z} = \mathbf{X}\mathbf{W}^\top + \mathbf{b}$$