Rodolfo Ferro
ferro@cimat.mx
https://rodolfoferro.xyz

Diplomado en Ciencia de Datos
Escuela Nacional de Estudios Superiores, Unidad León

Unidad León
Escuela Nacional de Estudios Superiores

Agosto, 2024

Aprendizaje profundo

Módulo 5

Tutor del módulo

Aprendizaje profundo

Rodolfo Ferro (ferro@cimat.mx)

- Sr. SWE (Data Engineer) @ Bisonic México
- Miembro del Consejo Consultivo para el Desarrollo Económico, Creatividad e Innovación de León
- **Formación:** BMath, CSysEng, StatMethodsSpc (*ongoing*)
- **Experiencia:** ML Engineer @ Vindoo.ai (España), Sherpa Digital en IA @ Microsoft México, AI Research Assistant @ CIMAT & AI Research Intern @ Harvard.

Tabla de contenidos

ferro@cimat.mx

Motivación

Intro al aprendizaje profundo

```
if(speed<4){
    status=WALKING;
}
```

Motivación

Intro al aprendizaje profundo

```
if(speed<4){
    status=WALKING;
}
```

```
if(speed<4){
    status=WALKING;
} else {
    status=RUNNING;
}
```

ferro@cimat.mx

Motivación

Intro al aprendizaje profundo

```
if(speed<4){
    status=WALKING;
}
```

```
if(speed<4){
    status=WALKING;
} else {
    status=RUNNING;
}
```

```
if(speed<4){
    status=WALKING;
} else if(speed<12){
    status=RUNNING;
} else {
    status=BIKING;
}
```

ferro@cimat.mx

Motivación

Intro al aprendizaje profundo

```
if(speed<4){
    status=WALKING;
}
```

```
if(speed<4){
    status=WALKING;
} else {
    status=RUNNING;
}
```

```
if(speed<4){
    status=WALKING;
} else if(speed<12){
    status=RUNNING;
} else {
    status=BIKING;
}
```

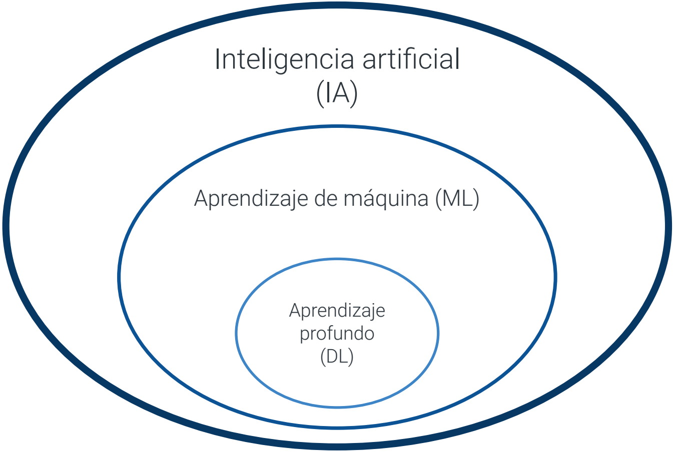
```
// ????
```

ferro@cimat.mx

¿Qué es el *Deep Learning*?

> *El aprendizaje profundo (Deep Learning) comprende algoritmos de Machine Learning que (particularmente) utilizan múltiples capas apiladas de unidades de procesamiento para aprender representaciones en un alto nivel sobre datos no estructurados.*
>
> David Foster (Generative Deep Learning)

¿Qué es el *Deep Learning*?



¿Qué es el *Deep Learning*?

- **Inteligencia artificial:** Cualquier técnica que permita a las computadoras emular o imitar el comportamiento humano.
- **Aprendizaje de máquina:** Capacidad de aprender sin ser programado explícitamente, enfoque en los algoritmos y la matemtica.
- **Aprendizaje profundo:** Extrae patrones de datos utilizando redes neuronales.

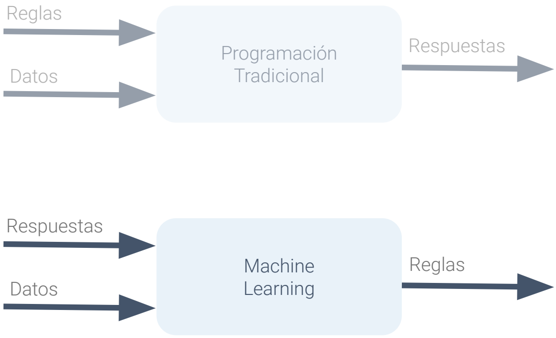
ferro@cimat.mx

¿Por qué el *Deep Learning*?

ferro@cimat.mx

¿Por qué el *Deep Learning*?

Intro al aprendizaje profundo



ferro@cimat.mx

¿Por qué el *Deep Learning*?

- La ingeniería de características requiere mucho tiempo, es suceptible a errores y no es escalable consistentemente con datos complejos. Mejor busquemos aprender las características subyacentes directamente de los datos.
- Las redes neuronales artificales existen desde hace décadas, pero su predominio actual reside principalmente en los siguientes tres aspectos:
  1. Hardware (GPUs, etc. + Paralelización)
  2. Software (Frameworks para trabajar con NNs)
  3. Grandes cantidades de datos
- El aprendizaje profundo está revolucionando muchos campos.

ferro@cimat.mx

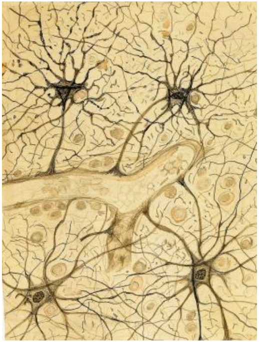
Contexto histórico

Intro al aprendizaje profundo

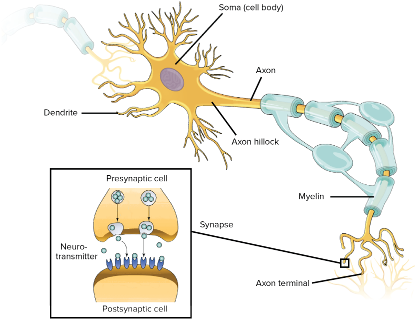
Figure: Santiago Ramón y Cajal

Contexto histórico

Intro al aprendizaje profundo



→



“Santiago Ramón y Cajal Drawings.” Janelia Research Campus. Accessed July 5, 2024. https://www.janelia.org

“Overview of Neuron Structure and Function (Article).” Khan Academy. Accessed July 5, 2024. https://www.khanacademy.org

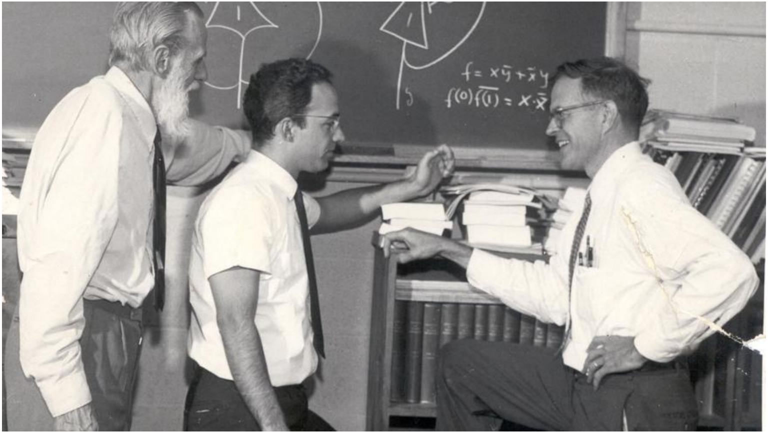
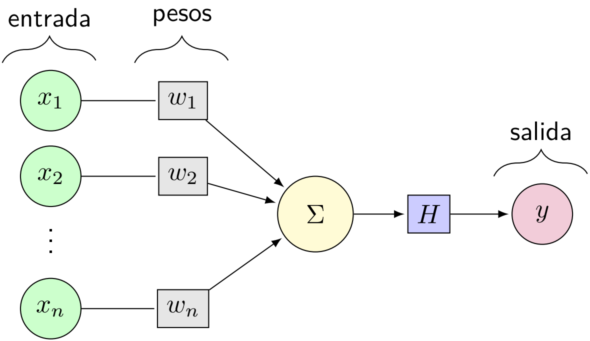
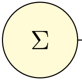


Figure: Warren McCulloch & Walter Pitts





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

Bias y función de activación

La operación matemática que realiza la neurona para la decisión de umbralización se puede escribir como:

$$f(\mathbf{x}) = \begin{cases} 0 & \text{si } \sum_i w_i x_i < \text{umbral o threshold} \\ 1 & \text{si } \sum_i w_i x_i \geq \text{umbral o threshold} \end{cases}$$

donde $i \in \{1, 2, ..., n\}$, y así, $\mathbf{x} = (x_1, x_2, ..., x_n)$.

ferro@cimat.mx

Bias y función de activación

De lo anterior, podemos despejar el umbral y escribirlo como b, obteniendo:

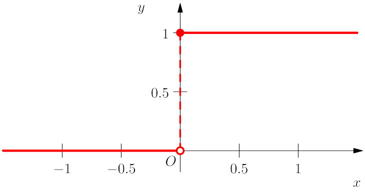
$$f(\mathbf{x}) = \begin{cases} 0 & \text{si } \sum_i w_i x_i + b < 0 \\ 1 & \text{si } \sum_i w_i x_i + b > 0 \end{cases}$$

donde $\mathbf{x} = (x_1, x_2, ..., x_n)$ y $i \in \{1, 2, ..., n\}$.
A esto que escribimos como b, también se le conoce como **bias**, y una interpretación es describir *qué tan susceptible es la neurona a **dispararse*** (como se exploró en el ejemplo práctico de la identificación de la actividad de Julieta).

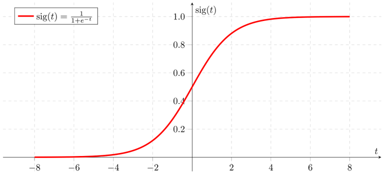
ferro@cimat.mx

Bias y función de activación

Intro al aprendizaje profundo



$\longrightarrow$



ferro@cimat.mx

El Perceptrón

Intro al aprendizaje profundo

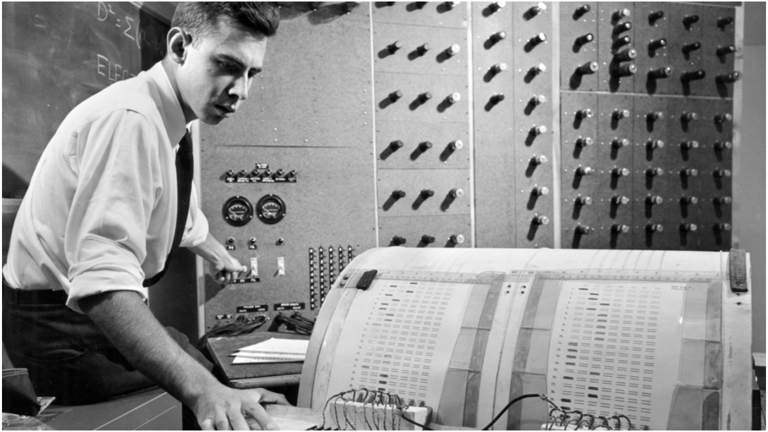
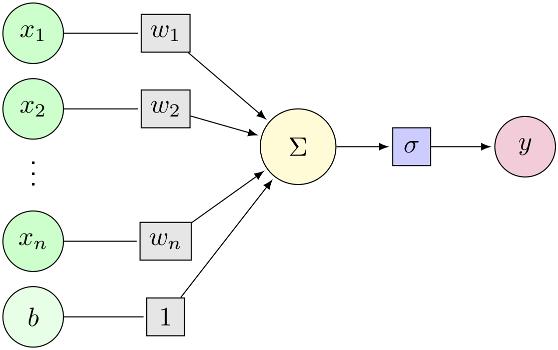


Figure: Frank Rosenblatt

ferro@cimat.mx

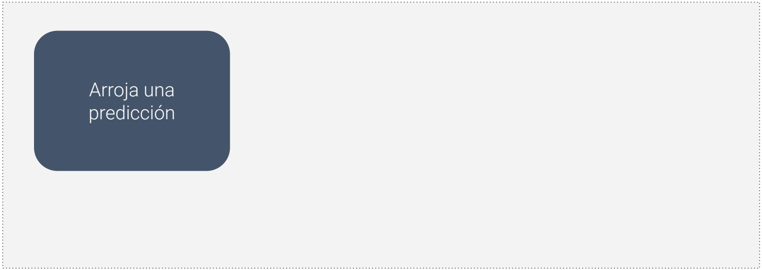
Unidad León Escuela Nacional de Estudios Superiores

El Perceptrón

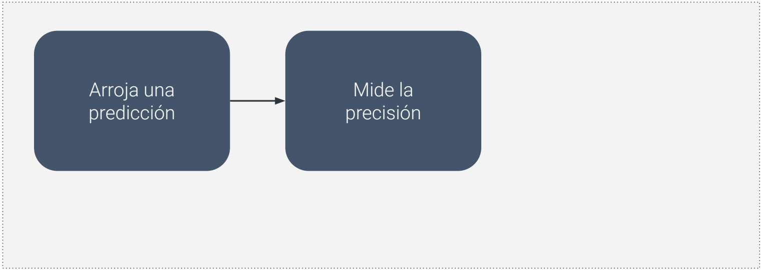


¡Y se agrega un algoritmo formal de entrenamiento!
(*Backpropagation*)

Idea intuitiva de entrenamiento

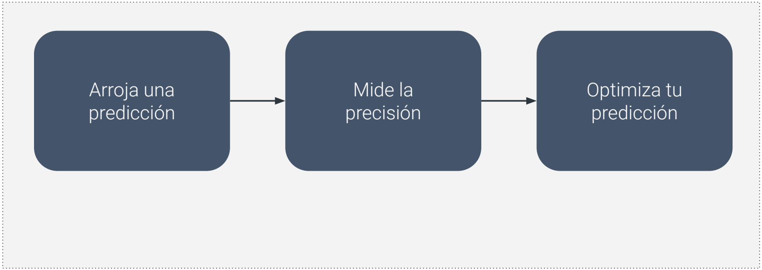


Idea intuitiva de entrenamiento

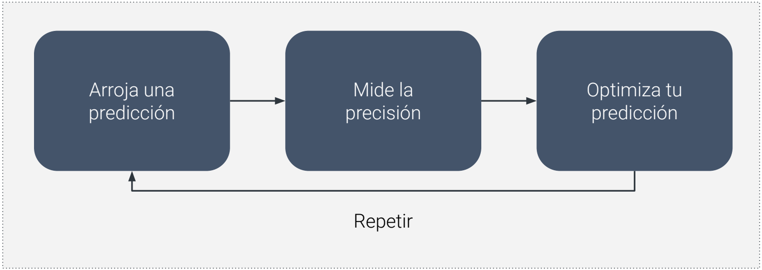


Idea intuitiva de entrenamiento

Intro al aprendizaje profundo



Idea intuitiva de entrenamiento



ferro@cimat.mx

Dado el vector X, **¿qué vector (**A, B, C**) se le *parece* más?**

$$X = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.7 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.3 \\ 0.3 \\ 0.3 \end{bmatrix}, B = \begin{bmatrix} 0.6 \\ 0.2 \\ 0.6 \end{bmatrix}, C = \begin{bmatrix} -0.5 \\ -0.3 \\ -0.7 \end{bmatrix}$$

Medición del error

Dado el vector X, **¿qué vector (**A, B, C**) se le *parece* más?**

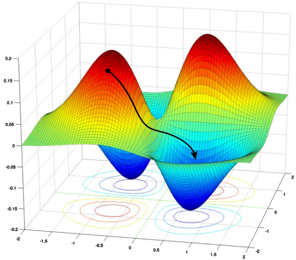
$$X = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.7 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.3 \\ 0.3 \\ 0.3 \end{bmatrix}, \; B = \begin{bmatrix} 0.6 \\ 0.2 \\ 0.6 \end{bmatrix}, C = \begin{bmatrix} -0.5 \\ -0.3 \\ -0.7 \end{bmatrix}$$

ferro@cimat.mx

Optimización del error

- **Error:** Es una función.
- **Optimizar:** Maximizar o minimizar.
- **Gradiente:** Derivada de una función vectorial, proporciona información sobre máximos o mínimos.
- **Descenso de gradiente:** Algoritmo para, iterativamente, buscar optimizar una función.
- **Limitantes:**
  - Max's/min's locales.
  - Tamaño de salto en gradiente



Deep Learning — R. Ferro (@rodo_ferro)

Escuela Nacional de Estudios Superiores Unidad León

Observaciones

Hasta este punto, debemos notar que hay algunas observaciones importantes:

- **TLUs:**
  - No existe un algoritmo de aprendizaje formal $\rightarrow$ Búsqueda de pesos.
  - Se limita a propagación hacia adelante (*forward pass/forward propagation*)
- **Perceptrón:** Puede utilizar retropropagación, introducido en 1958.
- **Retropropagación:** Algoritmo para realizar ajustes en los valores de los pesos.
- **Limitantes:** Separabilidad lineal.
- **¿Alguna otra observación?**

ferro@cimat.mx

Ejercicio: Manzanas vs. Naranjas

ferro@cimat.mx

Lecturas recomendadas

- Breve historia sobre el perceptrón
- Post sobre el perceptrón de Rosenblatt
- Post sobre la función de activación
- Selección de threshold para clasificadores binarios
- Post sobre redes neuronales por IBM

ferro@cimat.mx

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

Producto matricial

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & & \\ & & \\ & & \end{bmatrix}$$

$$(2 \cdot -2) + (5 \cdot -2) + (2 \cdot 0) = -14$$

ferro@cimat.mx

Producto matricial

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & \\ & & \\ & & \end{bmatrix}$$

$$(2 \cdot 1) + (5 \cdot 2) + (2 \cdot 0) = 12$$

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \\ & & \\ & & \end{bmatrix}$$

$$(2 \cdot 0) + (5 \cdot 1) + (2 \cdot 3) = 11$$

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \\ -2 & & \\ & & \end{bmatrix}$$

$$(1 \cdot -2) + (0 \cdot -2) + (-2 \cdot 0) = -2$$

Producto matricial

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \\ -2 & & \\ & & \end{bmatrix}$$

$$\ldots$$

ferro@cimat.mx

Producto matricial

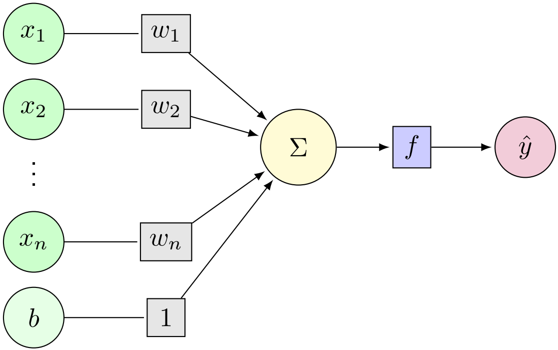
Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \\ -2 & 1 & -6 \\ -8 & 5 & 4 \end{bmatrix}$$

ferro@cimat.mx

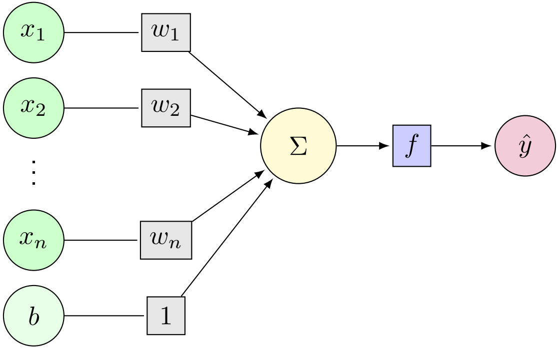
El Perceptrón

Intro al aprendizaje profundo



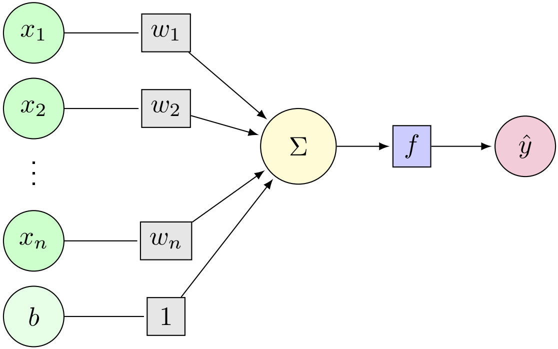
$$\hat{y} = f\left(\sum_{i}^{n} w_i x_i + b\right)$$

El Perceptrón



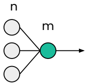
$$\sum_{i}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

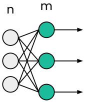
El Perceptrón



$$\mathbf{w^T x} = [w_1 w_2 \cdots w_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

Producto matricial



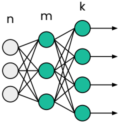


$$\mathbf{W}\cdot\mathbf{X}+\mathbf{b}=\begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}\cdot\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}+b \longrightarrow \mathbf{W}_{Layer}\cdot\mathbf{X}+\mathbf{b}=\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix}\cdot\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}+\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$f(\cdot)$$

Producto matricial

Intro al aprendizaje profundo



$$\mathbf{W}_{Layer_k} \cdot (\mathbf{W}_{Layer_m} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \dots & w_{k,m} \end{bmatrix} \left(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

$$\boxed{f(\cdot)}$$

Composición de funciones

Intro al aprendizaje profundo

$$\mathbf{W}_{Layer_k} \cdot (\mathbf{W}_{Layer_m} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \dots & w_{k,m} \end{bmatrix} \left(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

$$\downarrow$$

$$f_k(\mathbf{W}_{Layer_k} \cdot f_m(\mathbf{W}_{Layer_m} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k)$$

$$\downarrow$$

$$\boxed{f_n(\cdots (f_2(\mathbf{W}_{Layer_k} \cdot f_1(\mathbf{W}_{Layer_1} \cdot \mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2))) + \mathbf{b}_n}$$

ferro@cimat.mx

Composición de funciones

Intro al aprendizaje profundo

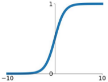
$$f_n(\cdots(f_2(\mathbf{W}_{Layer_k} \cdot f_1(\mathbf{W}_{Layer_1} \cdot \mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2))) + \mathbf{b}_n$$

$$\boxed{F'(x) = f'(g(x)) \cdot g'(x)}$$

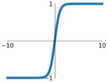
ferro@cimat.mx

Funciones de activación

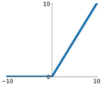
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



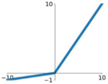
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

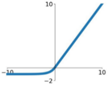


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

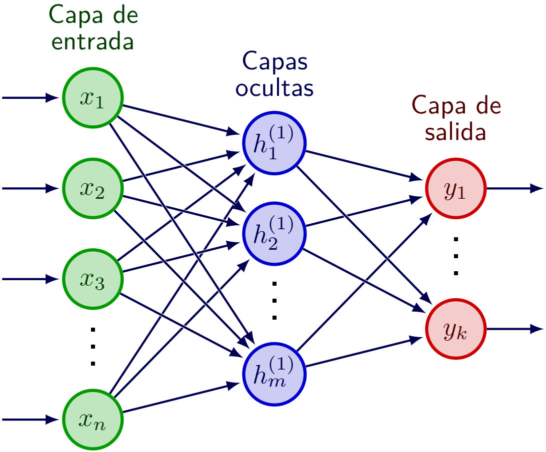
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



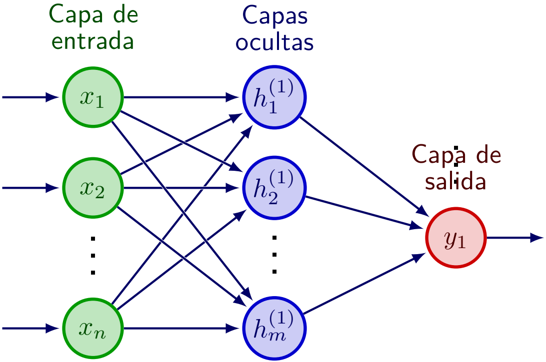
$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \Rightarrow f'(x) = f(x)(1 - f(x))$$

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \Rightarrow f'(x) = 1 - f(x)^2$$

El perceptrón multicapa



El perceptrón multicapa



Para $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$, $y^{(i)}$, tendríamos que la salida es $\hat{y}_1^{(i)} = f(x^{(i)})$.

Cuantificación del error

- La **función de pérdida (*loss*)** de nuestra red neuronal *mide* el *costo* asociado a predicciones incorrectas.
- Si observaciones (de entrada y salida) $(x^{(i)}, y^{(i)})$ y consideramos a la salida como función de $x^{(i)}$ y $\mathbf{W}$, entonces las salidas son $\hat{y} = f(x^{(i)}; \mathbf{W})$ y la función de pérdida puede escribirse como:

$$\mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

Es decir, una función que mida la salida *real* con la *predicción*. Todo esto para una observación i.

- Para todas las observaciones:

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(f(x^{(i)};\mathbf{W}), y^{(i)})$$

A esta función se le conoce como función de costo o función objetivo (lo que queremos minimizar).

Algunos ejemplos

- ***Binary Cross Entropy Loss***: Se puede utilizar con modelos que devuelven como salida una probabilidad entre 0 y 1.

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)}\log(f(x^{(i)};\mathbf{W})) + (1-y^{(i)})\log(1-f(x^{(i)};\mathbf{W}))$$

- ***Mean Squared Error (MSE) Loss***: Se puede utilizar con modelos de regresión que generan números reales continuos.

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f(x^{(i)};\mathbf{W})\right)^2$$

ferro@cimat.mx

- Queremos encontrar los pesos ideales de la red neuronal, los cuales minimizan $\mathbf{J}(\mathbf{W})$, es decir:

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

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \mathbf{J}(\mathbf{W})$$

Algoritmo: Descenso de gradiente

1. Inicializar los pesos aleatoriamente $\sim \mathcal{N}(0, \sigma^2)$
2. Repetir hasta converger:
3. Calcular el gradiente $\dfrac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
4. Actualizar los pesos $\mathbf{W} \leftarrow \mathbf{W} - \eta \dfrac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
5. Devolver pesos *óptimos*

Retropropagación

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \longrightarrow \mathbf{J}(\mathbf{W})$$

¿Cómo se calculan los gradientes?
Con la regla de la cadena.

ferro@cimat.mx

Retropropagación

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \longrightarrow \mathbf{J}(\mathbf{W})$$

$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial w_2} = \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

ferro@cimat.mx

Retropropagación

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \longrightarrow \mathbf{J}(\mathbf{W})$$

$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial w_1} = \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$$

ferro@cimat.mx

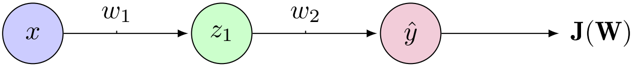
Retropropagación

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \longrightarrow \mathbf{J}(\mathbf{W})$$

$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial w_1} = \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

ferro@cimat.mx

Retropropagación



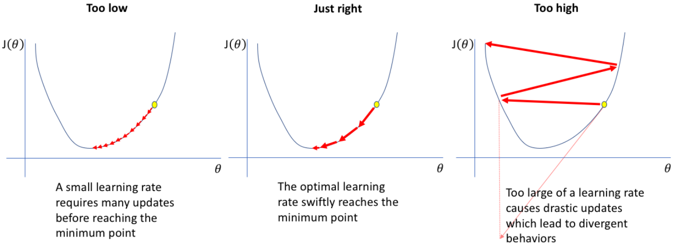
¿Cómo se calculan los gradientes?
Con la regla de la cadena.
Esto se repite para **cada peso** en la red neuronal, usando los gradientes de las capas posteriores.

ferro@cimat.mx

Learning rate

La actualización de pesos está dada por:

$$\mathbf{W} \leftarrow \mathbf{W} - \boxed{\eta} \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$$



Escuela Nacional de Estudios Superiores, Unidad León

Algoritmo: Descenso de gradiente

1. Inicializar los pesos aleatoriamente $\sim \mathcal{N}(0, \sigma^2)$
2. Repetir hasta converger:
3. Calcular el gradiente $\frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}^*$
4. Actualizar los pesos $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
5. Devolver pesos *óptimos*

- *Esto es muy pesado de calcular (computacionalmente).

ferro@cimat.mx

Descenso de gradiente estocástico

Algoritmo: Descenso de gradiente estocástico

1. Inicializar los pesos aleatoriamente $\sim \mathcal{N}(0, \sigma^2)$
2. Repetir hasta converger:
3. Seleccionar observación i
4. Calcular el gradiente $\frac{\partial \mathbf{J_i}(\mathbf{W})}{\partial \mathbf{W}}^*$
5. Actualizar los pesos $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
6. Devolver pesos *óptimos*

- *Esto es muy sencillo de calcular (computacionalmente), pero es estocástico.

ferro@cimat.mx

Algoritmo: Descenso de gradiente estocástico - *Mini batches*

1. Inicializar los pesos aleatoriamente $\sim \mathcal{N}(0, \sigma^2)$
2. Repetir hasta converger:
3. Seleccionar un batch B de observaciones
4. Calcular el gradiente $\boxed{\dfrac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B}\sum_{k=1}^{B} \dfrac{\partial \mathbf{J_k}(\mathbf{W})}{\partial \mathbf{W}}^{*}}$
5. Actualizar los pesos $\mathbf{W} \leftarrow \mathbf{W} - \eta \dfrac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
6. Devolver pesos *óptimos*

- *Esto es rápido de calcular (computacionalmente), y da una mejor estimación del gradiente.

Observaciones

- Los **frameworks** para aprendizaje profundo (como TensorFlow, PyTorch, etc.) ya hacen la diferenciación y optimización por nosotros, es decir, ya calculan el gradiente y actualizan los pesos.
- Nosotros exploraremos el uso de **TensorFlow** a través de su API de alto nivel, **Keras**, para las redes neuronales que estaremos construyendo.
- Comenzaremos retomando algunos de los problemas planteados en la sesión anterior.

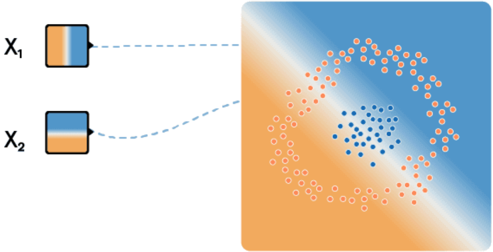
TensorFlow

TensorFlow

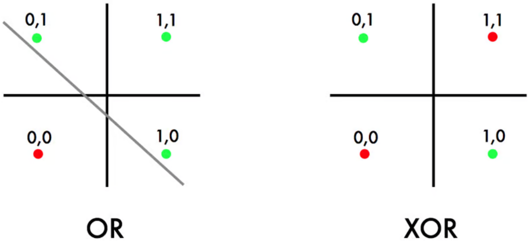
TensorFlow es un framework open-source para Machine Learning desarrollado por Google. Utilizado para construir y entrenar redes neuronales artificiales.

Deep Learning — R. Ferro (@rodo_ferro)
ferro@cimat.mx

Ejercicio: Exploración del TensorFlow Playground



Ejercicio: Problema de separabilidad lineal



Ejercicio: Exploración con TensorFlow

TensorFlow

Lecturas recomendadas

- Setting the learning rate of your neural network
- Retropropagación paso a paso
- TensorFlow Tutorials
- Libro Neural Networks and Deep Learning

ferro@cimat.mx