



Escuela
Nacional de
Estudios
Superiores

Rodolfo Ferro

ferro@cimat.mx

<https://rodolfoferro.xyz>

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

Agosto, 2024

Aprendizaje profundo

Módulo 5

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.



ferro@cimat.mx



Tabla de contenidos

- 1 Intro al aprendizaje profundo
- 2 Visión computacional profunda

- 3 Modelado profundo de secuencias
- 4 Modelado generativo profundo
- 5 Panorama actual y futuro

- 1 Intro al aprendizaje profundo
 - Motivación
 - Introducción
 - Contexto histórico
 - Perceptrón
 - Perceptrón multicapa
 - Aprendizaje
 - Regularización
 - Regularización
- 2 Visión computacional profunda
- 3 Modelado profundo de secuencias
- 4 Modelado generativo profundo
- 5 Panorama actual y futuro





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



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



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



```
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;  
}
```



```
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;  
}
```



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


¿Qué es el *Deep Learning*?

Intro al aprendizaje profundo

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.

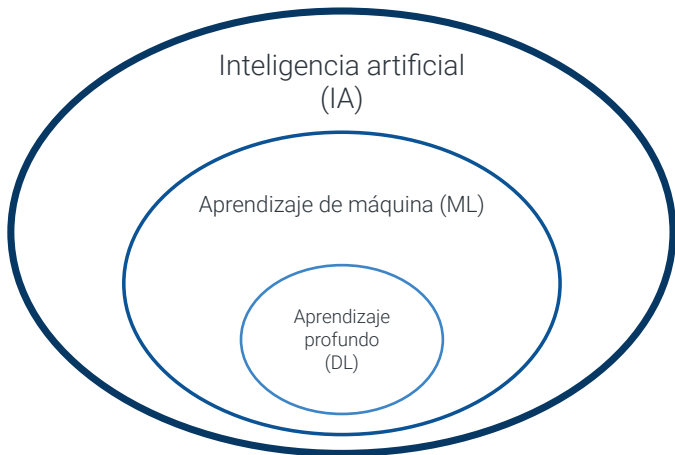
David Foster (Generative Deep Learning)



Escuela
Nacional de
Estudios
Superiores

¿Qué es el *Deep Learning*?

Intro al aprendizaje profundo



ferro@ciimat.mx



Escuela
Nacional de
Estudios
Superiores

¿Qué es el *Deep Learning*?

Intro al aprendizaje profundo

- **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 matemática.
- **Aprendizaje profundo:** Extrae patrones de datos utilizando redes neuronales.



Escuela
Nacional de
Estudios
Superiores

¿Por qué el *Deep Learning*?

Intro al aprendizaje profundo



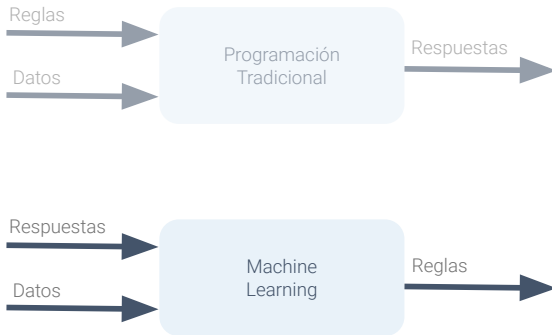
ferro@ciimat.mx



Escuela
Nacional de
Estudios
Superiores

¿Por qué el *Deep Learning*?

Intro al aprendizaje profundo



ferro@ciimat.mx



¿Por qué el *Deep Learning*?

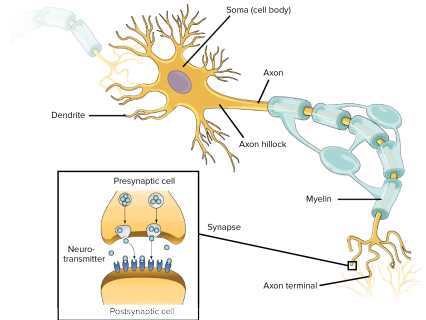
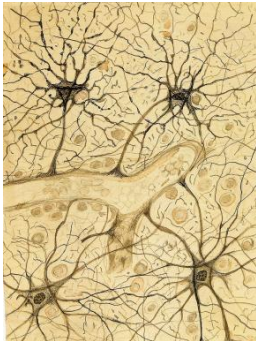
Intro al aprendizaje profundo

- La ingeniería de características requiere mucho tiempo, es susceptible a errores y no es escalable consistentemente con datos complejos. Mejor busquemos aprender las características subyacentes directamente de los datos.
- Las redes neuronales artificiales 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.





Figure: Santiago Ramón y Cajal



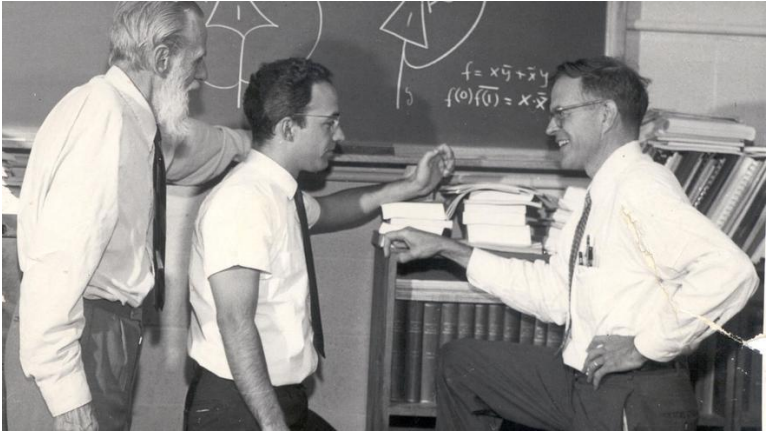
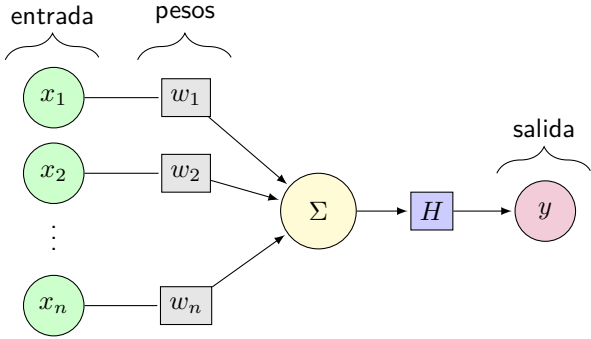


Figure: Warren McCulloch & Walter Pitts



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

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

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

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

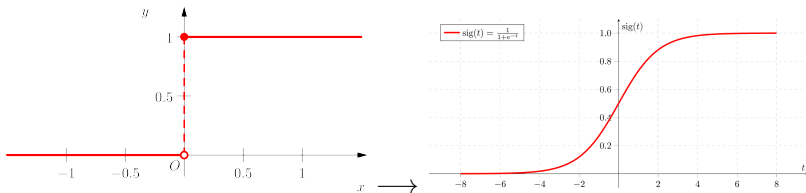
donde $\mathbf{x} = (x_1, x_2, \dots, x_n)$ y $i \in \{1, 2, \dots, n\}$.

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).



Bias y función de activación

Intro al aprendizaje profundo



El Perceptrón

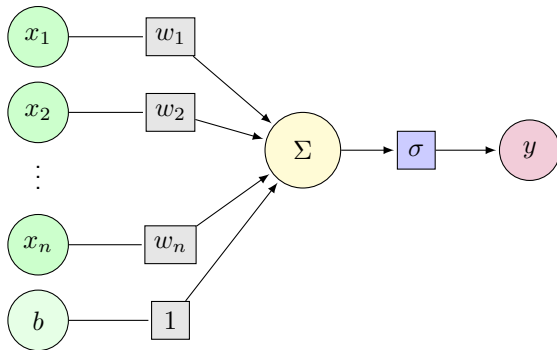
Intro al aprendizaje profundo



Figure: Frank Rosenblatt

El Perceptrón

Intro al aprendizaje profundo



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



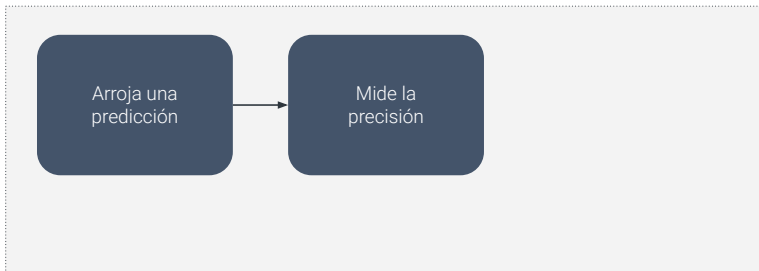
Idea intuitiva de entrenamiento

Intro al aprendizaje profundo

Arroja una
predicción

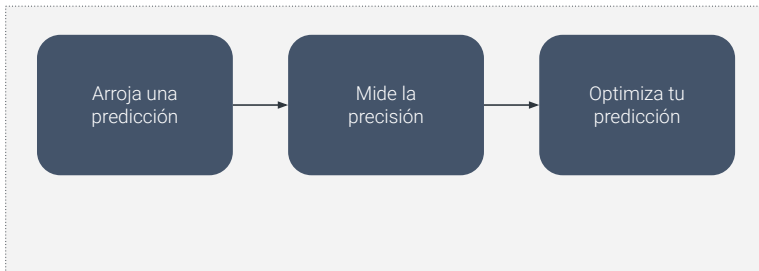
Idea intuitiva de entrenamiento

Intro al aprendizaje profundo



Idea intuitiva de entrenamiento

Intro al aprendizaje profundo

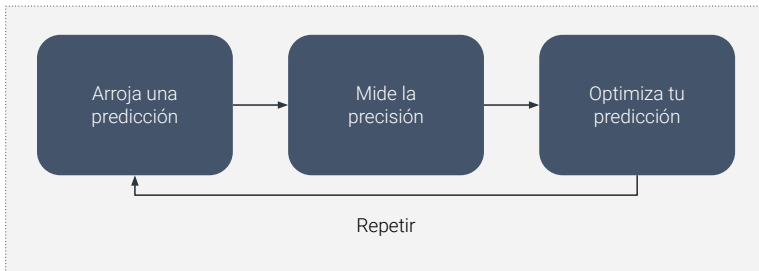


ferro@ciimat.mx



Idea intuitiva de entrenamiento

Intro al aprendizaje profundo



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

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

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



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

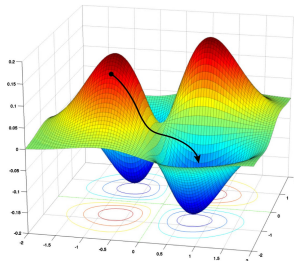
$$X = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.7 \end{bmatrix}$$

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

Optimización del error

Intro al aprendizaje profundo

- › **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



ferro@ciemat.mx



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

- **TLUs:**

- » No existe un algoritmo de aprendizaje formal → 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?**

Ejercicio: Manzanas vs. Naranjas



- › 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



Recordemos cómo opera el producto matricial:

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



Recordemos cómo opera el producto matricial:

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

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



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:

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

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



Recordemos cómo opera el producto matricial:

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

...



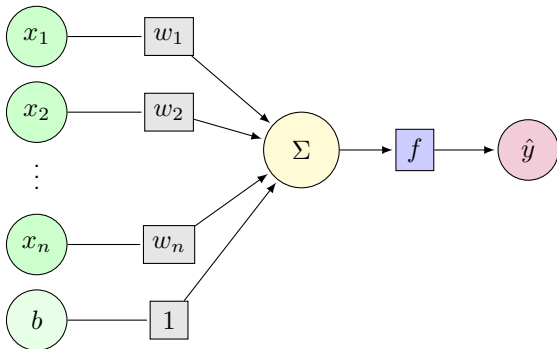
Recordemos cómo opera el producto matricial:

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



El Perceptrón

Intro al aprendizaje profundo

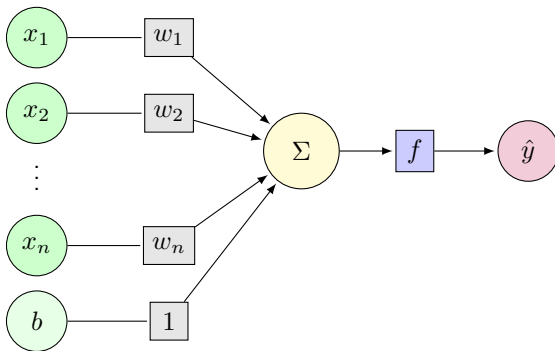


$$\hat{y} = f \left(\sum_i^n w_i x_i + b \right)$$



El Perceptrón

Intro al aprendizaje profundo

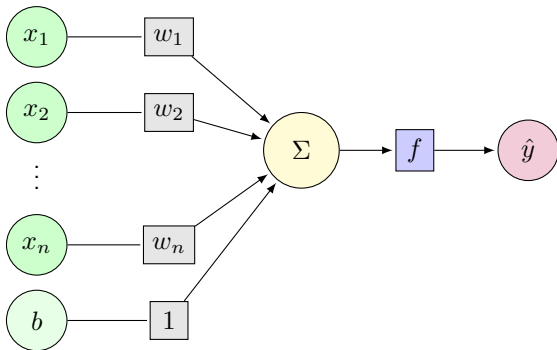


$$\sum_i^n w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$



El Perceptrón

Intro al aprendizaje profundo

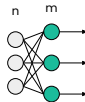


$$\mathbf{w}^T \mathbf{x} = [w_1 w_2 \cdots w_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$



Producto matricial

Intro al aprendizaje profundo



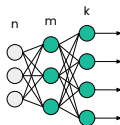
$$\mathbf{W} \cdot \mathbf{X} + \mathbf{b} = [w_1 \ w_2 \ \dots \ w_n] \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + b \rightarrow \mathbf{W}_{Layer} \cdot \mathbf{X} + \mathbf{b} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots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

$f(\cdot)$



Producto matricial

Intro al aprendizaje profundo



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



Composición de funciones

Intro al aprendizaje profundo

$$\begin{aligned} \mathbf{W}_{Layer_k} \cdot (\mathbf{W}_{Layer_m} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k &= \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \dots & w_{k,m} \end{bmatrix} \left(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \\ &\downarrow \\ &f_k(\mathbf{W}_{Layer_k} \cdot f_m(\mathbf{W}_{Layer_m} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k) \\ &\downarrow \\ &\boxed{f_n(\dots (f_2(\mathbf{W}_{Layer_k} \cdot f_1(\mathbf{W}_{Layer_1} \cdot \mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2))) + \mathbf{b}_n} \end{aligned}$$

Composición de funciones

Intro al aprendizaje profundo

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



$$F'(x) = f'(g(x)) \cdot g'(x)$$

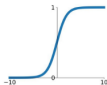


Funciones de activación

Intro al aprendizaje profundo

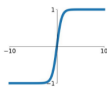
Sigmoid

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



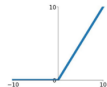
tanh

$$\tanh(x)$$



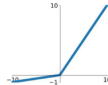
ReLU

$$\max(0, x)$$



Leaky ReLU

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

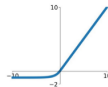


Maxout

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

ELU

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



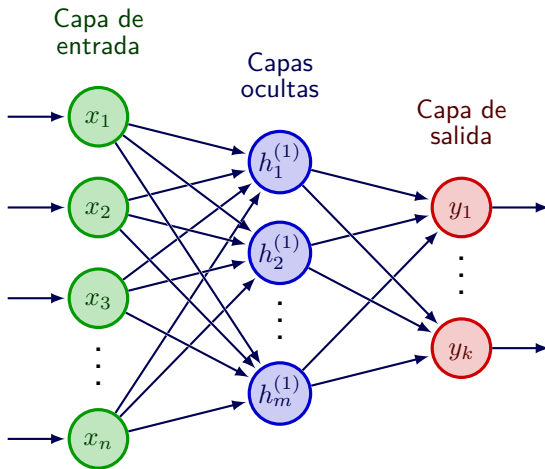
$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \Rightarrow f'(x) = f(x)(1-f(x))$$

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



El perceptrón multicapa

Intro al aprendizaje profundo



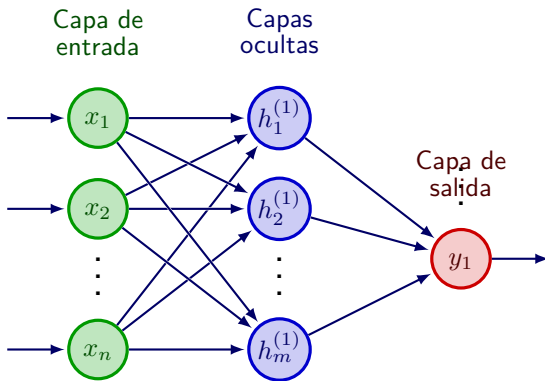
ferro@ciimat.mx



Escuela
Nacional de
Estudios
Superiores

El perceptrón multicapa

Intro al aprendizaje profundo



Para $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$, $y^{(i)}$, tendríamos que la salida es

$$\hat{y}_1^{(i)} = f(x^{(i)}).$$


- 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).



- **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.

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



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

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

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

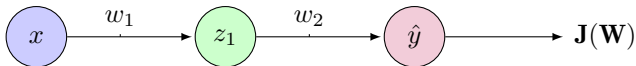


Algoritmo: Descenso de gradiente

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

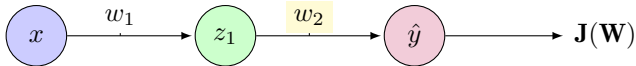
Retropropagación

Intro al aprendizaje profundo



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

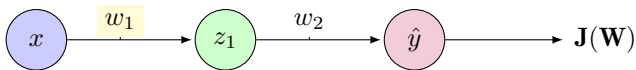




$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

Retropropagación

Intro al aprendizaje profundo

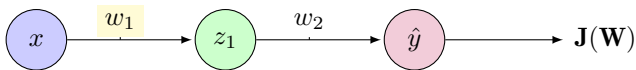


$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$$



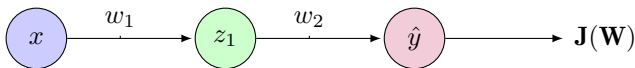
Retropropagación

Intro al aprendizaje profundo



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





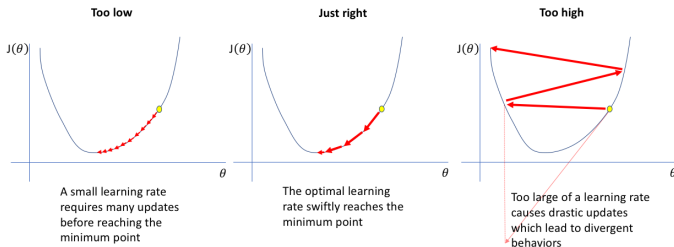
¿Cómo se calculan los gradientes?

Con la regla de la cadena.

Esto se repite para **cada peso** en la red neuronal, usando los gradientes de las capas posteriores.

La actualización de pesos está dada por:

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



Algoritmo: Descenso de gradiente

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

Descenso de gradiente estocástico

Intro al aprendizaje profundo

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.



Descenso de gradiente estocástico

Intro al aprendizaje profundo

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
$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial \mathbf{J}_k(\mathbf{W})}{\partial \mathbf{W}}^*$$

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 rápido de calcular (computacionalmente), y da una mejor estimación del gradiente.



- 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.



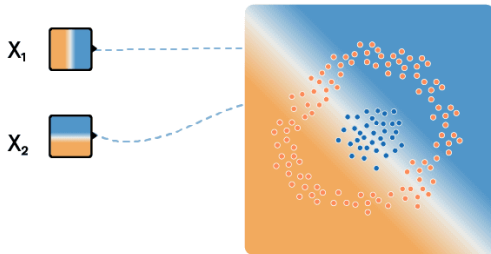


TensorFlow

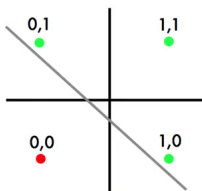
TensorFlow es un framework open-source para Machine Learning desarrollado por Google. Utilizado para construir y entrenar redes neuronales artificiales.



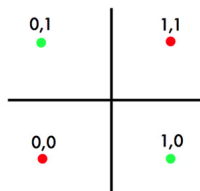
Ejercicio: Exploración del TensorFlow Playground



Ejercicio: Problema de separabilidad lineal



OR



XOR

Ejercicio: Exploración con TensorFlow



TensorFlow

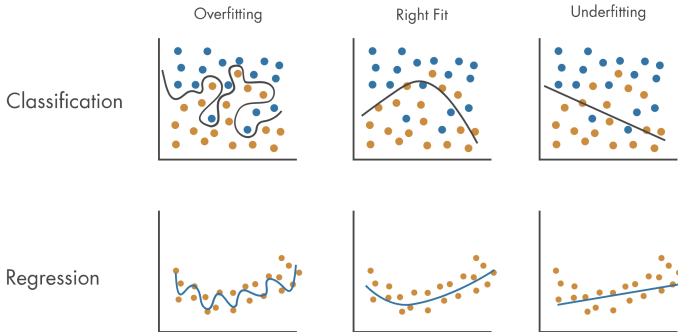


- › Setting the learning rate of your neural network
- › Retropropagación paso a paso
- › TensorFlow Tutorials
- › Libro Neural Networks and Deep Learning



El problema del overfitting

Intro al aprendizaje profundo

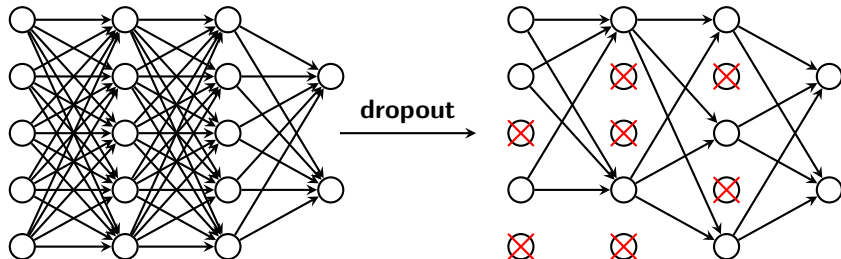


- › La **regularización** consiste en alguna técnica que sirve para evitar que un modelo se sobreajuste.
- › Es necesaria porque ayuda a mejorar la generalización de nuestro modelo con datos no vistos.
- › Los métodos de regularización que exploraremos serán:
 - › *Dropout*
 - › *Early stopping*



Dropout

Intro al aprendizaje profundo



ferro@cinat.mx

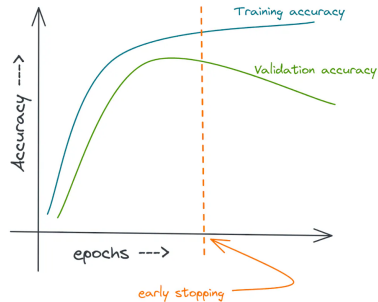
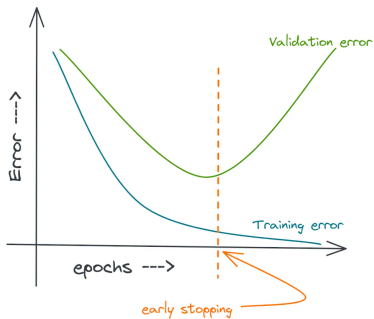


- › Durante el entrenamiento, establecemos aleatoriamente algunas activaciones en 0
 - » Típicamente hacemos "drop" del 50% de activaciones en una capa.
 - » Esto fuerza a la red a no depender de ningún nodo/neurona.
- › Podemos realizar el dropout en TensorFlow utilizando la capa `tf.keras.layers.Dropout(0.5)`, donde el 0.5 puede variar de acuerdo a lo especificado.



Early stopping

Intro al aprendizaje profundo



- El *Early Stopping* puede ser realizado en TensorFlow de manera sencilla creando un callback (función que se llama en cada iteración durante el entrenamiento de la red neuronal):

```
model = tf.keras.models.Sequential(...)
callback = tf.keras.callbacks.EarlyStopping(monitor='loss',
patience=3)
history = model.fit(..., callbacks=[callback])
```

- › Artículo "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"
- › Regularización en Redes Neuronales
- › Vanishing and Exploding Gradients in Neural Network Models: Debugging, Monitoring, and Fixing