

Aprendizaje Profundo

Facultad de Ingeniería
Universidad de Buenos Aires



Profesores:

Marcos Maillot
Gerardo Vilcamiza

Introducción

- Materia de 8 clases teórico-prácticas
- Trabajamos con diapositivas y notebooks
- Clases dinámicas, la participación y feedback son bien recibidos
- Clases:
 - 10 minutos resumen de la clase anterior y resolución de dudas de los TPs
 - 3 bloques de 50 minutos de trabajo teórico-práctico
 - 2 bloques de 10 minutos de descanso
 - Ejercicios clase a clase de práctica (sin nota)



Introducción

- **APROBACIÓN**
 - Trabajos prácticos offline, con entregas por Google Forms.
 - TP1: Envío al término de la clase 2, entrega hasta dos días después de la clase 4 (23 h).
 - TP2: Envío al término de la clase 4, entrega hasta dos días después de la clase 6 (23 h).
 - TP3: Envío al término de la clase 6, entrega hasta dos días después de la clase 8 (23 h).
 - Los 3 trabajos tiene el mismo peso, cada uno vale un tercio de la nota final.
- **MATERIAL DE CLASE**
 - Los contenidos estarán disponibles en el [repositorio](#).



- **CORREOS DE CONSULTA**

- Marcos Maillot: marcos_maillot@yahoo.com.ar
- Gerardo Vilcamiza: gvilcamiza.ext@fi.uba.ar
- Clase: ap_a22@cursoscapse.com

Contenido

Gerardo Vilcamiza

- **Clase 1:** Introducción al Deep Learning. Redes neuronales
- **Clase 2:** Funciones de activación, funciones de costo y algoritmos de optimización
- **Clase 3:** Introducción al framework PyTorch
- **Clase 4:** Regularización, hyperparameter tuning y embeddings

- **Clase 5:** Convolutional Neural Networks (CNNs)
- **Clase 6:** Recurrent Neural Networks (RNNs). Attention Layers
- **Clase 7:** Encoder-Decoder. Autoencoder. Transfer learning
- **Clase 8:** Generative Adversarial Networks (GANs)

Marcos Maillot



Referencias

Bibliografía solo a modo de sugerencia y no será obligatorio el uso de dicho material. La materia es completamente autocontenido.

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. <https://www.deeplearningbook.org>
- Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2022). Dive into Deep Learning. Cambridge University Press. <https://d2l.ai>
- Nielsen, M. A. (2015). Neural Networks and Deep Learning. Determination Press. <http://neuralnetworksanddeeplearning.com>



Herramientas de trabajo

- Lenguaje de programación:
 - Python 3.11
 - Pip / Conda para instalar paquetes y dependencias
- Librerías principales:
 - Numpy, Matplotlib, Pandas, Scikit-Learn, Scipy
 - **Pytorch**
- Consola interactiva:
 - **Google Colab**
 - (Opcional) Jupyter Notebook
- Herramientas:
 - Github para repositorios



Introducción a Deep Learning

“Deep learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. Because the computer gathers knowledge from experience, there is no need for a human computer operator to formally specify all the knowledge that the computer needs.”

Ian Goodfellow

“Deep learning is a subset of machine learning that uses multi-layered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain.”

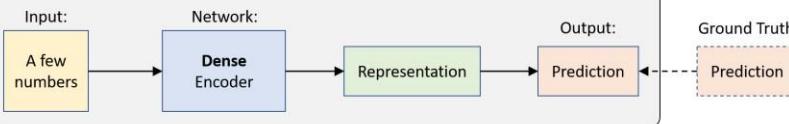
IBM



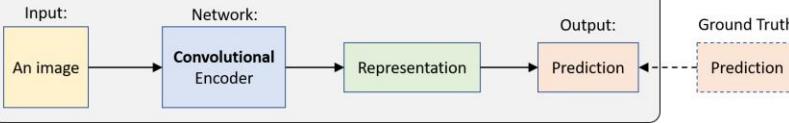
Introducción a Deep Learning

Supervised Learning

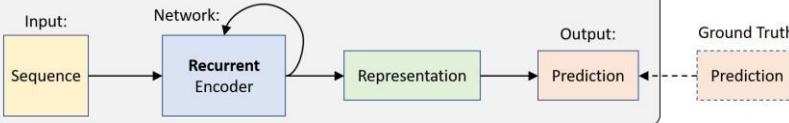
1. Feed Forward Neural Networks



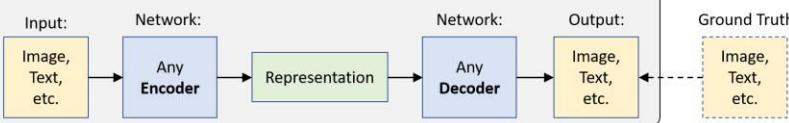
2. Convolutional Neural Networks



3. Recurrent Neural Networks

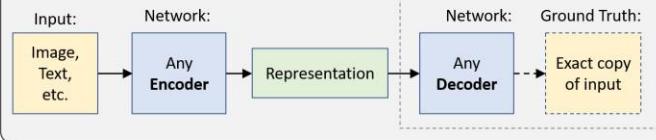


4. Encoder-Decoder Architectures

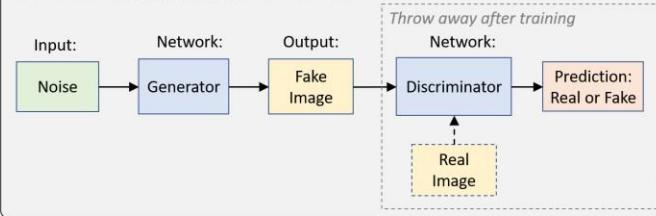


Unsupervised Learning

5. Autoencoder

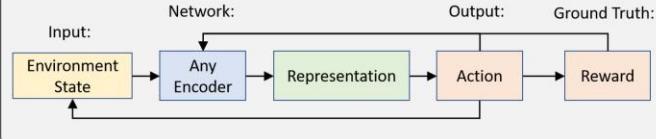


6. Generative Adversarial Networks



Reinforcement Learning

7. Networks for Learning Actions, Values, and Policies



Introducción a Deep Learning

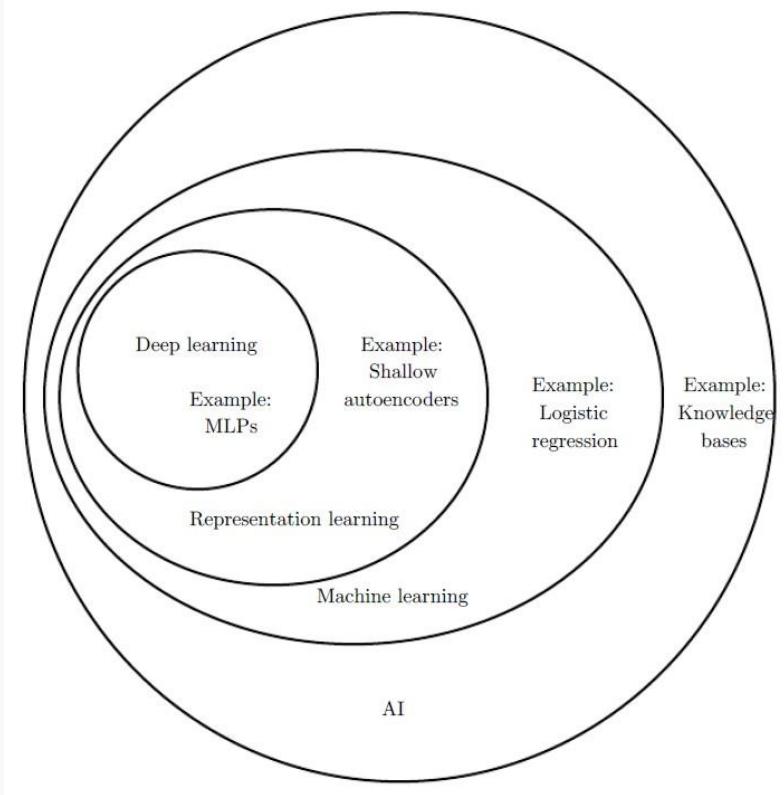
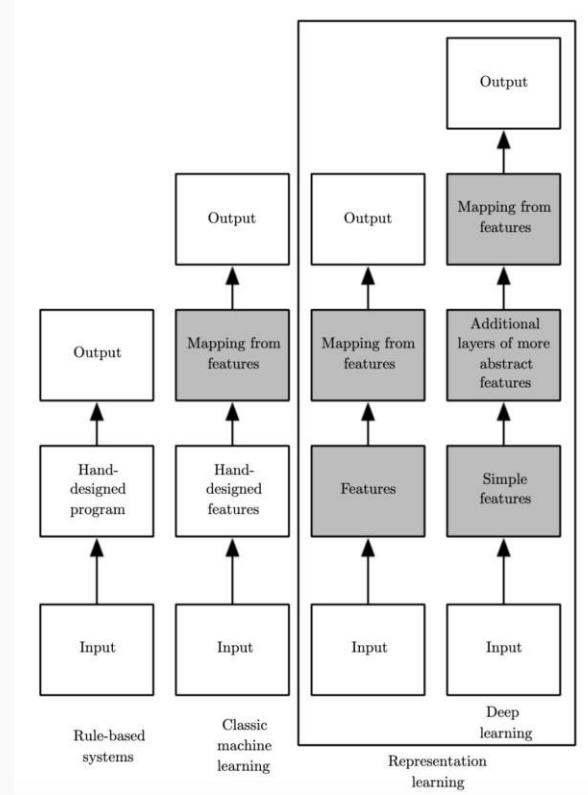


Diagrama de Venn de algoritmos



Enfoques de soluciones

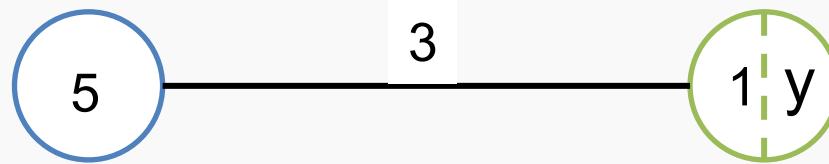
- Redes neuronales:**
- Aprendizaje end to end.
 - Aprenden Composiciones
 - NO LINEALES



Feed-Forward neural network

Feed-Forward neural network

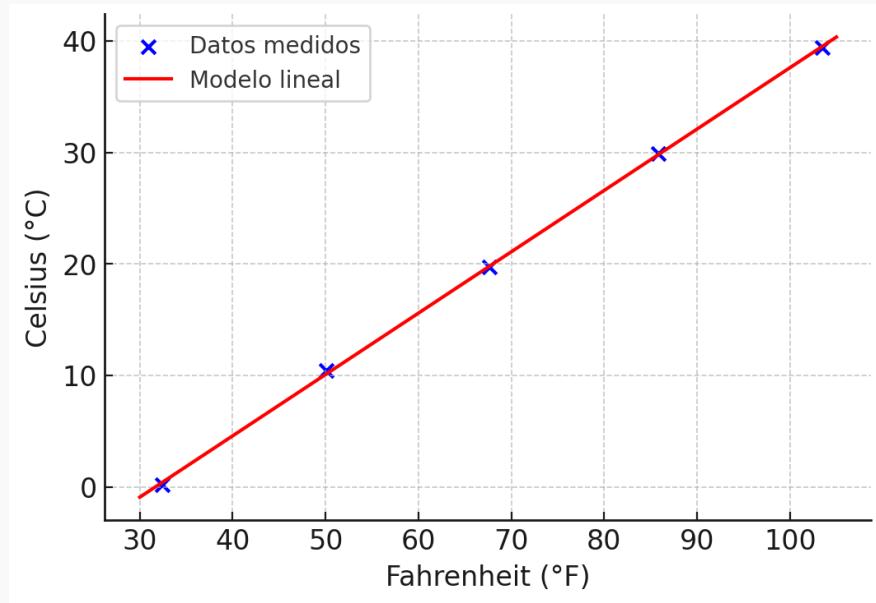
Neurona de entrada Neurona de salida



$$y = wx + b$$

$$y = 3(5) + 1 = 16$$

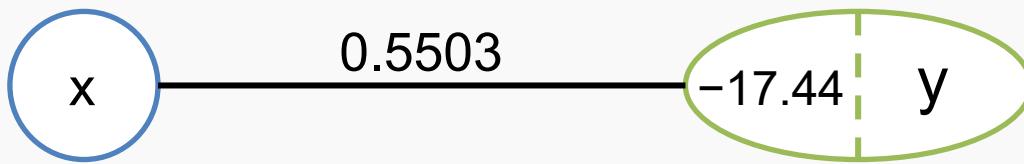
Feed-Forward neural network



Fahrenheit ($^{\circ}\text{F}$)	Celsius ($^{\circ}\text{C}$)
32.4	0.2
50.1	10.4
67.6	19.7
85.8	29.9
103.5	39.4

$$\hat{y} = 0.5503x - 17.44$$

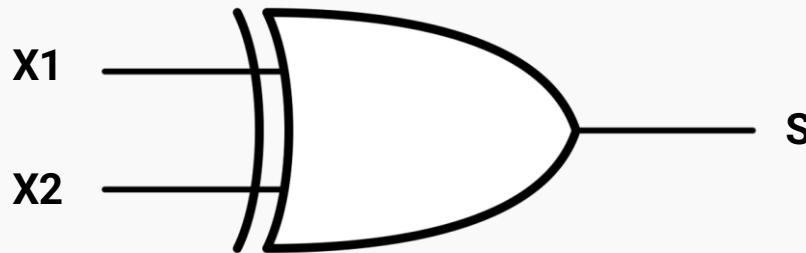
$$y_r = 0.5556x - 17.78$$



Feed-Forward neural network

¿Por qué necesitamos modelos no lineales?

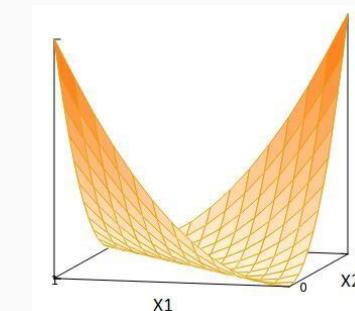
- Caso simple: Compuerta XOR



Nomenclatura :

- $\vec{X} \rightarrow \text{dataset de entrada} \in R^{n \times m}$
- $\vec{y} \rightarrow \text{salida} \in R^{n \times 1}$
- $m \rightarrow \text{cantidad de columnas}, n \rightarrow \text{cantidad de filas}$
- $X_{i,j} \rightarrow \text{parámetro } j \text{ de la muestra } i \in R$
- $\vec{X}_i \rightarrow \text{vector de la fila } i \in R^{1 \times m}$
- $\vec{X}_b \rightarrow \text{matriz de batch} \in R^{b \times m}$

	X1	X2	S	
X _i	0	0	0	y
X _B	0	1	1	y _B
	1	0	1	
X _{i,j}	1	1	0	
X				



Feed-Forward neural network

- Arquitectura: Modelo Lineal

$$\hat{f} : R^2 \rightarrow R / \hat{y}_i = \hat{f}(X_{i,1}, X_{i,2}) = W_1 \cdot X_{i,1} + W_2 \cdot X_{i,2} + b$$

- Loss function: Error cuadrático medio

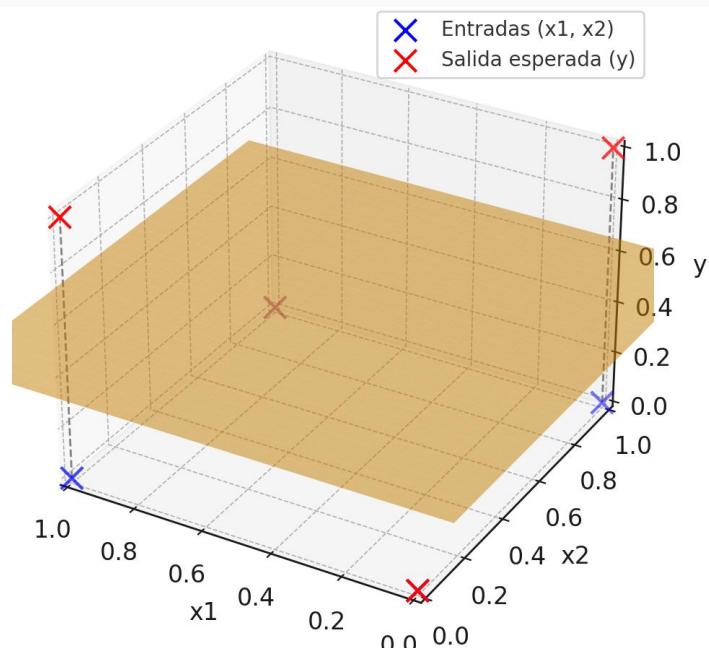
$$L(W_1, W_2, b, \vec{X}, \vec{y}) = L(W_1, W_2, b) = \frac{1}{4} \cdot \sum_{i=1}^4 (y_i - \hat{y}_i)^2$$

- Optimizador: Solución cerrada

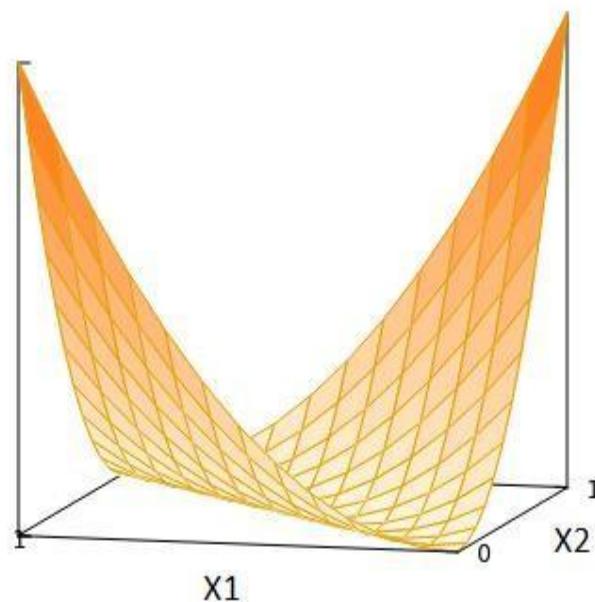
$$\vec{\nabla}_{\vec{w}} L = \vec{0} \rightarrow \vec{\nabla}_{\vec{w}} L = \left\{ \begin{array}{l} \frac{\partial L}{\partial W_1} \\ \frac{\partial L}{\partial W_2} \\ \frac{\partial L}{\partial b} \end{array} \right\} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{W} = \begin{bmatrix} W_1 \\ W_2 \\ b \end{bmatrix} = (\vec{X}^T \cdot \vec{X})^{-1} \cdot \vec{X}^T \cdot \vec{y} \rightarrow \vec{W} = \begin{bmatrix} 0 \\ 0 \\ 0.5 \end{bmatrix}$$

Feed-Forward neural network

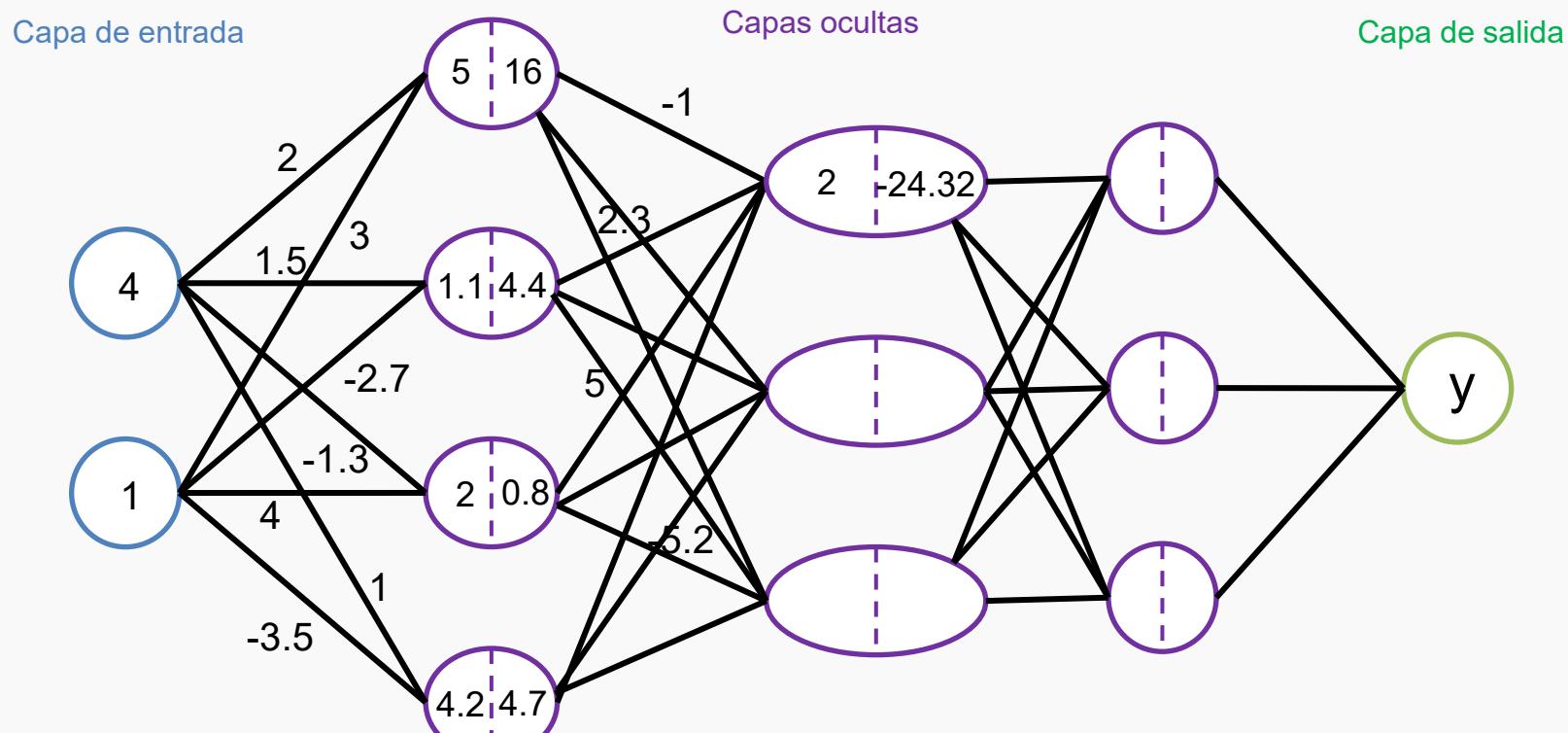


Lineal o Constante

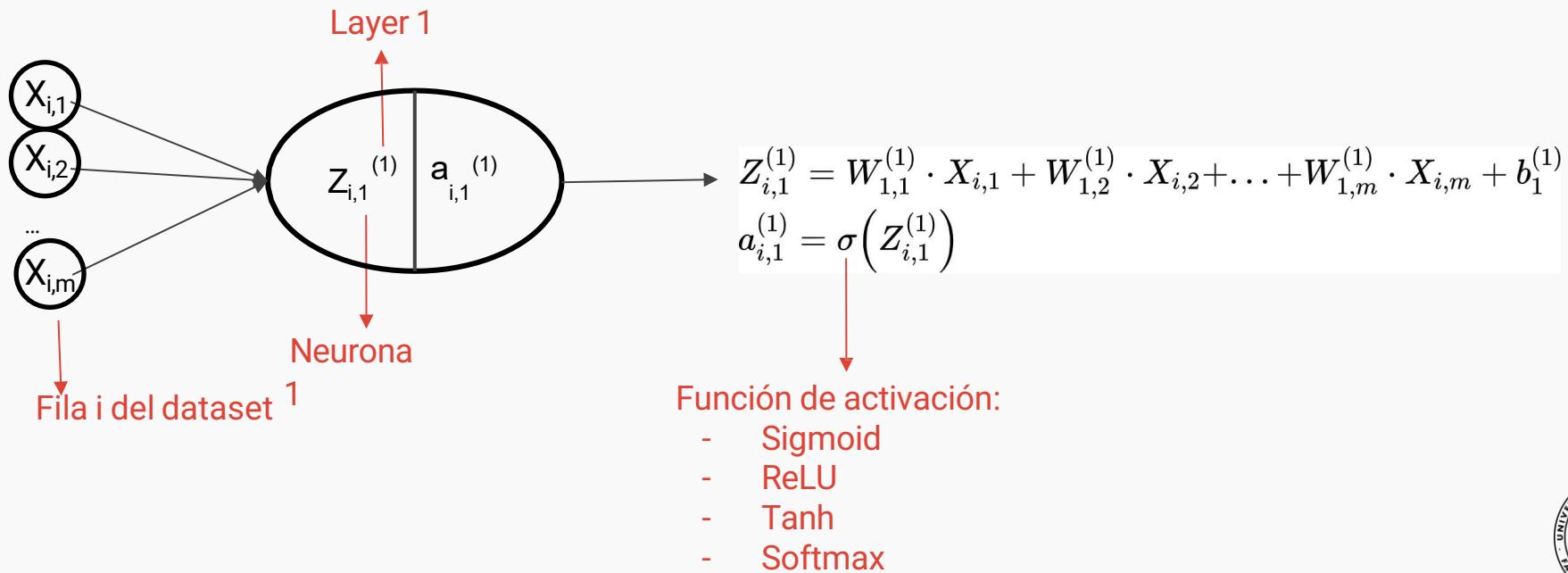


No lineal

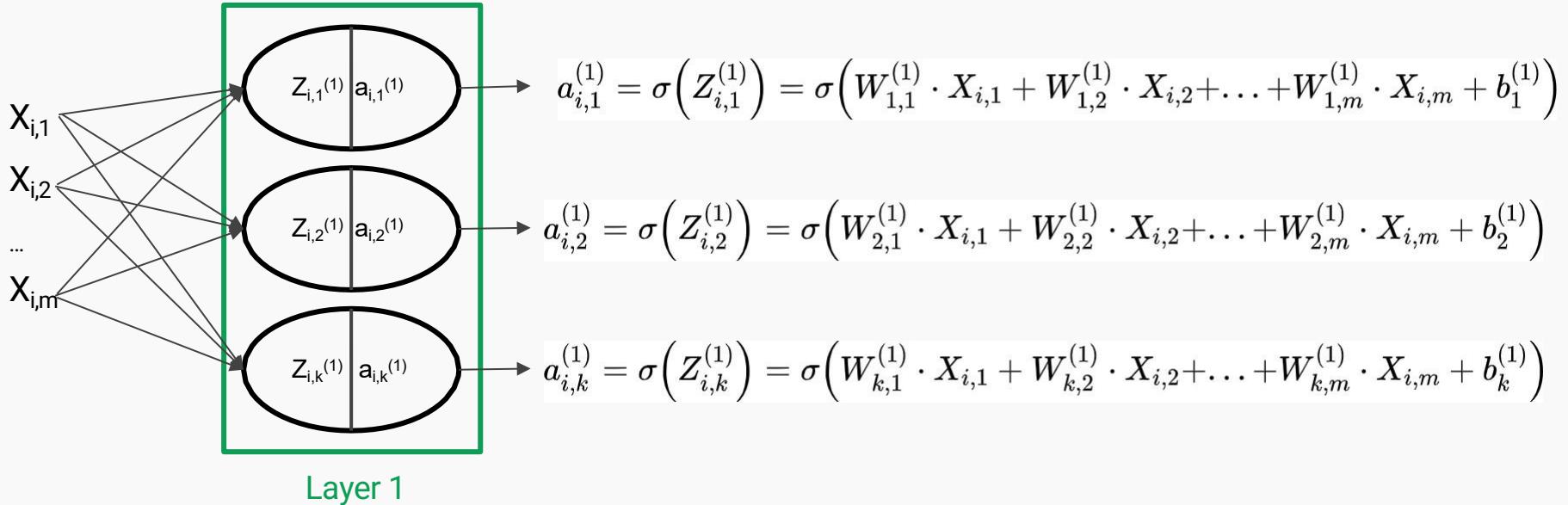
Feed-Forward neural network



Neurona



Layer Lineal con función de activación



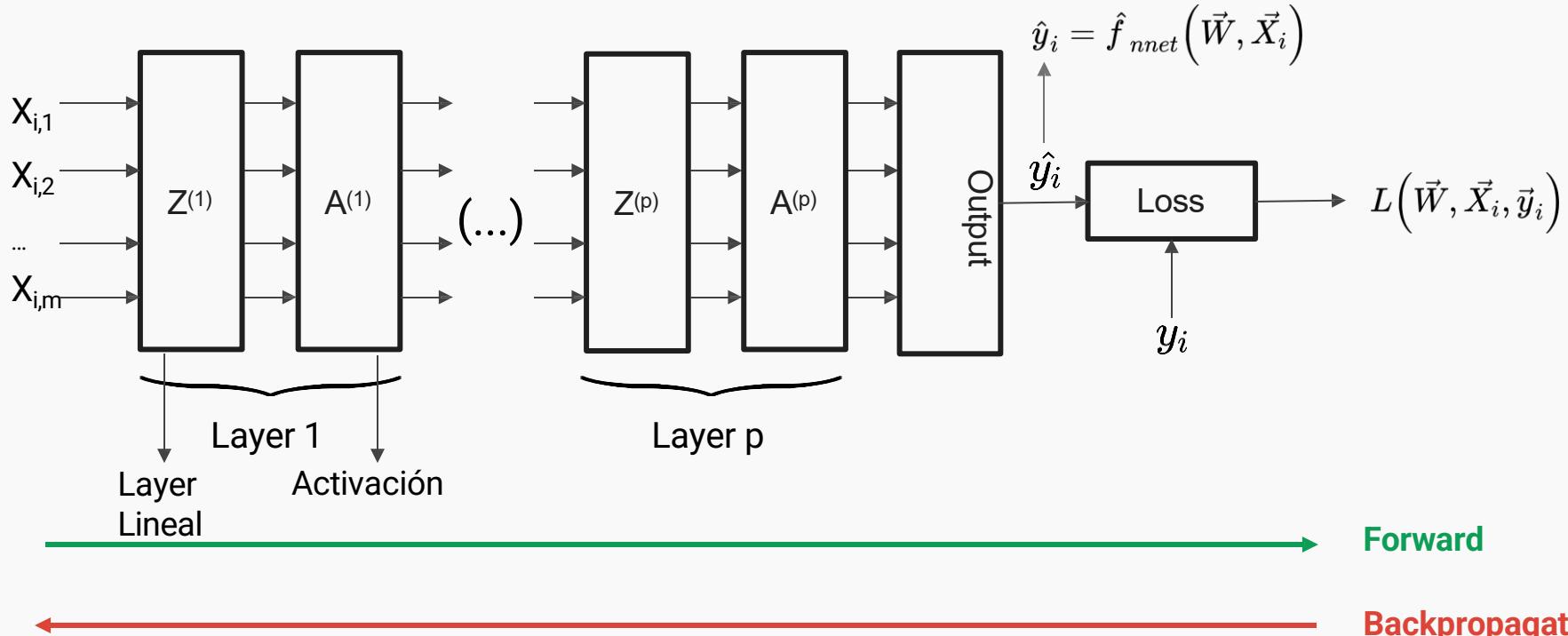
En formato matricial:

$$\vec{A}_i^{(1)} = \sigma(\vec{Z}_i^{(1)}) = \sigma(\vec{W}^{(1)} \cdot \vec{X}_i + \vec{b}^{(1)})$$

de parámetros : $k \times (m + 1)$

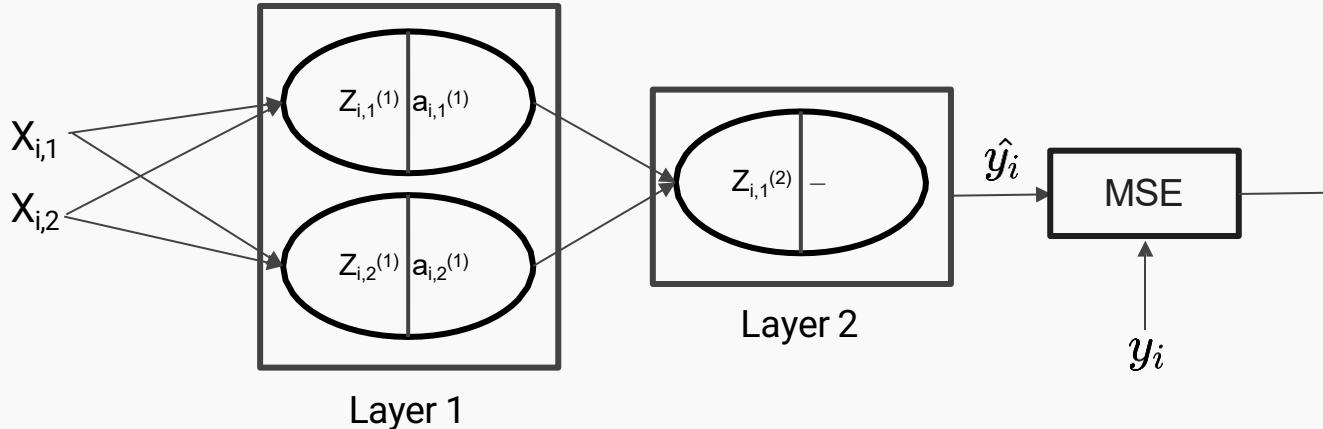
$$\vec{A}_i^{(1)}, \vec{Z}_i^{(1)} \in R^{k \times 1}$$
$$\vec{W}^{(1)} \in R^{k \times m}$$
$$\vec{X}_i \in R^{m \times 1}$$
$$\vec{b}^{(1)} \in R^{k \times 1}$$

Red neuronal feedforward con p layers



Resolvemos backpropagation en forma numérica: $\nabla_{\vec{W}} L \rightarrow \vec{W} = \vec{W} - \alpha \times \nabla_{\vec{W}} L$

Solución de XOR con red neuronal simple



- Arquitectura de 2 layers: 1 layer hidden, 1 layer de salida
- 2 neuronas en layer 1, función de activación sigmoid
- 1 neurona layer 2, sin activación
- ¿Cuántos parámetros entreno?

$$\left. \begin{array}{l} \vec{W}^{(1)} \in R^{2 \times 2} \rightarrow 4 \\ \vec{b}^{(1)} \in R^{2 \times 1} \rightarrow 2 \\ \vec{W}^{(2)} \in R^{1 \times 2} \rightarrow 2 \\ \vec{b}^{(2)} \in R^1 \rightarrow 1 \end{array} \right\} 9 \text{ parámetros}$$

Solución de XOR con red neuronal simple

- **Paso Forward:**

$$Z_{i,1}^{(1)} = W_{1,1}^{(1)} \cdot X_{i,1} + W_{1,2}^{(1)} \cdot X_{i,2} + b_1^{(1)}$$

$$Z_{i,2}^{(1)} = W_{2,1}^{(1)} \cdot X_{i,1} + W_{2,2}^{(1)} \cdot X_{i,2} + b_2^{(1)}$$

$$a_{i,1}^{(1)} = \sigma(Z_{i,1}^{(1)})$$

$$a_{i,2}^{(1)} = \sigma(Z_{i,2}^{(1)})$$

$$Z_{i,1}^{(2)} = W_{1,1}^{(2)} \cdot a_{i,1}^{(1)} + W_{1,2}^{(2)} \cdot a_{i,2}^{(1)} + b_1^{(2)}$$

$$\hat{y}_i = a_{i,1}^{(2)} = Z_{i,1}^{(2)}$$

$$L_{\vec{W}} = (y_i - \hat{y}_i)^2$$



Solución de XOR con red neuronal simple

- Debo encontrar $W_1^{(1)}, b_1^{(1)}, W_2^{(1)}, b_2^{(1)}, W_1^{(2)}, b_1^{(2)}$. Utilizo SGD:

> Inicializar los pesos $\vec{W} \rightarrow U(0, 1) \rightarrow 9$ variables

> for epoch in range(n_epochs) :

> for \vec{X}_i, y_i in (\vec{X}, \vec{y}) :

$$(1) \text{Forward} \rightarrow \hat{y}_i = \hat{f}_{nnet}(\vec{W}, \vec{X}_i)$$

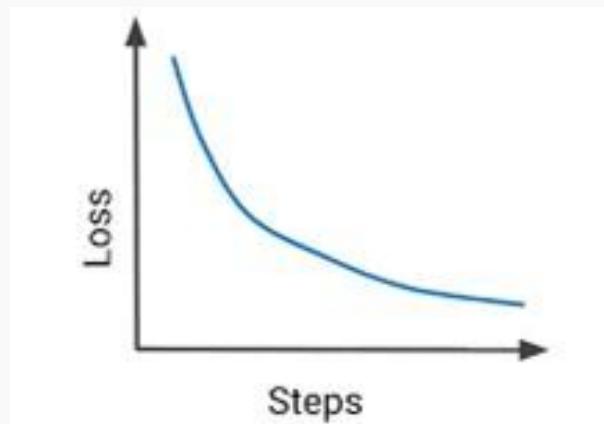
$$(2) \text{Error} \rightarrow e_i = (y_i - \hat{y}_i)^2$$

$$(3) \text{Backward} \rightarrow \begin{cases} \partial L / \partial W_{1,1}^{(1)}, \partial L / \partial W_{1,2}^{(1)}, \partial L / \partial b_1^{(1)} \\ \partial L / \partial W_{2,1}^{(1)}, \partial L / \partial W_{2,2}^{(1)}, \partial L / \partial b_2^{(1)} \\ \partial L / \partial W_{1,1}^{(2)}, \partial L / \partial W_{1,2}^{(2)}, \partial L / \partial b_1^{(2)} \end{cases}$$

$$(4) \text{Actualización} \rightarrow \begin{cases} W_{1,1}^{(1)} \leftarrow W_{1,1}^{(1)} - \alpha \cdot \partial L / \partial W_{1,1}^{(1)} \\ \dots \\ b_1^{(2)} \leftarrow b_1^{(2)} - \alpha \cdot \partial L / \partial b_1^{(2)} \end{cases}$$

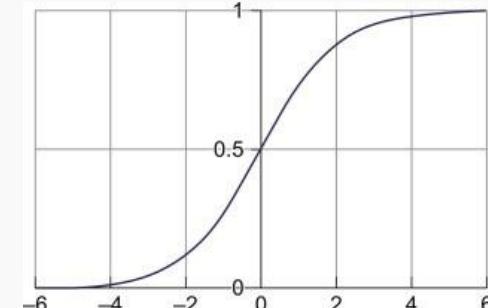
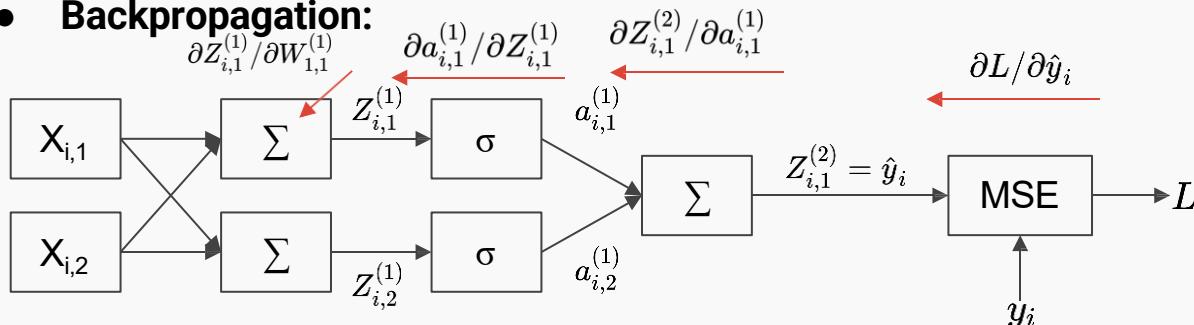
$$> \text{Calcular MSE} = \frac{1}{4} \sum_{i=1}^4 (y_i - \hat{y}_i)^2 \longrightarrow \text{Loss plot}$$

> Return \vec{W}



Solución de XOR con red neuronal simple

- **Backpropagation:**



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

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

$$\left\{ \begin{array}{l} \frac{\partial L}{\partial W_{1,1}^{(2)}} = \frac{\partial}{\partial W_{1,1}^{(2)}} \left((y_i - \hat{y}_i)^2 \right) = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial W_{1,1}^{(2)}} = 2 \cdot (y_i - \hat{y}_i) \cdot -1 \cdot a_{i,1}^{(1)} \\ \vdots \\ \frac{\partial L}{\partial W_{1,1}^{(1)}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial a_{i,1}^{(1)}} \cdot \frac{\partial a_{i,1}^{(1)}}{\partial Z_{i,1}^{(1)}} \cdot \frac{\partial Z_{i,1}^{(1)}}{\partial W_{1,1}^{(1)}} = \end{array} \right.$$

$$= -2 \cdot (y_i - \hat{y}_i) \cdot W_{1,1}^{(2)} \cdot \sigma(Z_{i,1}^{(1)}) \cdot (1 - \sigma(Z_{i,1}^{(1)})) \cdot X_{i,1}$$

9 derivadas parciales!

EJERCICIO

1. Completar las 9 derivadas parciales.
2. Implementar en Python la solución de XOR con la red neuronal planteada en clase. Utilizar SGD.
3. Graficar MSE por epoch.