

Aprendizaje Profundo

Facultad de Ingeniería
Universidad de Buenos Aires



Profesores:

Marcos Maillot
Maximiliano Torti

- Materia de 8 clases teórico-prácticas
- Trabajamos con diapositivas
- Clases dinámicas, la participación es muy bien recibida
- Clases:
 - 10 minutos resumen de clase anterior
 - 3 bloques de 50 minutos de trabajo teórico-práctico
 - 2 bloques de 10 minutos de descanso
 - Ejercicios clase a clase de práctica (sin nota)

- **APROBACIÓN**

- 2 trabajos prácticos offline
 - Se envía el enunciado en 4ta clase y 7ma clase.
 - Fecha límite de entrega 10 días posteriores al final de la cursada
 - La entrega consiste en un Jupyter Notebook o Colab en Github
 - Individuales
- Se aprecia el feedback de las clases



- Canal de Slack:
 - deep_learning: <https://ceaiworkspace.slack.com/archives/C019FT6NYCE>
- Repositorio de videos de las clases
- Repositorio de la materia
 - Github: https://github.com/FIUBA-Posgrado-Inteligencia-Artificial/aprendizaje_profundo
- Correos
 - Marcos Maillot: marcos_maillot@yahoo.com.ar
 - Maximiliano Torti: maxit1992@gmail.com



- **Clase 1:** Introducción a Deep Learning. Redes feed-forward
- **Clase 2:** Funciones de pérdida y optimización, activación, funciones de salida
- **Clase 3:** Pytorch
- **Clase 4:** Regularización, hyperparameter tuning, embedding layer
- **Clase 5:** Convolutional Neural Networks
- **Clase 6:** Recurrent Neural Networks. Attention Layers
- **Clase 7:** Encoder-Decoder. Autoencoder. Transfer learning
- **Clase 8:** Generative Adversarial Networks



Referencias

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

- Deep Learning. Ian Goodfellow. <https://www.deeplearningbook.org/>



Herramientas de trabajo

- Lenguaje de programación:
 - Python 3.8
 - Pip / Conda para instalar paquetes y dependencias
- Librerías principales:
 - Numpy, Pandas, Scikit-Learn, Scipy
 - **Pytorch**
- Consola interactiva:
 - **iPhyton y Google Colab**
 - (Opcional) Jupyter Notebook
- Herramientas:
 - Github para repositorios
- IDE:
 - VSCode o PyCharm



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

- Loss function

- Error cuadrático medio (ECM)
- Binary Cross Entropy
- Cross Entropy
- KL-Divergence
- Dice loss

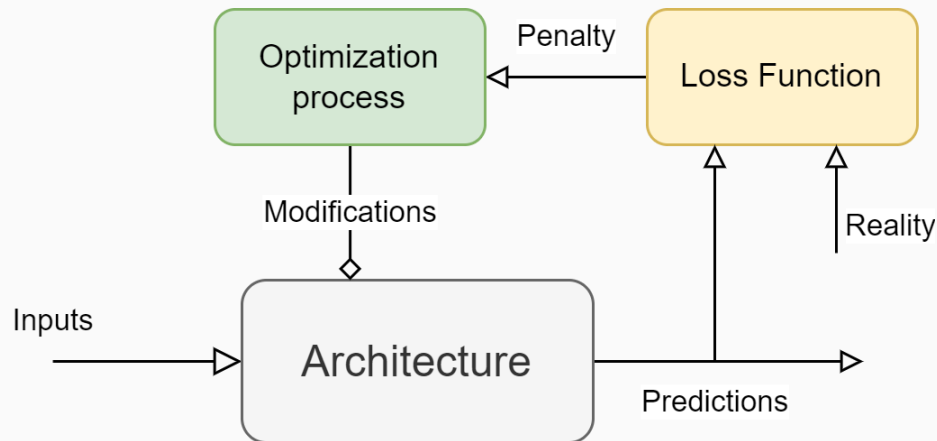
- Model Architecture

- Non-linear neural network
 - Layers: Linear, Convolutional, Recurrent
 - Activation Function: Sigmoid, Softmax, ReLu, Tanh
 - # of layers, # of hidden units

- Optimization

- Algorithms
 - Gradient Descent

- Metrics



Introducción a Deep Learning

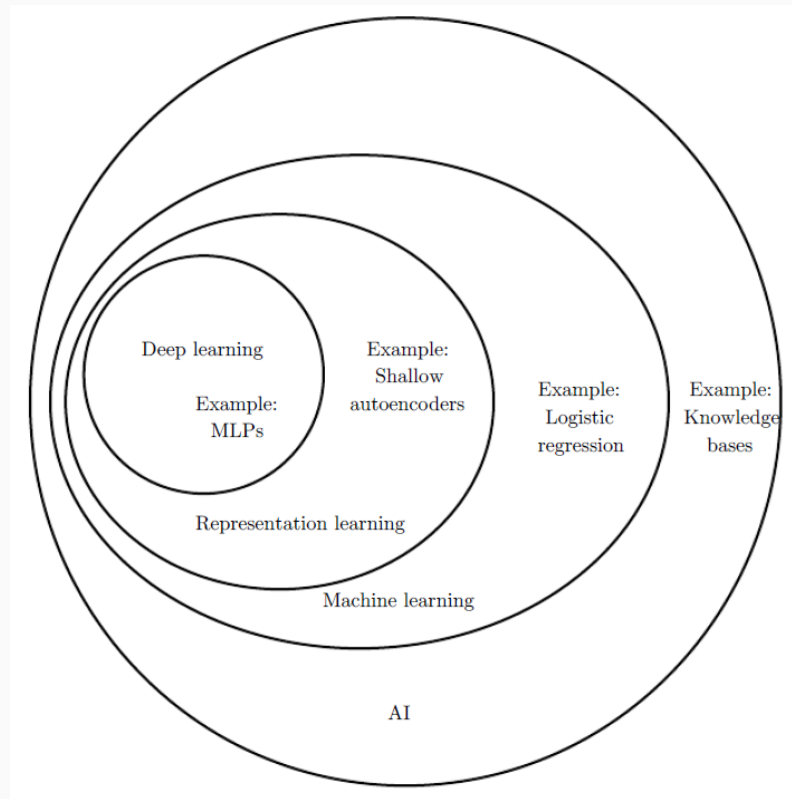
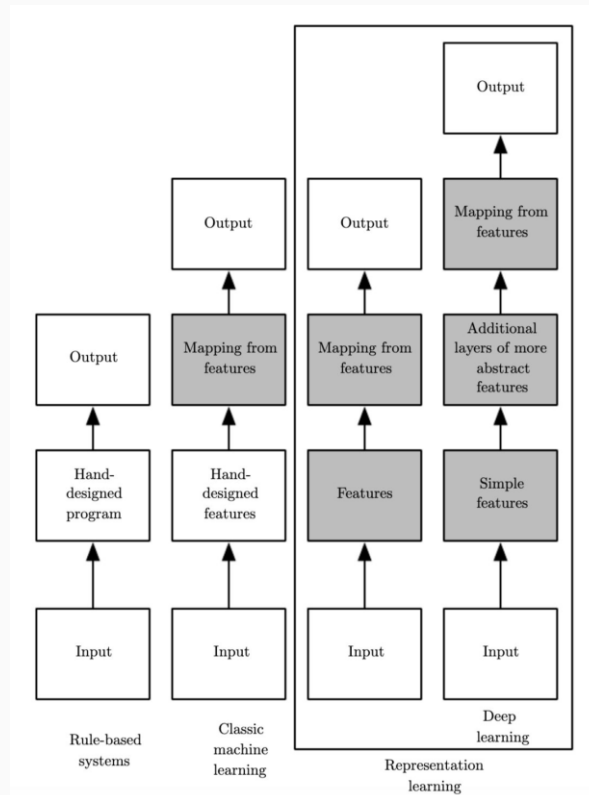


Diagrama de Venn de algoritmos

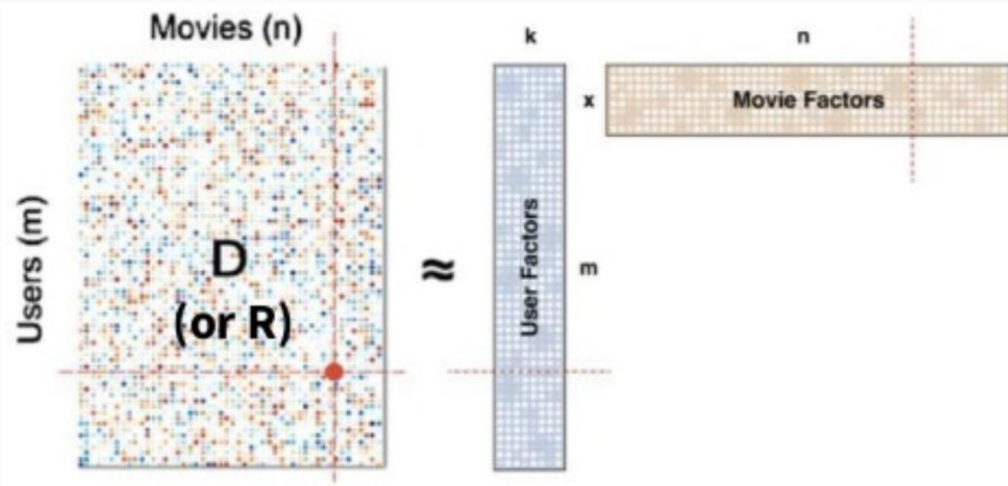


Enfoques de soluciones

Redes neuronales:

- Aprendizaje end to end.
- Aprenden Composiciones
- **NO LINEALES**

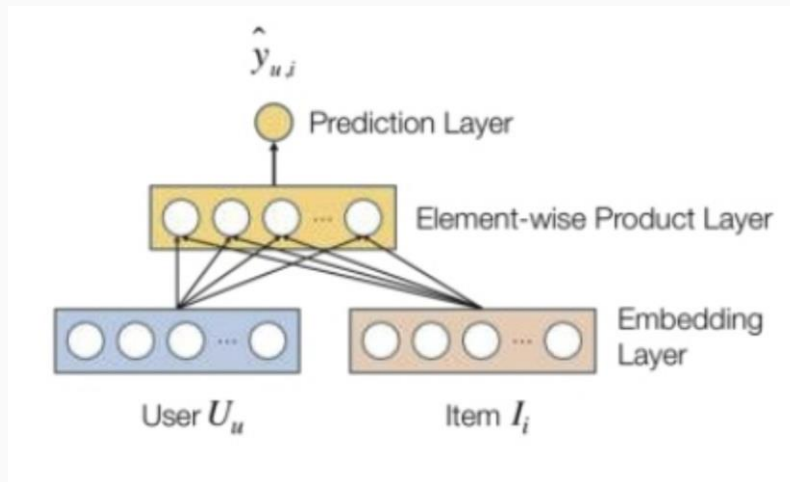
Ejemplo de solución con Deep Learning



- El objetivo es aprender todos los U_u e l_i .
- U_u e l_i son representaciones densas de usuarios y películas.

Usuario \ Pelicula				
	Monster Ink	Spider-Man	Inception	...
Cosme	9		8	...
Fulanito		6	7	...
...

Ejemplo de solución con Deep Learning

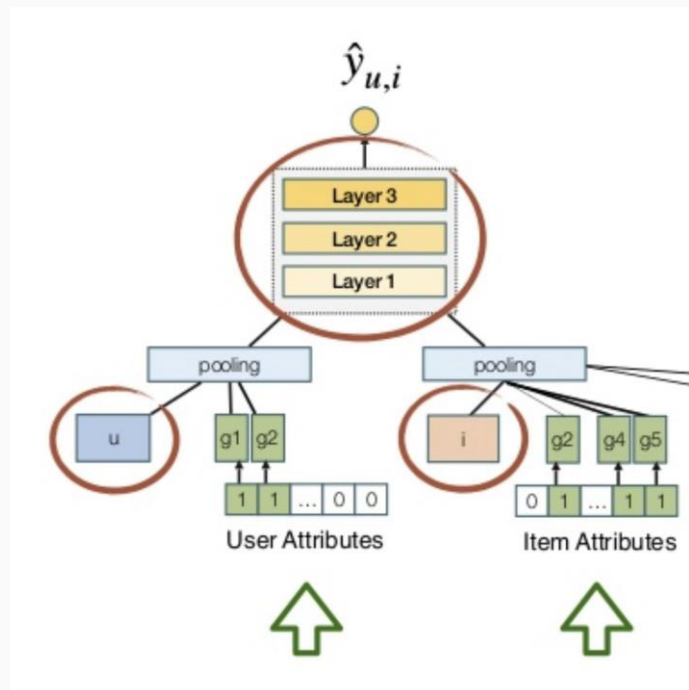


$$L = \sum_{u,i} (R_{u,i} - U_u I_i)^2$$

$$\nabla L = \nabla \left(\sum_{u,i} (R_{u,i} - U_u I_i)^2 \right)$$

- Lo que aprendemos son los “embeddings” de los usuarios y los “embeddings” de las películas.
- Producto interno entre U_u e I_i como arquitectura.
- ECM como función de costo.
- Gradient Descent como optimización.
- Podríamos utilizar una red neuronal para resolver el problema.

Ejemplo de solución con Deep Learning



$$L = \sum_{u,i} (R_{u,i} - f(U_u, I_i, g_1, g_2, \dots, g_5))^2$$

$$\nabla L = \nabla \left(\sum_{u,i} (R_{u,i} - f(U_u, I_i, g_1, g_2, \dots, g_5))^2 \right)$$

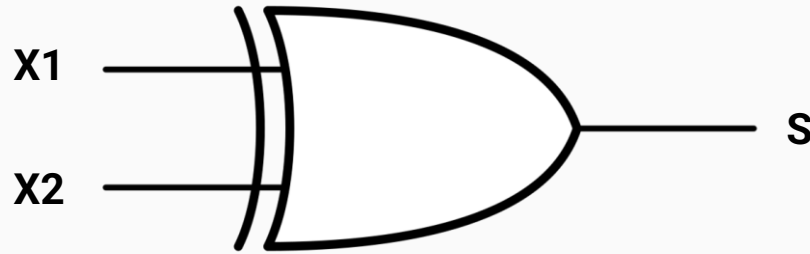
- Solución más cercana a una solución real, con una red neuronal compleja.
- Múltiples entradas de diferentes dominios -> Las redes neuronales son multimodales.
- Los elementos encerrados en rojo son los parámetros entrenables.

Feed-Forward neural network

Feed-Forward neural network

¿Por qué necesitamos modelos no lineales?

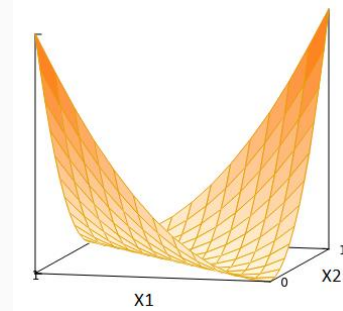
- Caso simple: Compuerta XOR



Nomenclatura :

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

	X1	X2	S	
\vec{X}_i	0	0	0	y
\vec{X}_B	0	1	1	y_B
	1	0	1	
	1	1	0	
	$X_{i,j}$	X		



Feed-Forward neural network

- Arquitectura: Modelo Lineal

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

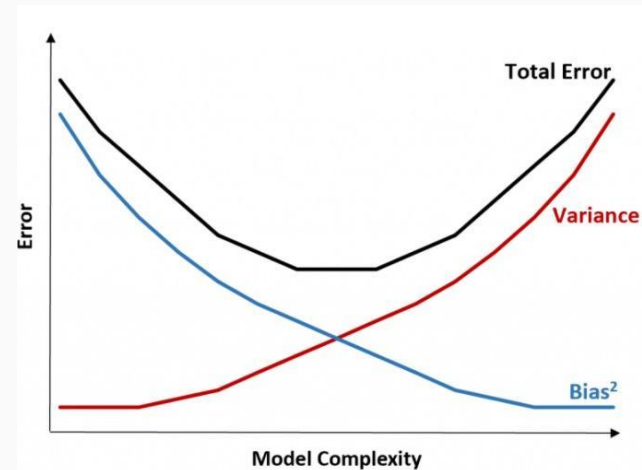
- Loss function: Error cuadrático medio

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

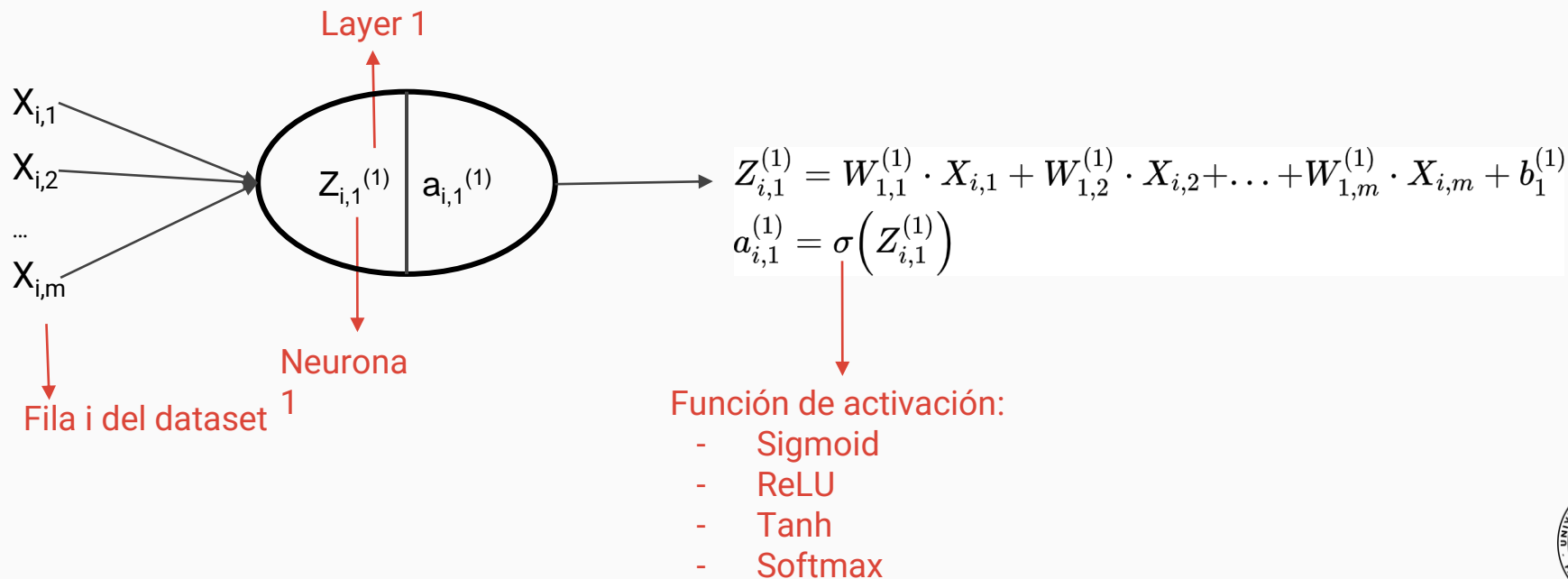
- Optimizador: Solución cerrada

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

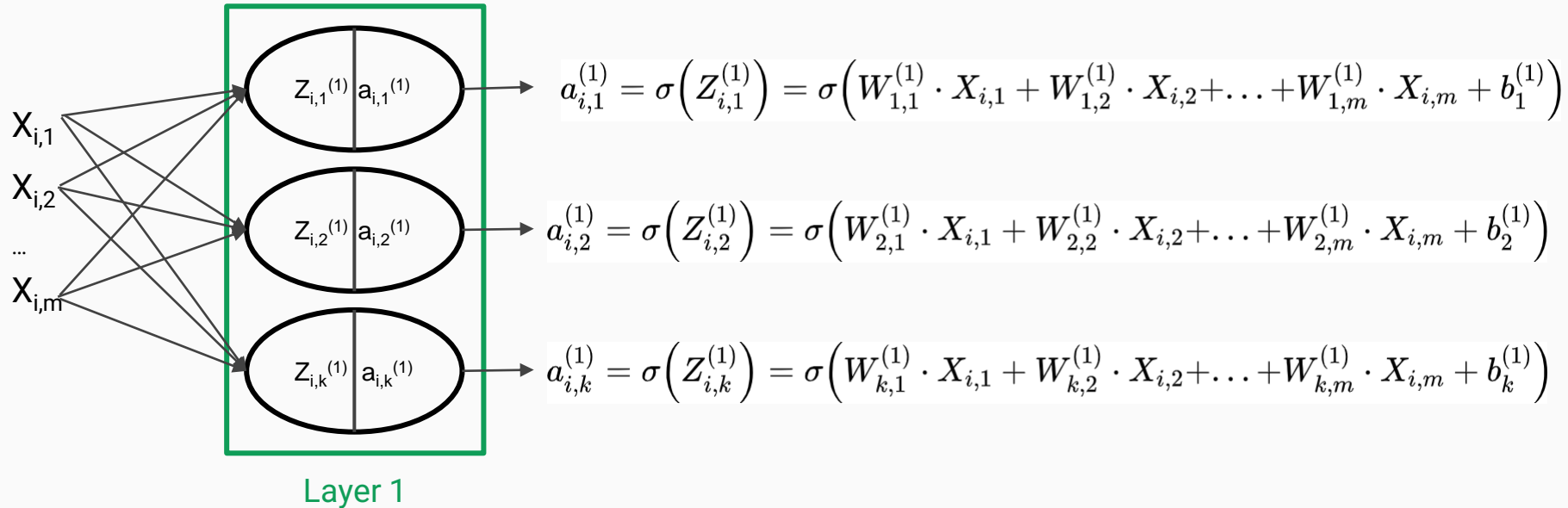
$$\vec{W} = \begin{bmatrix} W_1 \\ W_2 \\ b \end{bmatrix} = (\vec{X}^T \cdot \vec{X})^{-1} \cdot \vec{X}^T \cdot \vec{y} \rightarrow \vec{W} = \begin{bmatrix} 0 \\ 0 \\ 0.5 \end{bmatrix}$$



Neurona



Layer Lineal con función de activación

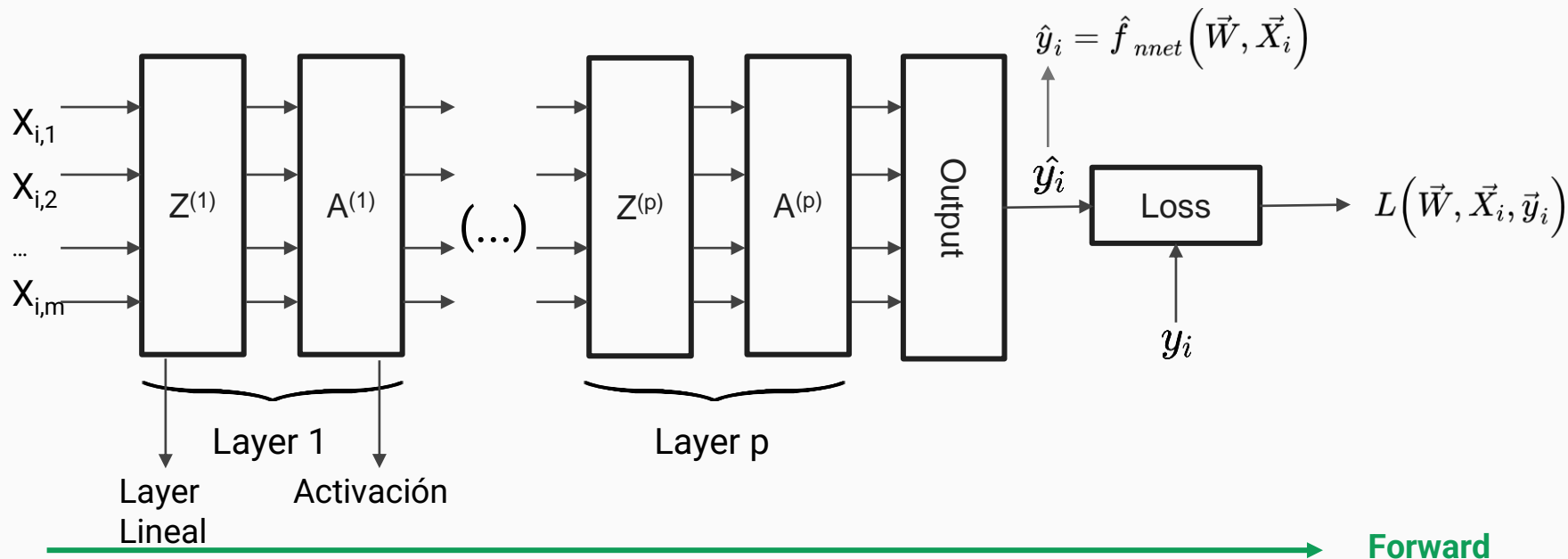


En formato matricial: $\vec{A}_i^{(1)} = \sigma(\vec{Z}_i^{(1)}) = \sigma(\vec{W}^{(1)} \cdot \vec{X}_i + \vec{b}^{(1)})$

\swarrow # de parámetros : $k \times (m + 1)$

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

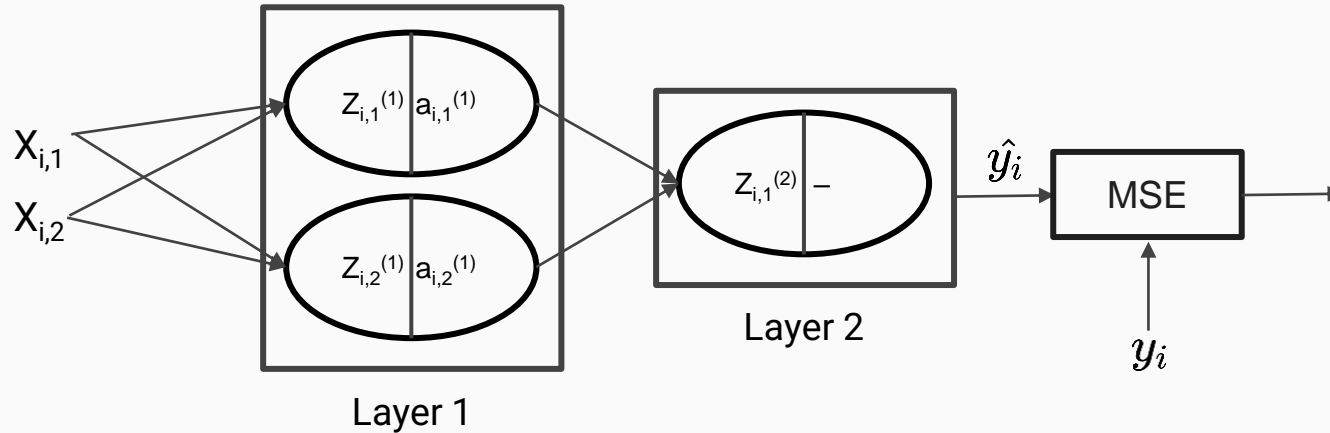
Red neuronal feedforward con p layers



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?

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

Solución de XOR con red neuronal simple

- **Paso Forward:**

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

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

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

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

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

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

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



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

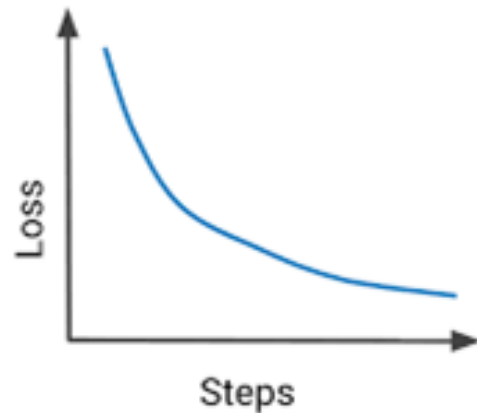
- (2)Error $\rightarrow e_i = (y_i - \hat{y}_i)^2$

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

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

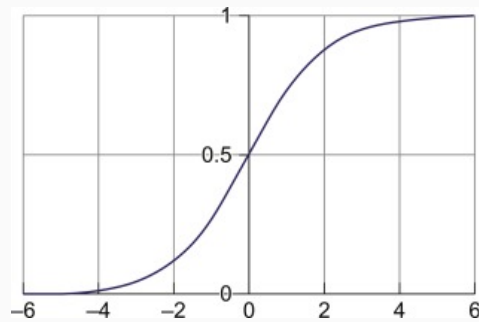
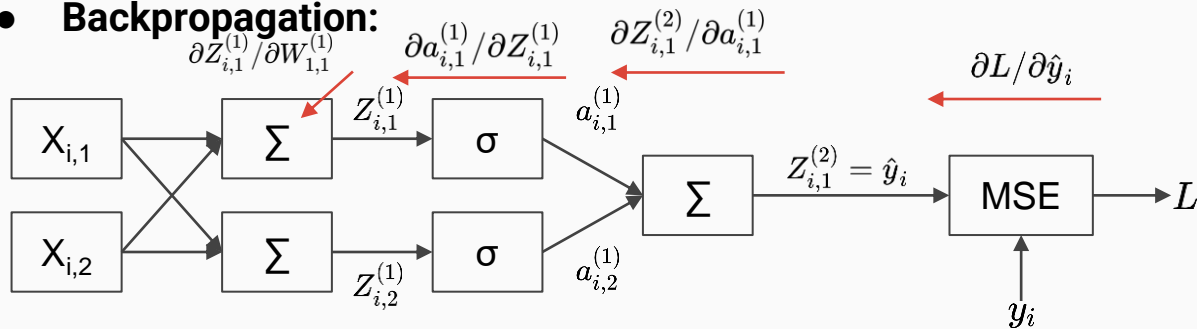
- > Return \vec{W}



Loss plot

Solución de XOR con red neuronal simple

• Backpropagation:



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

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

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

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.