IIMPI
Instituto de Ingeniería Mecánica y Producción Industrial

FACULTAD DE INGENIERÍA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

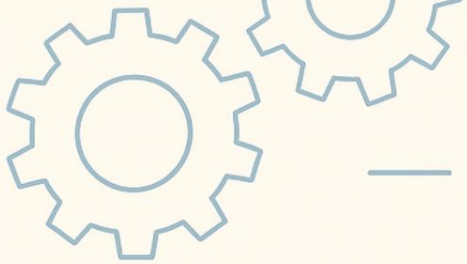
Bienvenidos

Aprendiendo Mecánica a través de Redes Neuronales Informadas por Física

$$div(k\nabla T)=0$$

$$-\nabla 2u = f$$

$$\frac{d^2x}{dt}$$



$$\nu, E$$

$$\sigma = E\varepsilon$$

$$Re = vD/\nu$$

Docentes:
Christian Diaz - Santiago Correa

Información general

Información general

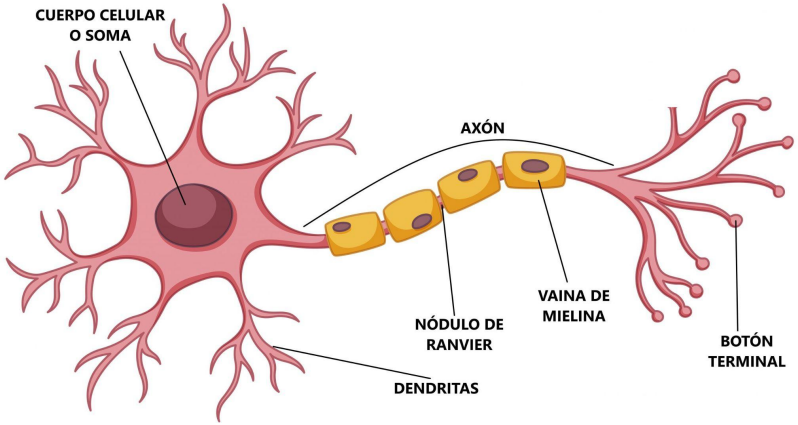
- Taller → 3 créditos en “Actividades Integradoras en Ingeniería Mecánica”
- Duración:
  - 5 Semanas de clases (12, 19, 26 de mayo, 2 y 9 de junio)
  - Lunes de 18:00 a 20:00 hs
- Clases con teórico y práctico en PC
- Aprobación con entrega de trabajo final:
  - Entrega de un informe con la resolución de un problema a elección*.
  - En grupos de a 2 personas.
- Temas:
  - Introducción a Redes Neuronales
  - Introducción a PINNs
  - Aplicación de PINNs a Mecánica del Sólido
  - Aplicación de PINNs a Transferencia de Calor
  - Aplicación de PINNs a Mecánica de los Fluidos

CLASE 1

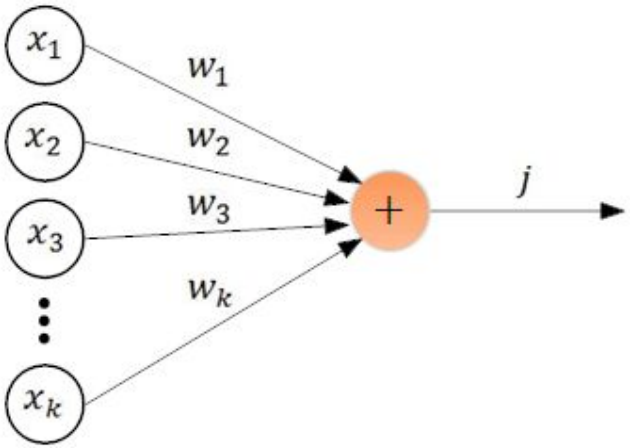
Temario

- Introducción al Machine Learning y a las PINNs
- Feed Forward Neural Networks (FFNN)
- Definiciones
- Forward propagation
- Backward propagation
- Entrenamiento

Introducción al Machine Learning y a las PINNs



Una neurona biológica Recibe señales a través de las dendritas, las procesa en el cuerpo celular y transmite una respuesta por el axón hacia otras neuronas.
La señal de salida depende del potencial eléctrico acumulado y se genera cuando se supera un umbral, activando una respuesta mediante una función tipo escalón.



Una Neurona Artificial es la unidad fundamental de una red neuronal, inspirada en el comportamiento de las neuronas biológicas. Su estructura incluye:

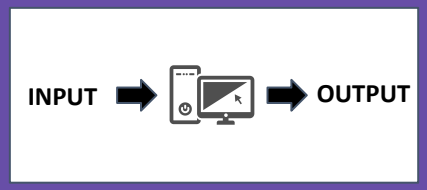
- **Pesos:** cada entrada se asocia a un peso que determina su importancia.
- **Sumador:** acumula las señales ponderadas de entrada.
- **Función de activación:** limita y regula la salida de la neurona.
- **Umbral (bias):** valor ajustable que se optimiza durante el entrenamiento.

APRENDIZAJE SUPERVISADO

- Se entrena con pares de datos entrada/salida.
- El modelo aprende una función que relaciona los pares de datos.
- El modelo es capaz de predecir a partir de una nueva entrada
- Resuelve problemas de clasificación o regresión.

APRENDIZAJE NO SUPERVISADO

- El modelo es capaz de identificar estructuras o patrones en los datos sin necesidad de conocimiento de los resultados.
- Permite explorar grandes conjuntos de datos sin la necesidad de etiquetar.
- Clustering, detección de anomalias



APRENDIZAJE POR REFUERZO

- El modelo aprende a tomar decisiones en un entorno para maximizar una recompensa acumulativa.
- No requiere datos etiquetados.
- El modelo aprende a través de la retroalimentación recibida en forma de recompensas o penalizaciones.

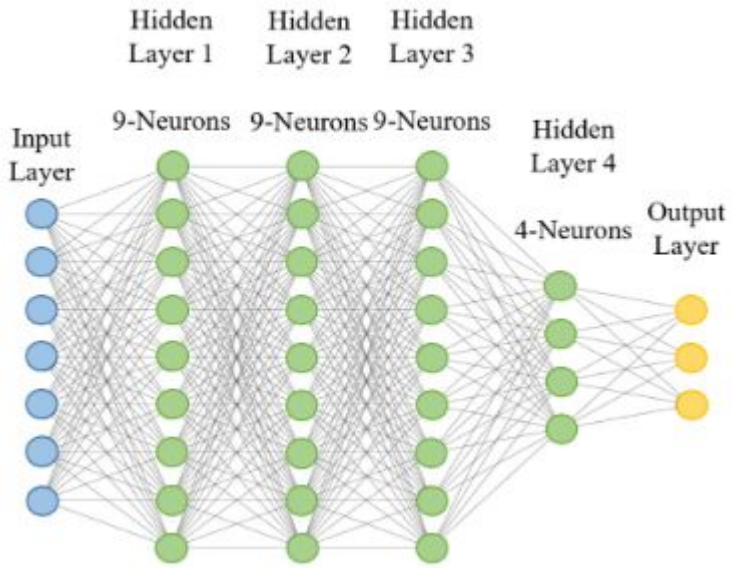
https://machinelearningmastery.com/types-of-learning-in-machine-learning/

Arquitecturas de Redes Neuronales y sus Aplicaciones

REDES NEURONALES PROFUNDAS (DNN)

Son modelos compuestos por múltiples capas de neuronas, donde cada neurona de una capa está densamente conectada con todas las neuronas de la siguiente capa. En este tipo de red cada neurona tiene asociado un peso (que ajusta la importancia de cada entrada) y una función de activación (que introduce no linealidad en la salida). Los datos de entrada se transforman progresivamente a medida que atraviesan las distintas capas del modelo y no hay conexiones directas entre capas no consecutivas; el flujo de información es secuencial, capa por capa.

La gran capacidad de una DNN radica en deducir y ajustar automáticamente las características relevantes de los datos de entrada con el fin de obtener la salida deseada, por lo que la red puede aprender representaciones complejas por sí misma.



Arquitecturas de Redes Neuronales y sus Aplicaciones

CONVOLUTIONAL NEURAL NETWORK (CNNs)

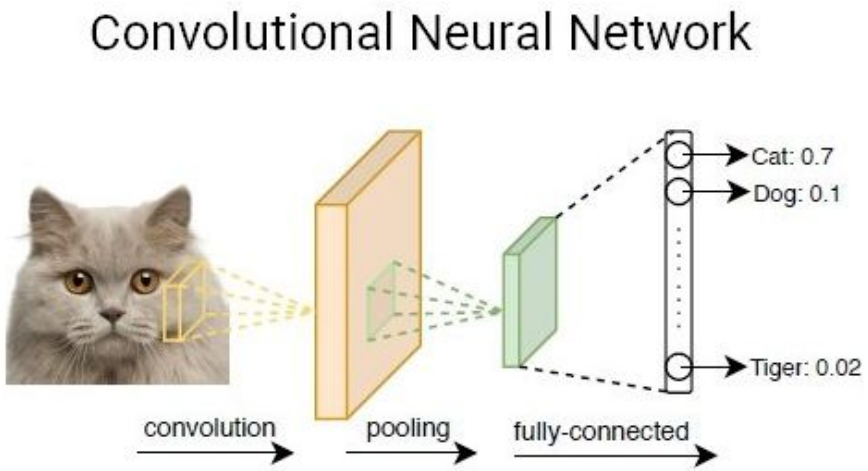
Es un tipo de red neuronal diseñada para trabajar con datos estructurados en forma de rejilla, como las imágenes. Son particularmente eficaces en tareas de visión por computadora, ya que están construidas para reconocer patrones y extraer características visuales automáticamente

Especializadas en datos con estructura espacial, como imágenes. Utilizan filtros convolucionales que extraen patrones locales (bordes, texturas, formas). Se emplean ampliamente en:

- Clasificación y segmentación de imágenes
- Detección de objetos
- Visión por computadora en medicina, vigilancia y vehículos autónomos

Aunque son muy potentes, también tienen limitaciones:

- Requieren grandes volúmenes de datos para entrenarse correctamente.
- Necesitan alto poder computacional (como GPUs) para entrenamientos eficientes
- Pueden sobreajustarse si los datos no son variados o no están bien etiquetados



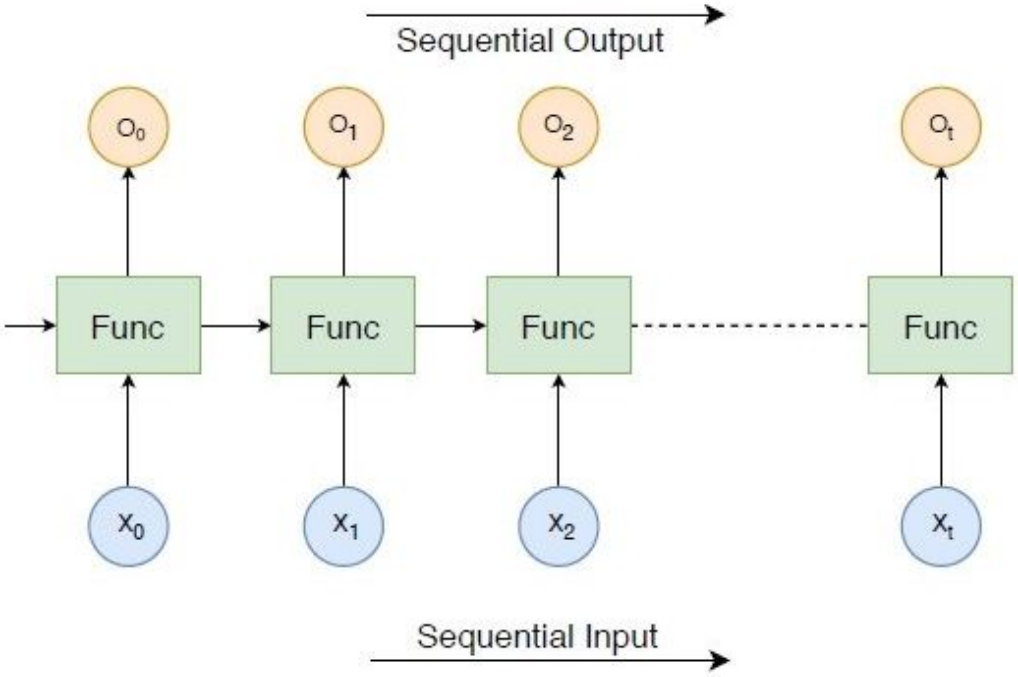
Arquitecturas de Redes Neuronales y sus Aplicaciones

REDES NEURONALES RECURRENTES (RNN/LSTM/GRU)

Las Redes Neuronales Recurrentes (RNNs) son una clase especial de redes diseñadas para procesar datos secuenciales, es decir, donde el orden de los datos importa. Son ampliamente utilizadas en Procesamiento de lenguaje natural (NLP), Análisis de series temporales, Reconocimiento de voz, Generación de texto

RNN poseen conexiones que se retroalimentan sobre sí mismas. Esto les permite:

- Mantener una “memoria interna” o estado oculto que se actualiza en cada paso de la secuencia.
- Capturar dependencias temporales o contextuales: por ejemplo, en una frase, el significado de una palabra puede depender de palabras anteriores.



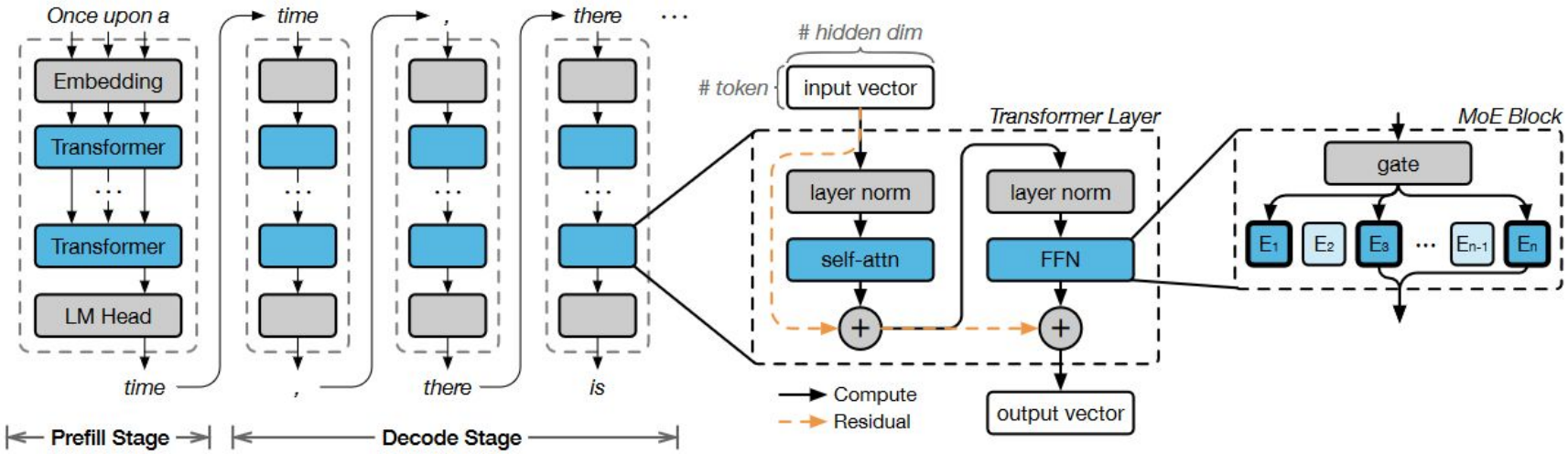
Arquitecturas de Redes Neuronales y sus Aplicaciones

TRANSFORMED BASED LLMs

Los Large Language Models (LLMs) basados en Transformers son modelos de aprendizaje profundo diseñados para procesar y generar texto en lenguaje natural utilizando grandes cantidades de datos y la arquitectura de Transformer como núcleo computacional.
Son los modelos que impulsan sistemas como ChatGPT, Claude, Mistral y Gemini.

El Transformer, introducido por Vaswani et al. (2017) en el trabajo "Attention is All You Need", cambió el paradigma del procesamiento secuencial (usado por RNNs) al introducir el mecanismo de auto-atención (self-attention), que:

- Evalúa la importancia de cada palabra en relación con todas las demás en una secuencia
- Captura dependencias a largo alcance
- Permite procesamiento paralelo (a diferencia de las RNN)



Arquitecturas de Redes Neuronales y sus Aplicaciones

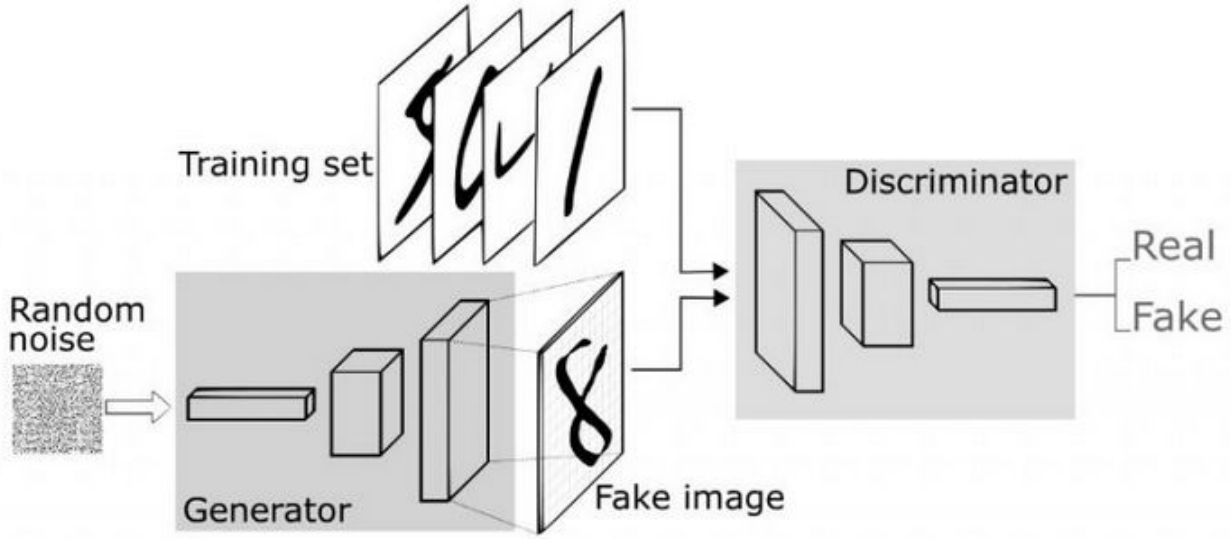
REDES GENERATIVAS ADVERSATIVAS (GANS)

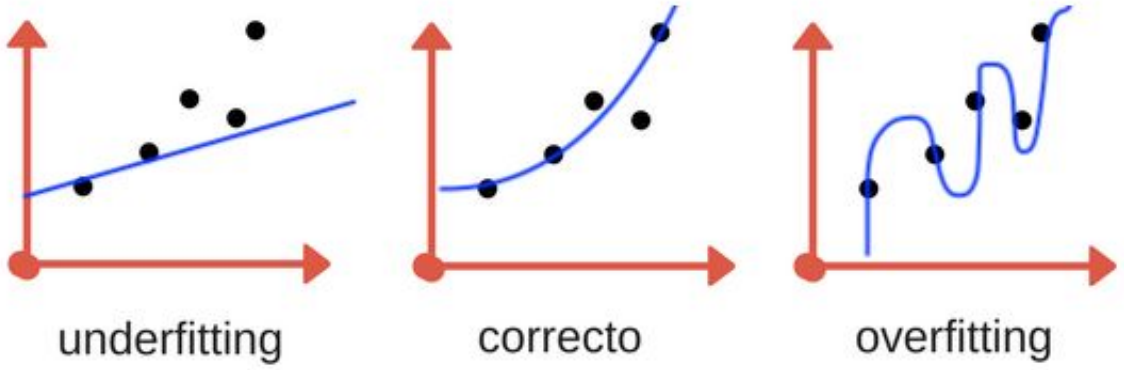
Funcionan a partir de un enfoque innovador de entrenamiento antagonico, en el que dos redes neuronales compiten entre sí: un **generador** y un **discriminador**.

El generador tiene la tarea de crear datos sintéticos que imiten lo mejor posible los datos reales. El discriminador, por su parte, intenta distinguir entre los datos verdaderos y los falsos generados. Durante el proceso de entrenamiento, ambos modelos se mejoran mutuamente. El generador aprende a producir resultados cada vez más convincentes, mientras que el discriminador se vuelve más preciso al detectar imitaciones.

Este tipo de arquitectura suele utilizarse para

- En visión por computadora, permiten generar imágenes realistas, realizar transferencia de estilo o mejorar la resolución de imágenes.
- Útiles para crear datos sintéticos cuando los conjuntos de datos reales son escasos o costosos de obtener.
- Se aplican en diseño de videojuegos, sistemas de reconocimiento facial y la generación de deepfakes.





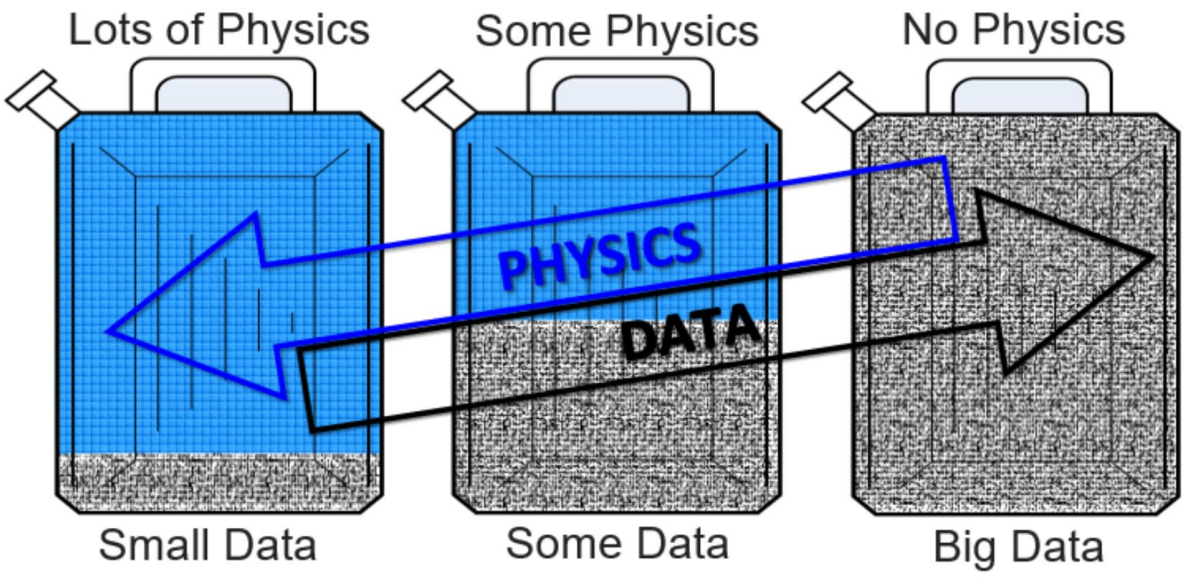
Limitaciones del enfoque clásico basado en datos (data-driven)

Los modelos tradicionales de machine learning **NO** generalizan bien fuera del dominio de entrenamiento.

Son altamente sensibles al ruido, lo que puede provocar:

- Sobreajuste (overfitting): el modelo memoriza datos ruidosos o irrelevantes.
- Subajuste (underfitting): el modelo no capta la complejidad del fenómeno.

Esto es especialmente problemático en contextos donde la adquisición de datos es costosa o limitada (ej. simulaciones físicas o experimentos) y donde se necesita extrapolar más allá de los datos observados.



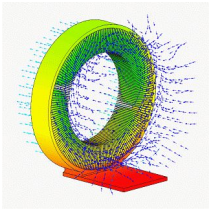
La "5D Law" en machine learning se refiere a cinco características problemáticas que suelen presentarse en los datos del mundo real y que afectan negativamente el rendimiento y la confiabilidad de los modelos.

- **D**inky (pequeños): los datos son insuficientes en cantidad, lo que dificulta generalizar o entrenar modelos robustos.
- **D**irty (sucios): los datos contienen errores, valores faltantes, duplicados o inconsistencias.
- **D**ynamic (dinámicos): los datos cambian con el tiempo (concept drift), por lo que los modelos pueden volverse obsoletos.
- **D**eceptive (engañosos): los datos pueden contener sesgos, correlaciones espurias o variables irrelevantes que engañan al modelo.
- **D**ata (datos en general): la simple presencia de datos no garantiza valor ni calidad; tener "muchos datos" no es lo mismo que tener "buenos datos".

Physics Informed Neural Networks (PINNs) son un enfoque de aprendizaje profundo que incorpora las leyes físicas que rigen un fenómeno directamente en el proceso de entrenamiento. Esto permite modelar y predecir sistemas complejos de manera más precisa, asegurando al mismo tiempo que las soluciones respeten los principios físicos fundamentales.

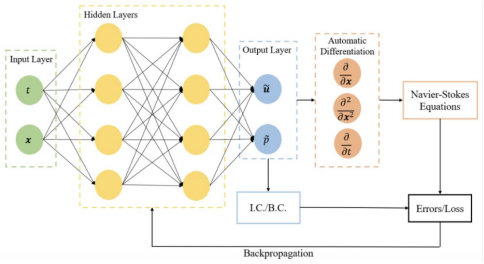
NEURAL NETWORK

CONOCIMIENTO DE LA FÍSICA

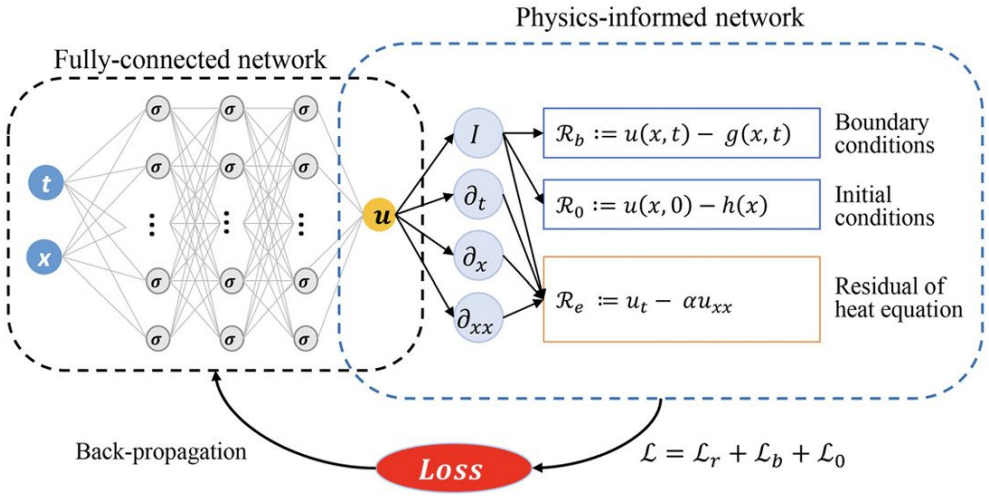


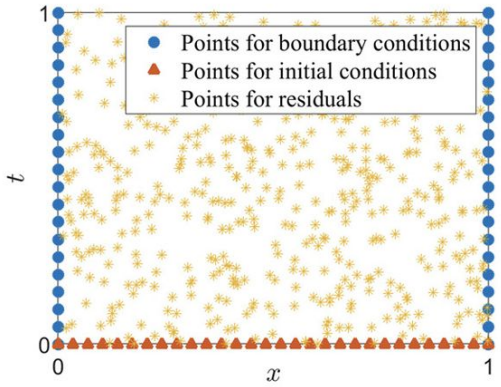
$$\frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = \nu\nabla^2\mathbf{u} - \frac{1}{\rho}\nabla p + \frac{1}{\rho}\mathbf{f}$$

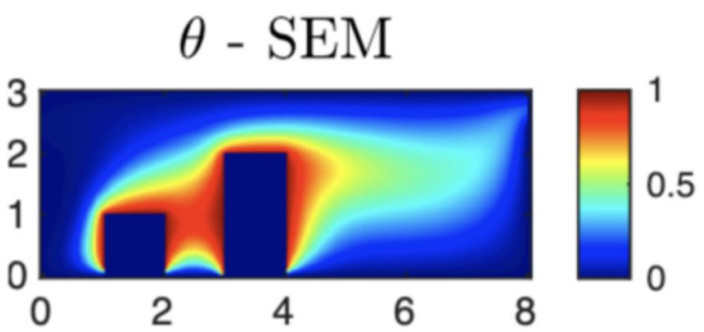
PHYSICS INFORMED NEURAL NETWORK

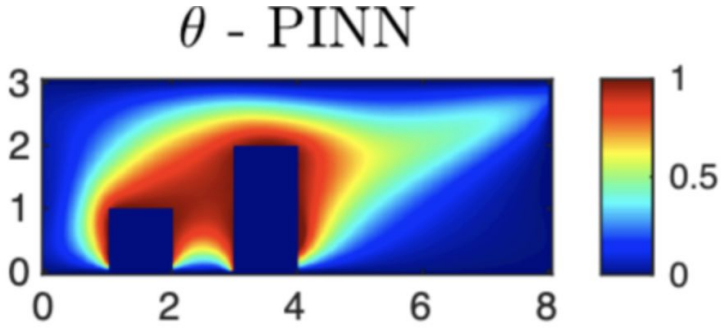


Cai, S., Wang, Z., Wang, S., Perdikaris, P., and Karniadakis, G. E. (April 21, 2021). "Physics-Informed Neural Networks for Heat Transfer Problems." ASME. J. Heat Transfer. June 2021; 143(6): 060801. https://doi.org/10.1115/1.4050542

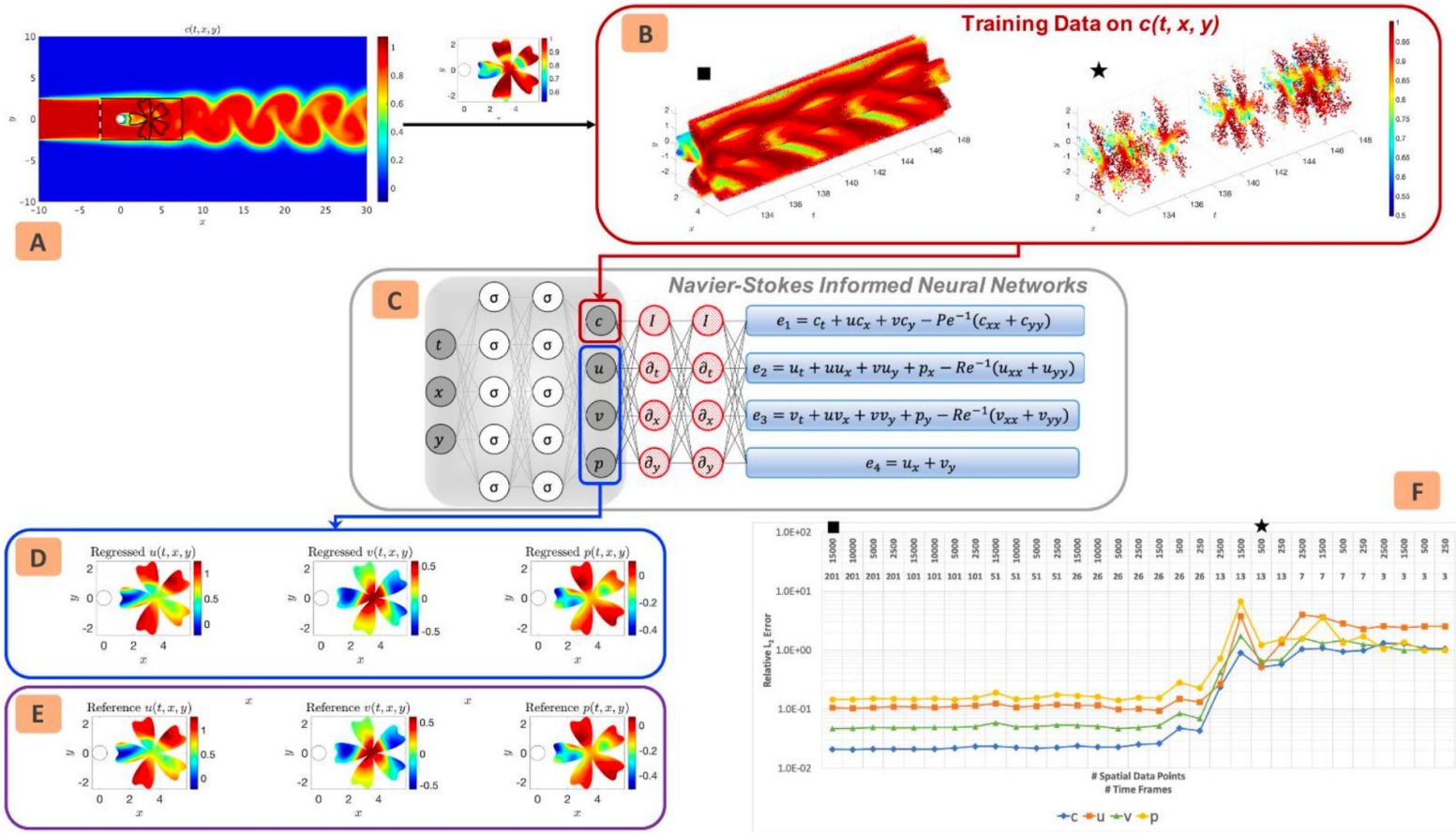




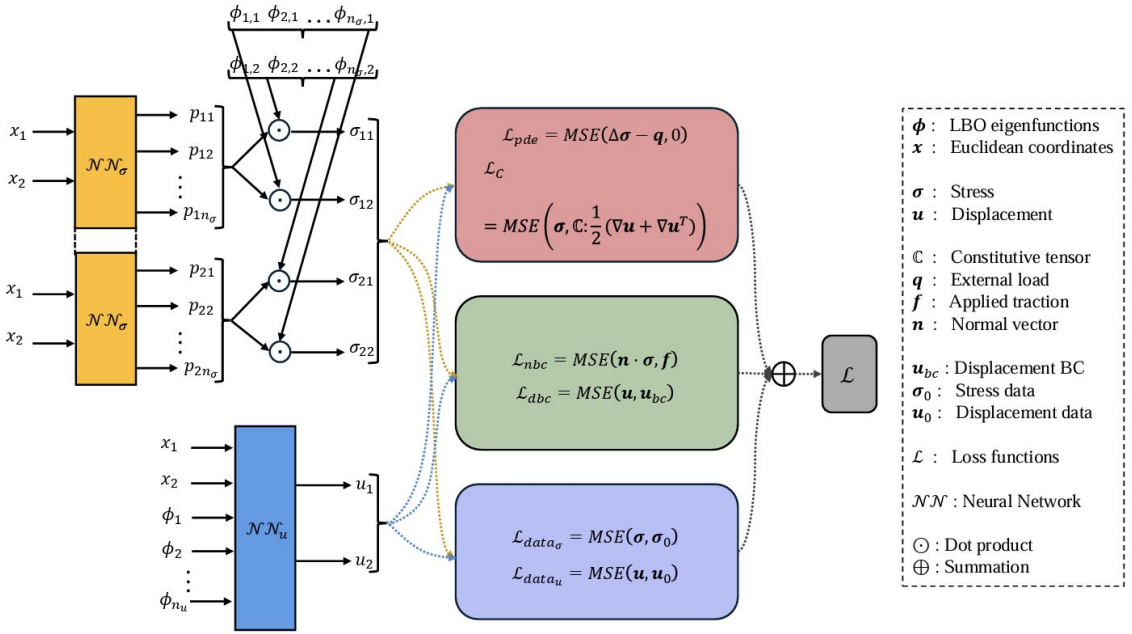


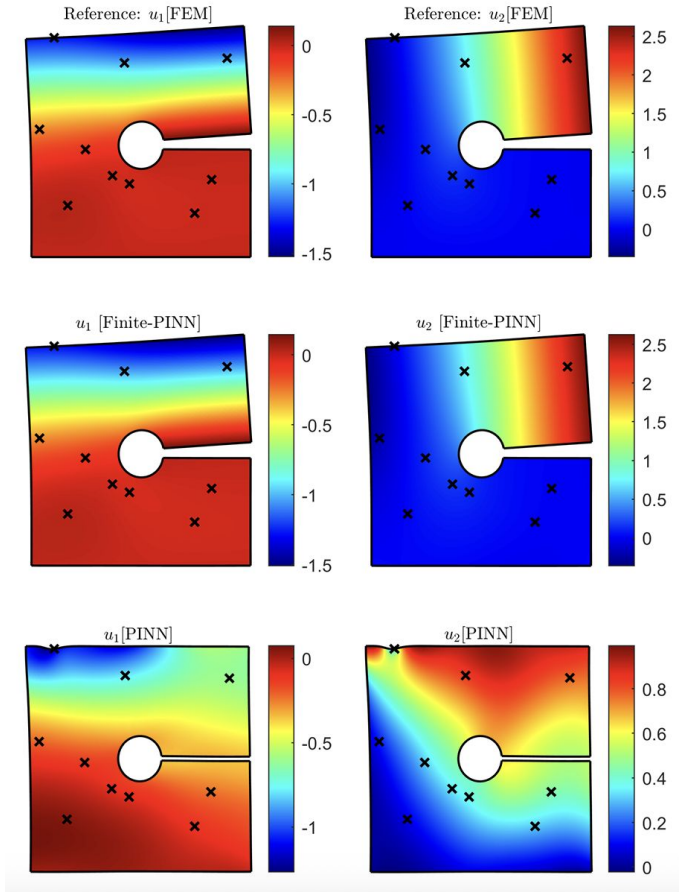


Maziar Raissi et al. ,Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science 367, 1026-1030(2020). DOI:10.1126/science.aaw4741



Li, H., Miao, Y., Khodaei, Z. S., & Aliabadi, M. H. (2024). Finite-PINN: A Physics-Informed Neural Network Architecture for Solving Solid Mechanics Problems with General Geometries. arXiv preprint arXiv:2412.09453.

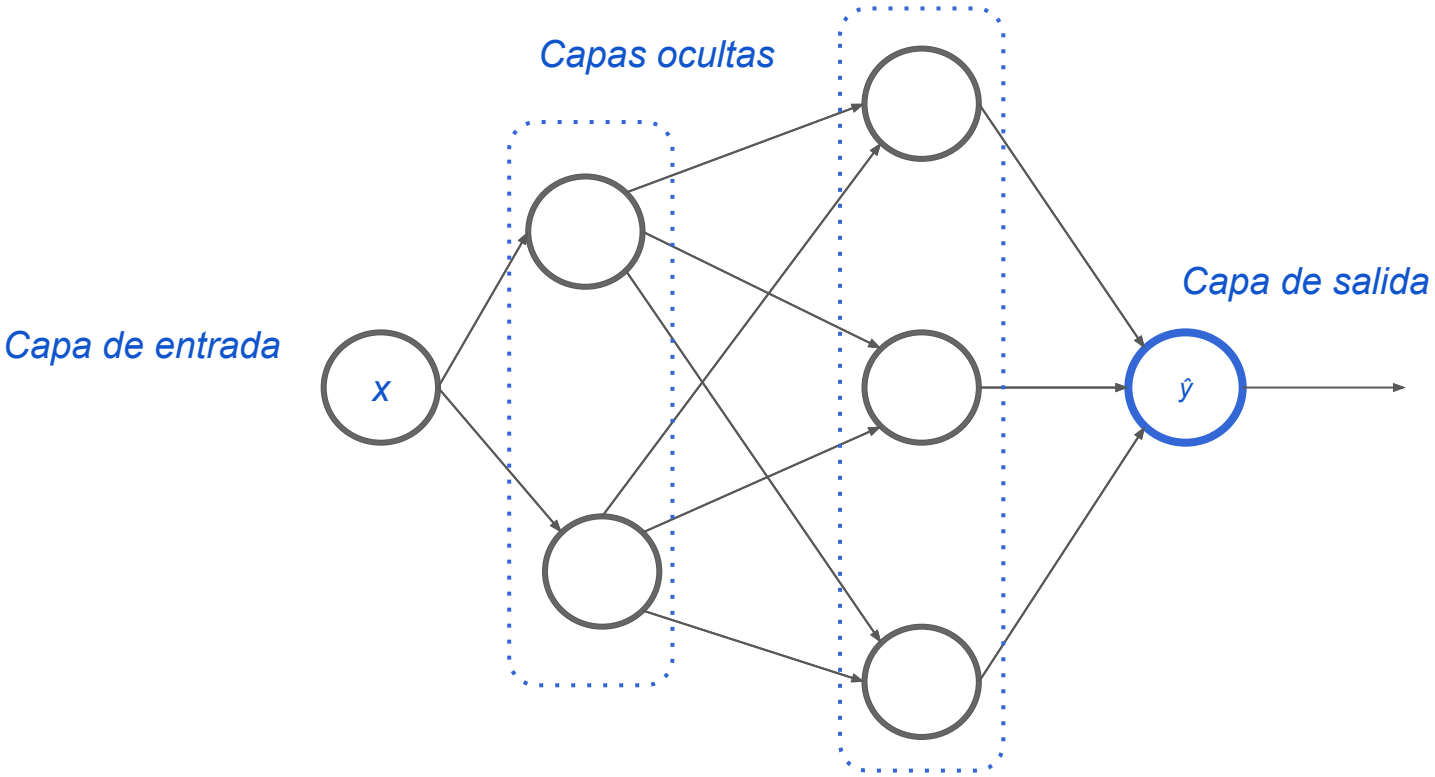




Feed Forward Neural Network (FFNN)

FEED FORWARD NEURAL NETWORK

Definiciones de arquitectura



Neuronas

Conexiones

Forward Propagation

Podemos decir que el objetivo de una Red Neuronal es representar una función $y=f(x)$ desconocida.

Entonces, se busca una red que para determinado conjunto de datos x, produzca una salida $\hat{y}$, tan parecida como sea posible a y.

El proceso por el cual la red toma x y "produce" $\hat{y}$ se conoce como "Forward propagation" y matemáticamente se puede reducir a: $\hat{y} = NN(x, W, b)$

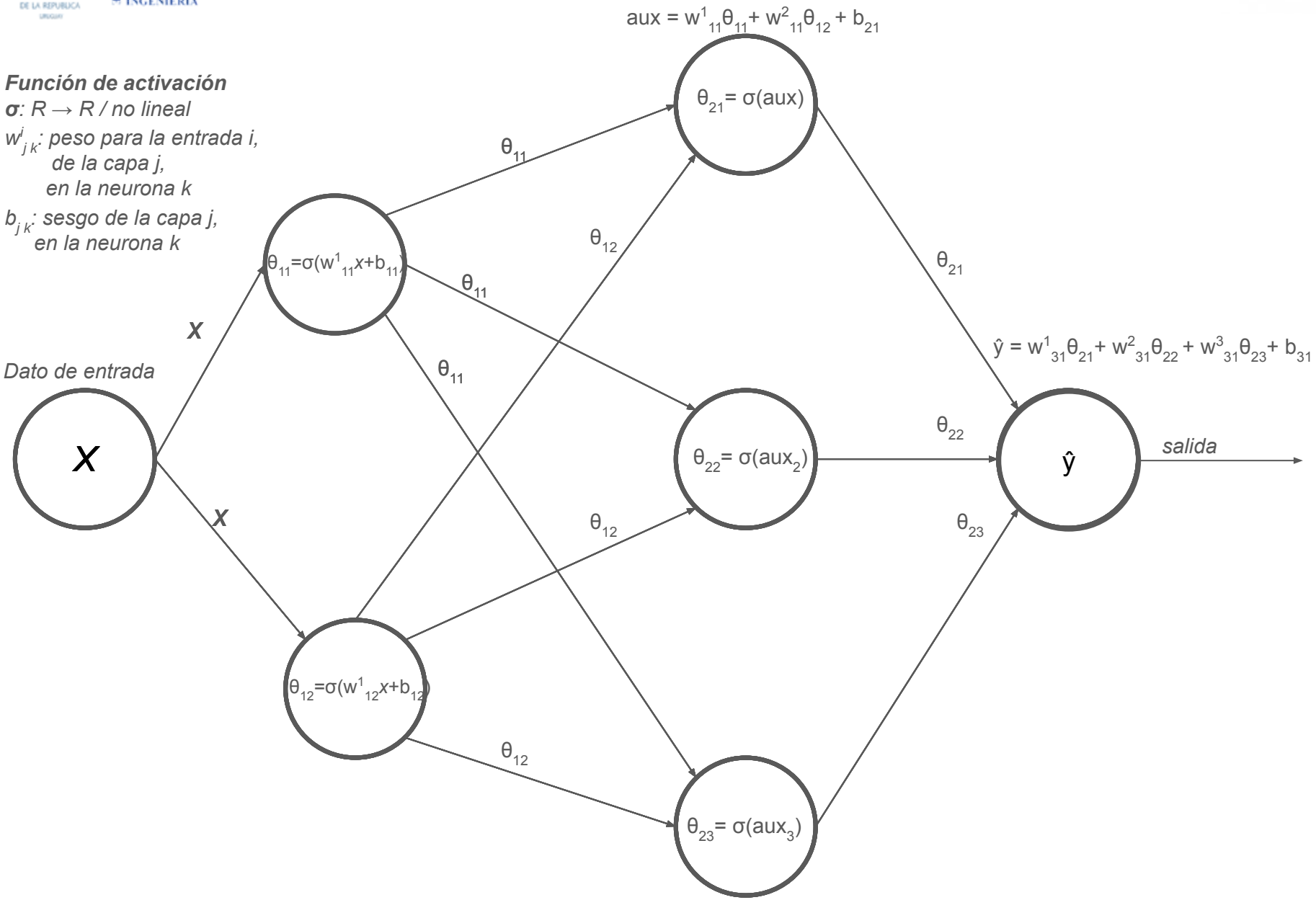
Donde W y b son parámetros de la red que se definirán más adelante.

Función de activación

***σ**: R → R / no lineal*

$w^{i}_{j\,k}$*: peso para la entrada i, de la capa j, en la neurona k*

$b_{j\,k}$*: sesgo de la capa j, en la neurona k*



Backward Propagation

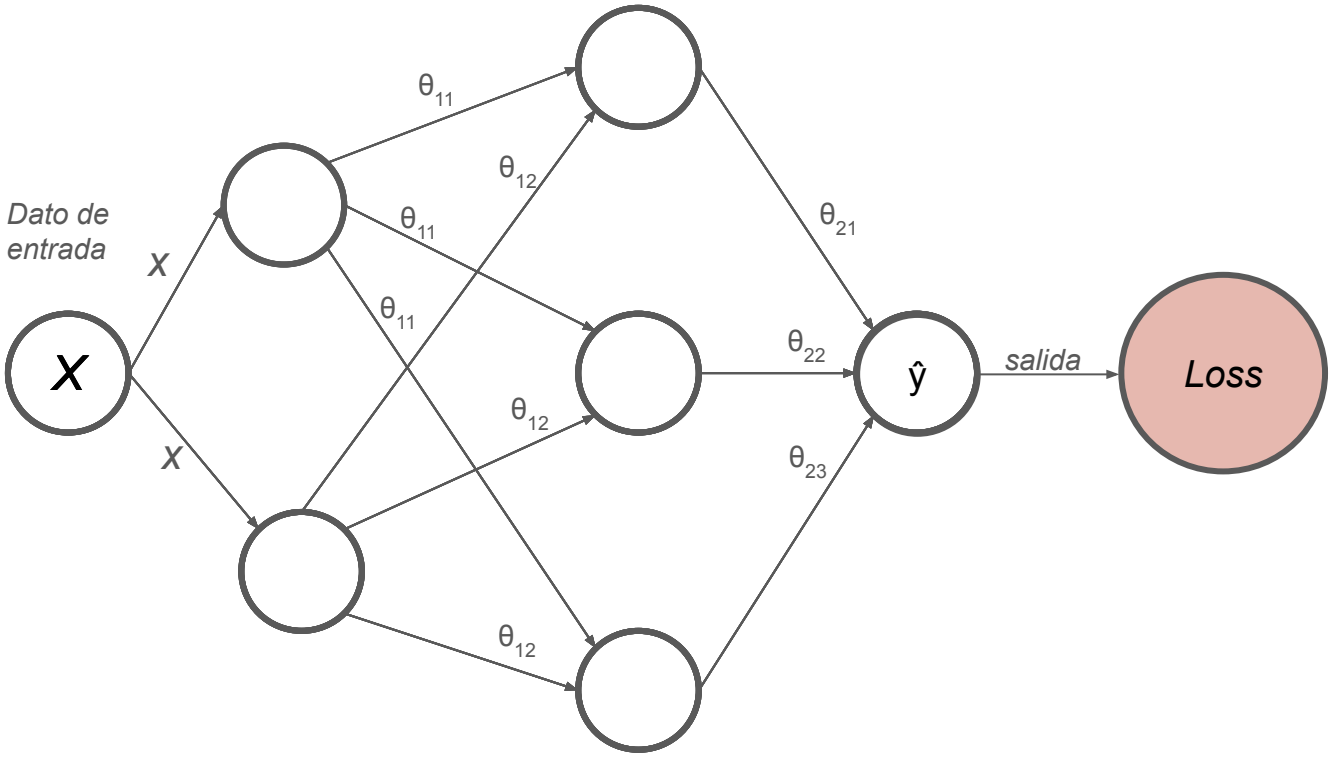
Dado que el objetivo de una Red Neuronal es representar una función $\mathbf{y}=f(\mathbf{x})$ desconocida, a través de $\hat{\mathbf{y}} = NN(\mathbf{x}, \mathbf{W}, \mathbf{b})$, lo que en definitiva se busca es minimizar el error entre $\hat{\mathbf{y}}$, e $\mathbf{y}$.

Además, como los únicos parámetros modificables de la red son $\mathbf{W}$ y $\mathbf{b}$, el problema se reduce a encontrar los pesos y sesgos que "mejor aproxima la solución" $(f(\hat{\mathbf{y}},\mathbf{y}))$.

Matemáticamente: $\min_{\mathbf{W},\mathbf{b}} f(\hat{\mathbf{y}}, \mathbf{y})$

Este es un problema de optimización para el cual existen diferentes estrategias. Una de ellas es modificar iterativamente los $\mathbf{W}$ y $\mathbf{b}$ (con alguna ley) y evaluar la $f(\hat{\mathbf{y}},\mathbf{y})$ hasta alcanzar un valor razonable.

Este proceso de actualización es conocido como "Backward Propagation".



Loss - Función de Pérdida.

Función que evalúa la diferencia entre el valor esperado (y), y el valor obtenido de la red ŷ.
Es una medida del error.
Ej.: *MSE*(y , ŷ)

→ Loss = L(y , ŷ)

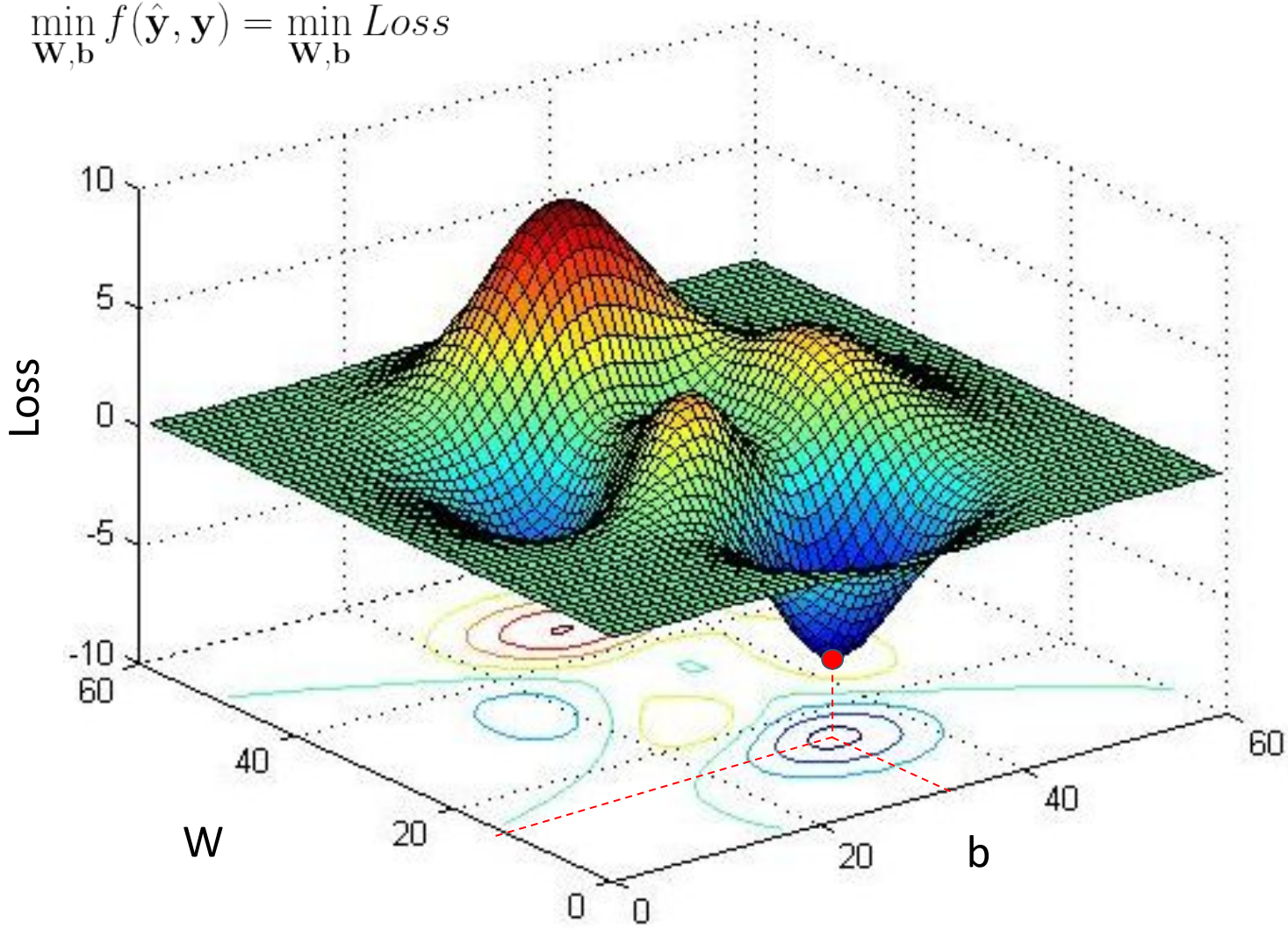
Backward Propagation.

Es un **procedimiento** algorítmico que, dado el valor de la **pérdida** en la salida de la red, propaga ese error hacia atrás a través de las capas, calculando en cada una los **gradientes** necesarios para **actualizar los parámetros** mediante un algoritmo de optimización (por ejemplo, descenso del gradiente).

Backward Propagation.

El procedimiento de actualización de los parámetros se basa en conocer como la Loss varía con respecto a cada parámetro porque el proceso de optimización busca:

$$\min_{\mathbf{W},\mathbf{b}} f(\hat{\mathbf{y}}, \mathbf{y}) = \min_{\mathbf{W},\mathbf{b}} Loss$$



Backward Propagation.

El procedimiento de actualización de los parámetros se basa en conocer como la Loss varía con respecto a cada parámetro porque el proceso de optimización busca:

$$\min_{\mathbf{W},\mathbf{b}} f(\hat{\mathbf{y}}, \mathbf{y}) = \min_{\mathbf{W},\mathbf{b}} Loss$$

Imaginemos solo 2 parámetros

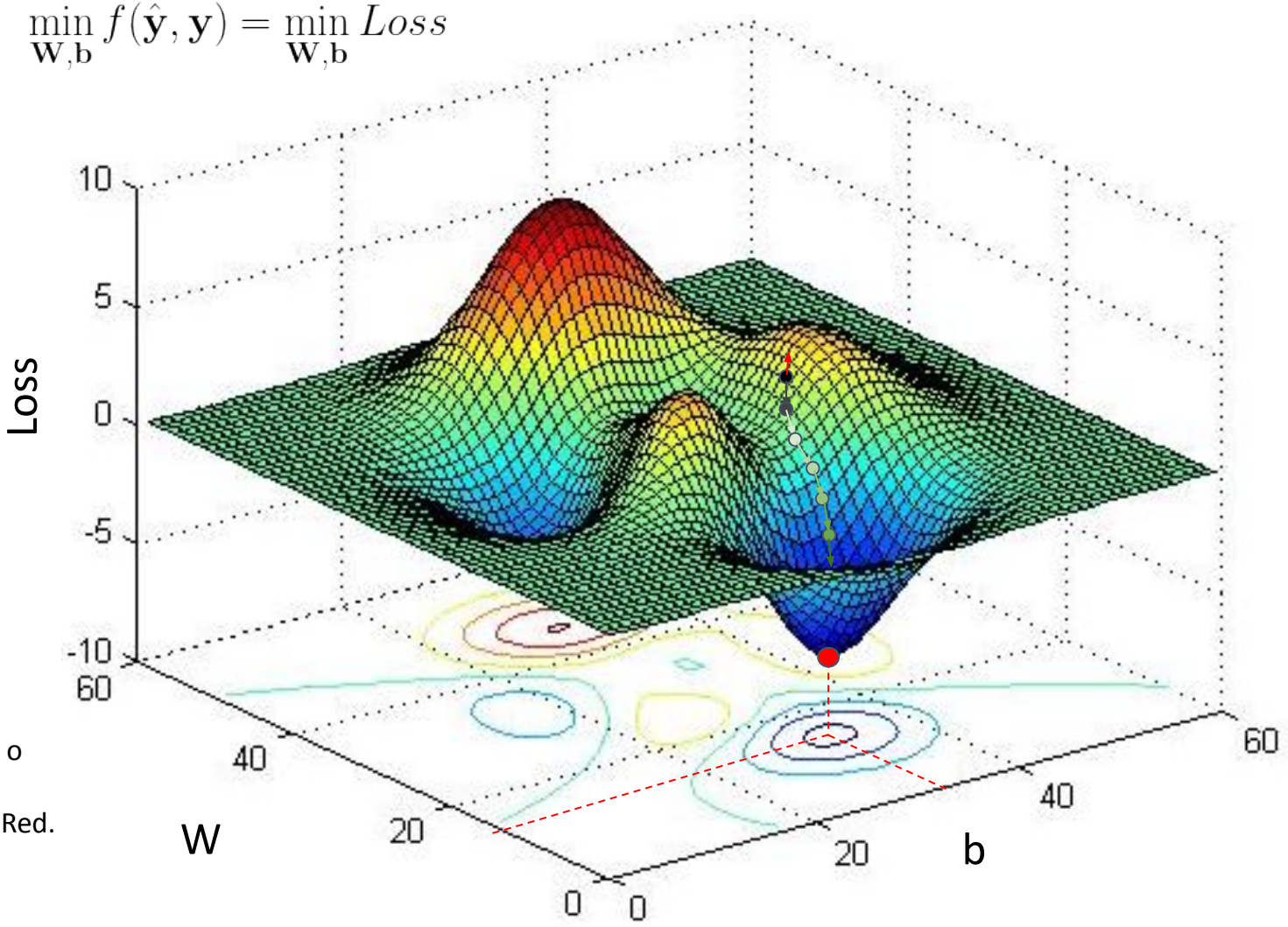
$$\begin{pmatrix} \frac{\partial Loss}{\partial W} \\ \frac{\partial Loss}{\partial b} \end{pmatrix} = \nabla Loss$$

Descenso por gradiente

$$b_{k+1} = b_k - \eta \frac{\partial Loss}{\partial b}$$

$$W_{k+1} = W_k - \eta \frac{\partial Loss}{\partial W}$$

Donde η es el learning rate, o taza de aprendizaje.
Es un hiperparámetro de la Red.



Backward Propagation.

El procedimiento de actualización de los parámetros se basa en conocer como la Loss varía con respecto a cada parámetro porque el proceso de optimización busca:

$$\min_{\mathbf{W},\mathbf{b}} f(\hat{\mathbf{y}}, \mathbf{y}) = \min_{\mathbf{W},\mathbf{b}} Loss$$

Cuando hay más parámetros se debe guardar el **grafo computacional.**

El **grafo computacional** es una representación estructurada de todas las operaciones matemáticas involucradas en la red para calcular la ***Loss***.

De esta forma, al conocer la dependencia de la ***Loss*** con cada parámetro, se pueden escribir las derivadas de

$$b_{k+1} = b_k - \eta \frac{\partial Loss}{\partial b}$$

para cada W y b.

$$W_{k+1} = W_k - \eta \frac{\partial Loss}{\partial W}$$

Backward Propagation.

El procedimiento de actualización de los parámetros se basa en conocer como la Loss varía con respecto a cada parámetro porque el proceso de optimización busca:

$$\min_{\mathbf{W},\mathbf{b}} f(\hat{\mathbf{y}}, \mathbf{y}) = \min_{\mathbf{W},\mathbf{b}} Loss$$

Existen diferentes procesos de actualización de parámetros conocidos como **Optimizadores.**

1. **Descenso por Gradiente**

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}$$

2. **RMSProp**

$$v_t = \gamma v_{t-1} + (1-\gamma)(\nabla_\theta \mathcal{L})^2$$
$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{v_t} + \epsilon} \cdot \nabla_\theta \mathcal{L}$$

3. **ADAM (Adaptative Moment Estimation)**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta \mathcal{L})^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta \leftarrow \theta - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

4. **Adagrad**

Cada parámetro tiene su propia tasa de aprendizaje que disminuye automáticamente a medida que se acumulan actualizaciones.

Como Adagrad suma cuadráticamente los gradientes, la tasa de aprendizaje puede decrecer demasiado con el tiempo.

Esto puede hacer que la optimización se detenga prematuramente (efecto de "parálisis").

5. **L-BFGS**

Es un método de segundo orden aproximado.

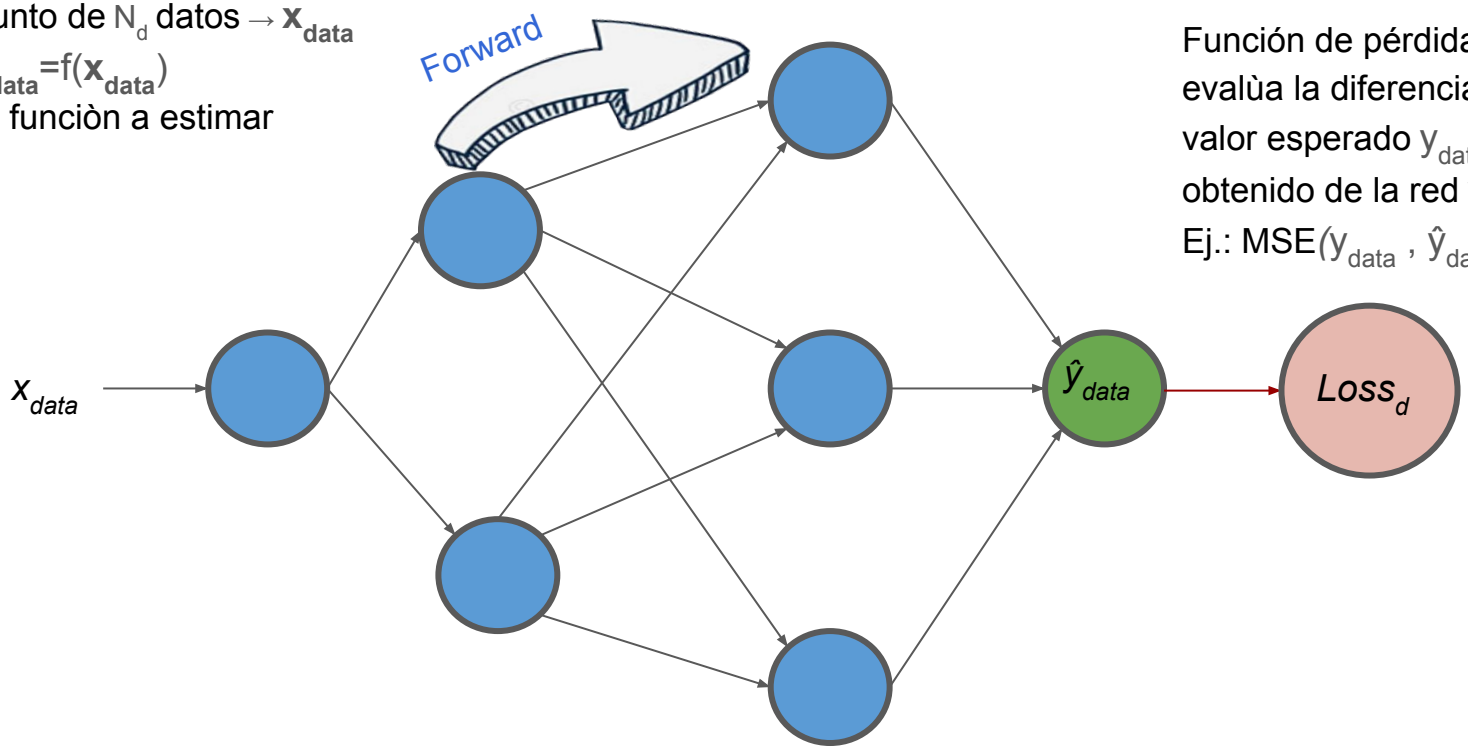
Usa una aproximación de la matriz Hessiana inversa (no la calcula explícitamente) para hacer actualizaciones más informadas que el descenso del gradiente estándar.

Es más preciso que SGD o Adam, aunque más costoso por iteración.

Entrenamiento

Entrenamiento:

Para un conjunto de N_d datos → $\mathbf{x}_{data}$
Se conoce $\mathbf{y}_{data}=f(\mathbf{x}_{data})$
Siendo f(x) la funciòn a estimar



Loss:

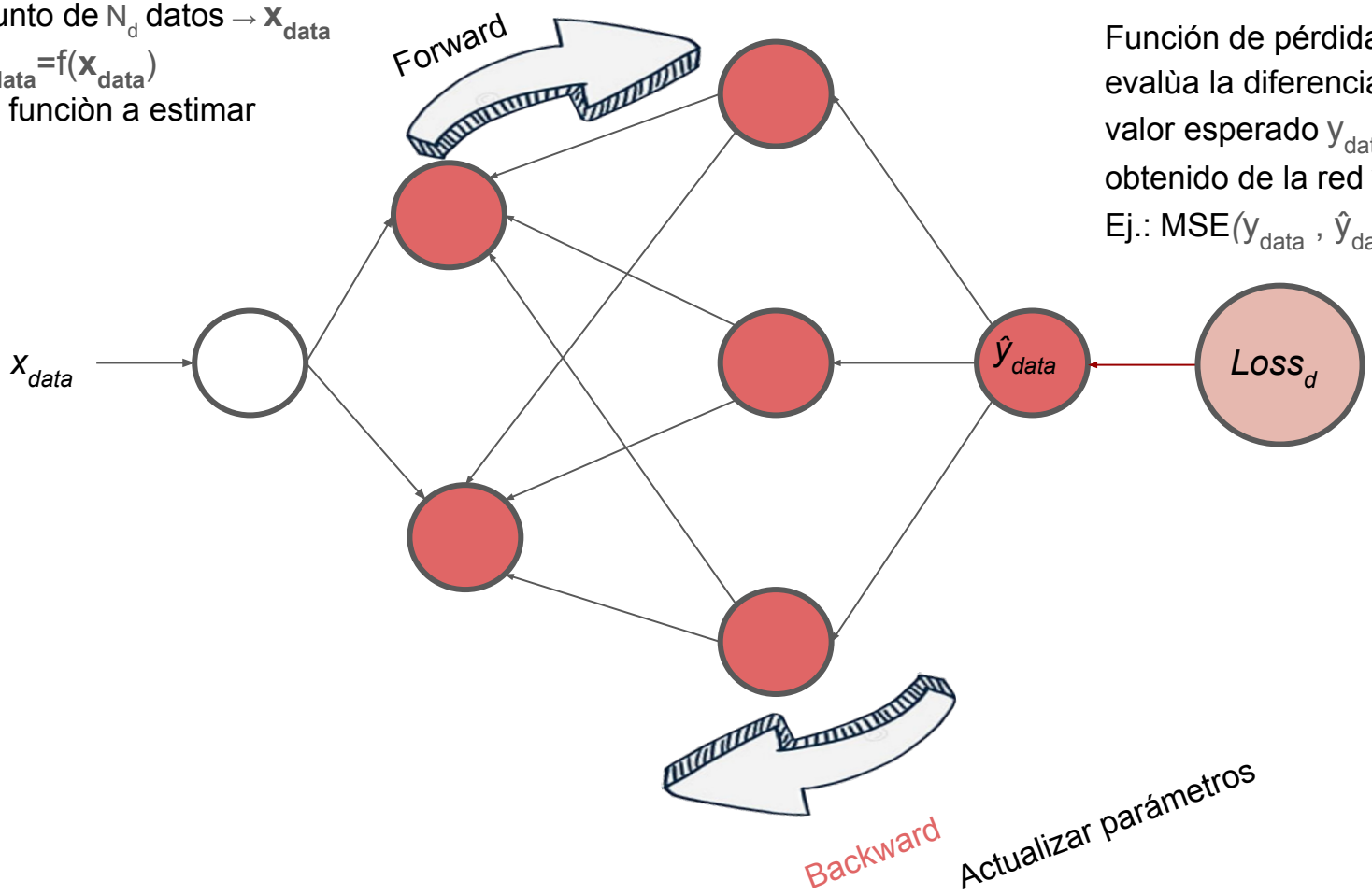
Función de pérdida que evalùa la diferencia entre el valor esperado y_{data} y el valor obtenido de la red $\hat{y}_{data}$
Ej.: MSE$(y_{data}, \hat{y}_{data})$

Entrenamiento:

Para un conjunto de N_d datos → $\mathbf{x}_{data}$
Se conoce $\mathbf{y}_{data}=f(\mathbf{x}_{data})$
Siendo f(x) la funciòn a estimar

Loss:

Función de pérdida que evalùa la diferencia entre el valor esperado y_{data} y el valor obtenido de la red $\hat{y}_{data}$
Ej.: MSE$(y_{data}, \hat{y}_{data})$

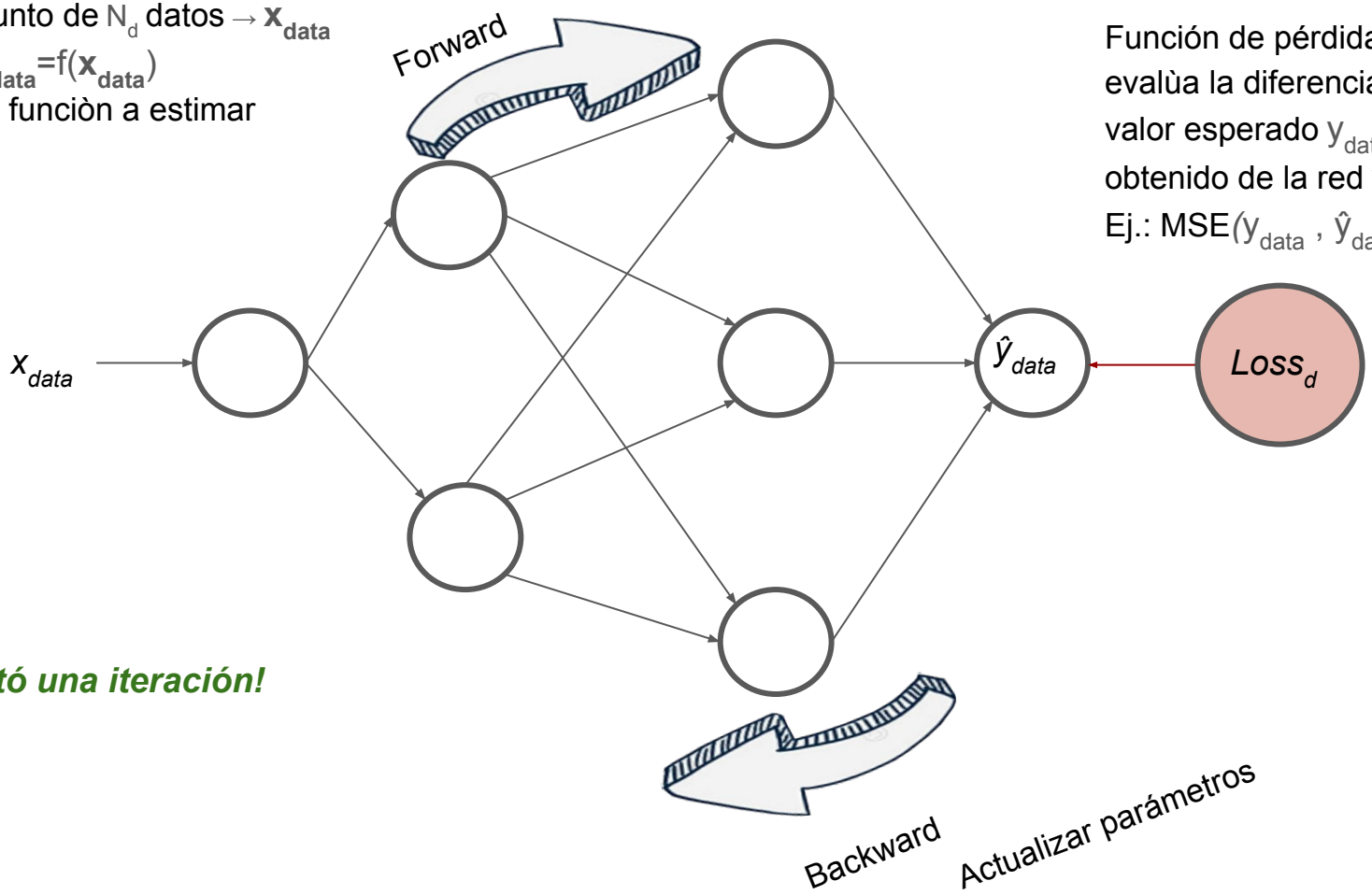


Entrenamiento:

Para un conjunto de N_d datos → $\mathbf{x}_{data}$
Se conoce $\mathbf{y}_{data}=f(\mathbf{x}_{data})$
Siendo f(x) la funciòn a estimar

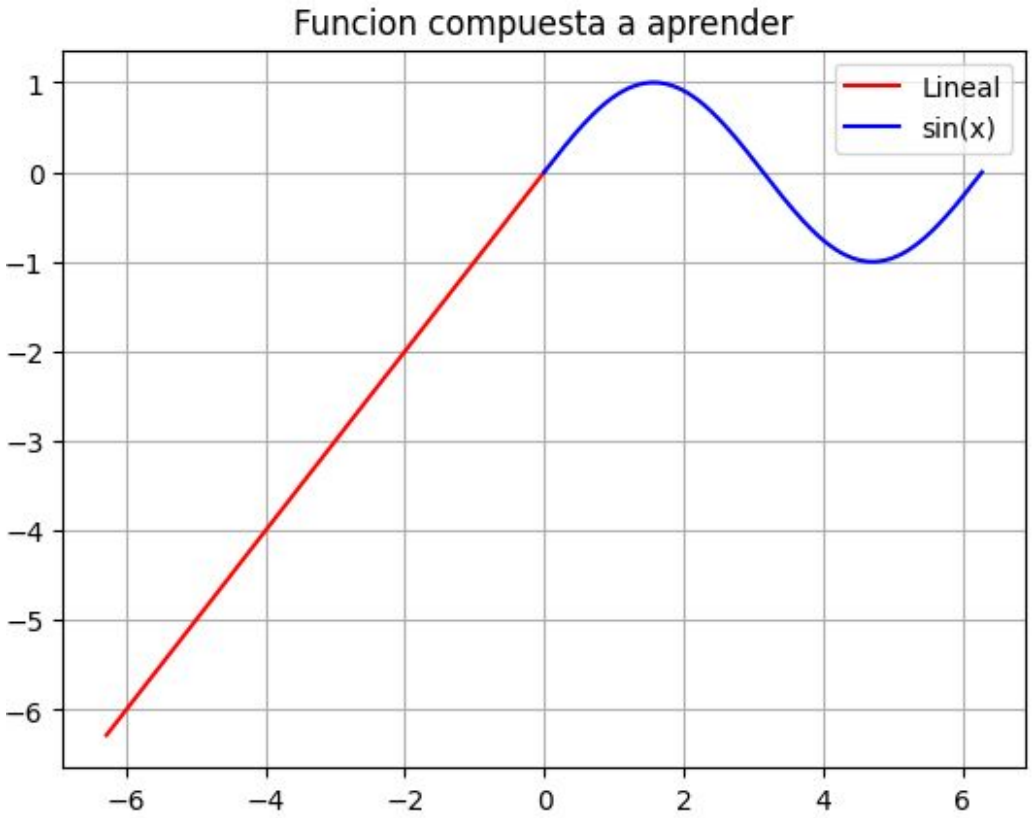
Loss:

Función de pérdida que evalùa la diferencia entre el valor esperado y_{data} y el valor obtenido de la red $\hat{y}_{data}$
Ej.: MSE$(y_{data}, \hat{y}_{data})$



Se completó una iteración!

Veamos el Ejemplo 1



Link a Google Colab (También en EVA)