HENRY

Data Science

M4L10 | Introducción al Deep Learning

→ soyhenry.com

Objetivos

- Explorar la introducción a PyTorch, analizando su estructura y utilidad para el desarrollo de modelos de aprendizaje profundo.
- Examinar la arquitectura y funcionamiento de una red neuronal, abordando conceptos clave como funciones de pérdida, algoritmos de optimización y propagación forward y backward.
- Implementar redes neuronales feedforward y densas, optimizando la representación y transformación de datos en distintos escenarios.

#TEMAS

Agenda

COMENCEMOS →

.01 DL vs ML tradicional

.02 Entrenamiento red neuronal

.03 Redes neuronales profundas

.04 Introducción a Pytorch

.05 Homework

<-->

¿Qué vimos en la lecture?

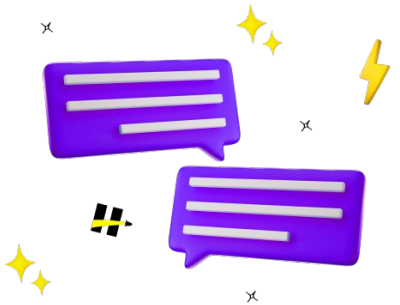
<01>

DL vs ML tradicional

Qué es el Deep Learning

El Deep Learning es una evolución del Machine Learning que lleva la idea de aprender de los datos a otro nivel. Mientras los modelos clásicos ajustan funciones simples (como árboles o regresiones), las redes neuronales profundas aprenden representaciones jerárquicas, capaces de captar relaciones no lineales y complejas entre variables.

Cómo aprende una red profunda



- Las primeras capas captan patrones básicos.
- Las intermedias combinan señales simples en estructuras más complejas.
- Las últimas generalizan el conocimiento, detectando comportamientos globales. En FinShield, por ejemplo, las capas iniciales identifican relaciones entre monto y horario, y las más profundas integran esa información para reconocer patrones de fraude.

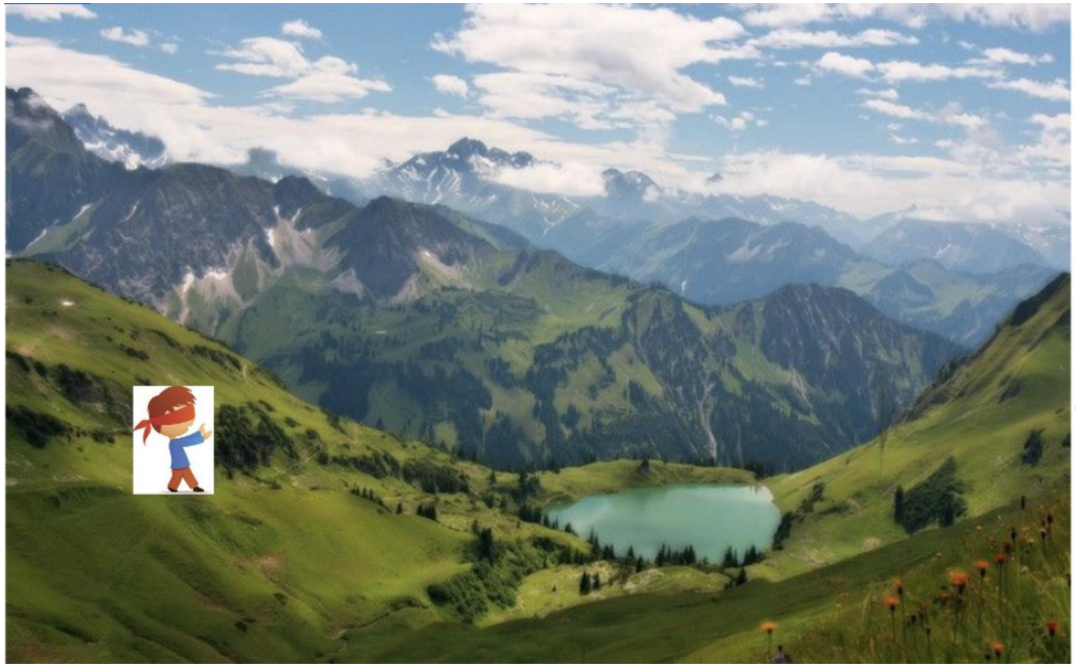
Ejemplos prácticos del salto de calidad

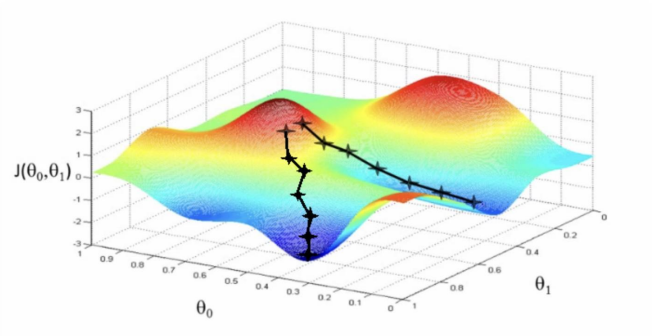
Ámbito	Modelos clásicos	Deep Learning
Lenguaje natural	Solo cuentan palabras	Comprenden significado y contexto
Visión artificial	Necesitan diseñar rasgos manuales	Aprenden automáticamente los patrones
Datos tabulares	Aprenden relaciones simples	Captan interacciones complejas no lineales

Descenso por gradiente: el corazón del aprendizaje

El modelo predice, mide su error con una función de pérdida y ajusta sus pesos para equivocarse menos. Este proceso es como descender una colina hasta el punto más bajo del error. En FinShield, los pesos se modifican en función del tipo de operación (país, monto, hora) para reducir las predicciones incorrectas en miles de ejemplos.

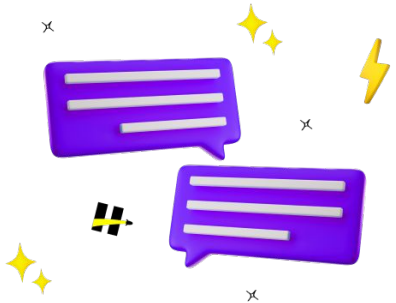
Cómo se ve el Descenso por gradiente





→ soyhenry.com

Cuándo conviene usar Deep Learning



- Cuando hay **grandes volúmenes de datos**.
- Cuando existen **relaciones no lineales o muy sutiles**.
- Cuando se requiere **detectar patrones complejos** (texto, imágenes, fraude).
  En cambio, los métodos clásicos son preferibles cuando los datos son simples o el objetivo es obtener interpretabilidad rápida.

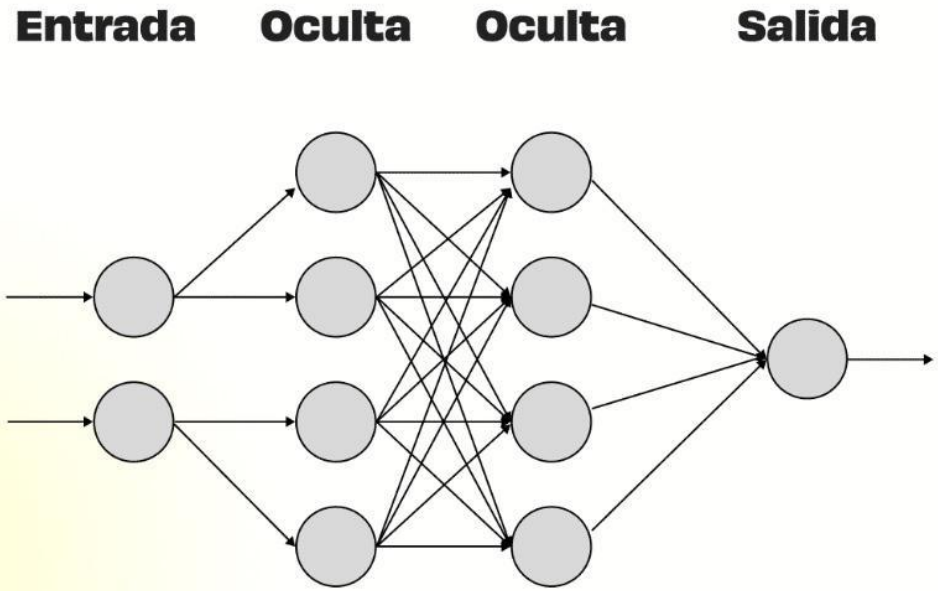
<02>

Entrenamiento red neuronal

Cómo está compuesta una red neuronal

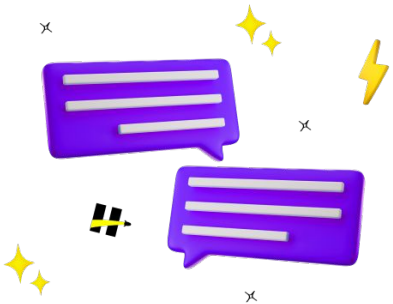
Una red neuronal tiene capas de neuronas conectadas por pesos. Las capas ocultas son las que realmente aprenden. Cada conexión se ajusta según el error cometido, y con el tiempo la red mejora su capacidad para distinguir entre transacciones legítimas y fraudulentas.

Cómo se ve una red neuronal



Proceso de entrenamiento

1. La red recibe los datos.
2. Calcula una predicción (forward).
3. Evalúa el error (función de pérdida).
4. Retropropaga ese error (backward).
5. Ajusta los pesos y repite el ciclo miles de veces.
   Así, cada iteración refina la red hasta alcanzar equilibrio entre precisión y generalización.



Función de pérdida: CrossEntropyLoss

Tipo de problema	Función más adecuada	Qué mide
Clasificación binaria	**CrossEntropyLoss**	Diferencia entre predicción y realidad
Regresión	**MSE (Error cuadrático medio)**	Distancia promedio entre valores

Funciones de activación más usadas

- **ReLU:** deja pasar valores positivos y bloquea negativos → acelera el aprendizaje.
- **Sigmoid:** transforma la salida en probabilidad (0–1) → ideal para salida binaria.
- **Tanh:** genera valores entre -1 y 1 → útil para mantener equilibrio interno.
  En FinShield, las capas internas usan ReLU y la salida, Sigmoid, para expresar probabilidad de fraude.

Optimización y control del aprendizaje

El optimizador **Adam** ajusta dinámicamente la tasa de aprendizaje de cada peso.

Combinado con **early stopping** y **checkpoints**, evita el sobreajuste.
El monitoreo de métricas por epoch —como pérdida, precisión o recall— permite detectar si el modelo realmente aprende o memoriza.

<03>

Redes neuronales profundas

Qué significa “profundidad”

La profundidad se mide por la cantidad de capas ocultas. Cada una añade capacidad para representar relaciones jerárquicas.

En FinShield, tres capas densas permiten pasar de variables crudas a representaciones abstractas de riesgo, detectando patrones cada vez más complejos.

Tipos de arquitecturas

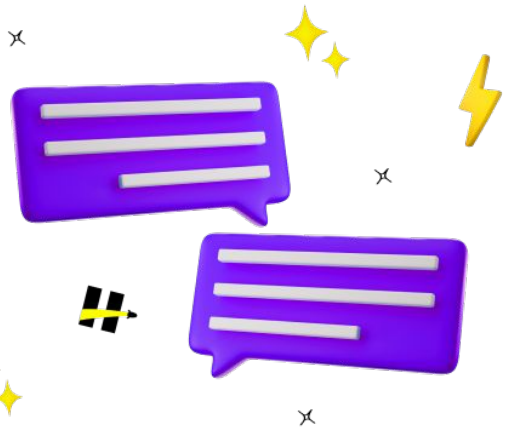
Tipo de red	Ideal para	Característica principal
Feedforward (MLP)	Datos tabulares	Flujo unidireccional de información
CNN	Imágenes o datos espaciales	Detecta patrones locales (bordes, texturas)
RNN / LSTM	Texto o series temporales	Conserva memoria y dependencias en el tiempo

Cómo aprenden CNNs y RNNs

Las **CNNs** ajustan filtros que se desplazan sobre los datos, detectando estructuras visuales.

Las **RNNs** transmiten información de un paso temporal al siguiente, aprendiendo secuencias.

En FinShield, una LSTM podría anticipar fraude analizando secuencias de compras previas.



Dimensionamiento e inicialización de redes

- Pocas capas → el modelo no aprende (underfitting).
- Demasiadas capas → memoriza los datos (overfitting).
- Inicialización **Xavier** o **He** estabiliza los gradientes al inicio.
  En FinShield, la primera capa tiene muchas neuronas y las siguientes se reducen gradualmente hasta la salida binaria.

Regularización: dropout y batch normalization

El **dropout** desactiva neuronas aleatoriamente, obligando al modelo a no depender de una sola ruta de aprendizaje.

La **batch normalization** ajusta internamente las activaciones, estabilizando y acelerando el entrenamiento.

Ambas mejoran la capacidad de generalización y evitan el sobreajuste.

<04>

Introducción a Pytorch

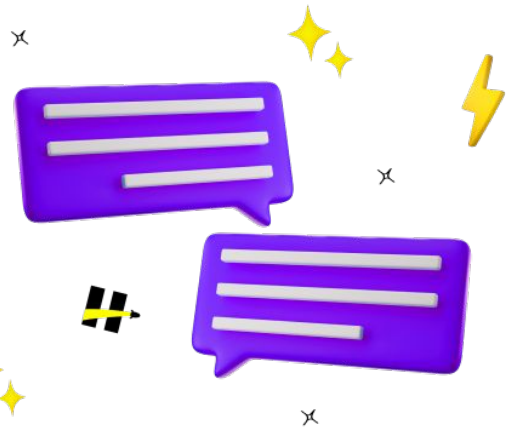
Qué es Pytorch

PyTorch es el framework más usado para redes neuronales por su claridad y flexibilidad.
Su base son los **tensores**, estructuras que almacenan los datos en múltiples dimensiones y pueden operar en GPU, acelerando el entrenamiento.

Autograd y nn.Module

- **Autograd:** calcula gradientes automáticamente y propaga errores sin derivar a mano.

- **nn.Module:** estructura las capas y define el flujo de datos del modelo.
  En FinShield, tres capas densas con ReLU producen una probabilidad de fraude a partir de cada transacción.



Dataset y DataLoader

El Dataset define cómo leer y transformar los datos; el DataLoader los agrupa en mini-batches.

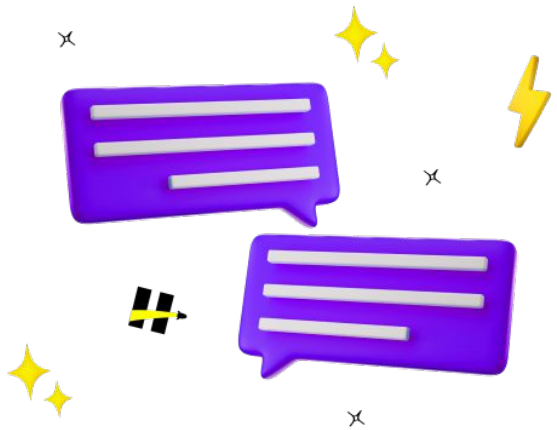
En FinShield, cada lote de 64 transacciones se procesa y actualiza los pesos, garantizando eficiencia y estabilidad durante el entrenamiento.

Autograd y nn.Module

PyTorch permite cambiar entre CPU y GPU sin alterar el modelo.

El **state_dict** guarda los pesos entrenados para reutilizarlos o desplegarlos.

En FinShield, esta estructura permitió conservar la mejor versión del modelo para predecir nuevos fraudes.



<DATA SCIENCE/>

Vayamos a la práctica

Homework

→ soyhenry.com

Consigna

Durante esta actividad trabajarás con datos del mercado financiero simulados a partir del índice **VIX**, conocido como el "índice del miedo" por reflejar la volatilidad esperada del mercado bursátil. El objetivo será construir un modelo de **red neuronal recurrente (LSTM)** utilizando **PyTorch**, capaz de predecir el valor del índice a corto plazo a partir de su comportamiento histórico.

Tu tarea consiste en:

1. Explorar los datos y realizar un **análisis exploratorio (EDA)** básico para detectar tendencias y estacionalidades.
2. Implementar una **validación cruzada de series temporales**, respetando la naturaleza secuencial de los datos.
3. Construir y entrenar una **red LSTM** en PyTorch para predecir el valor del VIX en los próximos días.
4. Evaluar el rendimiento del modelo con métricas adecuadas (RMSE, MAE, R^2) y comparar con un modelo base (por ejemplo, persistencia o regresión lineal).

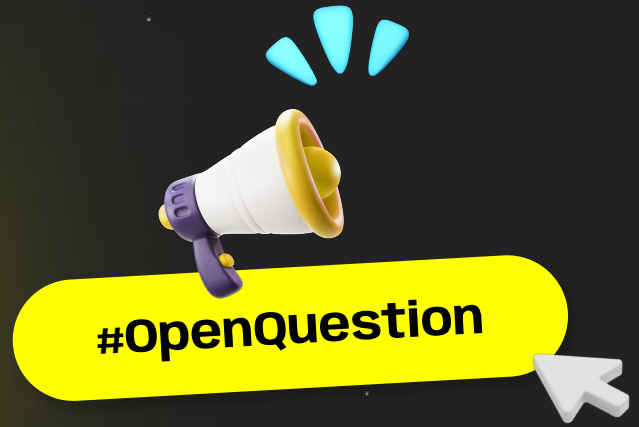
Tareas a realizar

- Cargar el dataset `econotrend_vix_sim.csv` y realizar una **limpieza y normalización** de los datos.
- Implementar una **ventana deslizante** para construir secuencias de entrenamiento (lookback = 10 días).
- Dividir los datos respetando el orden temporal (sin barajado).
- Construir una **red LSTM en PyTorch** con al menos una capa recurrente y una capa densa de salida.
- Entrenar el modelo y graficar la comparación entre **valores reales y predichos**.
- Evaluar el rendimiento con al menos dos métricas cuantitativas (MAE y RMSE).
- Explicar brevemente si la serie analizada muestra evidencia de ser predecible o si se comporta como un **random walk**.

Extra credit

- Implementar **una segunda arquitectura** (por ejemplo, LSTM bidireccional o GRU) y comparar resultados.
- Añadir **regresores externos**, como el rendimiento del S&P 500 o tasas de interés simuladas.
- Guardar y recargar los pesos del modelo con `torch.save()` y `torch.load()`.
- Realizar una **predicción multi-step** (por ejemplo, horizonte de 5 días).
- Incluir un gráfico de **curvas de entrenamiento (loss vs epoch)** para discutir overfitting o estabilidad del modelo.

HENRY



¿Preguntas?

HENRY

¡Muchas gracias!

→ soyhenry.com