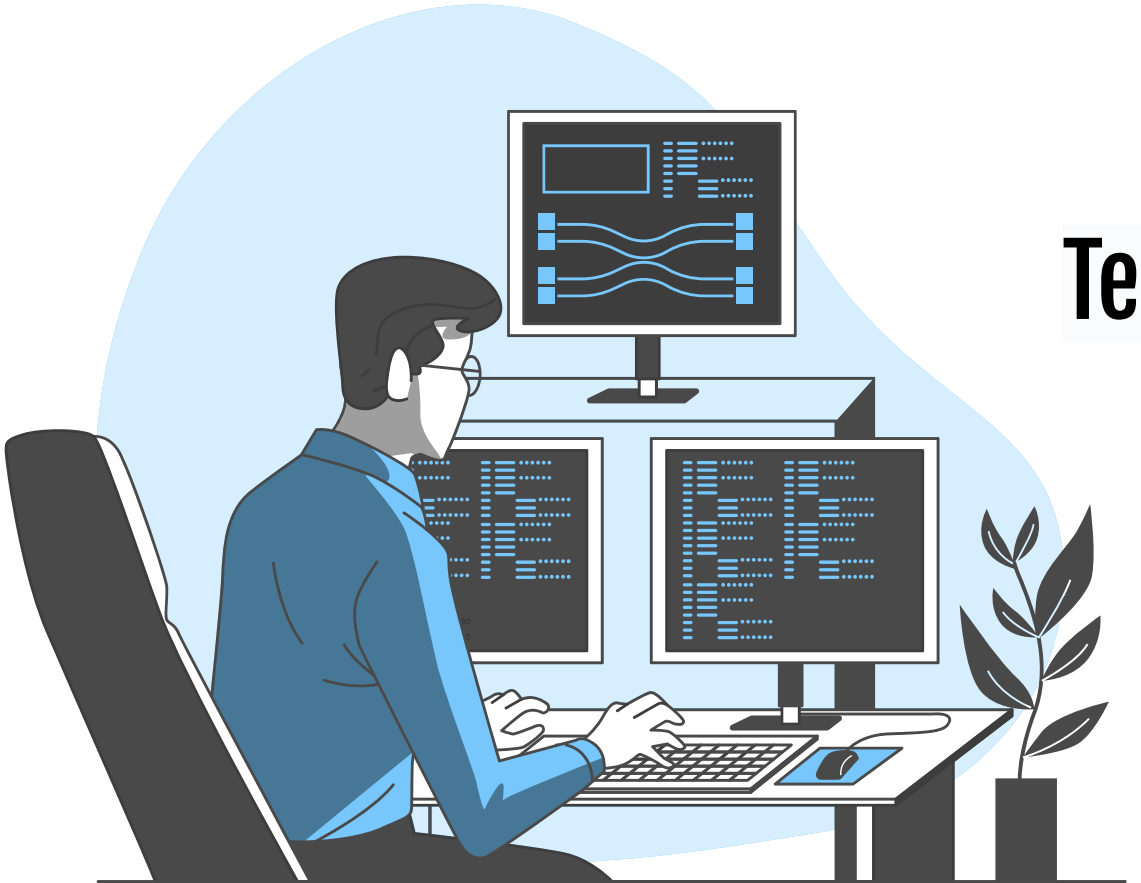
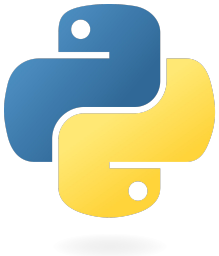
IPAR
ESTUDIOS



Tensorflow/keras



¿Qué es Tensorflow?

🔷 TensorFlow

- Framework de código abierto desarrollado por Google para:
  - Computación numérica
  - Modelado de redes neuronales
  - Entrenamiento y despliegue de modelos

¿Qué es Keras?

🔶 **Keras**

- API de alto nivel incluida en TensorFlow (tf.keras)
- Permite definir modelos de forma **rápida, simple y legible**
- Ideal para prototipado rápido y enseñanza

Pasos en la implementación de una RNA con keras

- 1. Define tu conjunto de datos de entrenamiento: vectores de entrada y de salida
- 2. Define la arquitectura de la Red Neuronal Artificial
- 3. Configura el proceso de aprendizaje mediante la seleccion de una funcion de error, una funcion de optimizacion y diferentes metricas para monitorizar el proceso
- 4. Entrena la RNA con tu conjunto de datos de entrenamiento mediante el uso del metodo **_fit()_**

Importaciones

Cómo se debe importar las librerías:

```
import tensorflow as tf
from tensorflow import keras
```

Keras también tiene un módulo de datasets, entre ellos mnist:

```
from keras import datasets
mnist = datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Como definir una RNA

```
from keras import models
from keras import layers

network = models.Sequential()

network.add(layers.Dense(16, activation='relu', input_shape=(4,)))
network.add(layers.Dense(8, activation='relu'))
network.add(layers.Dense(1, activation='sigmoid'))
```

Cómo definir una RNA

Para que la Red Neuronal Artificial funcione adecuadamente va a requerir tres componentes adicionales que se seleccionan durante el proceso de compilación:

1. **La función de error:** Se utiliza en el proceso de optimización de los parámetros del modelo para medir el error producido al modificar el valor de los parámetros del modelo en una dirección determinada
2. **La función de optimización:** Se corresponde con la función encargada de actualizar el valor de los parámetros del modelo en una dirección determinada en función del resultado de la función de error
3. **Métricas para monitorizar el proceso de entrenamiento:** Es interesante utilizar un conjunto de métricas durante el proceso de entrenamiento de la Red Neuronal Artificial de manera que podamos saber en cada iteración si el valor de los parámetros del modelo es adecuado o se están produciendo problemas como *overfitting*

Compilar y entrenar el modelo

Antes de entrenar, el modelo debe **compilarse**, indicando:

- El **optimizador**
- La **función de pérdida**
- Las **métricas a monitorizar**

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Entrenamiento:

```
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

epochs: veces que pasa por todo el dataset

batch_size: número de muestras por actualización

validation_split: reserva para validación

Evaluar y predecir

Evaluar el rendimiento:

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

Hacer predicciones:

```
predictions = model.predict(X_test)
```

Para regresión: activation='linear', loss='mse'

Para clasificación multiclase: activation='softmax', loss='categorical_crossentropy'

Evaluar y predecir

🧩 Elementos clave:

- **Capa (Dense)**: conecta entradas y salidas
- **Activación**: transforma la salida (ReLU, Sigmoid...)
- **Loss (pérdida)**: mide el error del modelo
- **Optimizer**: ajusta los pesos (SGD, Adam...)

🧠 Tipos de tareas:

- Clasificación binaria → sigmoid, binary_crossentropy
- Clasificación multiclase → softmax, categorical_crossentropy
- Regresión → linear, mse

Práctica

Añade en la práctica de Mnist el perceptrón, pero utilizando el framework de tensorflow.
Después prueba con una Red Neuronal Artificial formada por 2 capas.
Y por último define una Red Neuronal Artificial formada por 3 capas:
La primera capa estará formada por 300 neuronas.
La segunda capa estará formada por 100 neuronas.
La última capa estará formada por 10 neuronas.