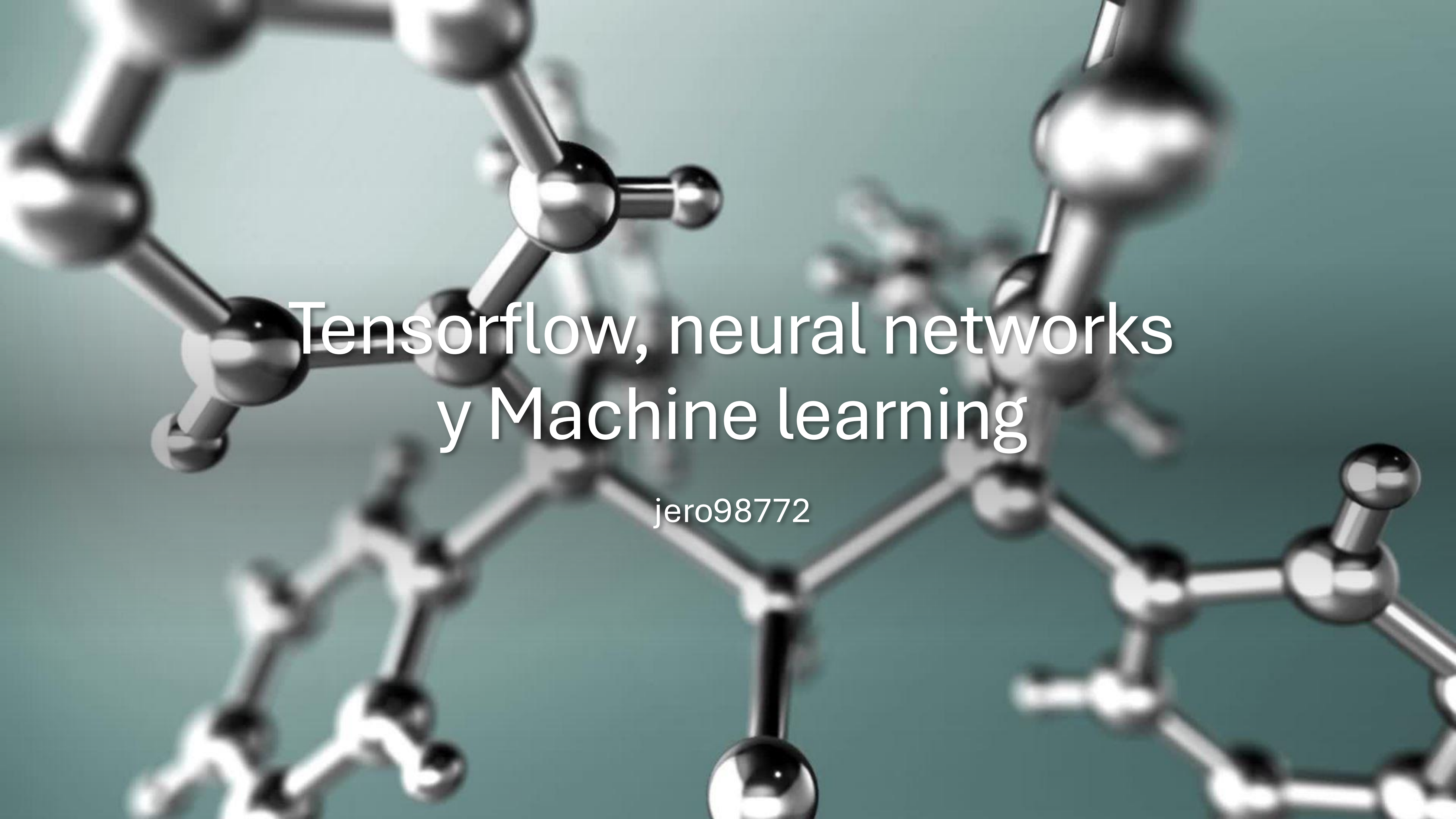
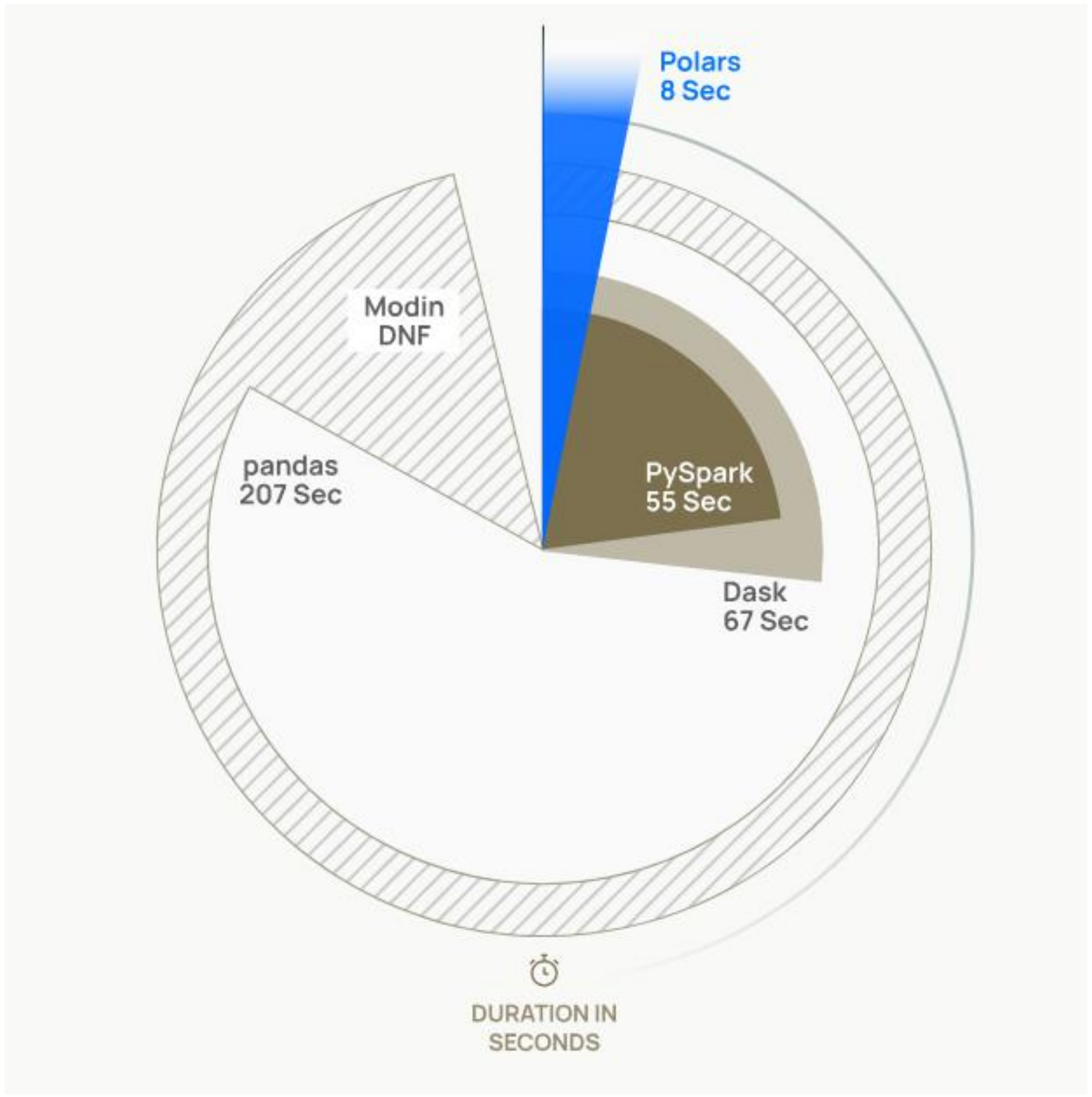


Tensorflow, neural networks y Machine learning

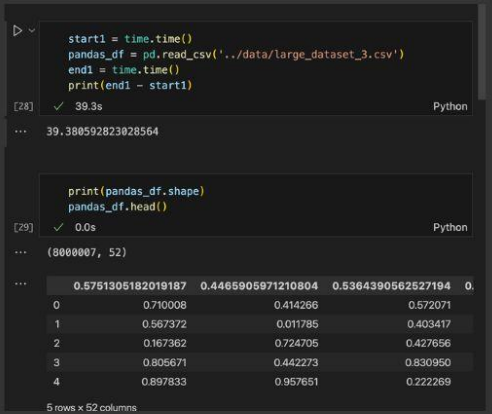
jero98772

Who use pandas?

https://pola.rs/



PANDAS



```
start1 = time.time()
pandas_df = pd.read_csv('../data/large_dataset_3.csv')
end1 = time.time()
print(end1 - start1)
```

[28] ✓ 39.3s Python

```
39.380592823028564
```

```
print(pandas_df.shape)
pandas_df.head()
```

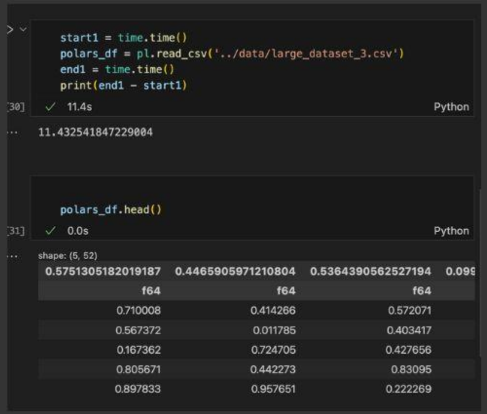
[29] ✓ 0.0s Python

```
(8000007, 52)
```

	0.5751305182019187	0.4465905971210804	0.5364390562527194	0.
0	0.710008	0.414266	0.572071	
1	0.567372	0.011785	0.403417	
2	0.167362	0.724705	0.427656	
3	0.805671	0.442273	0.830950	
4	0.897833	0.957651	0.222269	

5 rows × 52 columns

POLARS



```
start1 = time.time()
polars_df = pl.read_csv('../data/large_dataset_3.csv')
end1 = time.time()
print(end1 - start1)
```

[30] ✓ 11.4s Python

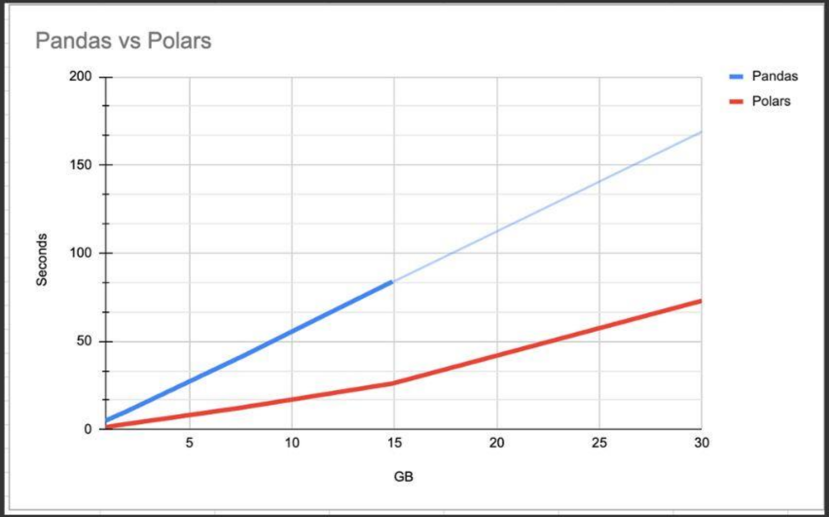
```
11.432541847229004
```

```
polars_df.head()
```

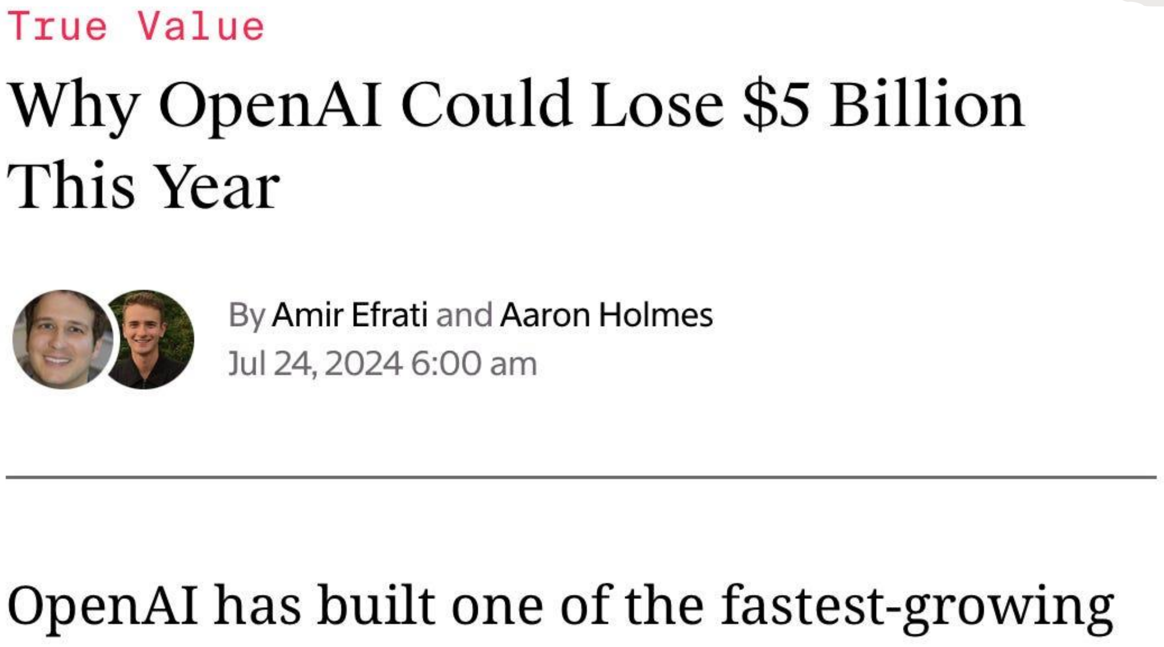
[31] ✓ 0.0s Python

shape: (5, 52)

0.5751305182019187	0.4465905971210804	0.5364390562527194	0.099
f64	f64	f64	
0.710008	0.414266	0.572071	
0.567372	0.011785	0.403417	
0.167362	0.724705	0.427656	
0.805671	0.442273	0.83095	
0.897833	0.957651	0.222269	



News



True Value

Why OpenAI Could Lose $5 Billion This Year

By Amir Efrati and Aaron Holmes
Jul 24, 2024 6:00 am

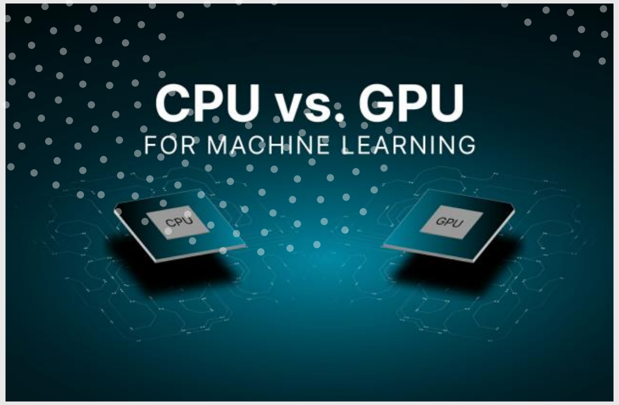
OpenAI has built one of the fastest-growing

Segment Anything - A Foundation Model for Image Segmentation

What is tensorflow

- TensorFlow makes it easy for beginners and experts to create machine learning is a library made by google and wide used in the word

Where can i use tensorflow



TensorFlow.js

TensorFlow Lite

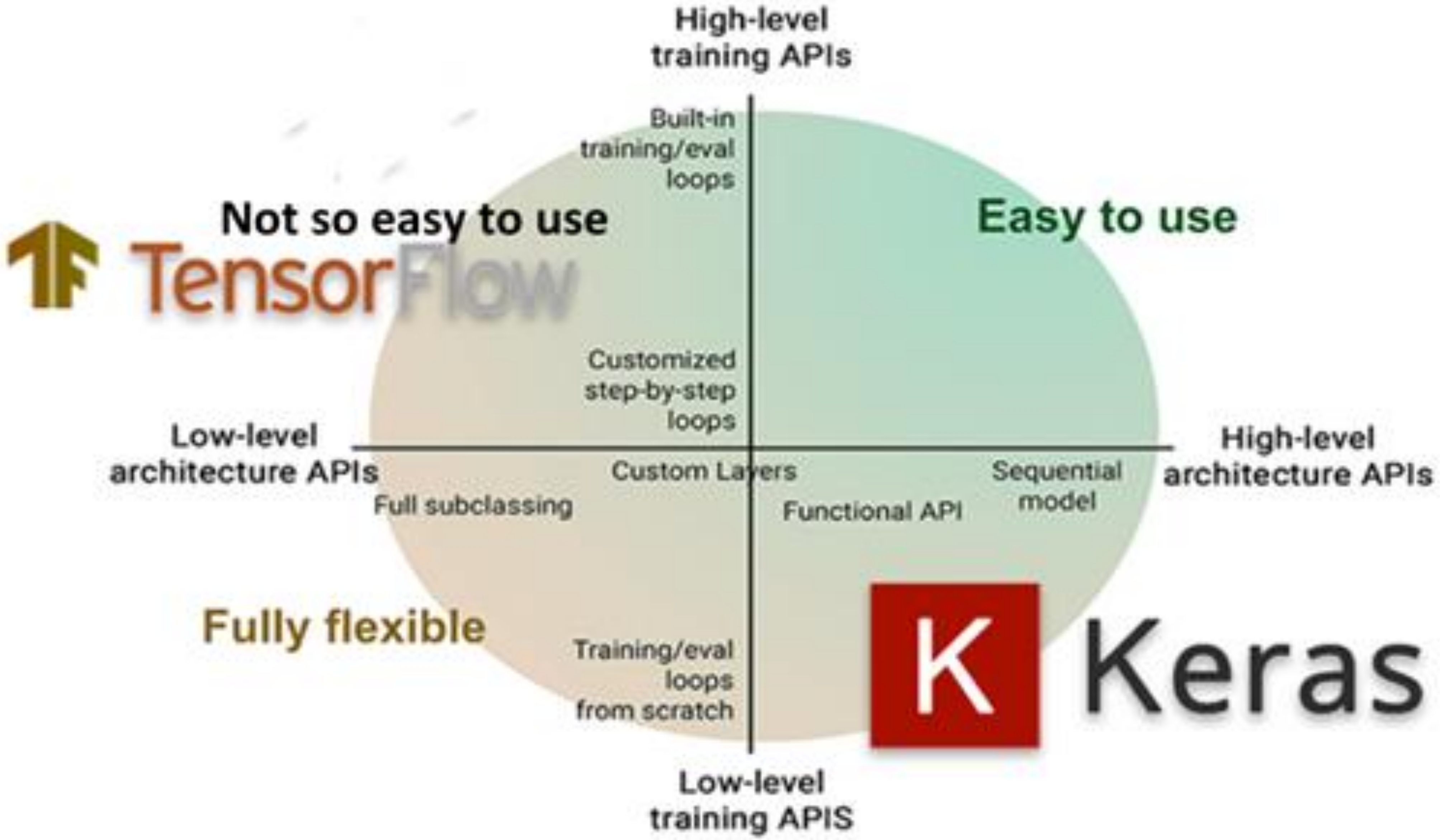
Alternatives

- **For Research**: PyTorch is often preferred due to its dynamic nature and ease of debugging.
- **For Production**: TensorFlow may be more suitable due to its mature ecosystem and robust deployment tools.
- **For Ease of Learning**: PyTorch's more straightforward API might make it easier for beginners to pick up.

Why keras? What is keras

- Keras is a high-level API for building and training deep learning models. It was originally a standalone library but is now integrated into TensorFlow as tf.keras.

- **TensorFlow**: Suitable for complex and custom machine learning workflows where performance and flexibility are crucial.
- **Keras**: Ideal for quickly building and experimenting with neural networks with less concern for low-level details.


High-level training APIs
Built-in training/eval loops
Not so easy to use
Easy to use
TensorFlow
Customized step-by-step loops
Low-level architecture APIs
High-level architecture APIs
Custom Layers
Full subclassing
Functional API
Sequential model
Fully flexible
Training/eval loops from scratch
Keras
Low-level training APIS

Install tensorflow

```
pip install tensorflow
```

```
pip install tf-nightly
```

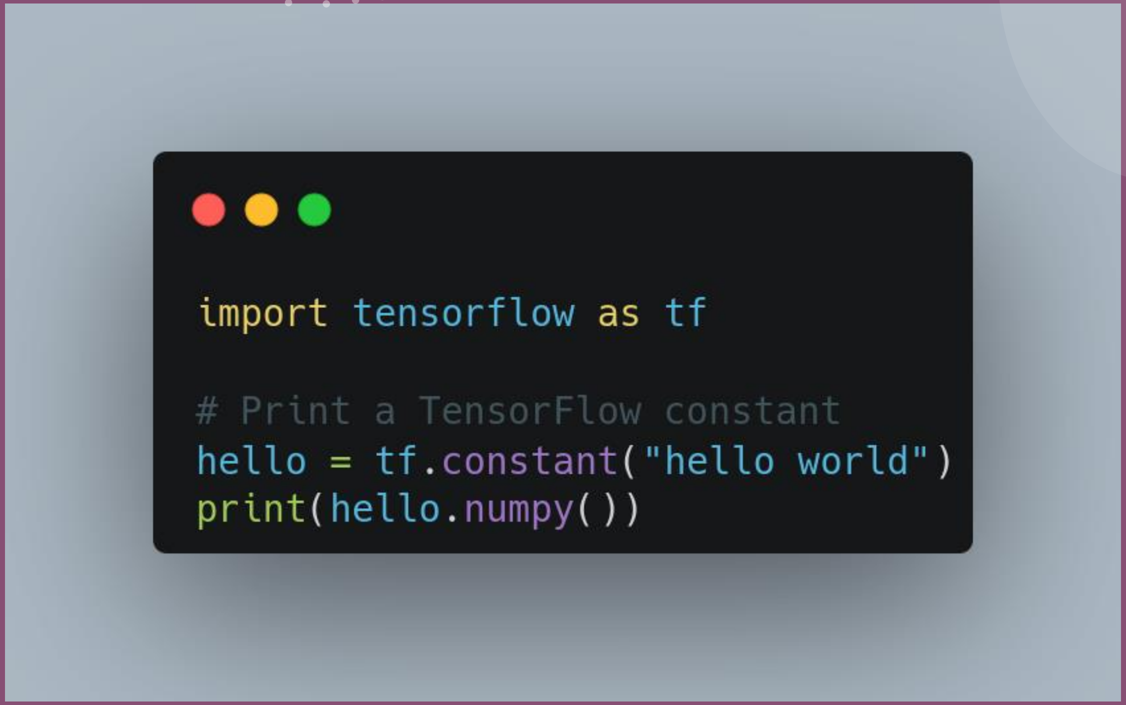
```
pip install tensorflow-gpu
pip install keras
```

```
npm install @tensorflow/tfjs
```

```
npm install @tensorflow/tfjs-node-gpu
```

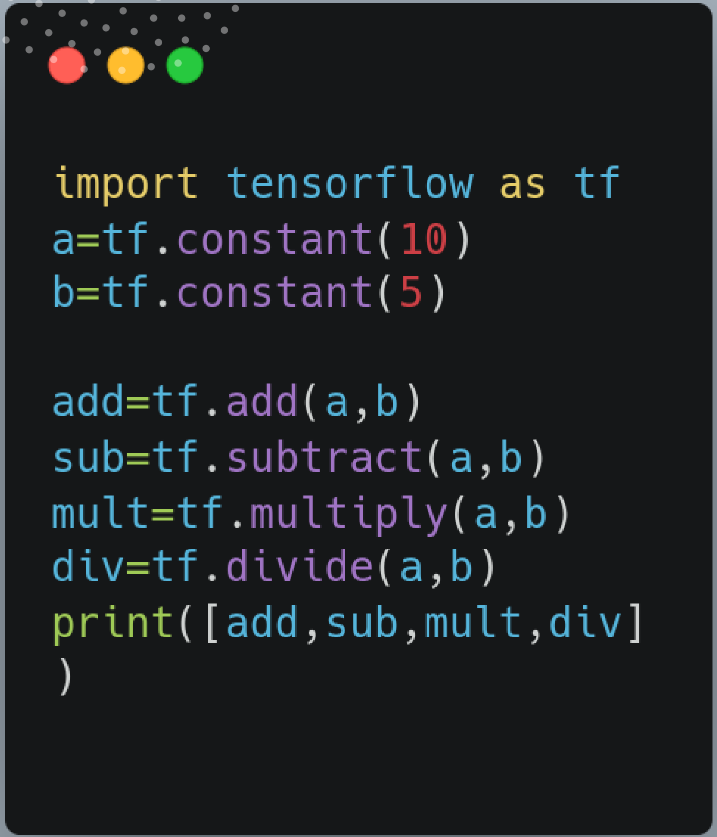
(there is no installation for tf-lite)

How to do hello word in tensorflow?



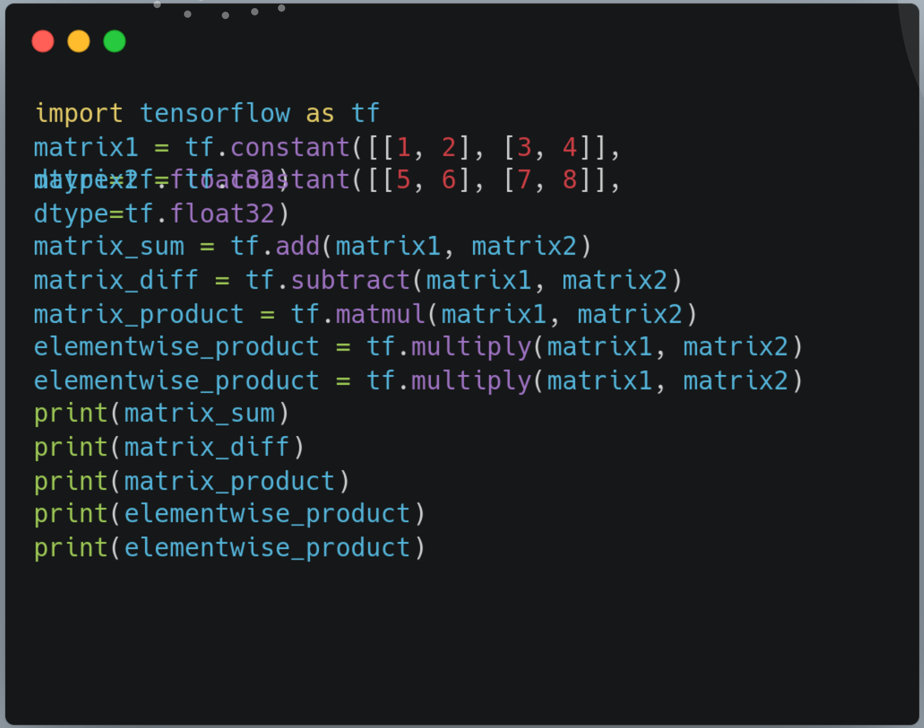
```
import tensorflow as tf

# Print a TensorFlow constant
hello = tf.constant("hello world")
print(hello.numpy())
```

Arimetic



```
import tensorflow as tf
a=tf.constant(10)
b=tf.constant(5)

add=tf.add(a,b)
sub=tf.subtract(a,b)
mult=tf.multiply(a,b)
div=tf.divide(a,b)
print([add,sub,mult,div]
)
```

Arimetic in matrixs



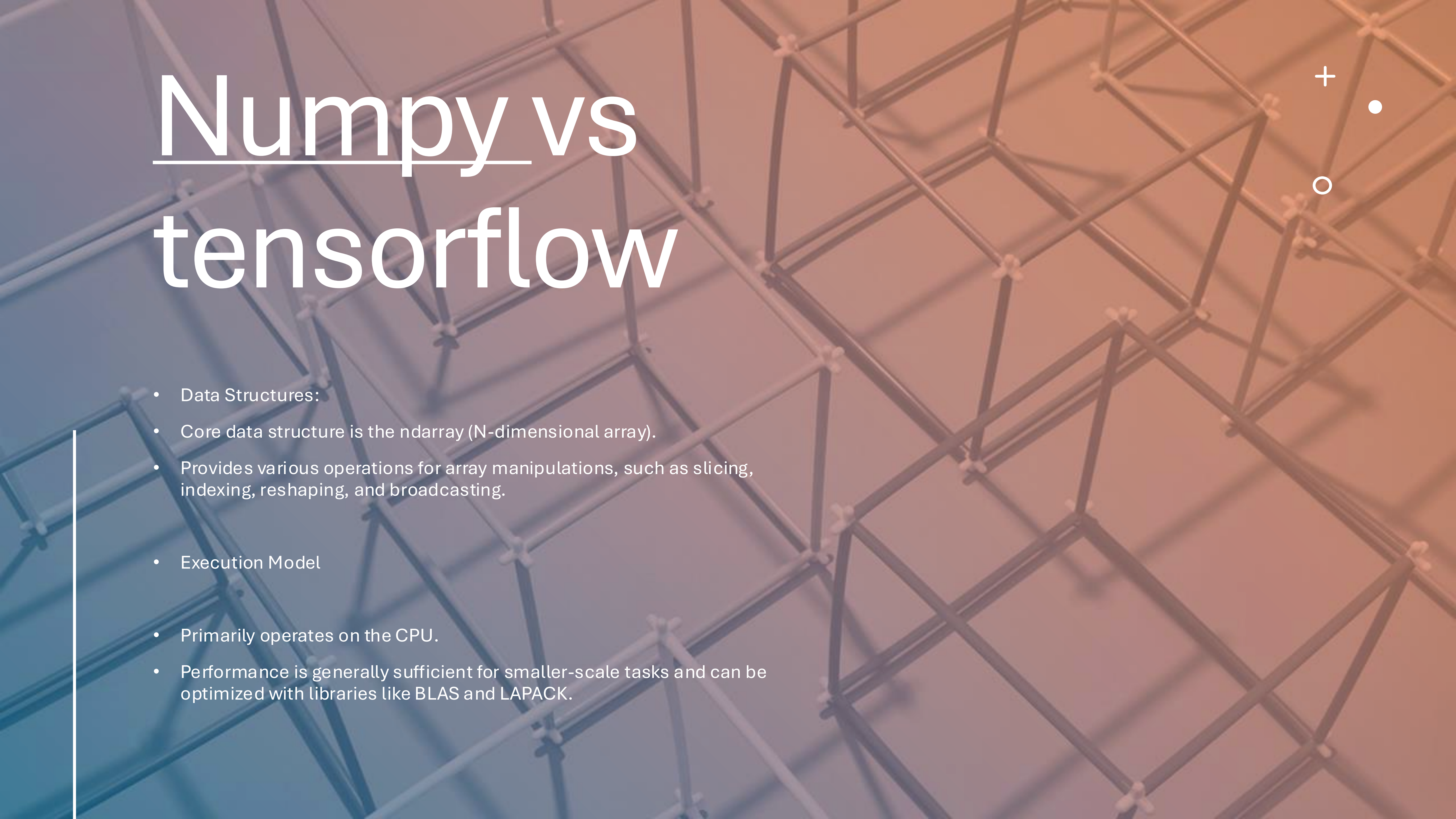
```
import tensorflow as tf
matrix1 = tf.constant([[1, 2], [3, 4]],
dtype=tf.float32)
matrix2 = tf.constant([[5, 6], [7, 8]],
dtype=tf.float32)
matrix_sum = tf.add(matrix1, matrix2)
matrix_diff = tf.subtract(matrix1, matrix2)
matrix_product = tf.matmul(matrix1, matrix2)
elementwise_product = tf.multiply(matrix1, matrix2)
elementwise_product = tf.multiply(matrix1, matrix2)
print(matrix_sum)
print(matrix_diff)
print(matrix_product)
print(elementwise_product)
print(elementwise_product)
```

Numpy and tensorflow are the same?

TensorFlow + Keras

Numpy vs tensorflow

- **Purpose and Use Case:**
- General-purpose library for numerical and matrix computations.
- Ideal for scientific computing, data analysis, and simple machine learning tasks.
- Provides a wide range of mathematical functions to operate on arrays and matrices.
- Typically used in environments where performance is not as critical, and the computations can be performed on a CPU.

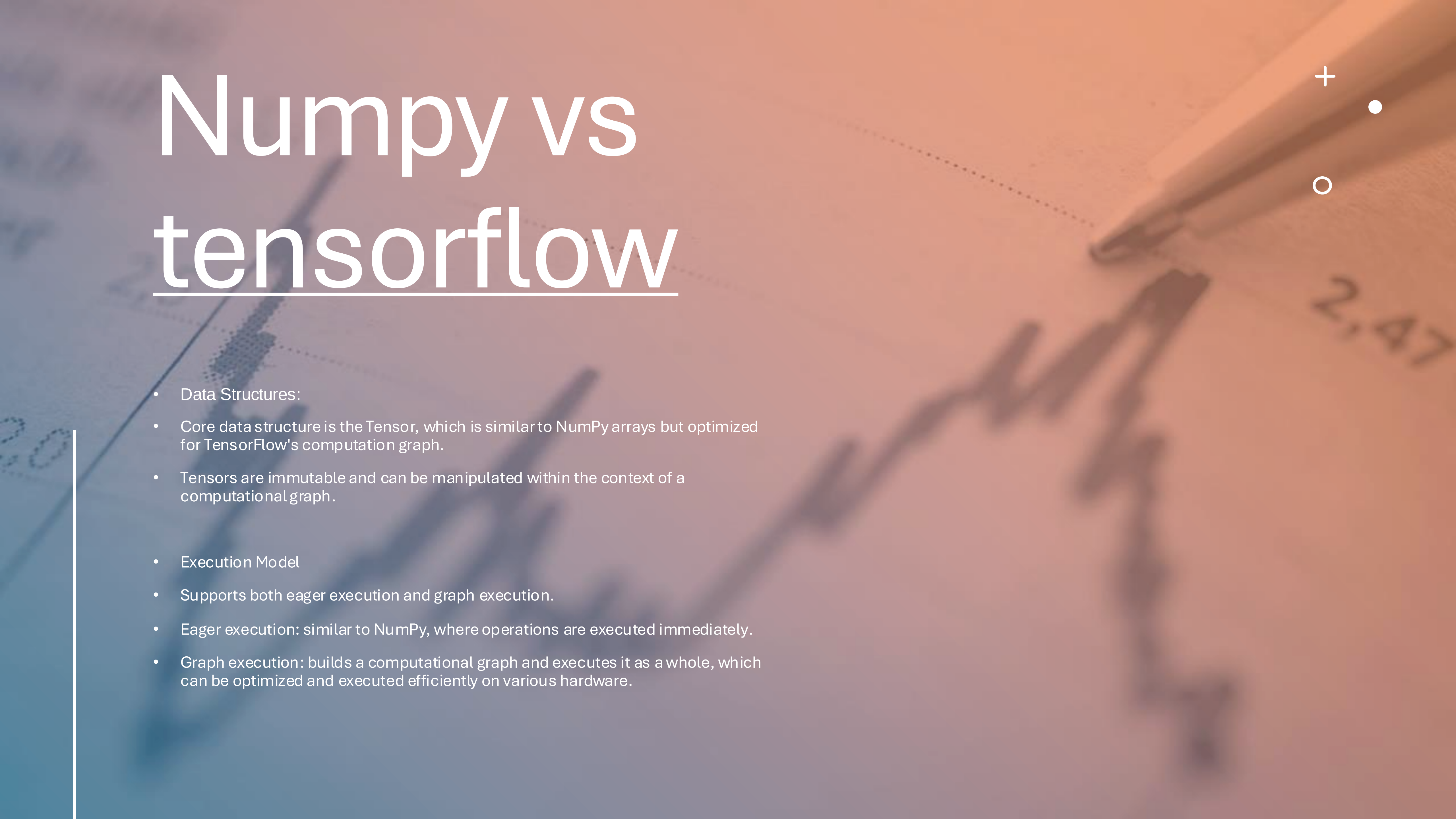


Numpy vs tensorflow

- Data Structures:
- Core data structure is the ndarray (N-dimensional array).
- Provides various operations for array manipulations, such as slicing, indexing, reshaping, and broadcasting.
- Execution Model
- Primarily operates on the CPU.
- Performance is generally sufficient for smaller-scale tasks and can be optimized with libraries like BLAS and LAPACK.

Numpy vs tensorflow

- **Purpose and Use Case:**
- Designed for large-scale machine learning and deep learning applications.
- Optimized for performance, supporting hardware acceleration (GPUs and TPUs).
- Provides high-level APIs for building and training machine learning models.
- Supports distributed computing, making it suitable for large datasets and complex models.



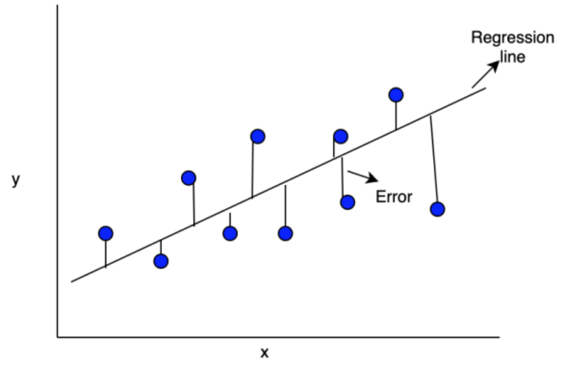
Numpy vs tensorflow

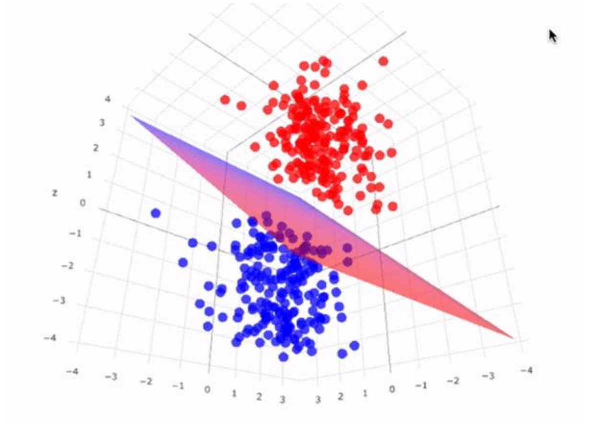
- Data Structures:
- Core data structure is the Tensor, which is similar to NumPy arrays but optimized for TensorFlow's computation graph.
- Tensors are immutable and can be manipulated within the context of a computational graph.

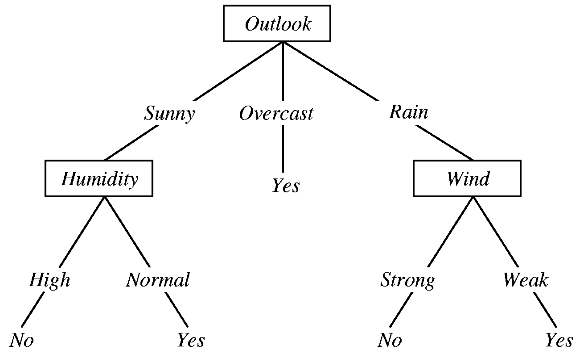
- Execution Model
- Supports both eager execution and graph execution.
- Eager execution: similar to NumPy, where operations are executed immediately.
- Graph execution: builds a computational graph and executes it as a whole, which can be optimized and executed efficiently on various hardware.

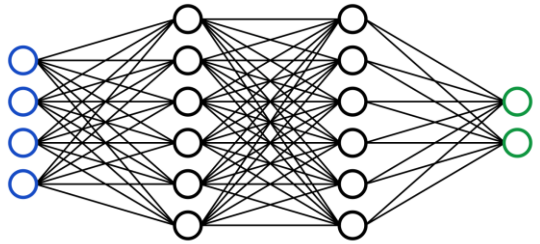
Create our models with tensorflow, but what is a model?

- In the context of machine learning, statistics, and data science, a model is a mathematical representation or abstraction used to make predictions or decisions based on data. Models learn patterns and relationships within the data and use this knowledge to predict outcomes or classify new data points. Here are a few key points about models:

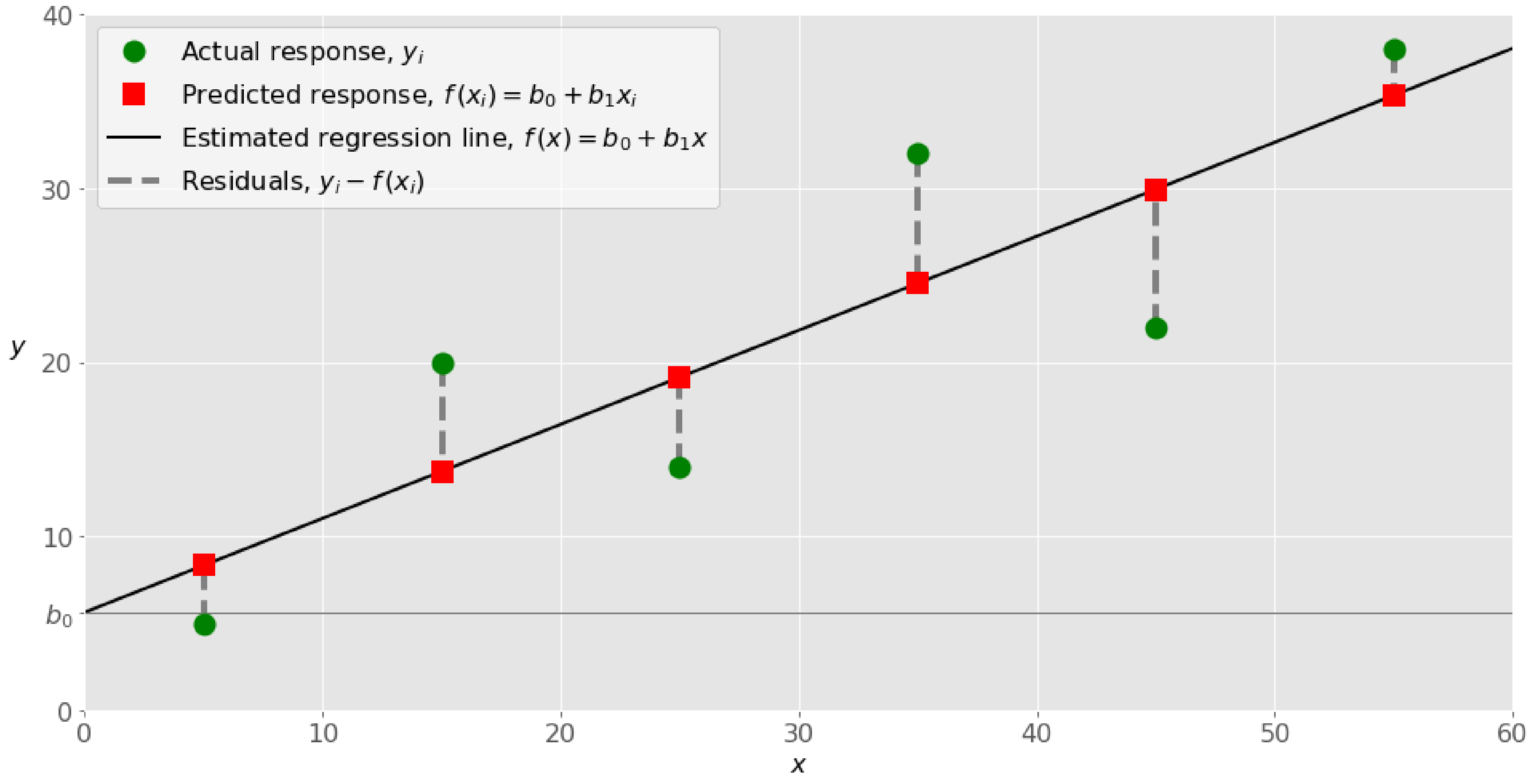




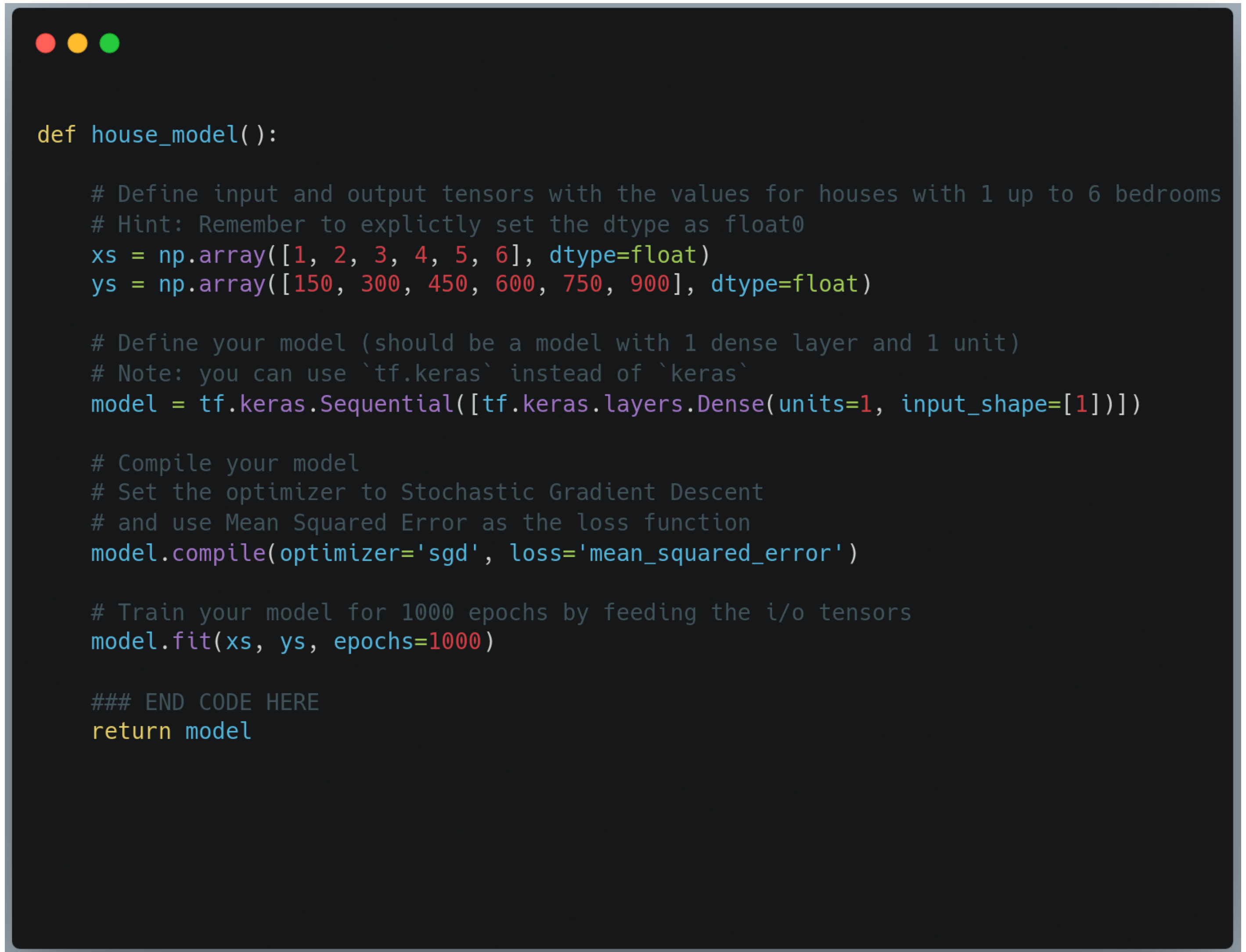




Ok, can i program a Linear regresion?

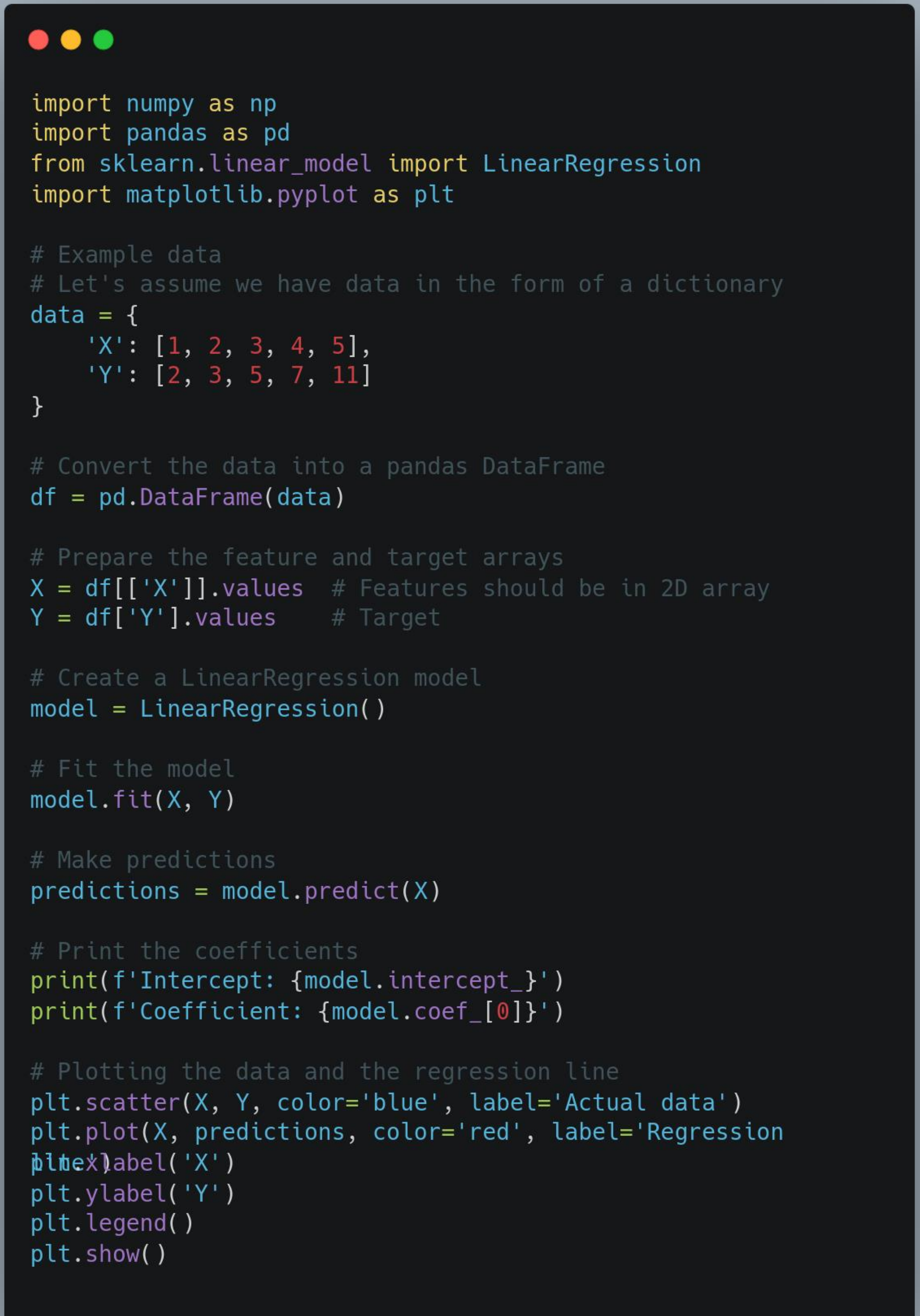


Linear regression in tf



```python
def house_model():

    # Define input and output tensors with the values for houses with 1 up to 6 bedrooms
    # Hint: Remember to explictly set the dtype as float0
    xs = np.array([1, 2, 3, 4, 5, 6], dtype=float)
    ys = np.array([150, 300, 450, 600, 750, 900], dtype=float)

    # Define your model (should be a model with 1 dense layer and 1 unit)
    # Note: you can use `tf.keras` instead of `keras`
    model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input_shape=[1])])

    # Compile your model
    # Set the optimizer to Stochastic Gradient Descent
    # and use Mean Squared Error as the loss function
    model.compile(optimizer='sgd', loss='mean_squared_error')

    # Train your model for 1000 epochs by feeding the i/o tensors
    model.fit(xs, ys, epochs=1000)

    ### END CODE HERE
    return model
```

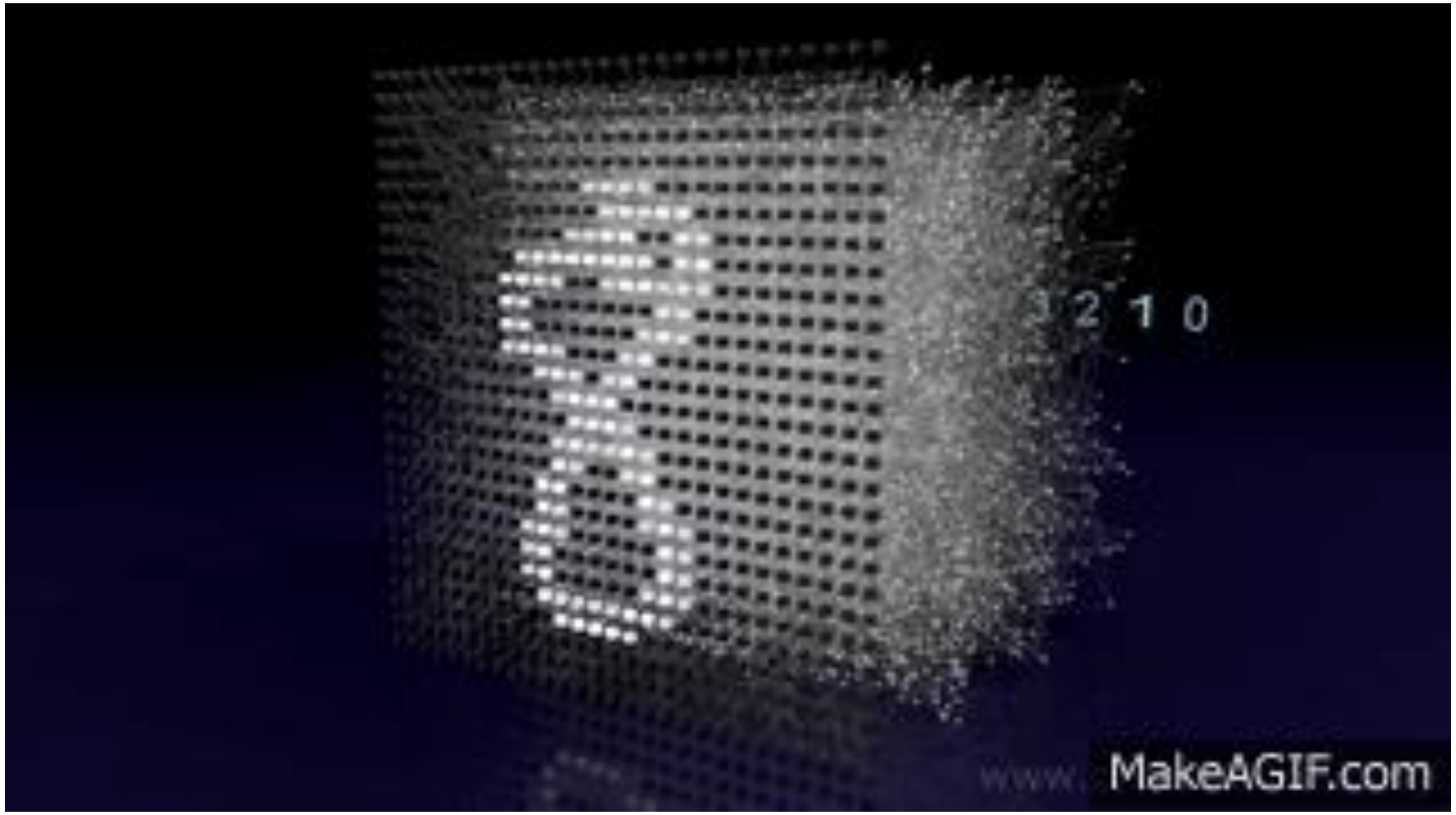
Not with tf



```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Example data
# Let's assume we have data in the form of a dictionary
data = {
    'X': [1, 2, 3, 4, 5],
    'Y': [2, 3, 5, 7, 11]
}

# Convert the data into a pandas DataFrame
df = pd.DataFrame(data)

# Prepare the feature and target arrays
X = df[['X']].values  # Features should be in 2D array
Y = df['Y'].values    # Target

# Create a LinearRegression model
model = LinearRegression()

# Fit the model
model.fit(X, Y)

# Make predictions
predictions = model.predict(X)

# Print the coefficients
print(f'Intercept: {model.intercept_}')
print(f'Coefficient: {model.coef_[0]}')

# Plotting the data and the regression line
plt.scatter(X, Y, color='blue', label='Actual data')
plt.plot(X, predictions, color='red', label='Regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Why we need tensorflow if there is alternatives like sklearns?

Neural networks



Neural networks Types

Deep neural networks

- Convolutional neural networks

Generative adversarial networks

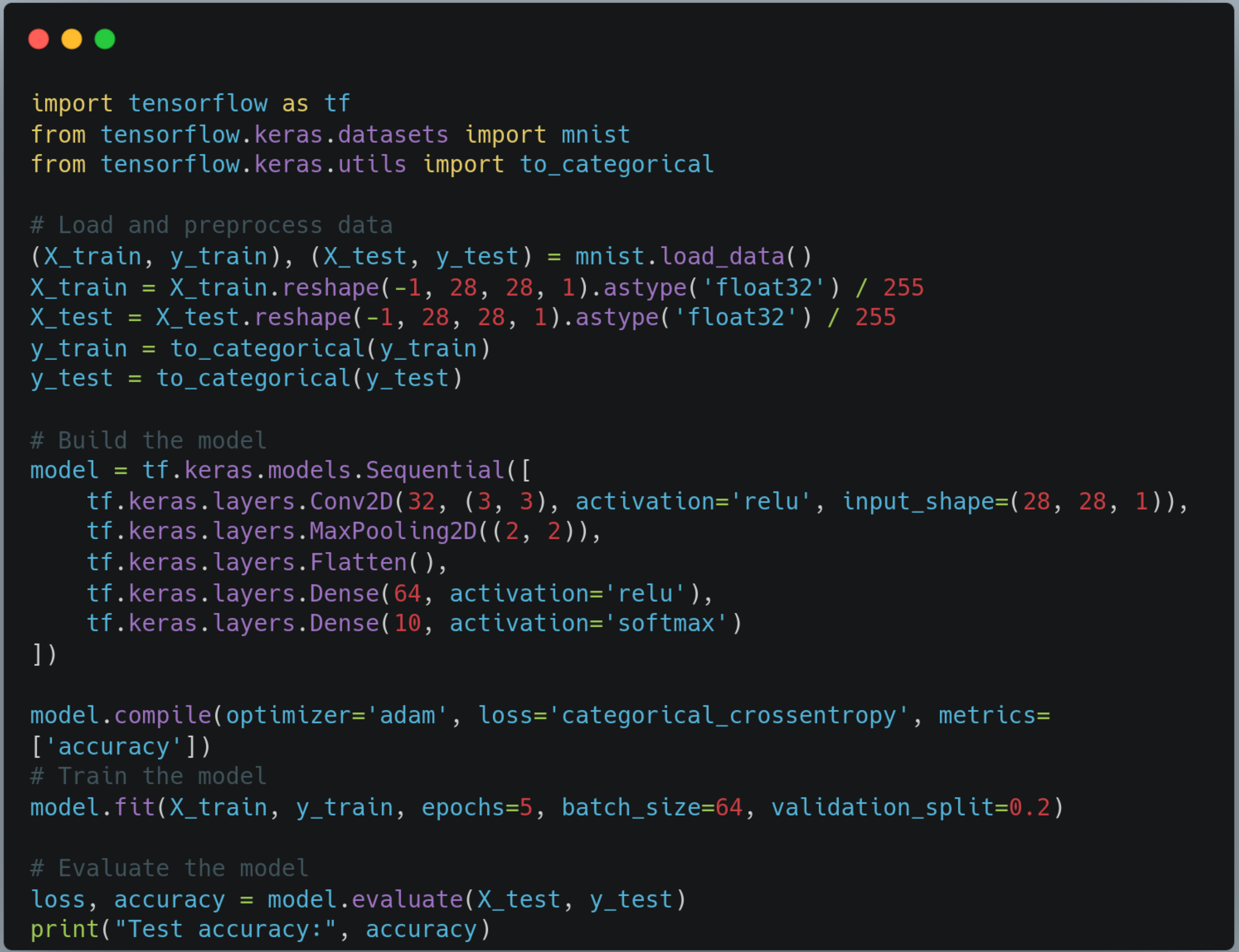
Recurrent neural networks

- Artificial neural networks

Long short-term memory networks

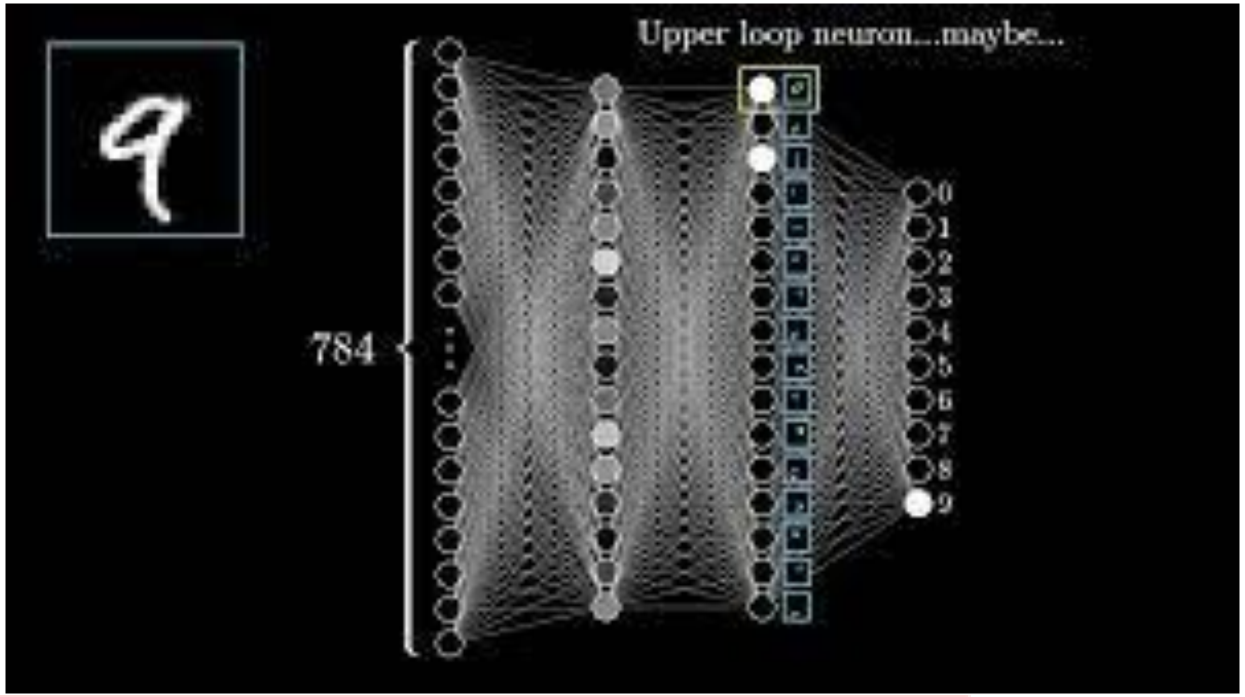
feedforward neural network

Code of the before example



```python
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load and preprocess data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])
# Train the model
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print("Test accuracy:", accuracy)
```

Sequential


Upper loop neuron...maybe...
784
0
1
2
3
4
5
6
7
8
9

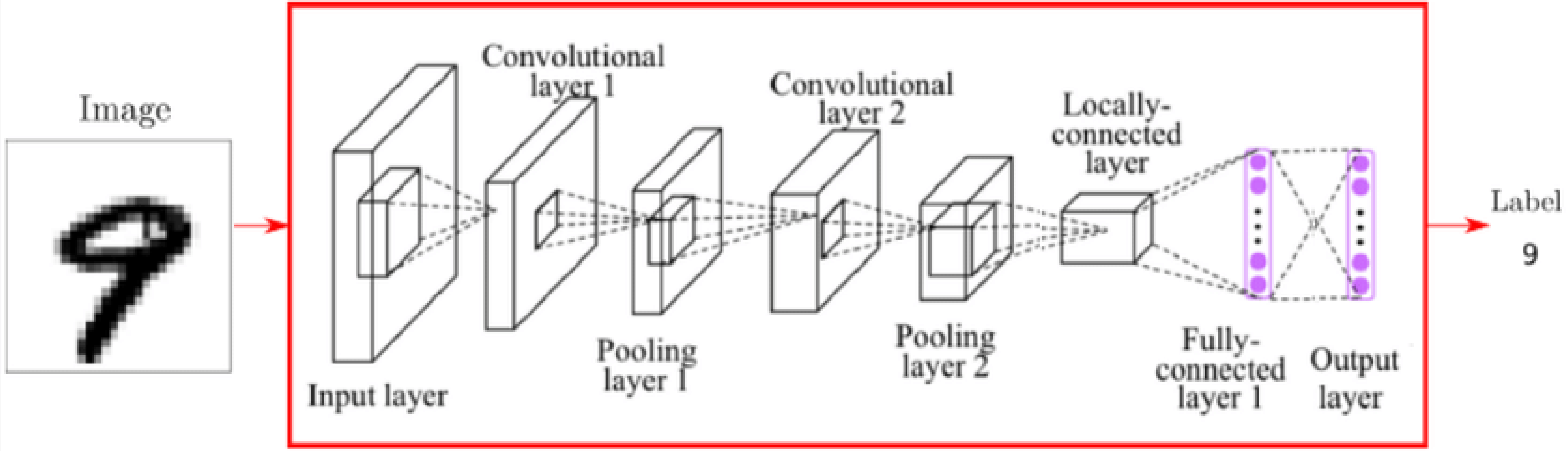


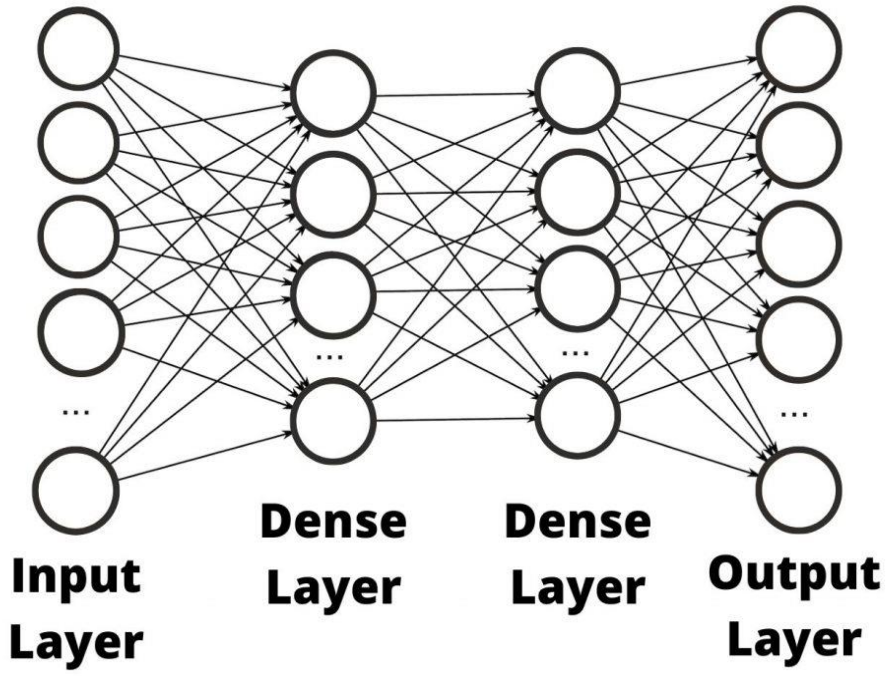
Image
Convolutional layer 1
Convolutional layer 2
Locally-connected layer
Input layer
Pooling layer 1
Pooling layer 2
Fully-connected layer 1
Output layer
Label
9

Layers

- **Dense (Fully Connected) Layer**: Each neuron in the layer is connected to every neuron in the previous layer.
- **Convolutional Layer (Conv2D/Conv3D)**: Used primarily in image processing; applies convolutional filters to the input.
- **Pooling Layer (MaxPooling/AveragePooling)**: Reduces the spatial dimensions of the input by taking the maximum or average value in a specified window.
- **Recurrent Layer (RNN, LSTM, GRU)**: Processes sequential data by maintaining a hidden state that is updated at each step in the sequence.
- **Batch Normalization Layer**: Normalizes the input of each mini-batch to improve training speed and stability.
- **Dropout Layer**: Randomly sets a fraction of the input units to zero during training to prevent overfitting.
- **Embedding Layer**: Converts categorical data into dense vectors of fixed size, commonly used in natural language processing.
- **Flatten Layer**: Flattens the input to a 1D vector, often used before a fully connected layer.
- **UpSampling Layer**: Increases the spatial dimensions of the input, commonly used in image generation tasks.
- **Locally Connected Layer**: Similar to a convolutional layer but without shared weights between the filters.
- **Separable Convolutional Layer**: Factorizes a convolution into two smaller operations, often used to reduce computation in deep convolutional networks.
- **Attention Layer**: Focuses on different parts of the input sequence, used in sequence-to-sequence models and transformers.

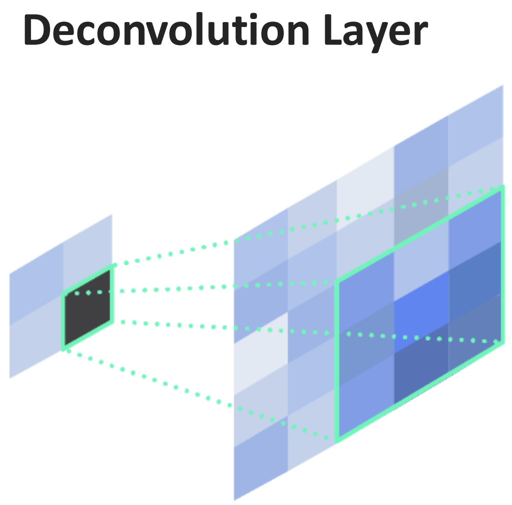
Dense

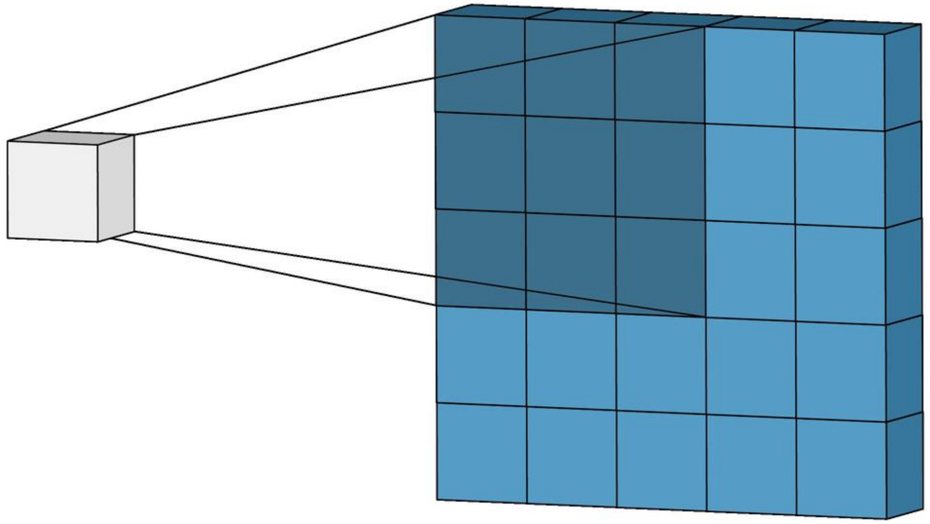
- **Dense (Fully Connected) Layer**: Each neuron in the layer is connected to every neuron in the previous layer.



Convolutional Layer (Conv2D/Conv3D)

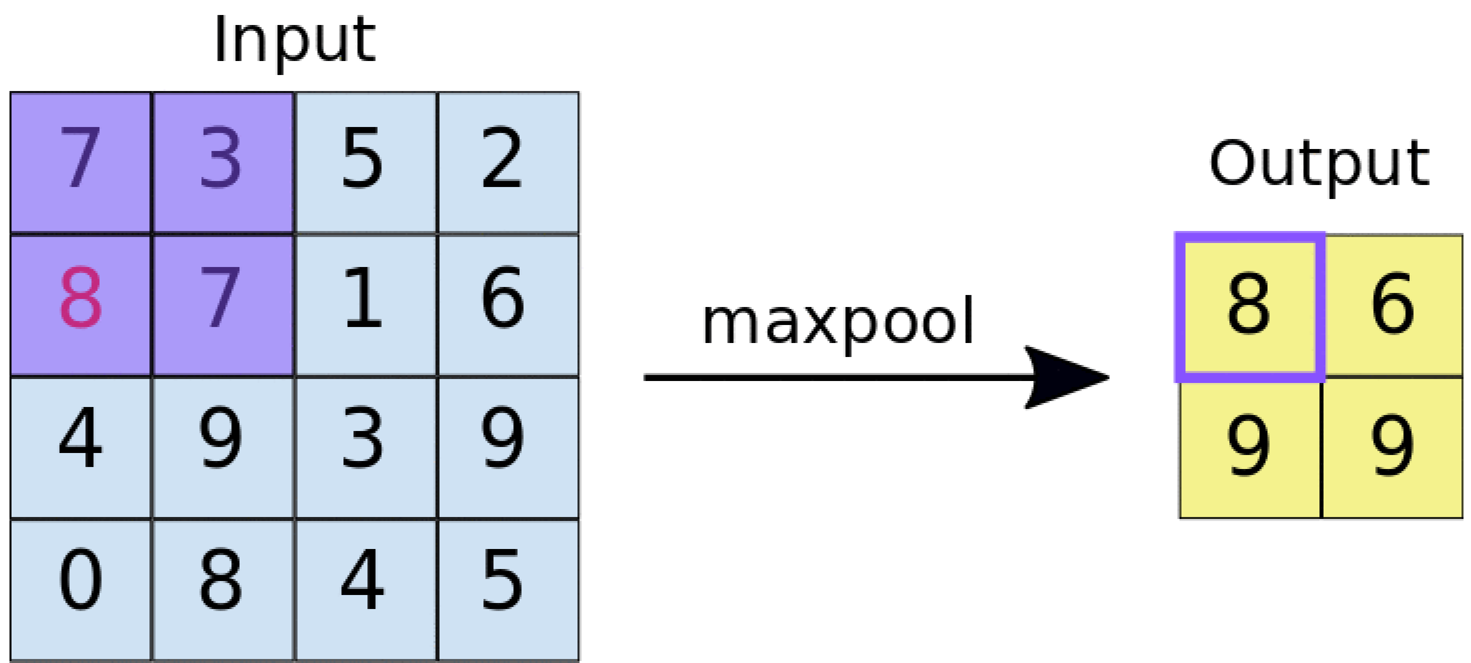
- **Convolutional Layer (Conv2D/Conv3D)**: Used primarily in image processing; applies convolutional filters to the input.
- Result dimension= (inputlayer.x-filter_size.x,inputlayer.y-filter_size.y)





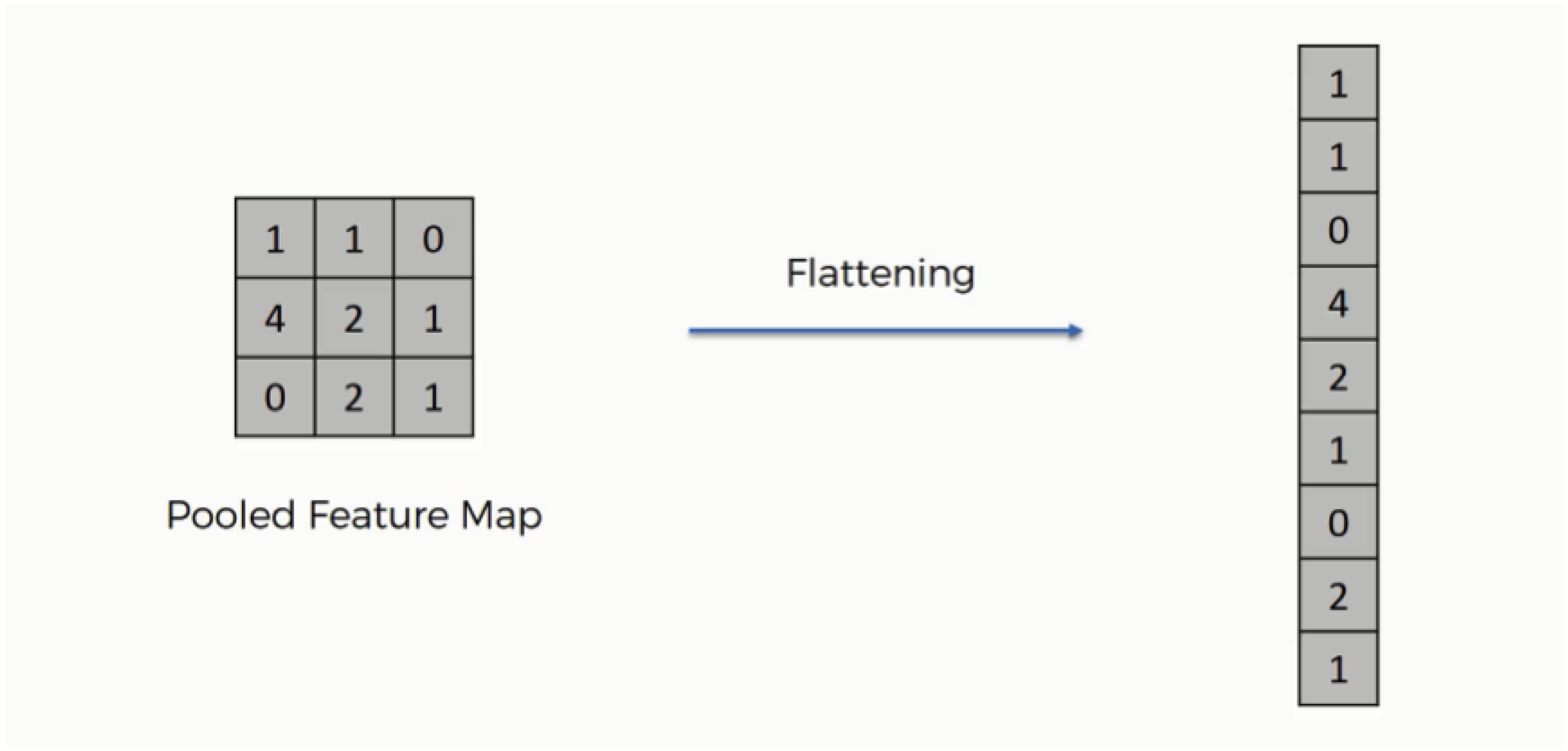
Pooling layer

- **Pooling Layer (MaxPooling/AveragePooling):** Reduces the spatial dimensions of the input by taking the maximum or average value in a specified window.
- Result dimension= (inputlayer.x/filter_size.x,inputlayer.y/filter_size.y)

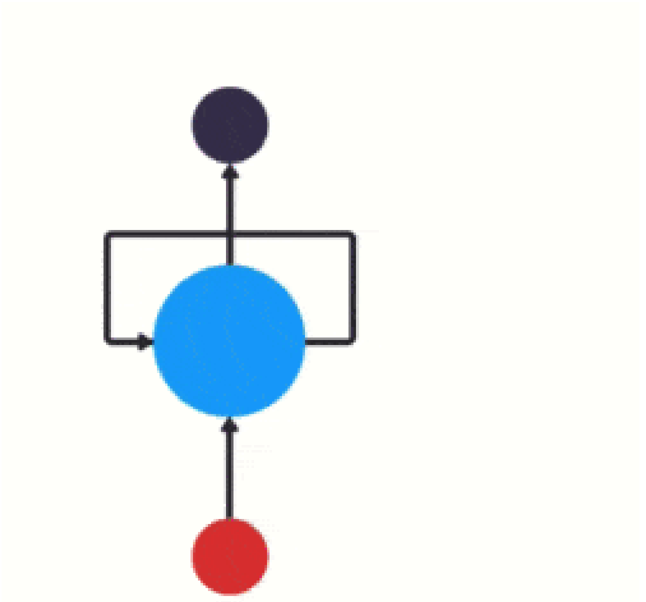


Flaten layer

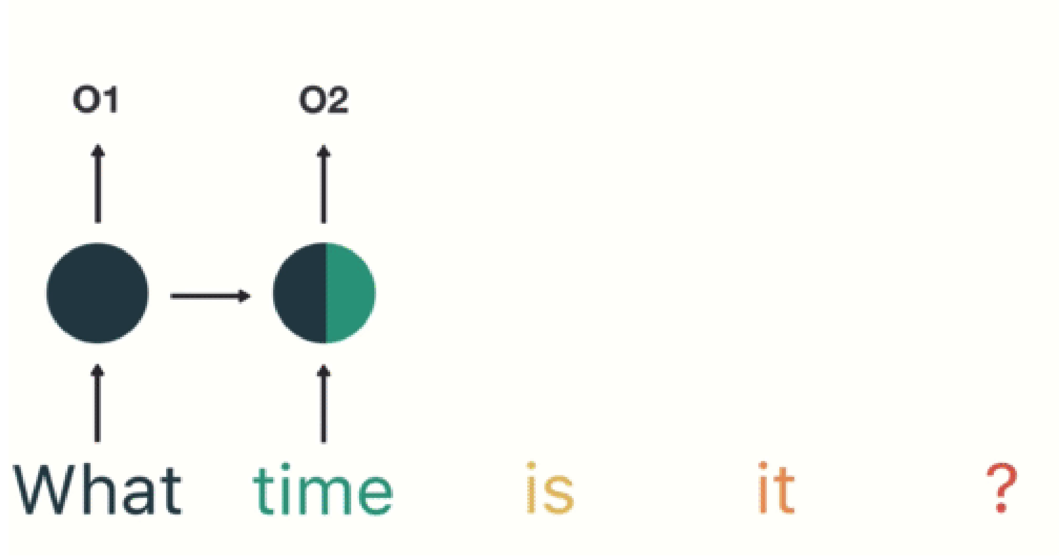
- **Flatten Layer**: Flattens the input to a 1D vector, often used before a fully connected layer.



Recurrent Layer



- **Recurrent Layer (RNN, LSTM, GRU)**: Processes sequential data by maintaining a hidden state that is updated at each step in the sequence.



Activations layers

- **ReLU (Rectified Linear Unit)**
- **Sigmoid**
- **Tanh (Hyperbolic Tangent)**
- **Leaky ReLU**
- **Parametric ReLU (PReLU)**Similar to Leaky ReLU but with a learnable parameter for the slope.
- **Exponential Linear Unit (ELU)**
- **Softmax** used for multi-class classification to output a probability distribution.
- **Swish** is the sigmoid function.
- **Softplus**
- **Hard Sigmoid**: An approximation of the sigmoid function with a linear piece-wise function for computational efficiency.
- **GELU (Gaussian Error Linear Unit)**: Combines properties of ReLU and sigmoid for smoother activation.

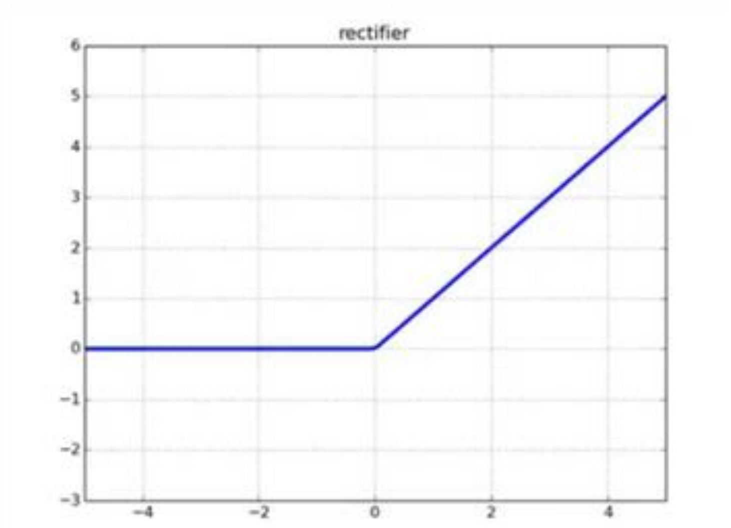
ReLU (Rectified Linear Unit)

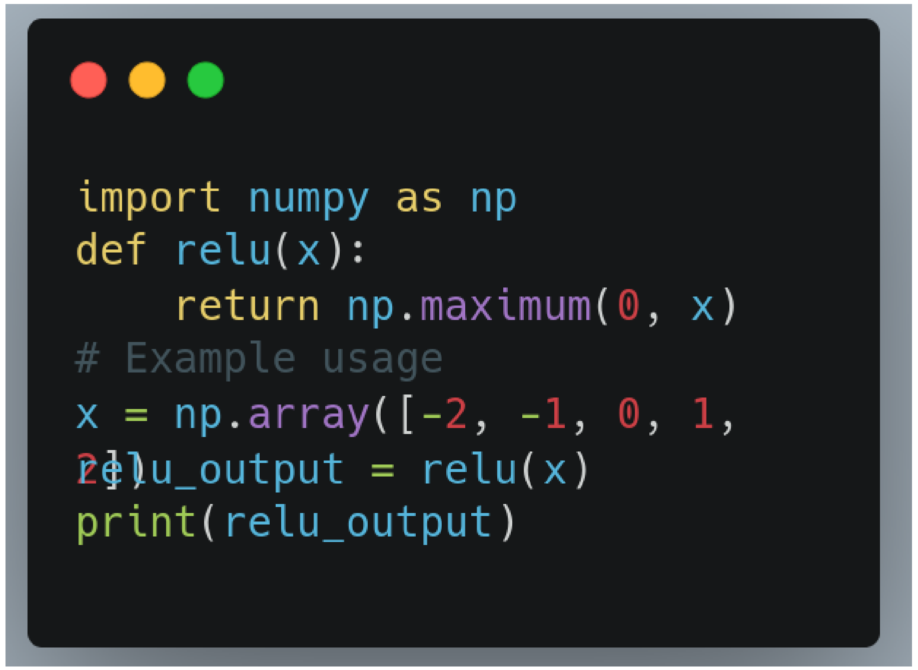
Advantages

•**Avoids Vanishing Gradient Problem**: Compared to sigmoid and tanh, ReLU helps mitigate the vanishing gradient problem, where gradients become very small, slowing down the learning process.

•**Sparsity**: Many neurons output zero, leading to a sparse network which can be computationally efficient and may help in generalization.

$$f(x)= \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x => 0 \end{cases}$$



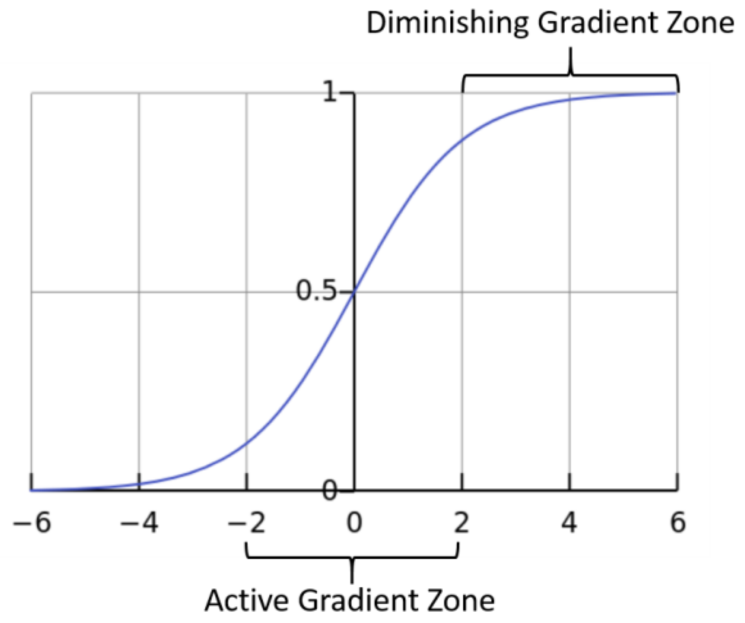


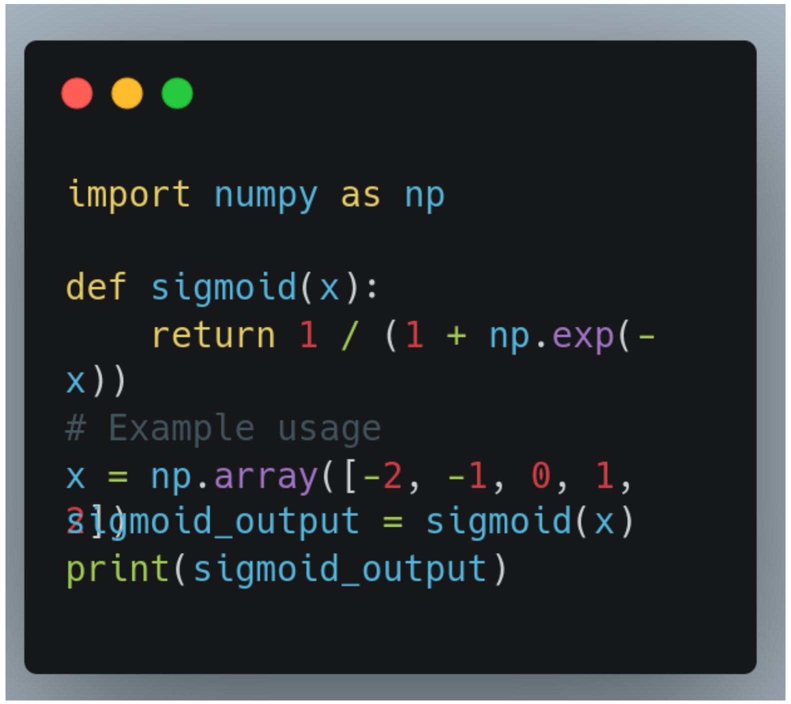
```
import numpy as np
def relu(x):
    return np.maximum(0, x)
# Example usage
x = np.array([-2, -1, 0, 1,
2])
relu_output = relu(x)
print(relu_output)
```

Sigmoid

- **Output Range**: between 0 and 1.
- **Usage**: used in binary classification problems, and occasionally in hidden layers of neural networks.
- **Interpretation**: The output can be interpreted as a probability, making it useful for binary classification.

$$A = \frac{1}{1+e^{-x}}$$





```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-
x))
# Example usage
x = np.array([-2, -1, 0, 1,
2])
sigmoid_output = sigmoid(x)
print(sigmoid_output)
```

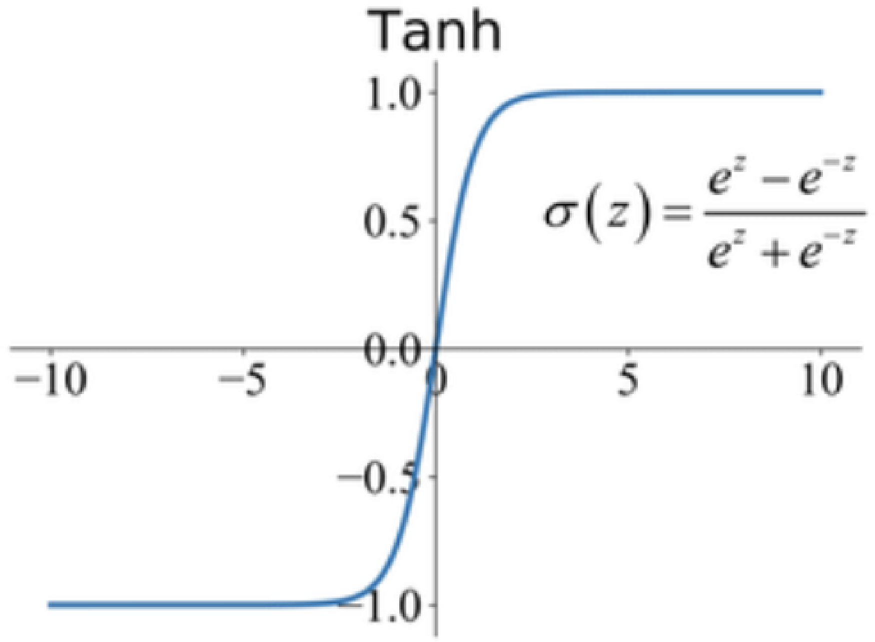
Tanh

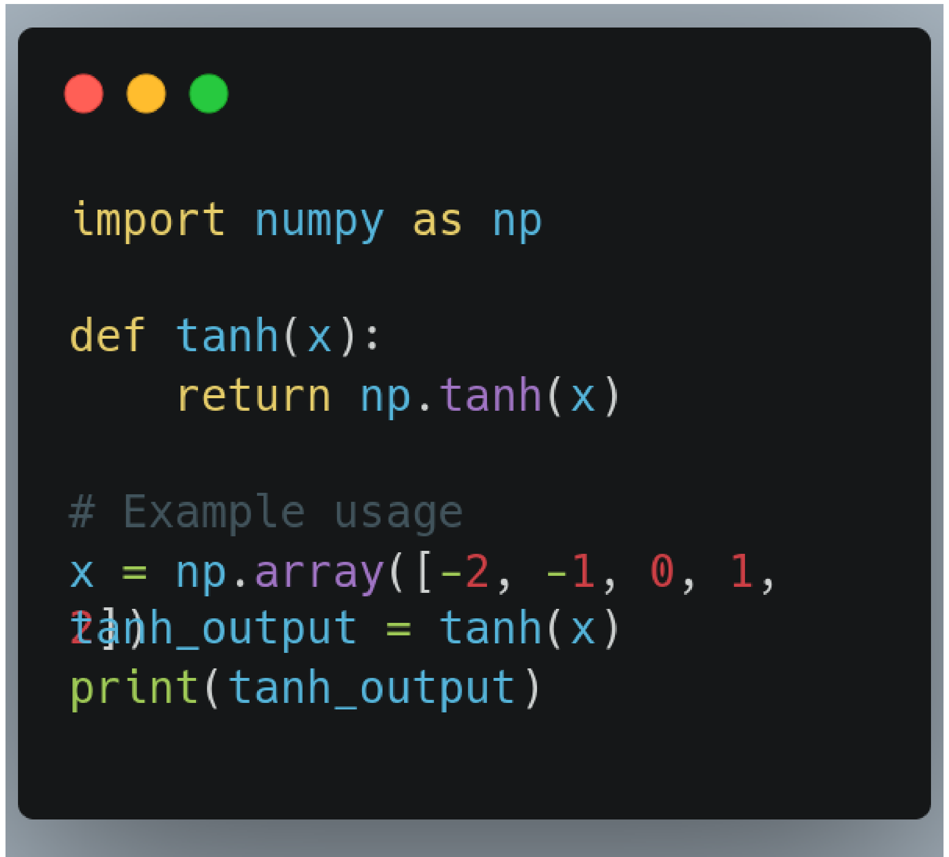
Characteristics

•**Output Range**: outputs values between -1 and 1.

•**Usage**: used in hidden layers.

•**Interpretation**: The output is centered around zero, which can make the optimization of the network easier.



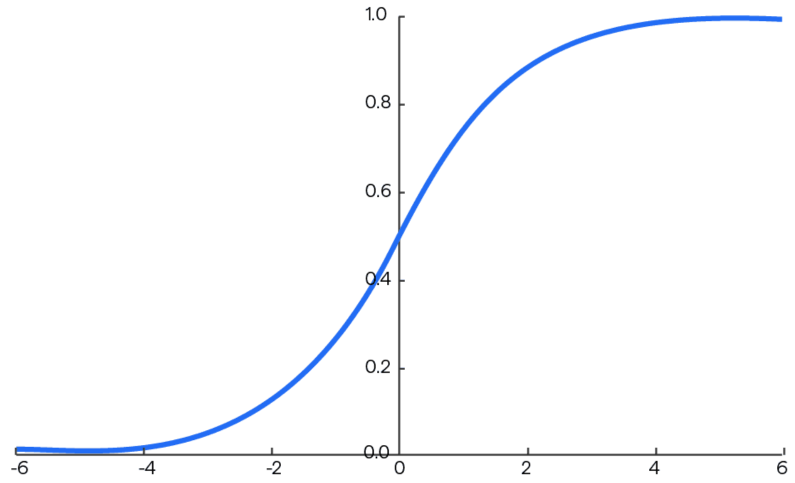


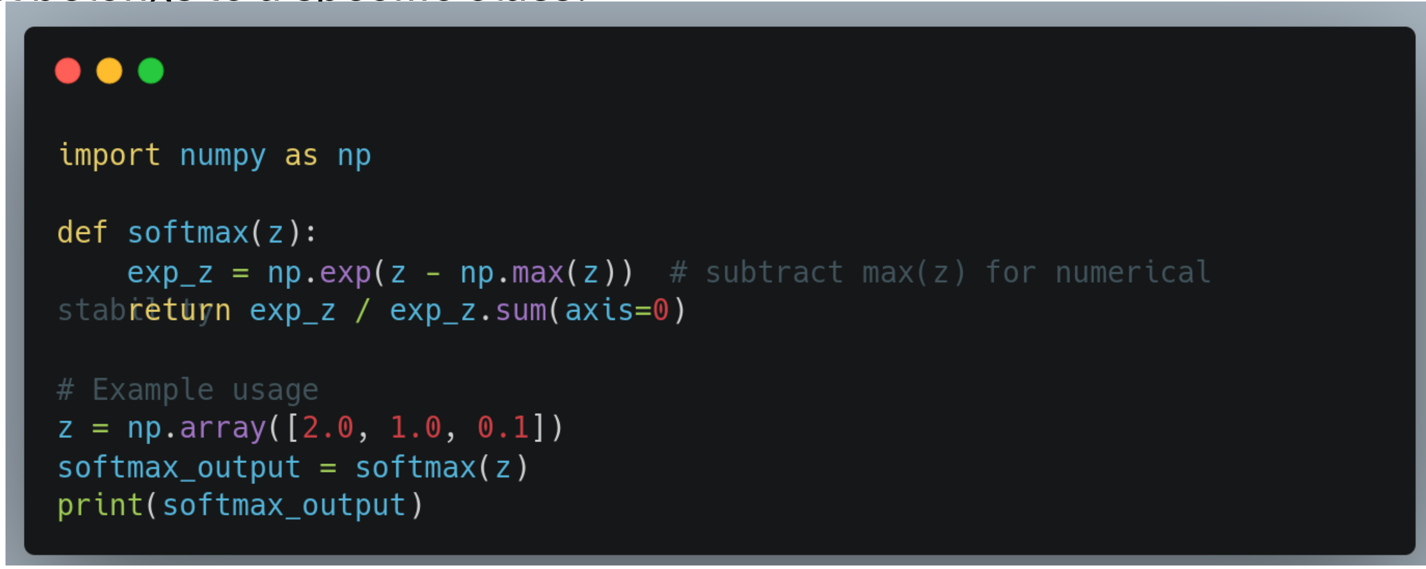
```python
import numpy as np

def tanh(x):
    return np.tanh(x)

# Example usage
x = np.array([-2, -1, 0, 1, 2])
tanh_output = tanh(x)
print(tanh_output)
```

Softmax

- **Output Range**: probability distribution over multiple classes, each value in the range (0, 1) and the sum of all outputs equal to 1.
- **Usage**: multi-class classification problems.
- **Interpretation**: Each output represents the probability that the input belongs to a specific class.

Softmax Function

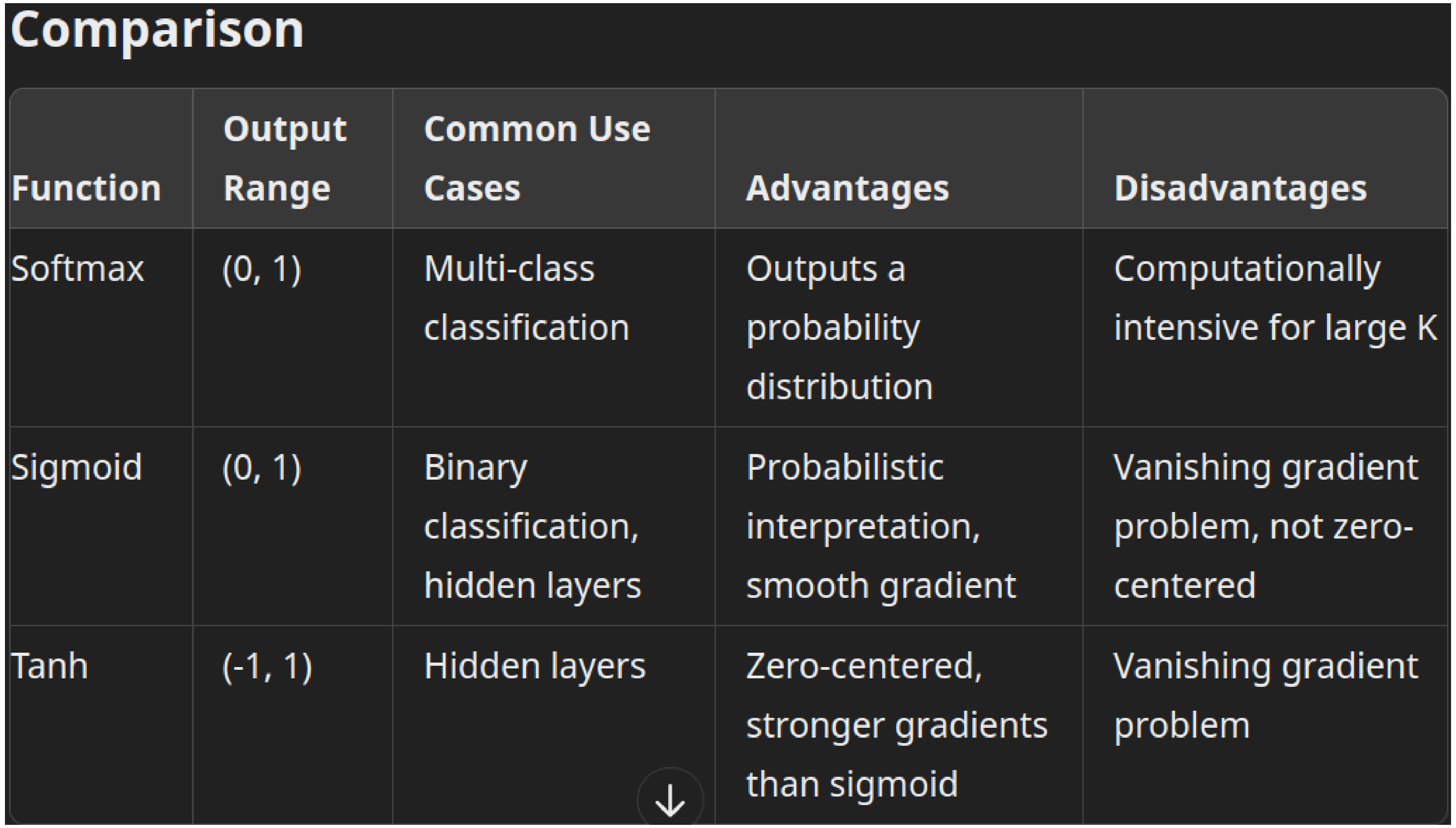




```python
import numpy as np

def softmax(z):
    exp_z = np.exp(z - np.max(z))  # subtract max(z) for numerical
stability
    return exp_z / exp_z.sum(axis=0)

# Example usage
z = np.array([2.0, 1.0, 0.1])
softmax_output = softmax(z)
print(softmax_output)
```

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

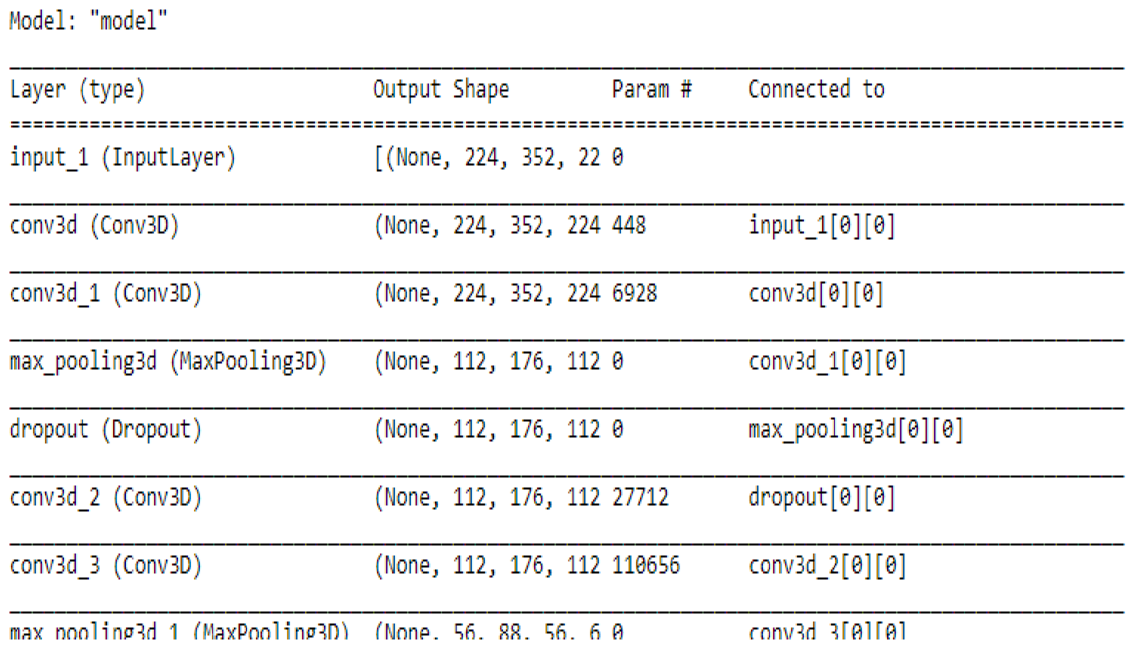
What is the diference between softmax, tanh, sigmoid?



Comparison

Function	Output Range	Common Use Cases	Advantages	Disadvantages
Softmax	(0, 1)	Multi-class classification	Outputs a probability distribution	Computationally intensive for large K
Sigmoid	(0, 1)	Binary classification, hidden layers	Probabilistic interpretation, smooth gradient	Vanishing gradient problem, not zero-centered
Tanh	(-1, 1)	Hidden layers	Zero-centered, stronger gradients than sigmoid	Vanishing gradient problem

Model.summary()

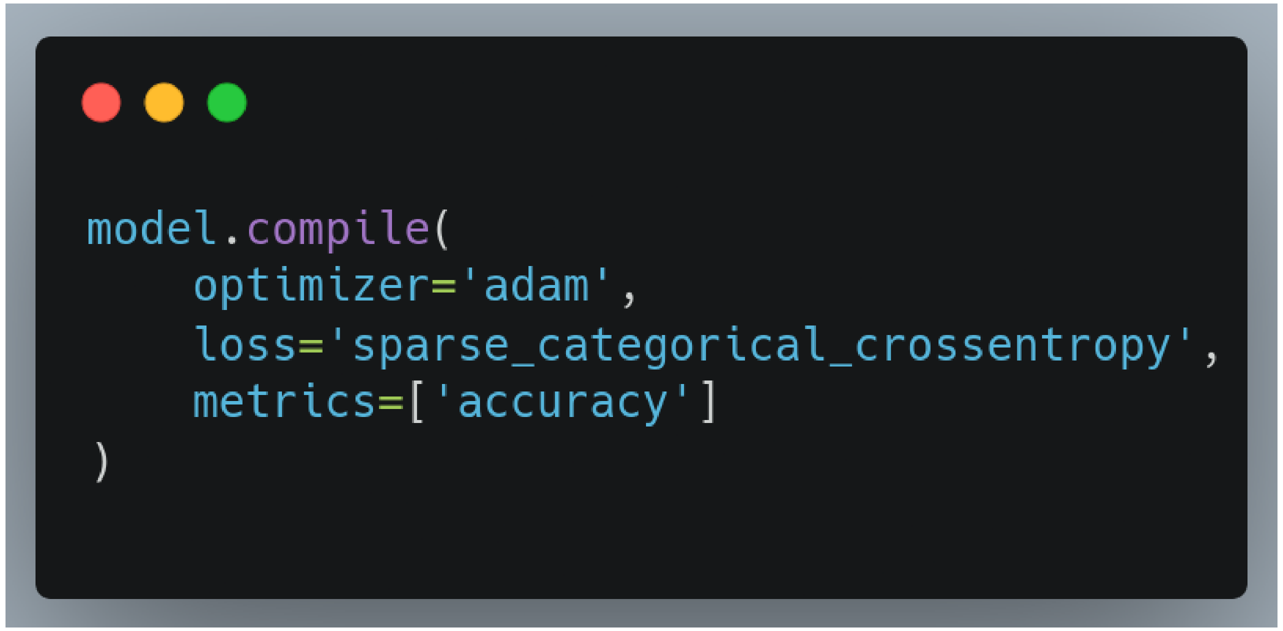


```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 352, 22 0

__________________________________________________________________________________________________
conv3d (Conv3D)                 (None, 224, 352, 224 448         input_1[0][0]
__________________________________________________________________________________________________
conv3d_1 (Conv3D)               (None, 224, 352, 224 6928        conv3d[0][0]
__________________________________________________________________________________________________
max_pooling3d (MaxPooling3D)    (None, 112, 176, 112 0           conv3d_1[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 112, 176, 112 0           max_pooling3d[0][0]
__________________________________________________________________________________________________
conv3d_2 (Conv3D)               (None, 112, 176, 112 27712       dropout[0][0]
__________________________________________________________________________________________________
conv3d_3 (Conv3D)               (None, 112, 176, 112 110656      conv3d_2[0][0]
__________________________________________________________________________________________________
max_pooling3d_1 (MaxPooling3D)  (None, 56, 88, 56, 6 0           conv3d_3[0][0]
```

method for get information about the layers of our model

Model compile

This method specifies how the model will be trained, evaluated, and optimized.



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

Optimizers

- Optimizer update the model in response to the loss function
- are algorithms
- used to change the attributes of your neural network such as weights

Gradient Descent Optimizer

- You can use gradient descent if your function is convex and difrensiable

Loss functions

- In most learning networks, error is calculated as the difference between the actual output $\boldsymbol{y}$ and the predicted output $\hat{\boldsymbol{y}}$.
- $\boldsymbol{Y}$-$\hat{\boldsymbol{Y}}$

metrics