What is TensorFlow ?

a) ~~Library~~ ✗
b) ~~Package~~ ✗
c) Ecosystem ✗
d) Framework ✓

What is a framework ?

A framework in software (including DL) is a comprehensive set of tools, libraries, rules and structures that provides everything you need to build, train and deploy models or applications

TENSORBOARD → Track models' performance leveraging Tensor board.

some framework examples: TensorFlow, PyTorch, Django (Web), React.js, Spring (Java)

TensorFlow is a deep learning framework.

- TensorFlow(TF) is an open-source developed by Google Brain in 2015
  - [for their internal research & production]
- TF is widely used for building DL models due to its flexibility, scalability and ability to run on CPUs: central Processing Units
  GPUs: Graphics Processing Units
  TPUs: Tensor Processing Units (by Google)
- TF supports building models at multiple abstraction levels
  - from low level operations to high level APIs like tf.keras

(TensorFlow)

Low level operations
For research & development

High level operations
* For production ready models

TENSORFLOW (low level)

```python
import tensorflow as tf
import numpy as np

# Dummy data
X = np.random.randn(100, 2).astype(np.float32)
y = (np.sum(X, axis=1) > 0).astype(np.float32).reshape(-1, 1)
```

1st Nov.

TF-KERAS HIGH LEVEL API

```python
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
```

26 Oct

```python
import numpy as np

# Dummy data
X = np.random.randn(100, 2).astype(np.float32)
y = (np.sum(X, axis=1) > 0).astype(np.float32).reshape(-1, 1)

# Initialize weights and biases
W1 = tf.Variable(tf.random.normal([2, 4]))
b1 = tf.Variable(tf.zeros([4]))
W2 = tf.Variable(tf.random.normal([4, 1]))
b2 = tf.Variable(tf.zeros([1]))

# Define model
def forward_pass(X):
    z1 = tf.matmul(X, W1) + b1
    a1 = tf.nn.relu(z1)
    z2 = tf.matmul(a1, W2) + b2
    output = tf.nn.sigmoid(z2)
    return output

# Loss and optimizer
def loss_fn(y_true, y_pred):
    return tf.reduce_mean(tf.keras.losses.binary_crossentropy(y_true, y_pred))

optimizer = tf.optimizers.Adam(learning_rate=0.01)

# Training Loop
for epoch in range(100):
    with tf.GradientTape() as tape:
        y_pred = forward_pass(X)
        loss = loss_fn(y, y_pred)
    grads = tape.gradient(loss, [W1, b1, W2, b2])
    optimizer.apply_gradients(zip(grads, [W1, b1, W2, b2]))
    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss.numpy():.4f}")
```

26 oct

```python
from tensorflow.keras import layers
import numpy as np

# Dummy data
X = np.random.randn(100, 2)
y = (np.sum(X, axis=1) > 0).astype(np.float32)

# Build model
model = keras.Sequential([
    layers.Dense(4, activation='relu', input_shape=(2,)),
    layers.Dense(1, activation='sigmoid')
])

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

# Train
model.fit(X, y, epochs=100, verbose=1)
```

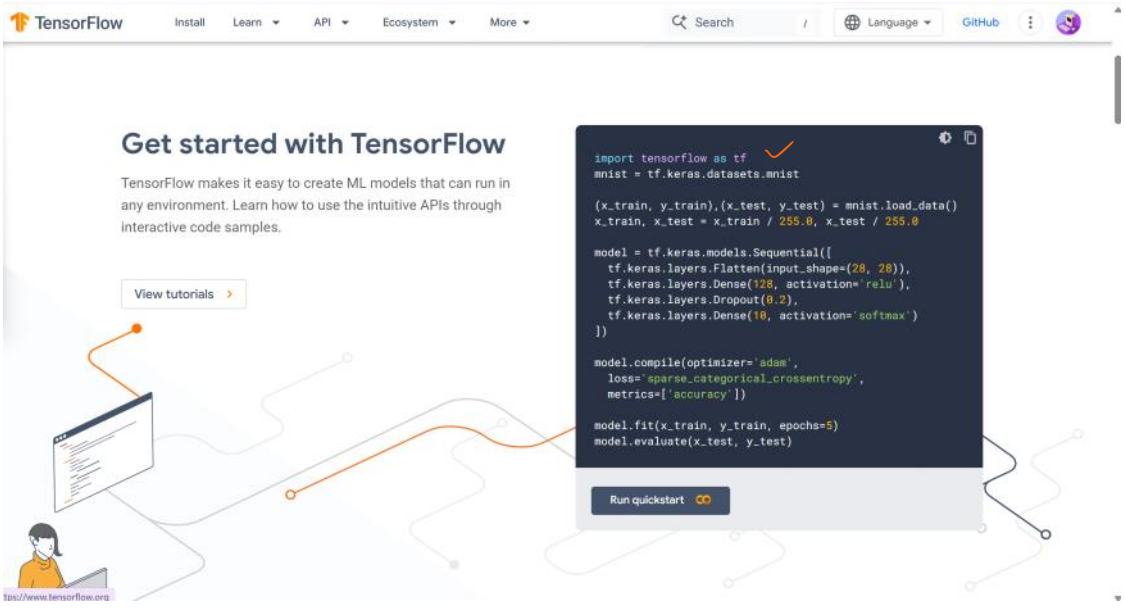
MLP from scratch – APC Model (even lower level)

25 oct

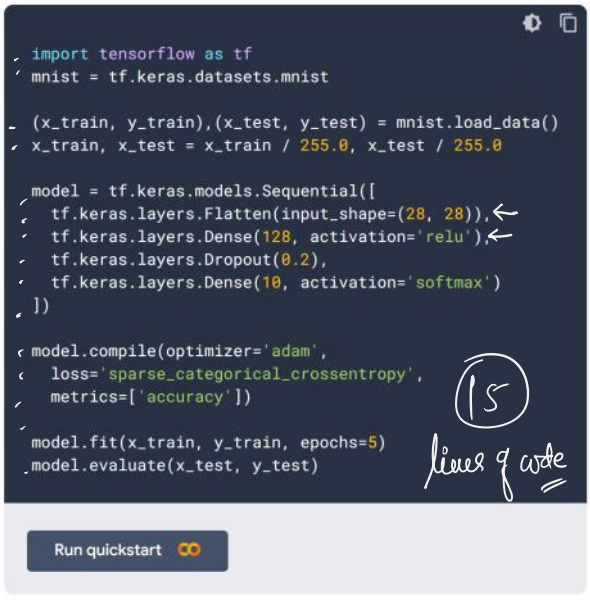
INPUT LAYER ---> (1) HIDDEN LAYER ---> OUTPUT LAYER

```python
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate = 0.01, epochs=100):
        self.input_size = input_size # no. of input vars or features in the input layer
        self.hidden_size = hidden_size #no. of neurons in the hidden layer (in here we have only one hidden layer)
        self.output_size = output_size #no. of neurons in the output layer
        self.learning_rate = learning_rate #learning rate for the gradient descent step
        self.epochs = epochs #no. of training epochs

        ### INITIALIZE WEIGHTS AND BIASES

        ##############################
        # INPUT LAYER TO HIDDEN LAYER
        ##############################
        self.W1 = np.random.randn(self.input_size, self.hidden_size)*0.01 #random weights drawn from std. normal distribution
        self.b1 = np.zeros((1, self.hidden_size)) # zero bias values for the neurons in the hidden layer

        ##############################
        # HIDDEN LAYER TO OUTPUT LAYER
        ##############################
        self.W2 = np.random.randn(self.hidden_size, self.output_size)*0.01 #random weights drawn from std. normal distribution
        self.b2 = np.zeros((1, self.output_size)) # zero bias values for the neurons in the output layer

        ### LOSS & ACCURACY HISTORY for PLOTTING
        self.loss_history = [] #empty list initialized to store the losses during training epochs
        self.accuracy_history = [] #empty list initialized to store the accuracy values during training epochs


    ### ADD SOME ACTIVATION FUNCTIONS

    ###  ReLU activation function
    def relu(self, z):
        return np.maximum(0,z)

    ### Derivative of ReLU for backpropagation
    def relu_derivative(self, z):
        return np.where(z>0, 1, 0)

    ### Softmax activation function
    def softmax(self, z):
        exp_values = np.exp(z - np.max(z, axis=1, keepdims=True)) #subtract max for numerical stability
        return exp_values/np.sum(exp_values, axis=1, keepdims= True)


    ##############################
    # 1.FORWARD PROPAGATION
    ##############################
    def forward(self, X):

        ##############################
        # INPUT LAYER TO HIDDEN LAYER
        ##############################
        self.z1 = np.dot(X, self.W1) + self.b1 # compute z = WiX + b1 #
        self.a1 = self.relu(self.z1) #plug z1 it into ReLU activation funtion to get output as a1

        ##############################
        # HIDDEN LAYER TO OUTPUT LAYER
        ##############################
        self.z2 = np.dot(self.a1, self.W2) + self.b2 # compute z = Wiai + b2 #
        self.probs = self.softmax(self.z2) #plug z2 it into SoftMax activation funtion to get output as a2
        return self.probs

    ##############################
    # 2.COMPUTE LOSS & ACCURACY
    ##############################
    ### Cross-entropy loss for multi-class classification
    def compute_loss(self, y_true, probs):
        loss = -np.mean(np.sum(y_true * np.log(probs), axis=1))
        return loss
    ### Compute accuracy
    def compute_accuracy(self, y_true, probs):
        predictions = np.argmax(probs, axis=1)
        true_labels = np.argmax(y_true, axis=1)
        return np.mean(predictions == true_labels)

    ##############################
    # 3.BACKPROPAGATION
    ##############################
    def backward(self, X, y):

        ## Using Batch Gradient Descent (BGD)
        ### Number of rows/training examples
        m = X.shape[0] #all rows in the data

        ### Gradients of the loss w.r.t. weights and biases of the output layer
        delta3 = self.probs - y #error at the output layer
        dW2 = np.dot(self.a1.T, delta3)/m #gradient of the loss w.r.t. weights of the output layer --> dW2
        db2 = np.sum(delta3, axis=0, keepdims=True)/m #gradient of the loss w.r.t bias of the output layer --> db2

        ### Gradients of the loss w.r.t weights and biases of the hidden layer
        delta2 = np.dot(delta3, self.W2.T) * self.relu_derivative(self.z1) #using derivative of ReLU
        dW1 = np.dot(X.T, delta2)/m #gradient of the loss w.r.t weights of the hidden layer --> dW1
        db1 = np.sum(delta2, axis=0, keepdims=True)/m #gradient of the loss w.r.t bias of the hidden layer --> db1

        ### Updating the parameters --> weights and biases across the layers (Hidden Layer & Output Layer)
        self.W2 -= self.learning_rate * dW2
        self.b2 -= self.learning_rate * db2
        self.W1 -= self.learning_rate * dW1
        self.b1 -= self.learning_rate * db1
```

https://www.tensorflow.org/

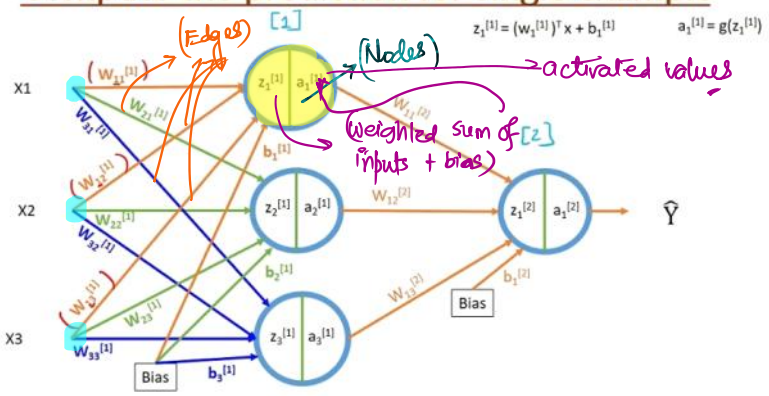


https://github.com/tensorflow/tensorflow/tree/master/tensorflow



What do you mean by 'Tensor' and 'Flow' in TensorFlow?

Compute the prediction for single example



In any n/w graph, there are nodes and edges:

$$W = \begin{bmatrix} W_{11} & W_{21} & W_{31} \cdots \\ W_{12} & W_{22} & W_{32} \cdots \\ \vdots & & \\ W_{15} & W_{25} & W_{35} \cdots \end{bmatrix}$$

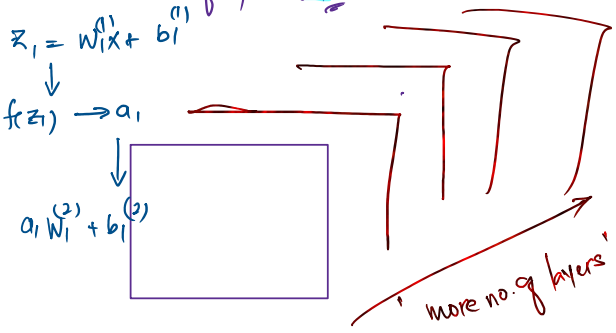
(edges)

represented by an array.

- edges in any n/w graph represent → multi-dimensional arrays
  ↓
  called as TENSOR.

Tensor: represents multi-dimensional data structure (n-dim arrays)

Flow: represents the way data moves through the n/w due to series of operations.



KERAS

What is Keras?

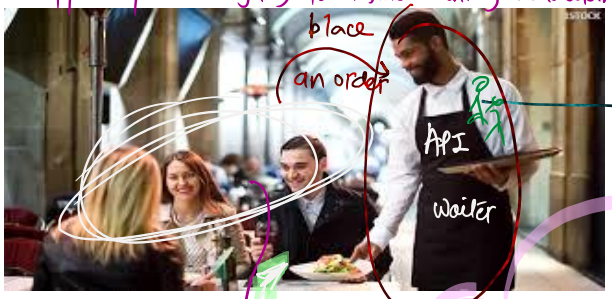
- library / API layer
  (High level neural n/w API)

- It is an open source high level neural n/w API (Application Programming Interface) developed by Francois Chollet and it released in 2015.

What's an API ?

- Stands for Application Programming Interface.

Suppose you are going to a fine-dining restaurant on a fine sunday.

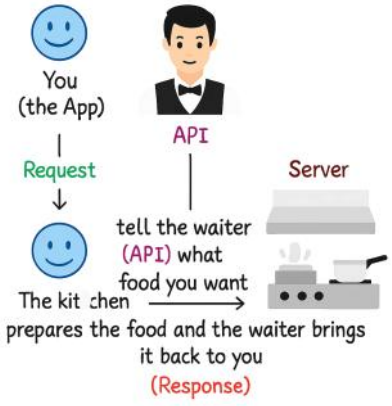


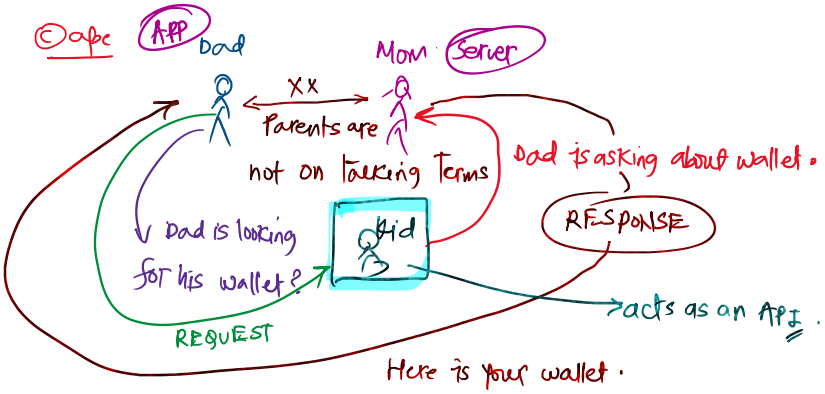
Kitchen (server)

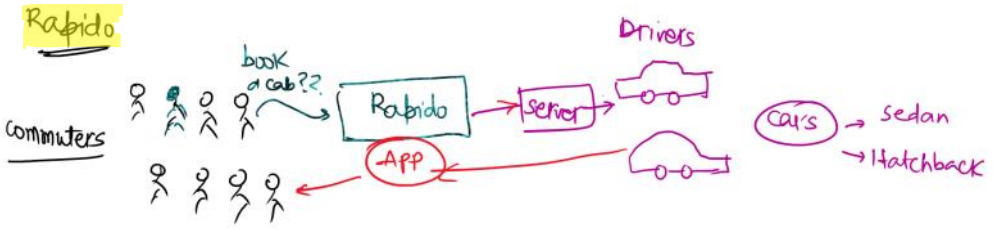
there are chefs multi-tasking on a lot of orders

order is READY

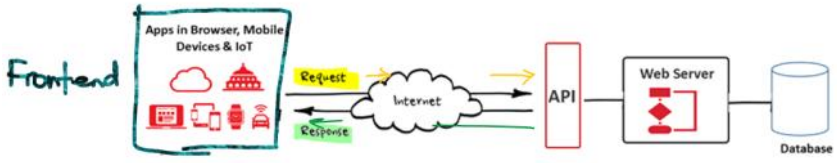
Application → Food (Sizzlers) Request







An API is a software which can be used by other softwares, to communicate with other softwares or even hardwares. It acts as a bridge between different softwares and devices.



API is a software intermediary that allows two or more applications to talk to each other

Why do we need Keras?

① User-friendly: Easy to use and understand → making it ideal platform for the beginners

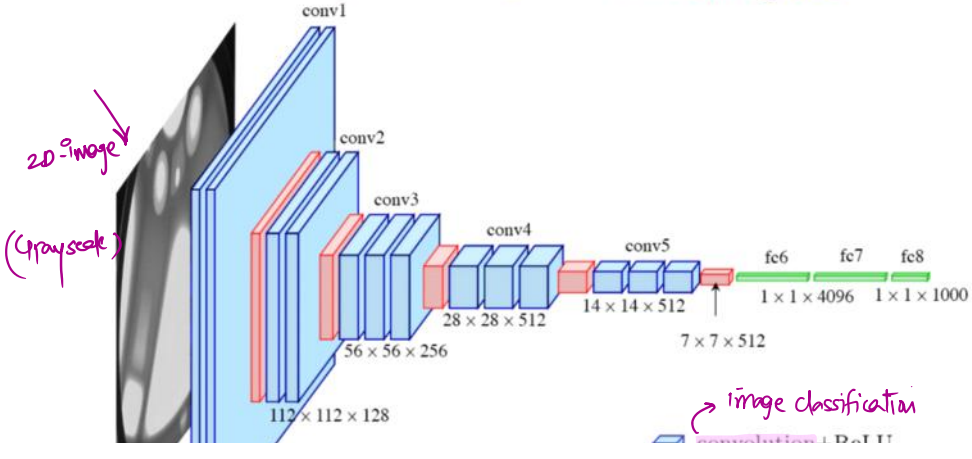
② Modularity: Building blocks for neural n/w models that's been saved after previously being trained on a large-scale datasets

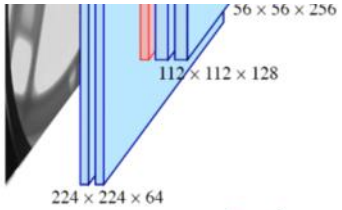
ChatGPT → can run on android/iOS phones/tablets
can run on very low configured laptops/computers

Note: Keras does provide several pre-trained models
- ResNet 50
- VGG 16
- YOLO v12
- YOLO v08
- deepseek r1

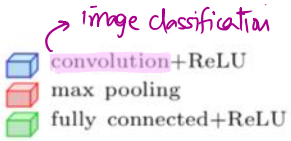
VGG16 is a deep convolutional neural network (CNN)

VGG16 is a deep convolutional neural network (CNN)





(Pre-trained model)



Manav - difference between API and ABI

TENSORFLOW vs KERAS

	TENSORFLOW	KERAS
◦ Type:	Deep learning framework	High level API for DL or High level library
◦ Level:	Low-level + High-level (via Keras)	High-level only.
◦ Origin:	Developed by Google Brain in 2015	Developed by Francois Chollet in 2015 too.
◦ Purpose	Build, train, deploy → DL models. (R&D) work → TensorFlow	Quickly build & train or for experimenting - Quick & dirty PoC
◦ Flexibility	very flexible, allows for custom operations and full control	Designed for simplicity, and limited or less control over low-level operations
◦ Standalone:	Yes, can be run without Keras.	Keras is tightly integrated with Tensor Flow

-MNIST Use-case - we'll build using TF first only & then using TF-Keras