**TensorFlow lab**

**Objective of the Program**

The primary objective of this program is to create, train, and evaluate a neural network model using TensorFlow to classify handwritten digits from the MNIST dataset.

**Steps and Explanation**

1. **Load and Preprocess the Data:**

**Goal:** To prepare the data for training the neural network.

* + The MNIST dataset is loaded, which contains 60,000 training images and 10,000 test images of handwritten digits (0-9).
  + Each image is a 28x28 pixel grayscale image.
  + The pixel values are normalized to the range [0, 1] by dividing by 255.0, which helps in faster convergence during training.
  + The labels (0-9) are one-hot encoded to create binary vectors. For example, the label '3' is converted to [0, 0, 0, 1, 0, 0, 0, 0, 0, 0].

1. **Build the Neural Network Model:**

**Goal:** To define the architecture of the neural network.

* + **Sequential Model:** A linear stack of layers.
  + **Flatten Layer:** Converts the 2D 28x28 pixel images into a 1D vector of 784 elements. This makes it easier to pass the data into the dense layers.
  + **Dense Layer (Hidden Layer):** Fully connected layer with 128 neurons and ReLU (Rectified Linear Unit) activation function. This layer learns to detect features from the input data.
  + **Dense Layer (Output Layer):** Fully connected layer with 10 neurons (one for each class) and softmax activation function. The softmax function converts the outputs into probabilities for each class.

1. **Compile the Model:**

**Goal:** To configure the learning process.

* + **Optimizer (Adam):** Algorithm to adjust the weights of the network to minimize the loss function.
  + **Loss Function (Categorical Crossentropy):** Measures the difference between the predicted and actual labels. It is used for classification problems.
  + **Metrics (Accuracy):** Metric to evaluate the performance of the model. It measures the proportion of correct predictions.

1. **Train the Model:**

**Goal:** To adjust the weights of the model based on the training data to minimize the loss function.

* + The fit method trains the model for a specified number of epochs (iterations over the entire training dataset).
  + The training data is split into training and validation sets. The model trains on the training set and validates on the validation set to monitor its performance.

1. **Evaluate the Model:**

**Goal:** To assess the performance of the trained model on unseen test data.

* + The evaluate method computes the loss and accuracy of the model on the test dataset.
  + The test accuracy is printed, indicating how well the model can classify new, unseen handwritten digits.

**Summary**

The program aims to build a neural network that can accurately classify handwritten digits from the MNIST dataset. By following the steps of loading data, building the model, training, and evaluating, the program demonstrates the process of developing a machine learning model using TensorFlow. The final output is the test accuracy, which tells us how well the model performs on new data.

**Code:**

# Import necessary libraries

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.utils import to\_categorical

# Load and preprocess the data

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Build the model

model = Sequential([

Flatten(input\_shape=(28, 28)),

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(x\_train, y\_train, epochs=5, batch\_size=32, validation\_split=0.2)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print('Test accuracy:', test\_acc)

# Activation Functions in Neural Networks

## Introduction

Activation functions play a crucial role in neural networks. They help the network learn complex patterns and make decisions. Two commonly used activation functions are ReLU (Rectified Linear Unit) and Softmax. Here's a simple explanation for each:

## ReLU (Rectified Linear Unit)

What is ReLU?  
- ReLU is an activation function that is defined as ReLU(x) = max(0, x).  
- This means that it outputs the input value if it's positive; otherwise, it outputs zero.

Why use ReLU?  
- Simplicity: ReLU is simple and computationally efficient.  
- Sparse Activation: It introduces non-linearity to the model, allowing the network to learn complex patterns.  
- Avoids Vanishing Gradient Problem: Unlike some other activation functions (like Sigmoid and Tanh), ReLU helps mitigate the vanishing gradient problem, which can slow down or halt the learning process in deep networks.

How does ReLU work?  
- If the input is positive, the output is the same as the input.  
- If the input is negative, the output is zero.

Example:  
- Input: [-2, -1, 0, 1, 2]  
- Output: [0, 0, 0, 1, 2]

Visual Representation:  
 |  
 | /  
 | /  
 | /  
 |\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_  
 -2 -1 0 1 2

## Softmax

What is Softmax?  
- Softmax is an activation function that converts a vector of raw scores (logits) into probabilities. The probabilities of the output classes sum to 1.  
- It is defined as: Softmax(z\_i) = e^(z\_i) / sum(e^(z\_j)), where z\_i is the input score for the i-th class, and the denominator is the sum of exponentials of all input scores.

Why use Softmax?  
- Probability Distribution: Softmax is used in the output layer of a neural network for multi-class classification problems. It provides a probability distribution over classes.  
- Interpretable Outputs: The output probabilities make it easy to interpret the model's predictions.

How does Softmax work?  
- Each score in the input vector is exponentiated.  
- These exponentiated values are then normalized by dividing by the sum of all exponentiated values.  
- This process ensures that the output values are between 0 and 1 and sum to 1.

Example:  
- Input: [2.0, 1.0, 0.1]  
- Exponentiated values: [e^2.0, e^1.0, e^0.1] ≈ [7.39, 2.72, 1.11]  
- Sum of exponentiated values: 7.39 + 2.72 + 1.11 = 11.22  
- Output: [7.39/11.22, 2.72/11.22, 1.11/11.22] ≈ [0.66, 0.24, 0.10]

Visual Representation:  
 Class 1: 0.66  
 Class 2: 0.24  
 Class 3: 0.10

## Summary

- ReLU (Rectified Linear Unit): Outputs the input if it's positive; otherwise, it outputs zero. It's simple and helps in learning complex patterns by introducing non-linearity.  
- Softmax: Converts raw scores into a probability distribution over multiple classes, useful for multi-class classification problems.  
  
Both activation functions have specific roles and are chosen based on the layer's purpose in the neural network. ReLU is typically used in hidden layers to introduce non-linearity, while Softmax is used in the output layer for classification tasks.