VCET TECHZETTE विसीईटी ज्ञानपत्र

ISSN: 2584-0886 (Online)

Building a Simple Multi-Layer Perceptron (MLP): A Step-by-Step Guide

Shalaka Kadam

- **Introduction**

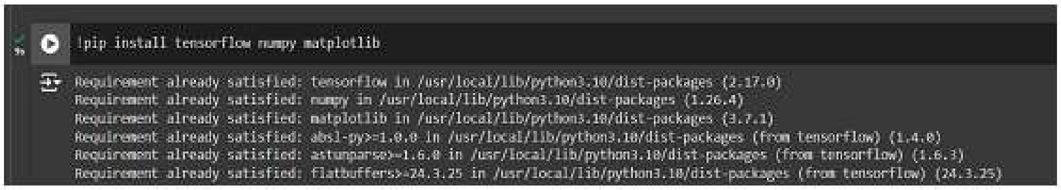
In this tutorial, we will guide you through the process of building a simple Multi-Layer Perceptron (MLP) from scratch using Python and TensorFlow. We'll use the MNIST dataset of handwritten digits to train and test our MLP, and every step will be accompanied by a screenshot to make it easier for beginners to follow along. This article is perfect if you are new to neural networks and want a hands-on introduction to building your first MLP model. Let's dive in!

- **What is an MLP?**

A Multi-Layer Perceptron (MLP) is a fundamental type of neural network that consists of three main layers: an input layer, one or more hidden layers, and an output layer. The input layer receives the data, the hidden layers process the data, and the output layer generates the final prediction. An MLP "learns" by adjusting its weights and biases during the training phase, allowing it to make more accurate predictions over time.

- **Setting Up the Environment**

To start building our MLP, we need to set up the development environment by installing a few necessary libraries. Open your terminal and run the following command to install TensorFlow, NumPy, and Matplotlib:



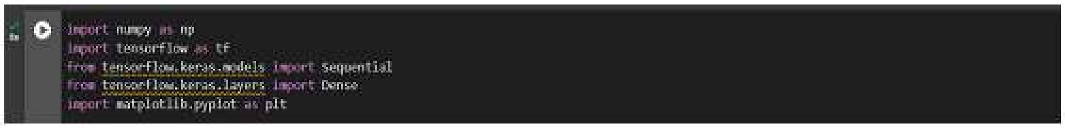
```
!pip install tensorflow numpy matplotlib

Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
```

Figure : install libraries

VCET TECHZETTE विसीईटी ज्ञानपत्र
ISSN: 2584-0886 (Online)

Building a Simple Multi-Layer Perceptron (MLP): A Step-by-Step Guide

After installing the libraries, we can begin by importing them into our Python script. These libraries will help us load the data, build the MLP model, and visualize the results. Here's the code to import them:

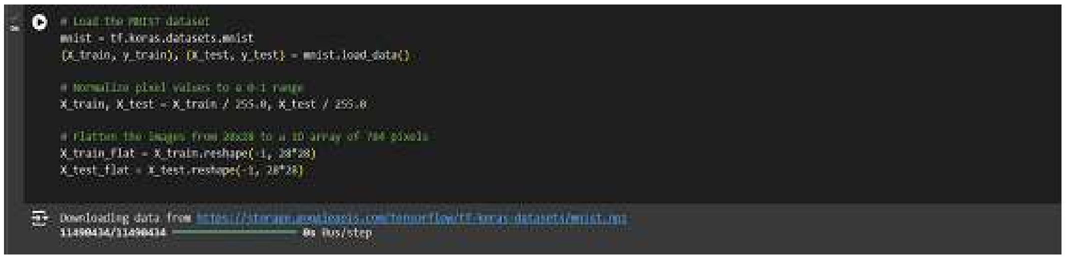


```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
```

Figure : import libraries

• **Building the MLP**

The first step in building the MLP is to load and preprocess the data. We'll be using the MNIST dataset, which consists of 28x28 pixel images of handwritten digits ranging from 0 to 9. TensorFlow provides this dataset out-of-the-box, so loading it is simple. After loading the data, we need to normalize the pixel values to be between 0 and 1. This makes the training process smoother. We'll also flatten each 28x28 image into a single vector of 784 pixels, which will serve as input for our MLP model.



```
# Load the MNIST dataset
mnist = tf.keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize pixel values to a 0-1 range
X_train, X_test = X_train / 255.0, X_test / 255.0

# Flatten the images from 28x28 to a 1D array of 784 pixels
X_train_flat = X_train.reshape(-1, 28*28)
X_test_flat = X_test.reshape(-1, 28*28)
```

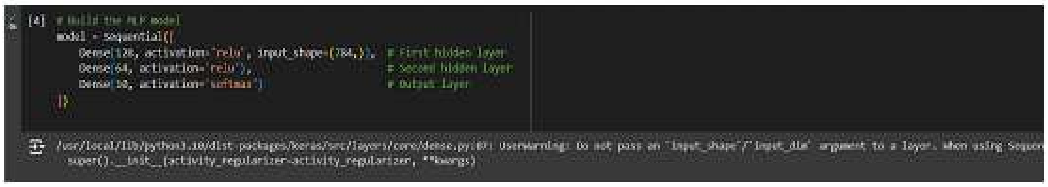
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Figure : data being loaded, normalized, and flattened

With the data prepared, we can now define the architecture of the MLP model. We'll use the Sequential API provided by TensorFlow's Keras module, which allows us to easily stack layers. Our MLP will have an input layer, two hidden layers, and an output layer. The first

Building a Simple Multi-Layer Perceptron (MLP): A Step-by-Step Guide

hidden layer will contain 128 neurons, and the second will have 64 neurons. Both of these layers will use the ReLU activation function, which helps the model learn complex patterns. The output layer will have 10 neurons, one for each possible digit (0-9), and will use the softmax activation function to output probabilities for each class.

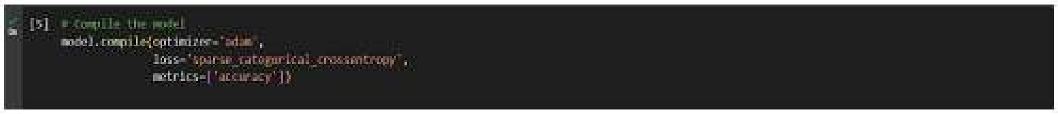


```
[4] # Build the MLP model
    model = Sequential([
        Dense(128, activation='relu', input_shape=(784,)),  # First hidden layer
        Dense(64, activation='relu'),                       # Second hidden layer
        Dense(10, activation='softmax')                     # Output layer
    ])

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seque
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Figure : MLP model structure

- **Training the MLP**

Before we can train the MLP, we need to compile the model. Compiling involves specifying the loss function, which helps the model understand how far its predictions are from the actual labels. In this case, we'll use sparse_categorical_crossentropy, which is commonly used for classification problems where the labels are integers (like in our MNIST dataset). We also define the optimizer, which is the algorithm that adjusts the model's weights during training to minimize the loss. We'll use the popular Adam optimizer for this. Lastly, we'll track the accuracy metric to evaluate how well the model is performing.



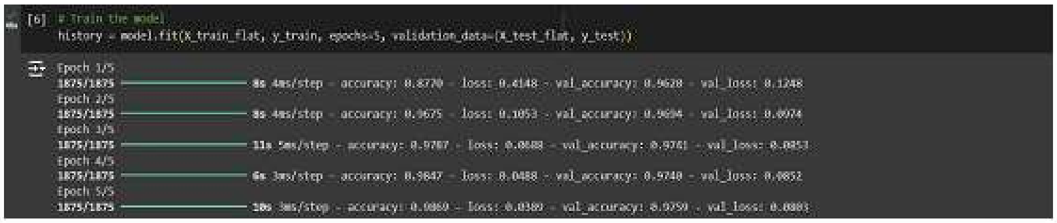
```
[5] # Compile the model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

Figure : Compile the model

Once the model is compiled, we can begin the training process. We'll train the MLP using the fit() method, which will allow the model to learn from the training data. We'll train it for 5 epochs, meaning the model will see the entire dataset 5 times. We'll also evaluate the model on the test dataset after each epoch to see how well it generalizes to unseen data.

VCET TECHZETTE विसीईटी ज्ञानपत्र
ISSN: 2584-0886 (Online)

Building a Simple Multi-Layer Perceptron (MLP): A Step-by-Step Guide

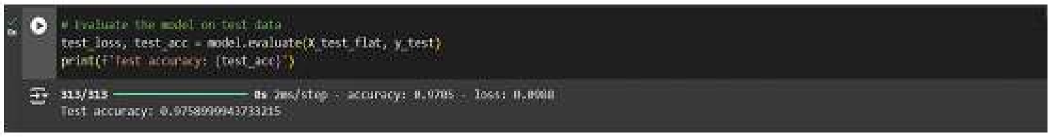


```
[6] # Train the model
    history = model.fit(X_train_flat, y_train, epochs=5, validation_data=(X_test_flat, y_test))

    Epoch 1/5
    1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8770 - loss: 0.4148 - val_accuracy: 0.9628 - val_loss: 0.1248
    Epoch 2/5
    1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9675 - loss: 0.1053 - val_accuracy: 0.9694 - val_loss: 0.0974
    Epoch 3/5
    1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9787 - loss: 0.0688 - val_accuracy: 0.9741 - val_loss: 0.0853
    Epoch 4/5
    1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9847 - loss: 0.0488 - val_accuracy: 0.9740 - val_loss: 0.0852
    Epoch 5/5
    1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9869 - loss: 0.0389 - val_accuracy: 0.9759 - val_loss: 0.0803
```

Figure : Training the model

- **Testing the MLP**

Now that the model has been trained, it's time to evaluate its performance on the test data. We can use the evaluate() method to calculate the test accuracy, which will tell us how well the MLP performs on data it has never seen before. A good test accuracy indicates that the model has learned to generalize well beyond the training examples.



```
# Evaluate the model on test data
test_loss, test_acc = model.evaluate(X_test_flat, y_test)
print(f'Test accuracy: {test_acc}')

313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9705 - loss: 0.0988
Test accuracy: 0.9758999943733215
```

Figure : Evaluate the model

In this guide, we built a simple Multi-Layer Perceptron (MLP) using Python and TensorFlow, covering everything from loading the MNIST dataset to training and testing the model. The MLP is a basic yet powerful neural network. Now that you've completed this project, feel free to experiment by adding layers, adjusting activation functions, or trying new datasets. Happy coding!