Name : Tambe Tanmay Jitendra

PRN: 22420169

Roll No : 382072

Batch: B3

Assignment No 1

Feedforward Neural Network

Problem Statement:

Implementing Feedforward neural networks in Python using Keras and TensorFlow

Objective:

- To understand the architecture and working of Feedforward Neural Networks (FNNs).
- To preprocess and normalize data for neural network training.
- To implement and train a feedforward model using Keras and TensorFlow.
- To evaluate the performance of the model using accuracy and loss metrics.
- To visualize training and validation performance.

Technical Apparatus used:

- **Operating System:** Windows/Linux/MacOS
- **Kernel:** Python 3.x
- **Tools:** Jupyter Notebook, Anaconda, or Google Colab
- **Hardware:** CPU with minimum 4GB RAM; optional GPU for faster processing

Libraries and Packages used:

- **TensorFlow**
- **Keras**
- **NumPy**
- **Matplotlib**

Theory:

A **Feedforward Neural Network (FNN)** is one of the simplest types of artificial neural networks. It consists of an input layer, one or more hidden layers, and an output layer. Information moves in

only one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes. There are no cycles or loops in the network

Structure:

Input Layer: Accepts flattened 28x28 pixel images (784 input nodes).

Hidden Layers: Dense (fully connected) layers with ReLU activation functions.

Fully Connected Layers: After flattening the pooled feature maps, these layers connect every neuron in one layer to every neuron in the next layer, leading to the output layer.

Output Layer: 10 output neurons with SoftMax activation for classifying 10 fashion categories.

Loss Function: Sparse Categorical Crossentropy.

Activation Functions:

Common activation functions used in FNN include ReLU (Rectified Linear Unit) and SoftMax, which introduce non-linearity into the model and help in classifying multiple classes.

Methodology:

1. **Data Acquisition:**

- The Fashion MNIST dataset is loaded using keras.datasets.fashion_mnist.
- It contains 70,000 grayscale images (60,000 training, 10,000 testing) across 10 classes (e.g., T-shirt/top, trousers, bag, etc.).

2. **Data Preparation:**

- Images are reshaped and normalized (scaled between 0 and 1) by dividing pixel values by 255.0.
- Each 28x28 image is flattened to a 1D array of size 784.
- Labels are kept as integers for use with sparse categorical loss.

3. **Define FNN Model Architecture:**

A **Sequential** model is built using Keras:

- **Input Layer**: Flatten (784 features).
- **First Dense Layer**: 128 neurons with ReLU activation.
- **Second Dense Layer**: 64 neurons with ReLU activation.
- **Output Layer**: 10 neurons with SoftMax activation.

4. **Model Compilation:**

- Compile the model using the Adam optimizer and Sparse Categorical Crossentropy as the loss function.

5. **Model Training:**
   - Fit the model on the training dataset while validating on the test dataset.
   - Track accuracy and loss over epochs.

6. **Model Evaluation:**
   - Evaluate the model on the test dataset to measure performance.

7. **Loss Visualization:**
   - Plot training and validation accuracy and loss over epochs to assess model performance.

Advantages:

- Easy to implement and train.
- Performs well on structured and moderately complex image datasets.
- Good baseline model for comparison with more complex architectures.
- Fully connected layers are powerful for general-purpose learning.

Limitations:

- Does not consider spatial structure (unlike CNNs).
- Prone to overfitting on image data.
- Lacks translation invariance.
- Requires manual flattening and loses image context.

Applications:

- Handwritten digit recognition (MNIST)
- Fashion item classification (Fashion MNIST)
- Simple tabular data classification
- Binary and multiclass classification tasks

Working / Algorithm:

Step 1: Load Dataset

The Fashion MNIST dataset, which contains 70,000 grayscale images of clothing items across 10 different classes, is loaded using the TensorFlow/Keras datasets API. The dataset is divided into 60,000 training images and 10,000 test images. Each image is of size 28x28 pixels

Step 2: Preprocess Data

Each image is normalized by dividing pixel values by 255 to scale them between 0 and 1. This helps in faster and more stable training.
Since the feedforward network expects 1D input, each 28x28 image is flattened into a 784-element vector.

Step 3: Visualize Data

A sample of images from the training dataset is displayed using Matplotlib. Each image is labeled with its corresponding class (e.g., T-shirt/top, trousers, bag, etc.), helping in understanding the dataset before training.

Step 4: Define CNN Model Architecture

A Feedforward Neural Network (FNN) is created using the Keras Sequential API:

- **Flatten Layer**: Converts 28x28 pixel images into 784-dimensional vectors.
- **First Dense Layer**: 128 neurons with ReLU activation function.
- **Second Dense Layer**: 64 neurons with ReLU activation.
- **Output Layer**: 10 neurons (for 10 classes) with SoftMax activation for multiclass classification.

Step 5: Compile the Model

The model is compiled with the Adam optimizer, sparse categorical cross entropy loss (appropriate for multiclass classification), and accuracy as the performance metric.

Step 6: Train the Model

The model is trained on the training dataset for **10 epochs** with a **batch size of 32**.
During training:

- The model updates weights after each batch.
- Validation is performed using the test dataset at the end of each epoch.

- Training and validation loss and accuracy are recorded.

Step 7: Evaluate the Model

After training, the model is evaluated on the test data to compute the final test loss and accuracy.

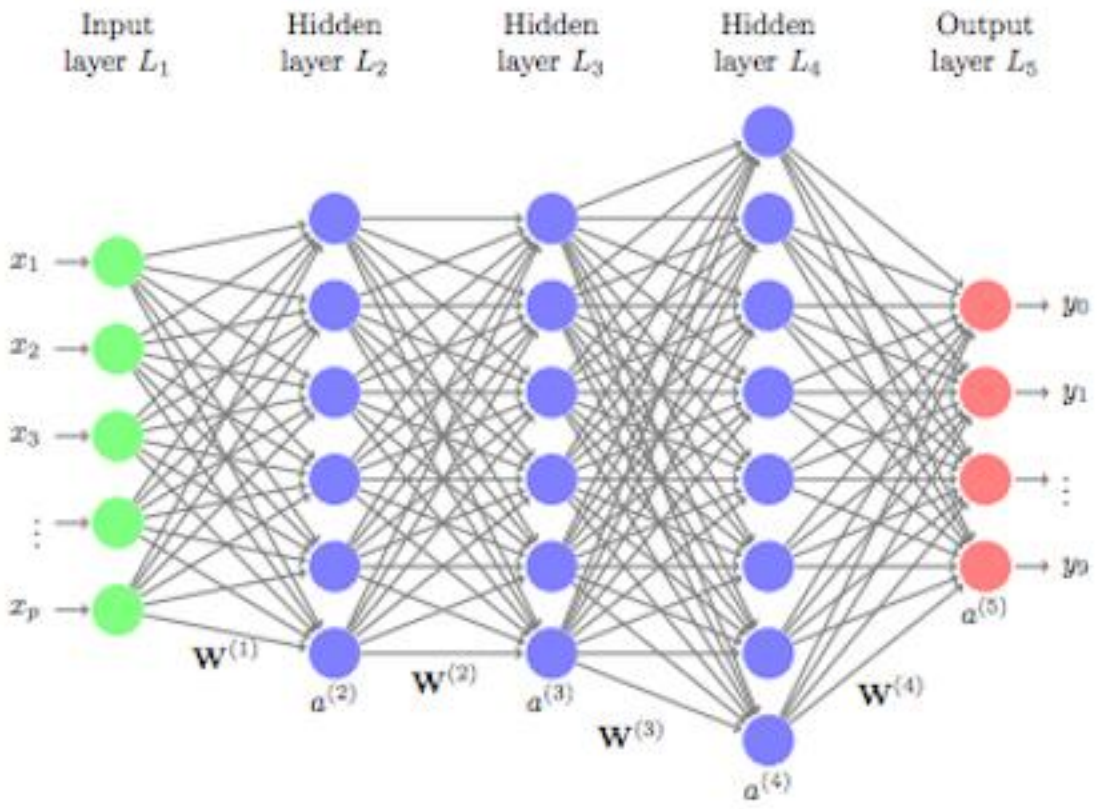
Step 8: Visualize Training History

A plot is generated showing the model's accuracy and validation accuracy over the training epochs.

Step 9: Print Test Accuracy

The final test accuracy is printed, indicating how well the model performs on unseen test data.

Diagram:



Conclusion:

In this assignment, we implemented a basic Feedforward Neural Network using Keras and TensorFlow to perform multiclass classification on the Fashion MNIST dataset. The model successfully learned to classify various fashion items, achieving good accuracy on unseen test data. While FNNs are limited in capturing image patterns, they serve as a strong foundation in understanding deep learning workflows before advancing to more complex models like CNNs.