**Assignment No: - 1**

**Name: Asiya Hasan Attar**

**Div:B**

**Roll No : 382048**

**Prn : 22311467**

**Feed-Forward Neural Network**

**Problem Statement:**

Implementing Feedforward neural networks in Python using Keras and TensorFlow.

**Objective:**

* To understand the basic structure of feedforward neural networks.
* To learn how to preprocess data for training neural networks.
* To implement a feedforward neural network model using Keras and TensorFlow.
* To evaluate model performance using validation data.
* To visualize training loss and validation loss over epochs.

**S/W Packages and H/W 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 training

**Libraries and packages used:** TensorFlow, Keras, NumPy, Pandas, Matplotlib, Scikit-Learn

**Theory:**

**Definition:** A feedforward neural network is a type of artificial neural network where connections between the nodes do not form cycles. The information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes.

**Structure:** It consists of:

* Input Layer: Receives the input features.
* Hidden Layers: One or more layers where computation occurs. Each neuron in a layer is connected to every neuron in the next layer.
* Output Layer: Produces the output of the network.

**Activation Functions:** Functions like ReLU (Rectified Linear Unit), Sigmoid, and SoftMax are used to introduce non-linearity into the model.

**Backpropagation:** A key algorithm used for training the network, where the error is propagated backward through the network to update weights.

**Methodology:**

**1. Data Acquisition**

* We use the MNIST dataset which is already available inside TensorFlow.
* This dataset contains images of handwritten digits (0–9).
* It is divided into training data (images + labels) and test data.

**2. Data Preparation**

* The pixel values of images are between 0–255.
* We normalize them to a range between 0 and 1 by dividing by 255.
* This makes training faster and more accurate.

**3. Model Architecture**

* We build a Sequential model in Keras.
* Layers are:
  + Flatten layer: Converts each 28×28 image into a 1‑D array (784 values).
  + Dense layer (128 units, ReLU): First hidden layer to learn patterns.
  + Dense layer (64 units, ReLU): Second hidden layer for deeper learning.
  + Dense layer (10 units, Softmax): Output layer with 10 units (one for each digit 0–9). Softmax is used to output probabilities.

**4. Model Compilation**

* Optimizer: Adam (helps in faster and efficient learning).
* Loss Function: Sparse Categorical Crossentropy (used because we are classifying digits into categories 0–9).
* Metric: Sparse Categorical Accuracy (to measure how many predictions are correct).

**5. Model Training**

* The model is trained on the training dataset.
* Training adjusts weights inside the model so it correctly predicts digits.

**6. Model Evaluation**

* After training, the model is tested on the test dataset (images the model hasn’t seen before).
* The model outputs test loss and test accuracy.
* Accuracy tells how well the model identifies handwritten digits.

**7. Output**

* Finally, the test accuracy is printed.
* This shows how accurate the model is at recognizing digits.

**Advantages:**

* **Non-linearity Handling:**

Feedforward neural networks use activation functions (like ReLU, sigmoid, or tanh) that introduce non-linearities, allowing them to learn complex relationships in data that linear models cannot capture.

* **Flexibility in Architecture:**

These networks can be easily modified to suit various tasks by adjusting the number of layers, neurons, and types of activation functions, making them versatile for different applications.

* **Scalability:**

Feedforward neural networks can scale well with the addition of more hidden layers and neurons, which can improve the model's ability to learn from large datasets.

* **Robustness:**

When properly trained, feedforward neural networks can generalize well to unseen data, making them effective for various prediction tasks in real-world applications.

* **Parallel Processing:**

The structure of feedforward neural networks allows for parallel computation of neurons, making them suitable for implementation on modern hardware like GPUs, significantly speeding up training times.

**Limitations:**

* **Data Requirements:**

Feedforward neural networks require large amounts of labelled data for effective training. Limited data can lead to overfitting, where the model performs well on training data but poorly on unseen data.

* **Computational Cost:**

Training deep networks can be computationally expensive, requiring significant time and resources, especially with large datasets and complex architectures.

* **Black-Box Nature:**

The inner workings of feedforward neural networks are often opaque, making it challenging to interpret how decisions are made. This can be a limitation in fields requiring explainability, like healthcare.

* **Overfitting Risk:**

If the network is too complex for the dataset, it can overfit, capturing noise instead of the underlying pattern, which degrades performance on new data.

* **Hyperparameter Sensitivity:**

Performance can be significantly influenced by the choice of hyperparameters (learning rate, number of layers, etc.), making tuning crucial yet time-consuming.

**Applications:**

* **Non-linearity Handling:**

Feedforward neural networks use activation functions (like ReLU, sigmoid, or tanh) that introduce non-linearities, allowing them to learn complex relationships in data that linear models cannot capture.

* **Flexibility in Architecture:**

These networks can be easily modified to suit various tasks by adjusting the number of layers, neurons, and types of activation functions, making them versatile for different applications.

* **Scalability:**

Feedforward neural networks can scale well with the addition of more hidden layers and neurons, which can improve the model's ability to learn from large datasets.

* **Robustness:**

When properly trained, feedforward neural networks can generalize well to unseen data, making them effective for various prediction tasks in real-world applications.

* **Parallel Processing:**

The structure of feedforward neural networks allows for parallel computation of neurons, making them suitable for implementation on modern hardware like GPUs, significantly speeding up training times.

**Working / Algorithm:**

**Step 1: Import the necessary libraries.**

* Import TensorFlow and Keras modules needed to build the neural network.

**Step 2: Load the dataset.**

* Use the built‑in MNIST dataset provided by TensorFlow.
* This dataset contains handwritten digit images (0–9) along with their labels.

**Step 3: Preprocess the data.**

* Normalize the image pixel values by dividing them by 255.0.
* This scales the input values between 0 and 1, making training faster and more accurate.

**Step 4: Define the model architecture.**

* Use a Sequential model with the following layers:
  + Flatten layer: Converts the 28×28 image into a 1‑dimensional vector of 784 values.
  + Dense layer with 128 units and ReLU activation: Learns important features from the input data.
  + Dense layer with 64 units and ReLU activation: Learns deeper patterns.
  + Dense layer with 10 units and Softmax activation: Output layer that gives probability
  + distribution across 10 digit classes.

**Step 5: Compile the model.**

* Optimizer: Adam (helps faster convergence).
* Loss Function: Sparse Categorical Crossentropy (suitable for multi‑class classification).
* Evaluation Metric: Sparse Categorical Accuracy (measures accuracy of predictions).

**Step 6: Train the model.**

* Train the model using the training dataset (x\_train, y\_train).
* The model adjusts its internal weights to improve prediction accuracy.

**Step 7: Evaluate the model.**

* Test the model using the test dataset (x\_test, y\_test).
* Evaluate and compute metrics like test loss and test accuracy.

**Step 11: Visualize training loss.**

* Create a Data Frame to store the loss history and plot the training and validation loss to visualize model performance over epochs.

**Step 8: Output the accuracy.**

* Print the final test accuracy, which shows how well the model can recognize handwritten digits it has never seen

**Diagram:**



**Conclusion:**

n this assignment, we successfully implemented a deep learning model using TensorFlow and Keras to classify handwritten digits from the MNIST dataset. The dataset was preprocessed by normalizing the image pixel values, and a sequential neural network model was built consisting of a Flatten layer, two hidden Dense layers with ReLU activation, and an output Dense layer with Softmax activation. The model was compiled using the Adam optimizer, Sparse Categorical Crossentropy as the loss function, and accuracy as the performance metric.

After training and evaluation, the model achieved a good level of accuracy on the test dataset, which shows that it is able to effectively recognize and classify digits (0–9). This demonstrates the power of neural networks in solving real‑world image classification problems. The experiment also highlights the importance of preprocessing, proper model architecture, and evaluation in building accurate machine learning models.