**Assignment No: - 1**

**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.

**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:**

**1. Data Acquisition:**

* The dataset used in this project is the MNIST dataset, which contains 70,000 grayscale images of handwritten digits from 0 to 9.
* The dataset was loaded using tensorflow.keras.datasets.mnist, which provides a pre-split version of the dataset:
  + 60,000 samples for training
  + 10,000 samples for testing

**2. Data Preparation:**

* The image pixel values were originally in the range 0–255. These values were normalized by dividing by 255.0 to bring them to the range 0–1. This ensures faster and more stable training.
* The labels (0 to 9) were converted to one-hot encoded format using to\_categorical() to match the format expected by the neural network.
* No additional data cleaning or feature engineering was required since the dataset is already pre-processed and balanced.

**3. Model Architecture:**

* A Sequential Feedforward Neural Network (FNN) was created using Keras.
* The network contains the following layers:
  + Flatten Layer: Converts 2D input (28x28) into 1D array (784) for Dense layer input.
  + Dense Layer 1: 64 neurons with ReLU activation.
  + Dense Layer 2: 10 neurons with Softmax activation to output the probability of each digit class (0 to 9).

**4. Model Compilation:**

* The model was compiled using the following parameters:
  + Optimizer: Adam (adaptive and widely used optimizer)
  + Loss Function: categorical\_crossentropy (used for multi-class classification)
  + Metric: accuracy (to monitor the percentage of correct predictions)

**5. Model Training:**

* The model was trained using the fit() method.
* The training was performed for 5 epochs with a batch size of 32, which means the model updates its weights after every 32 samples.
* During training, the model automatically learned to minimize the loss and improve accuracy using backpropagation and gradient descent.

**6. Model Evaluation:**

* After training, the model was evaluated using the evaluate() method on the test dataset (10,000 images).
* The final output included the test accuracy, which indicates how well the model generalized to new, unseen data.

**7. Prediction and Result Interpretation:**

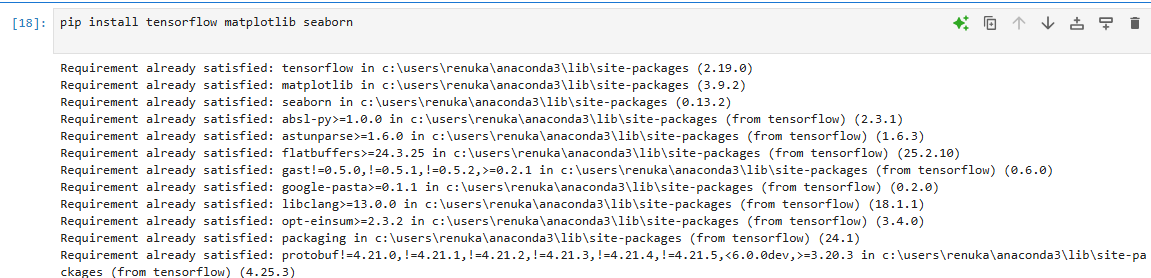
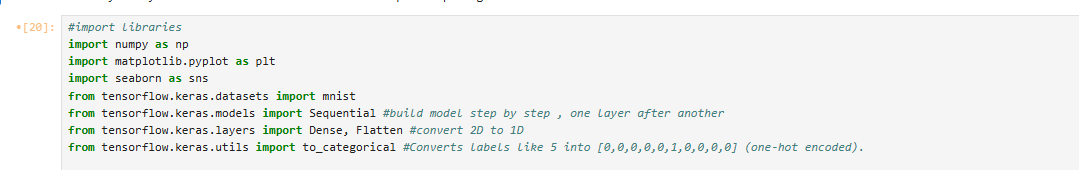
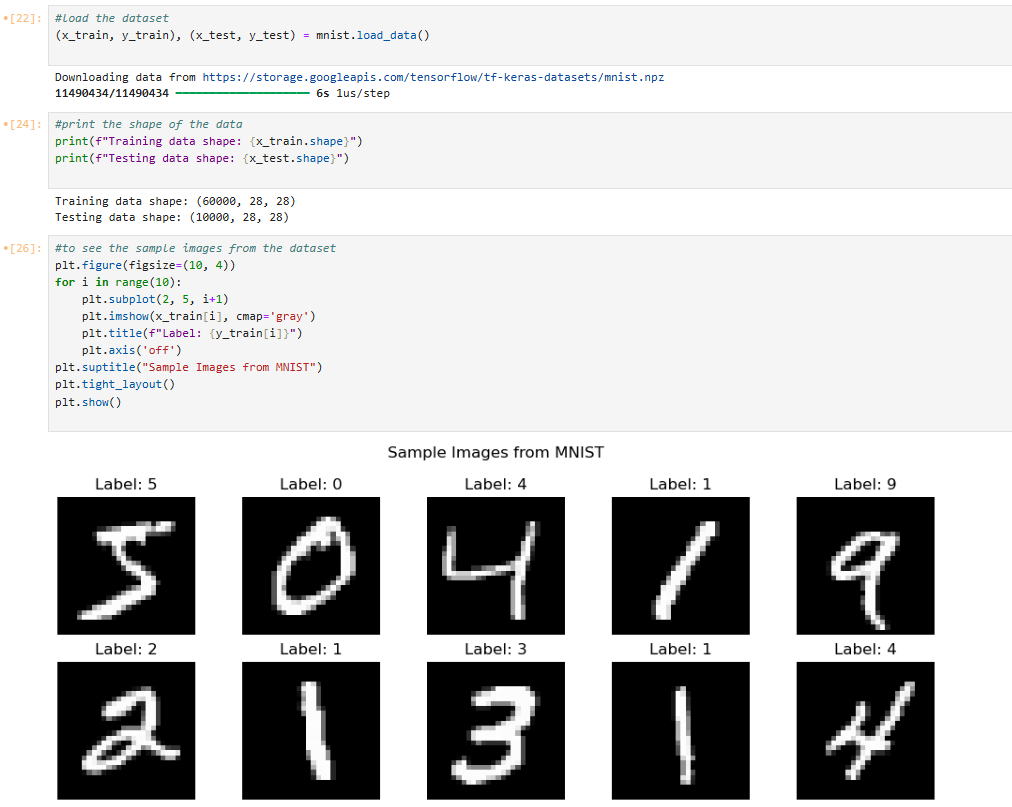
* The model was used to predict the classes of sample images from the test set.
* The predicted digit (with the highest probability) was displayed along with the actual image, allowing for visual confirmation of correct/incorrect classifications.

**8. Loss & Accuracy Visualization (Optional):**

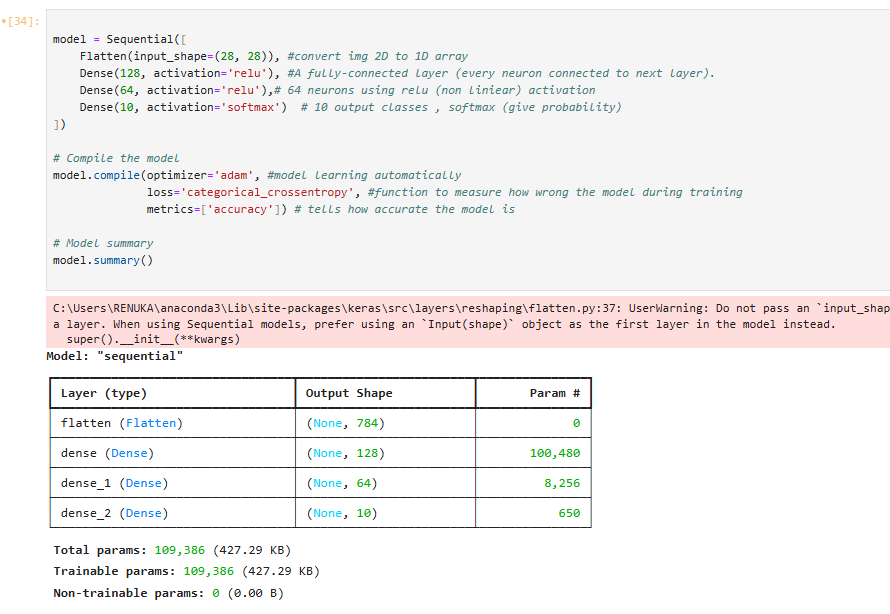
* The history object returned by model.fit() contains logs of training accuracy and loss.
* These values can be plotted using matplotlib to visualize how the model performed over each epoch and check for overfitting or underfitting.

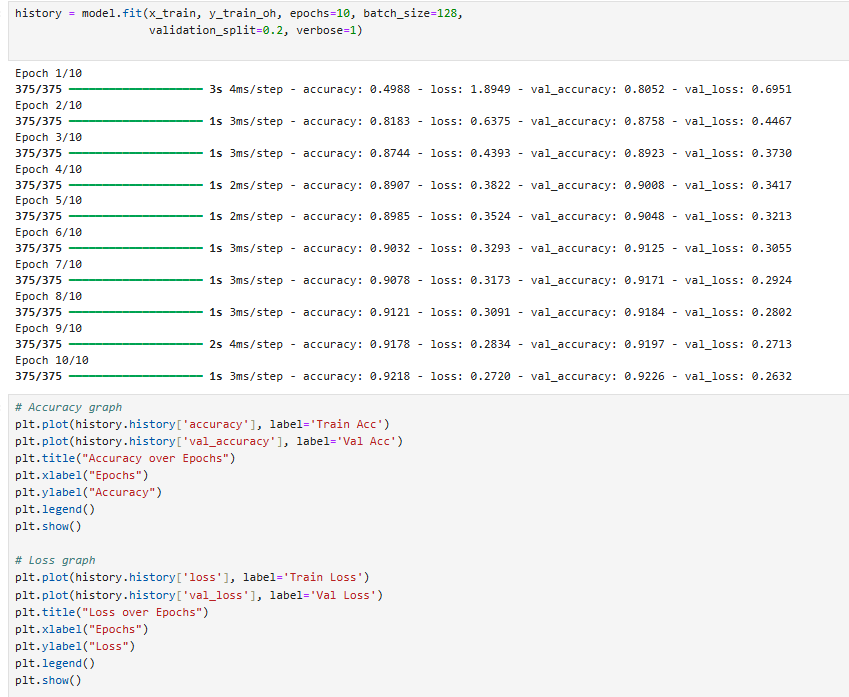
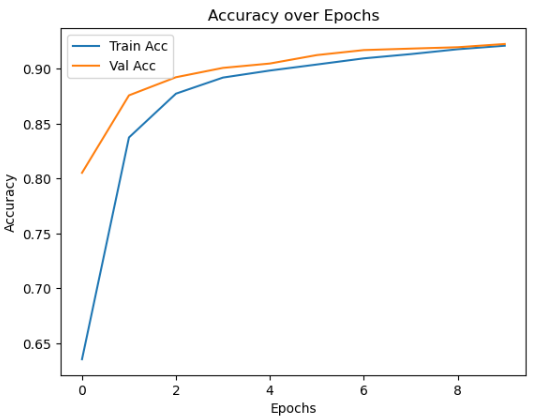
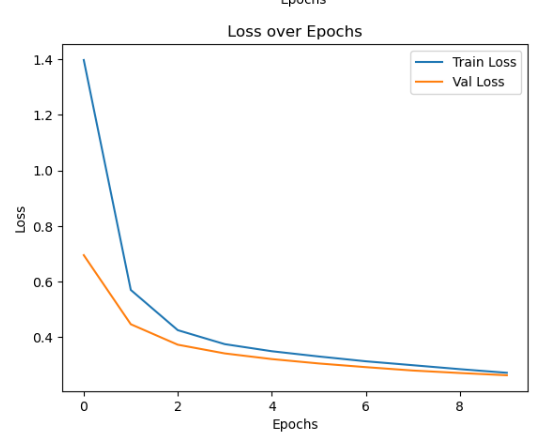
**Diagram:**

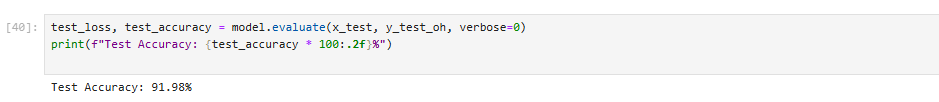


**Output:**

****

****

****

****

**Conclusion:**

In conclusion, the Feedforward Neural Network (FNN) algorithm is a powerful and versatile approach to predictive modeling, especially suited for both classification and regression tasks, such as predicting wine quality based on various chemical features. By utilizing its ability to learn complex, non-linear relationships between inputs and outputs, FNN can provide valuable predictions and insights that can drive decision-making. However, it's important to consider the limitations of FNN, such as the need for significant computational resources, the risk of overfitting, and the necessity of careful tuning of hyperparameters to ensure optimal performance. Despite these challenges, FNN remains a highly effective model for complex data scenarios when appropriately managed and applied.