Student name- Neha Sawant **PRN-** 22210877

Assignment No: - 1

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:**
    - Load the dataset (winequality-red.csv) using Pandas to analyze the chemical properties of red wine and their quality.
2. **Data Preparation:**
    - Split the dataset into training (75%) and validation (25%) sets.
    - Normalize the feature values to a range between 0 and 1 to facilitate faster convergence during training.
3. **Model Architecture:**
    - Create a sequential model using Keras.
    - Add multiple dense layers:
        - Input layer with 64 units and ReLU activation function.
        - Hidden layer with 64 units and ReLU activation function.
        - Output layer with a single unit for regression.
4. **Model Compilation:**
    - Compile the model using the Adam optimizer and Mean Absolute Error (MAE) as the loss function.
5. **Model Training:**
    - Fit the model to the training data while validating on the validation set.
    - Track the loss metrics over a specified number of epochs.
6. **Model Evaluation:**
    - Use the trained model to predict the quality of wine based on the validation dataset.
    - Compare the predicted values with actual values for assessment.
7. **Loss Visualization:**
    - Plot the training and validation loss over epochs to visualize the model's performance and check for overfitting.

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.

Use libraries such as NumPy, Pandas, and TensorFlow.

Step 2: Load the dataset.

Read the dataset (e.g., winequality-red.csv) into a Pandas Data Frame.

Step 3: Split the dataset.

Randomly select 75% of the data for training (train_df) and use the remaining 25% for validation (val_df).

Step 4: Normalize the data.

Scale the feature values to a range of (0, 1) using min-max normalization.

Step 5: Separate features and target variable.

Split the training and validation DataFrames into feature sets (X_train, X_val) and target labels (y_train, y_val).

Step 6: Define the input shape.

Determine the input shape based on the number of features in X_train.

Step 7: Build the model.

Create a sequential model with the following layers:

First hidden layer: Dense layer with 64 units and ReLU activation function.

Second hidden layer: Dense layer with 64 units and ReLU activation function.

Output layer: Dense layer with 1 unit for regression output.

Step 8: Compile the model.

Compile the model using the Adam optimizer and mean absolute error (MAE) as the loss function.

Step 9: Train the model.

Fit the model to the training data for a specified number of epochs, using the validation data to monitor performance.

Step 10: Evaluate the model.

Use the model to make predictions on the validation set and compare predicted values with actual target values.

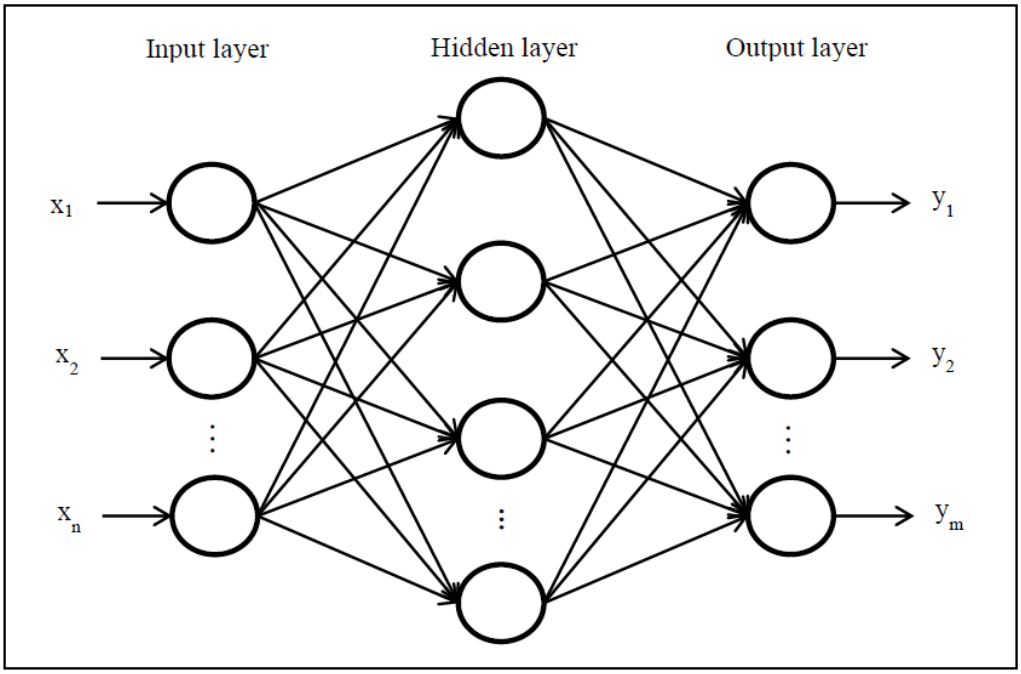
Step 11: Visualize training loss.

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

Step 12: Model readiness.

The model is now ready for use in making predictions on new data.

Diagram:



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.