**REG NO.220701055**

**EX.NO : 9** **DATE : 05.11.24**

# IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION

**Regression using Artificial Neural Networks**

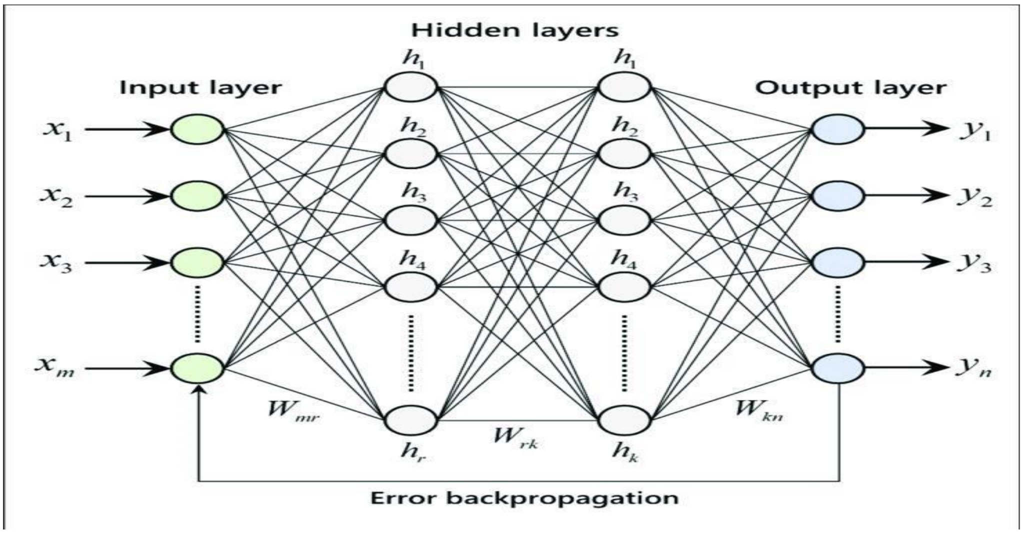
Why do we need to use Artificial Neural Networks for Regression instead of simply using Linear Regression?

The purpose of using Artificial Neural Networks for Regression over Linear Regression is that the linear regression can only learn the linear relationship between the features and target and therefore cannot learn the complex non-linear relationship. In order to learn the complex non-linear relationship between the features and target, we are in need of other techniques. One of those techniques is to use Artificial Neural Networks. Artificial Neural Networks have the ability to learn the complex relationship between the features and target due to the presence of activation function in each layer. Let's look at what are Artificial Neural Networks and how do they work.

## Artificial Neural Networks

Artificial Neural Networks are one of the deep learning algorithms that simulate the workings of neurons in the human brain. There are many types of Artificial Neural Networks, Vanilla Neural Networks, Recurrent Neural Networks, and Convolutional Neural Networks. The Vanilla Neural Networks have the ability to handle structured data only, whereas the Recurrent Neural Networks and Convolutional Neural Networks have the ability to handle unstructured data very well. In this post, we are going to use Vanilla Neural Networks to perform the Regression Analysis.

## Structure of Artificial Neural Networks




The Artificial Neural Networks consists of the Input layer, Hidden layers, Output layer. The hidden layer can be more than one in number. Each layer consists of n number of neurons. Each layer will be having an Activation Function associated with each of the neurons. The activation function is the function that is responsible for introducing non-linearity in the relationship. In our case, the output layer must contain a linear activation function. Each layer can also have regularizers associated with it. Regularizers are responsible for preventing overfitting.

Artificial Neural Networks consists of two phases,

- Forward Propagation
- Backward Propagation

Forward propagation is the process of multiplying weights with each feature and adding them. The bias is also added to the result. Backward propagation is the process of updating the weights in the model.
Backward propagation requires an optimization function and a loss function.

## AIM :

To implementing artificial neural networks for an application in Regression using python.

## CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
```

```
[ ] # Generate synthetic data for regression
    np.random.seed(42)
    X = np.random.rand(1000, 1) * 10  # Features: 1000 samples, single feature scaled between 0 and 10
    y = 2 * X + np.random.randn(1000, 1) * 2  # Target: linear relation with some noise

    # Convert to DataFrame for better visualization (optional)
    data = pd.DataFrame(np.hstack((X, y)), columns=['Feature', 'Target'])
    print(data.head())
```

```
[ ] # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Scale the features (important for neural networks)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

```
[ ] # Create the ANN model
    model = keras.Sequential()
    model.add(layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)))  # Input layer
    model.add(layers.Dense(32, activation='relu'))  # Hidden layer
    model.add(layers.Dense(1))  # Output layer (for regression)

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
```

```
[ ] # Train the model
    history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

```
[ ] # Evaluate the model on test data
    test_loss = model.evaluate(X_test, y_test)
    print(f'Test Loss (MSE): {test_loss}')
```

```
[ ] # Make predictions on test data
    y_pred = model.predict(X_test)

    # Plotting the results
    plt.scatter(X_test, y_test, color='blue', label='True Values')
    plt.scatter(X_test, y_pred, color='red', label='Predicted Values')
    plt.title('True vs Predicted Values')
    plt.xlabel('Feature')
    plt.ylabel('Target')
    plt.legend()
    plt.show()
```

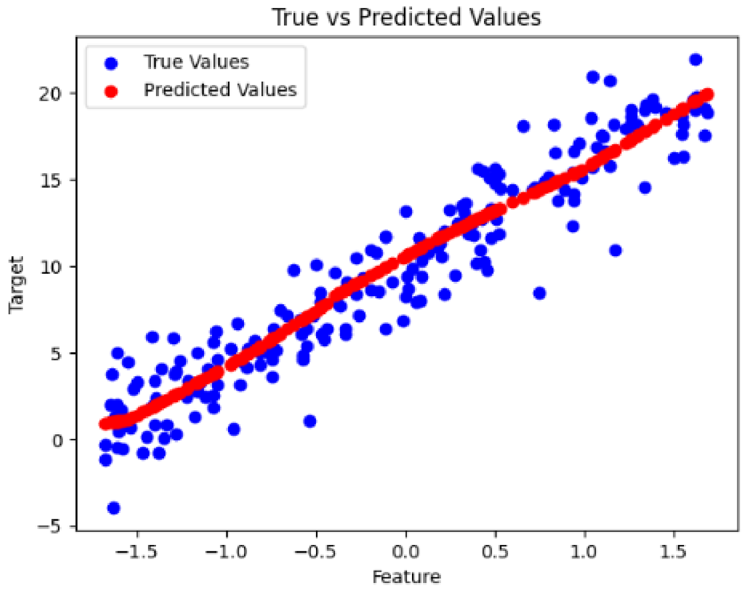
# OUTPUT:

```
    Feature     Target
0  3.745401   7.846204
1  9.507143  16.343597
2  7.319939  15.400275
3  5.986585  13.194341
4  1.560186   4.239954
```

```
Epoch 1/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 143.6730 - val_loss: 128.6559
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 125.6925 - val_loss: 115.8342
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 104.8570 - val_loss: 97.2831
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 93.5006 - val_loss: 73.8218
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 69.3045 - val_loss: 46.1074
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 41.1903 - val_loss: 21.3986
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 17.2995 - val_loss: 8.3409
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 7.8356 - val_loss: 5.9572
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 6.0510 - val_loss: 5.5160
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 5.7683 - val_loss: 5.1246
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 5.6023 - val_loss: 4.8443
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 5.7596 - val_loss: 4.5965
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 4.7886 - val_loss: 4.4421
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 5.0777 - val_loss: 4.3117
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 4.6154 - val_loss: 4.1926
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 4.4176 - val_loss: 4.1209
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 4.2199 - val_loss: 4.0598
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 4.2792 - val_loss: 4.0215
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 4.4246 - val_loss: 4.0034
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.0891 - val_loss: 3.9996
Epoch 21/100
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 3.9068
Test Loss (MSE): 3.5400872230529785
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```




**RESULT:** Thus to implement ANN for an application using regression in python is executed successfully.