EX.NO :08 **DATE :27.09.2024**
Reg.no:220701054

IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION

Regression using Artificial Neural Networks

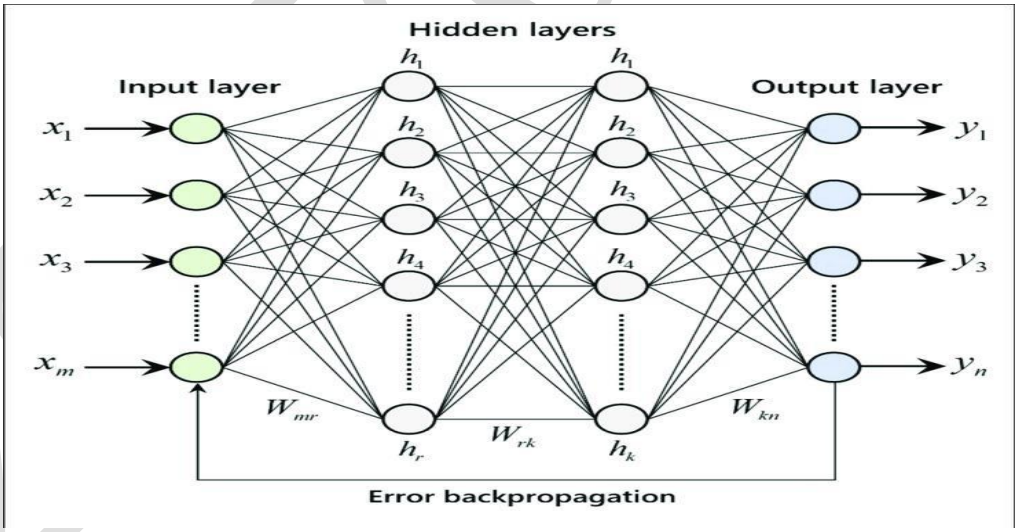
Why do we need to use Artificial Neural Networks for Regression instead of simply using Linear Regression?

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.

1.X_train:- Matrix of features for the training dataset

2.Y_train:- Dependent variable vectors for the training dataset

3.batch_size: how many observations should be there in the batch. Usually, the value for this parameter is 32 but we can experiment with any other value as well.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

# Generating synthetic dataset for demonstration
# Replace this with your own dataset
np.random.seed(42)
X = np.random.rand(1000, 3)  # 1000 samples, 3 features
y = 3 * X[:, 0] + 2 * X[:, 1] ** 2 + 1.5 * np.sin(X[:, 2] * np.pi) + np.random.normal(0, 0.1, 1000)  # Non-linear relationship

# 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)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Building the ANN model
model = Sequential()

# Input layer and first hidden layer with 10 neurons and ReLU activation
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
```

```
# Second hidden layer with 10 neurons and ReLU activation
model.add(Dense(10, activation='relu'))

# Output layer with linear activation for regression
model.add(Dense(1, activation='linear'))

# Compiling the model
model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')

# Training the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)

# Making predictions
y_pred = model.predict(X_test)

# Evaluating the model
mse = np.mean((y_test - y_pred.flatten()) ** 2)
print(f'Mean Squared Error: {mse:.4f}')

# Plotting training history
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

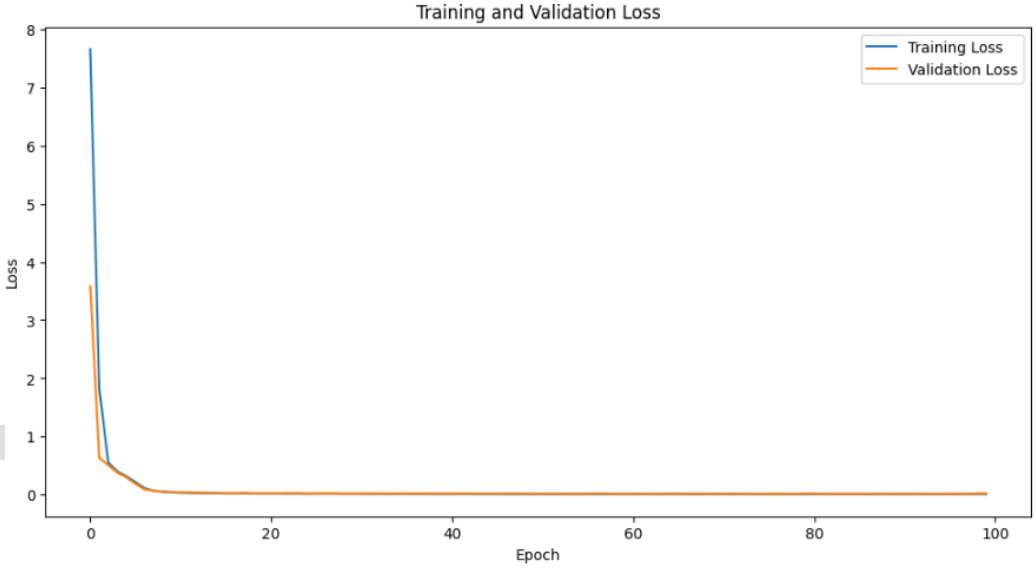
Output:

220701054.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text

```
Epoch 1/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - loss: 9.6805 - val_loss: 3.5762
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 2.4427 - val_loss: 0.6325
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.6179 - val_loss: 0.5068
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.4315 - val_loss: 0.3798
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.3330 - val_loss: 0.2982
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.2338 - val_loss: 0.1869
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1378 - val_loss: 0.0936
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0728 - val_loss: 0.0689
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0601 - val_loss: 0.0555
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0420 - val_loss: 0.0492
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0397 - val_loss: 0.0417
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0328 - val_loss: 0.0395
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0302 - val_loss: 0.0357
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0264 - val_loss: 0.0338
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0263 - val_loss: 0.0324
```



Result:

Thus the implementation of Artificial Neural Networks for an application using python -Regression is executed successfully.