Regression with TensorFlow and Keras

Overview

In this lesson, you will learn how to build and evaluate a neural network for a regression problem using TensorFlow and Keras. We'll work with a CSV file containing housing data, where the goal is to predict the **Price** of a house. The dataset includes the following columns:

- **Avg. Area Income:** Average income of residents in the area.
- **Avg. Area House Age:** Average age of houses in the area.
- **Avg. Area Number of Rooms:** Average number of rooms in the houses.
- **Avg. Area Number of Bedrooms:** Average number of bedrooms.
- **Area Population:** The population of the area.
- **Price:** The target variable representing the house price.

---

1. Understanding the Dataset

Dataset Contents

- **Features:**
  - **Avg. Area Income:** Indicates the economic status of the area.
  - **Avg. Area House Age:** Provides a sense of how old the houses are.
  - **Avg. Area Number of Rooms:** May suggest the overall size or luxury of homes.
  - **Avg. Area Number of Bedrooms:** Complements the number of rooms by focusing on sleeping areas.
  - **Area Population:** Higher populations may affect prices due to demand and other factors.
- **Target:**
  - **Price:** The median value of the house in that area (usually in dollars).

Why Use This Dataset?

This dataset is well-suited for regression because:

- **Multiple Features:** It provides several variables that can influence house prices.
- **Real-World Context:** It simulates real-world scenarios where you might want to predict property values.
- **Learning Opportunity:** It helps you learn data preprocessing (including normalization), model building, training, and evaluation in a regression context.

---

2. Setting Up the Environment

Importing Libraries

We will use Python libraries such as **Pandas** for data manipulation, **Scikit-learn** for splitting the data, and **TensorFlow/Keras** for building the neural network.

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

---

3. Loading and Preparing the Dataset

Step 3.1: Load the CSV File

Assuming your dataset is saved as **housing.csv**, we can load it using Pandas:

```
# Load the dataset from a CSV file
data = pd.read_csv('housing.csv')

# Display the first few rows to understand its structure
print(data.head())
```

Explanation:
Here, we load the CSV file into a Pandas DataFrame so that we can easily explore and manipulate the data.

Step 3.2: Explore and Extract Features and Target

```
# Define the features (input variables) and the target (output variable)
features = ['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of
Rooms',
            'Avg. Area Number of Bedrooms', 'Area Population']
target = 'Price'

# Extract features and target from the DataFrame
X = data[features]
y = data[target]
```

Explanation:
We separate the independent variables (features) from the dependent variable (target) that we want to predict.

Step 3.3: Split the Data into Training and Testing Sets

```
# Split the data into training and testing sets (80% training, 20% testing)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Explanation:
Using an 80/20 split ensures that we train our model on the majority of the data while reserving a portion for evaluating the model's performance on unseen data.

Step 3.4: Normalize the Features

Normalization (or standardization) is important for regression problems because it scales the features so that each has a similar range, which helps the model learn more efficiently.

```
# Calculate the mean and standard deviation from the training data
mean = X_train.mean()
std = X_train.std()

# Standardize the training and testing features
X_train_norm = (X_train - mean) / std
X_test_norm = (X_test - mean) / std
```

Explanation:
By normalizing, we ensure that features with larger scales do not dominate the learning process.

---

4. Building the Neural Network Model

Step 4.1: Define the Model Architecture

For a regression task, we'll use a neural network with a couple of hidden layers and a single neuron in the output layer.

```
# Define the neural network model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train_norm.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)  # Output layer for predicting a continuous value (Price)
])
```

Explanation:

- **Dense Layers:** Fully connected layers that learn the relationships between the features and the target.
- **ReLU Activation:** Introduces non-linearity, allowing the model to learn complex relationships.
- **Output Layer:** Contains one neuron because we are predicting a single continuous value (price).

Step 4.2: Compile the Model

```
# Compile the model with the Adam optimizer, Mean Squared Error loss, and Mean
Absolute Error as a metric
model.compile(optimizer='adam',
              loss='mse',      # Mean Squared Error (MSE) measures the average
squared difference between predictions and true values
              metrics=['mae']) # Mean Absolute Error (MAE) measures the average
absolute error in the original units
```

Explanation:

- **Optimizer (Adam):** Efficiently adjusts the weights during training.
- **Loss (MSE):** The primary function to minimize during training.
- **Metric (MAE):** Provides a direct measure of prediction error.

---

5. Training the Model

Step 5.1: Train the Model

```
# Train the model on the normalized training data, using 20% of the training
data for validation
history = model.fit(X_train_norm, y_train,
                    epochs=100,
                    validation_split=0.2,
                    verbose=1)
```

Explanation:

- **Epochs:** Each epoch represents one complete pass through the training data.
- **Validation Split:** Allows you to monitor the model's performance on data it hasn't seen during training.

---

6. Evaluating the Model

Step 6.1: Evaluate on the Test Data

```
# Evaluate the model on the test set
test_loss, test_mae = model.evaluate(X_test_norm, y_test, verbose=2)
print("\nTest Mean Squared Error (MSE):", test_loss)
print("Test Mean Absolute Error (MAE):", test_mae)
```

Explanation:
This step gives you an estimate of how well the model performs on unseen data by reporting the MSE and MAE.

---

7. Visualizing the Training Process

Step 7.1: Plot Training and Validation Metrics

```
# Plot training & validation loss values (MSE) and MAE values over epochs
plt.figure(figsize=(12, 4))

# Plot Loss (MSE)
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss (MSE)')
plt.plot(history.history['val_loss'], label='Validation Loss (MSE)')
plt.title('Model Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.legend()

# Plot MAE
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Train MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model MAE over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.legend()

plt.show()
```

Explanation:
Visualizing the training process helps you identify whether the model is overfitting (performing much better on training data than on validation data) or underfitting (poor performance on both).

Conclusion

In this lesson, you have learned how to:

- **Understand the Dataset:** Explore a CSV file containing housing data with various features and a target price.
- **Preprocess Data:** Load the data, split it into training and testing sets, and normalize the features.
- **Build and Compile a Model:** Construct a neural network using TensorFlow and Keras, compiling it with appropriate loss and metrics for regression.
- **Train and Evaluate:** Train the model on the training data, evaluate its performance on the test data, and visualize the training process.
- **Error Metrics:** Use Mean Squared Error (MSE) and Mean Absolute Error (MAE) to gauge the model's prediction accuracy.

This hands-on exercise demonstrates the complete process for solving a regression problem using a custom CSV dataset. Feel free to experiment with different model architectures, training durations, or preprocessing techniques to further enhance your understanding.