🌟 Day 14 – Introduction to Neural Networks and TensorFlow Implementation

---

What are Neural Networks?

A **Neural Network (NN)** is a type of machine learning model inspired by how the human brain works. In a neural network, we have layers of interconnected "neurons" (nodes), and each neuron processes information and passes it on to the next layer. These connections have **weights**, which the model adjusts during training in order to learn patterns in the data.

Neural networks are capable of handling complex problems like image classification, speech recognition, and autonomous driving — problems where traditional algorithms struggle.

Difference Between Machine Learning, Neural Networks, and Deep Learning

Understanding how neural networks fit into the broader field of AI is important:

Feature	Machine Learning	Neural Networks	Deep Learning
Definition	Traditional ML algorithms like decision trees, SVMs	A model inspired by the human brain with neurons and layers	Multi-layered neural networks capable of feature extraction and learning
Data Requirement	Can work with small to moderate datasets	Moderate	Requires large datasets
Performance	Good for structured data	Better at capturing patterns	Best for complex data like images, text
Feature Engineering	Manual feature selection needed	Less manual effort	Almost none needed (automatic feature learning)
Examples	Logistic regression, KNN, SVM	Single-layer Perceptron	CNN, RNN, Transformers
Computation	Light	Moderate	Heavy (requires GPU/TPU)

🔧 Implementation: Linear Regression using TensorFlow

Today, we built a **simple neural network model** using TensorFlow to predict the **miles per gallon (mpg)** of a car based on its **horsepower**.

🛠 Step-by-Step Explanation of Code

1 Importing Libraries

```
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

We use TensorFlow for neural networks, pandas for data handling, and seaborn/Matplotlib for visualization.

2️⃣ Loading and Preprocessing the Dataset

```
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-
mpg/auto-mpg.data"
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower',
'weight','acceleration', 'model_year', 'origin', 'car_name']
raw_dataset = pd.read_csv(url, names=column_names,na_values='?',
comment='\t', sep=' ', skipinitialspace=True)
```

We load the **Auto MPG dataset** from UCI repository and assign proper column names. We also handle missing values.

3️⃣ Feature Selection and Normalization

```
dataset = raw_dataset.copy()
dataset = dataset.dropna()

X = dataset[['horsepower']].astype('float32')
y = dataset[['mpg']].astype('float32')

# Normalize horsepower
X = (X - X.mean()) / X.std()
```

- Only **horsepower** is used as input.
- Data is **normalized** to help the model converge faster and perform better.

4️⃣ Splitting Data for Training and Testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

We split the data into 80% training and 20% testing using `train_test_split()`.

5️⃣ Defining the Model

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=[1])
])
```

- This is a **Sequential model** with **1 input** and **1 output neuron**
- It's equivalent to a **simple linear regression** in structure

---

6️⃣ Compiling and Training the Model

```
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
              loss='mean_squared_error')
history = model.fit(X_train, y_train, epochs=100, verbose=0)
```

- We use **Stochastic Gradient Descent (SGD)** with learning rate `0.01`
- Model is trained for `100 epochs`

7️⃣ Evaluating the Model

```
loss = model.evaluate(X_test, y_test)
print(f"\nTest Loss: {loss:.4f}")
```

We calculate **Mean Squared Error (MSE)** on test data.

8️⃣ Viewing Learned Parameters

```
weights = model.get_weights()
print(f"\nLearned weight: {weights[0][0][0]:.4f}, bias:
{weights[1][0]:.4f}")
```

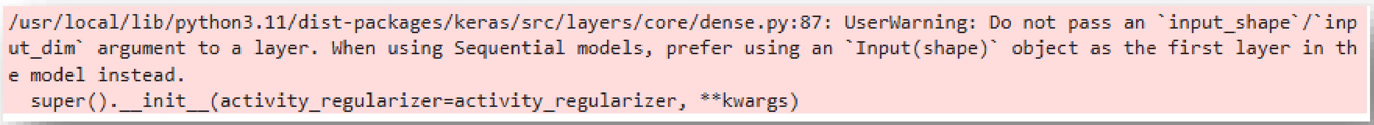
This gives us the model’s learned **slope** and **intercept**.

9️⃣ Making Predictions and Plotting

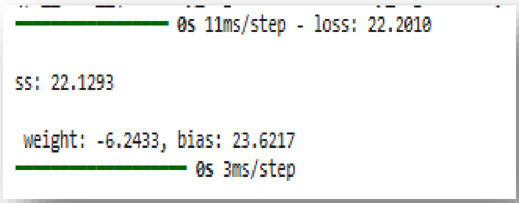
```
y_pred = model.predict(X).flatten()

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X['horsepower'], y=y['mpg'], label="Actual Data")
sns.lineplot(x=X['horsepower'], y=y_pred, color='red', label="Predicted
Line")
plt.xlabel('Normalized Horsepower')
plt.ylabel('MPG')
plt.title('Linear Regression using TensorFlow')
plt.grid(True)
plt.legend()
plt.show()
```

📈 This shows how well the predicted values align with the actual data using a line fit.

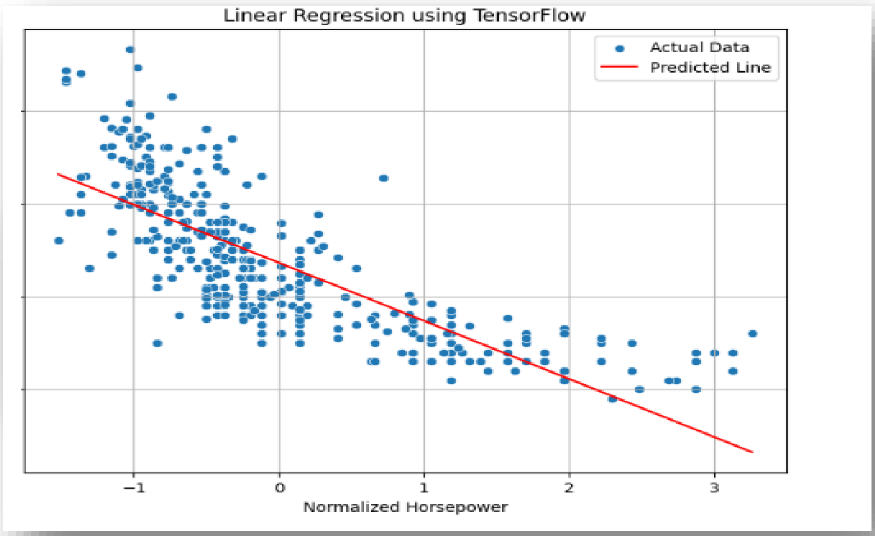
📌 OUTPUTS :



```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`inp
ut_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in th
e model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



```
━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 22.2010

ss: 22.1293

 weight: -6.2433, bias: 23.6217
━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```



📌 Summary

- Understood the **basics of Neural Networks**
- Implemented a **single-layer neural network** using TensorFlow
- Applied it to the **Auto MPG dataset**
- Visualized and evaluated the predictions