Carnegie Mellon Unversity
17-644: Applied Deep Learning
Assignment#1 Report

A. Sklearn Traditional Linear Regression Model vs. TensorFlow Single Neuron Model

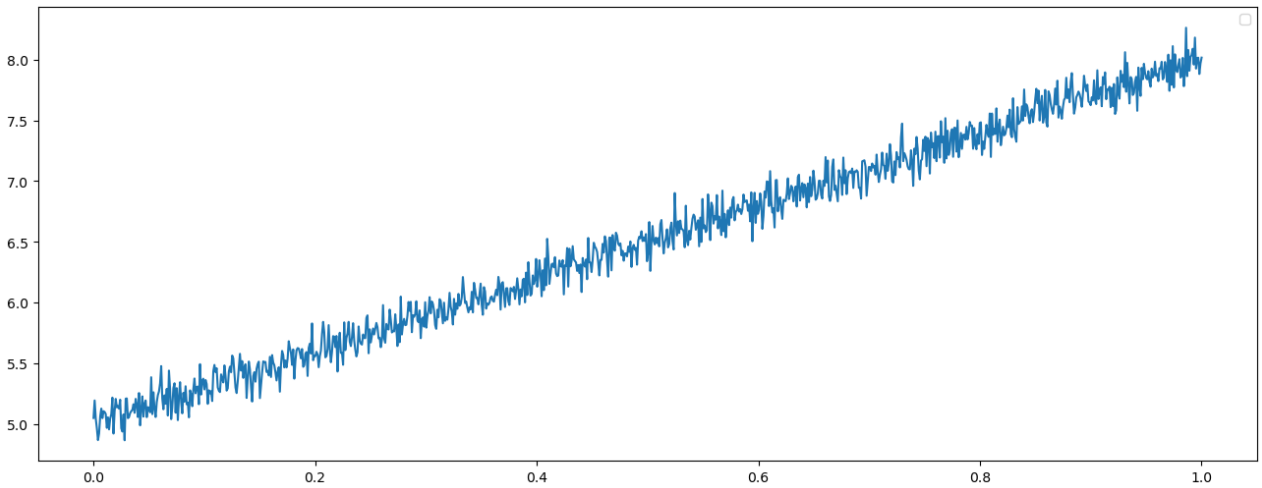
Since it's the first assignment of the course, I started with creating a virtual conda environment called "applieddeeplearning" and then installing required packages as below:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from tensorflow import keras
from tensorflow.keras import layers
```

Then, I created random x values and the y linear regression function with my selected a and b values as such:

```
x = np.random.rand(1000, 1)
a = 3
b = 5
noise = np.random.normal(0, 0.1, size=(1000, 1))
y = a * x + b + noise
```

I decided to add normal noise since it's the most common noise type I could imagine to add. The noise line plotted is below:



Then performed sklearn linear regression and TensorFlow single neural network respectively. Since I had only one neuron, it wasn't a network but a single, linear regression function node. The resulting plots with R^2 values, weight, bias and coefficients are plotted below:

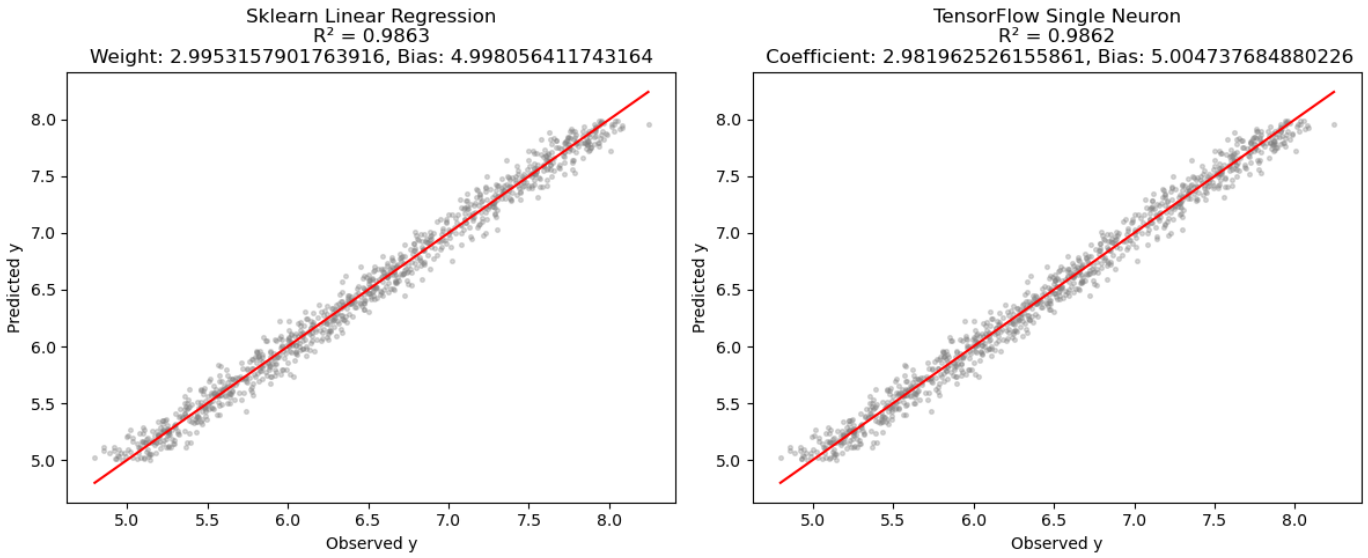


Figure 1 *sklearn traditional linear regression model vs. TensorFlow single neuron model.*

It wasn't surprising to see TensorFlow performed very close to the traditional sklearn since we didn't use the capacity of deep learning but instead only used one neuron, therefore, launched almost a linear regression algorithm, although in a relatively longer timeframe. As discussed in the lecture, in situations like this, where our dataset isn't that complex, a simple linear regression model could work better than a complicated-look deep learning model.

One important step I had to ensure the TensorFlow code reached the accuracy of the simple linear regression model was to set up the epochs correctly. For this, I had to adjust the epoch value from 200 to 400 as shown in the below code snippet and the plots:

```
tf_model = keras.Sequential([layers.Dense(units=1, input_shape=[1])])
tf_model.compile(optimizer='adam', loss='mean_squared_error')
tf_model.fit(x, y, epochs=400, verbose=0) # played with epochs. From 200 to 400
y_pred = tf_model.predict(x)
r2_tf = r2_score(y, y_pred)
print(f"tensorflow r2_tf: {r2_tf}")
```

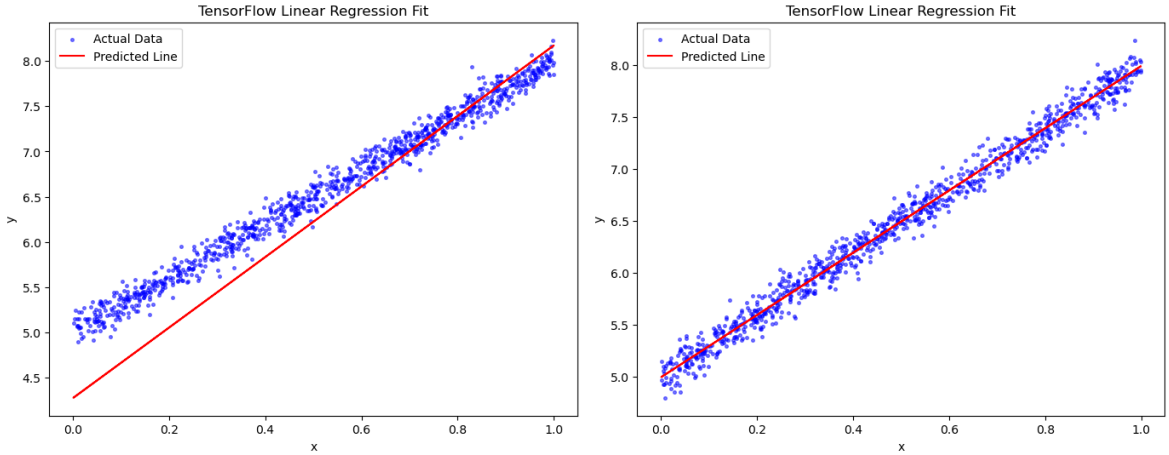


Figure 2 *TensorFlow model with epochs = 200 (on the left) and epochs = 400 (on the right)*

B. Appendix

The whole code I used to prepare and train the models:

```
x = np.random.rand(1000, 1)
a = 3
b = 5
noise = np.random.normal(0, 0.1, size=(1000, 1))
y = a * x + b + noise
```

```
sk_model = LinearRegression()
sk_model.fit(x, y)
y_pred = sk_model.predict(x)
r2_sk = r2_score(y, y_pred)
print(f"sklearn r2_sk: {r2_sk}")

plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='blue', label='Actual Data', alpha=0.5, s=7)
plt.plot(x, y_pred, color='red', label='Predicted Line')
plt.title('sklearn Linear Regression Fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

print(f"Coefficient: {sk_model.coef_[0][0]}, Bias: {sk_model.intercept_[0]}")
```

```
tf_model = keras.Sequential([layers.Dense(units=1, input_shape=[1])])
tf_model.compile(optimizer='adam', loss='mean_squared_error')
tf_model.fit(x, y, epochs=400, verbose=0) # play with epochs
y_pred = tf_model.predict(x)
r2_tf = r2_score(y, y_pred)
print(f"tensorflow r2_tf: {r2_tf}")

plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='blue', label='Actual Data', alpha=0.5, s=7)
plt.plot(x, y_pred, color='red', label='Predicted Line')
plt.title('TensorFlow Linear Regression Fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

tf_weight, tf_bias = tf_model.layers[0].get_weights()
print(f"Weight: {tf_weight[0][0]}, Bias: {tf_bias[0]}")
```