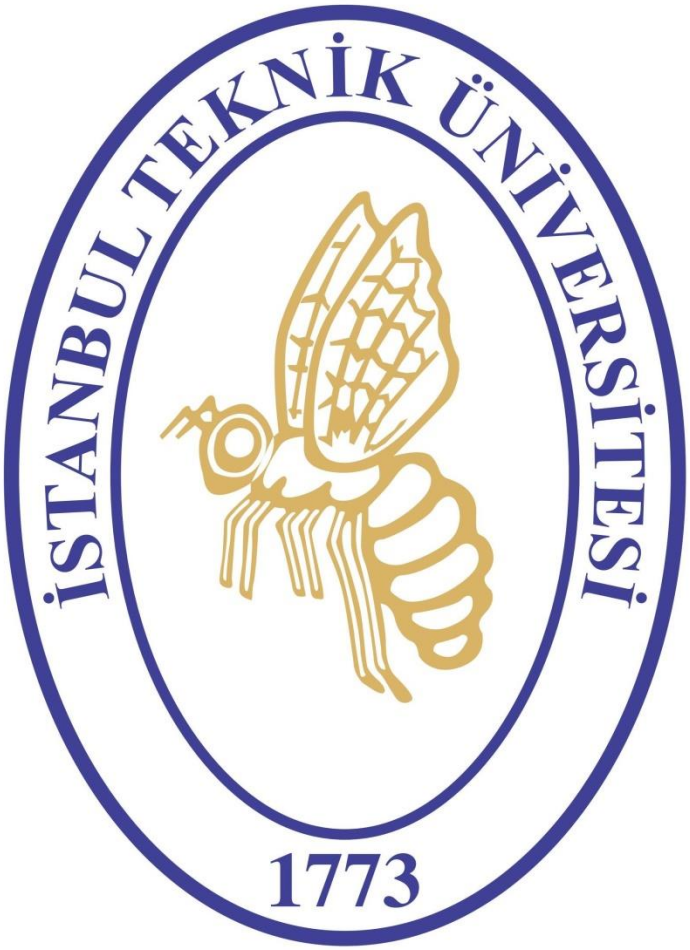
EHB420E Artificial Neural Networks

Final Exam Report

Ataberk Demirkaya

040200252



The Github repository of this project:

https://github.com/AtaberkDemirkaya/Artificial-Neural-Network-Final-Project

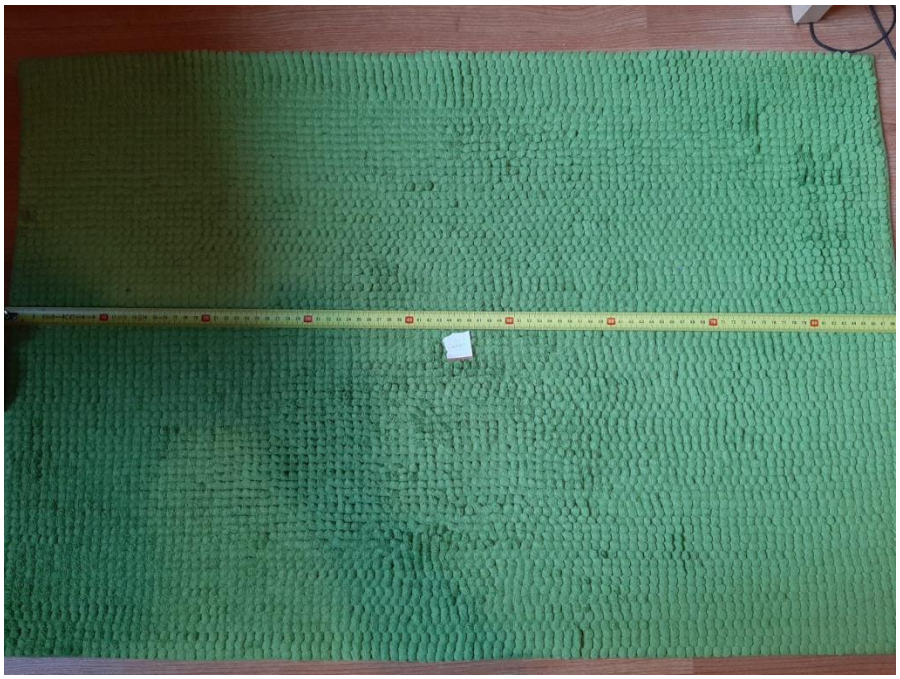
In this report, I am going to talk about my process and experience whilst working on this homework. There were several challenging parts to accomplish in this assignment as a Python beginner myself. For example, data collection was the most challenging part in my opinion since manually gathering data by a minimum of 100 iterations is quite time consuming due to the fact that I am performing measurements and entering the data to my excel sheet.

1. Data Collection and Labeling:

Here are the coins I used for data gathering:

As you may realize, those coins are put in order like heads-tails-heads-tails and so on. I chose to structure my data in this shape in my excel sheet. Therefore I will be dropping the coins following that specific order.

The surface that I performed my operations on (my rug):



I preferred to do the dropping operation on my rug for the sole reason to protect my parquet. Since the coins are made of metal and they were going to be dropped from a meter, I figured that using a soft surface such as my rug would be smart for damage prevention.

As you can see on top of my rug, there is a meter and a piece of paper. In this picture, I used the meter for determining the center of the surface which I will be considering as my origin point from now on. (For your consideration, the rug is 88 to 60 cm's which sets the origin at (44, 30))

Setup of the system:

The idea behind it is pretty simple. I grabbed a meter and fixed it at 100 centimeters so I can drop my coins consistently. In the image, I am holding 9 of the 10 coins I showed in the very first image of this report. One of them is in the ground which is the very first data I obtained.

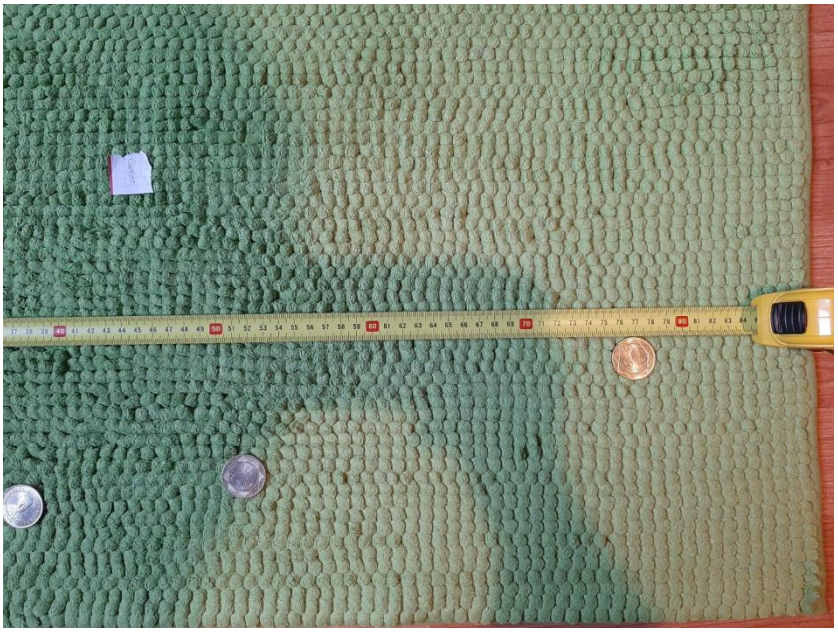
As you can see my meter is connected with my surface where I marked my origin with that small piece of paper. Therefore I can ensure that the initial contact with the surface is always done through the origin precisely.

I keep dropping those coins until I don't have any more of them and simultaneously entering my data to the excel sheet. I found this method to be effective and time efficient rather than dropping the coin one by one than measuring it.

An example result of the experiment:

Unfortunately, my camera quality is horrible and that is one of the main reasons why I chose to do this task my manually entering the data. As you can see, it is not clear whether the coin is tails or heads in the end although the position may be relatively deducible, it would be hard to distinguish the paper which marks my origin from the coins.

Gathering the Data:



As you can see I am measuring its x coordinate in this picture. It is measured as 76.75 cm.

Then I subtract its value from the origin x coordinate which is x=44. We obtain the result of 76.75 – 44 = +32.25 cm for the x coordinate.

Although it is not seen here, I perform the same operation for the y coordinate as well. Then have my final data point written to my excel sheet.

The Excel Sheet:

	A	B	C	D	E
1	Initial state of the coin	X coordinate	Y coordinate	Distance to release point	Final state of the coin
2	Y	4	-2,25	4,589389938	Y
3	T	-20,75	-2,25	20,87163146	Y
4	Y	27,5	-3,5	27,72183255	Y
5	T	-15,75	-4	16,25	T
6	Y	-34,25	27,5	43,92393994	T
7	T	32,75	13	35,23581275	Y
8	Y	9,5	20	22,14158983	T
9	T	-43,75	5	44,03478738	T
10	Y	-6,75	-22	23,01222501	T
11	T	8,5	-20,25	21,96161424	Y
12	Y	-4,25	-12,5	13,20274593	T
13	T	-7,75	-20,25	21,68236611	T
14	Y	-7,25	18,5	19,86988928	T
15	T	-8	-10,5	13,20037878	T
16	Y	-0,5	5,75	5,77169819	T
17	T	-9	-7,25	11,55692433	T
18	Y	15,75	3,75	16,19027486	T
19	T	3,75	21,75	22,07090845	T
20	Y	35,25	12,5	37,40070187	T

Here are the first 20 rows from my data sheet. As you can see I have the initial state of the coin (the face facing upwards during the throw), X and Y coordinates of the landing (calculated with respect to the origin point), absolute distance to the origin and finally the face facing upwards after the landing.

2. Data Visualization:

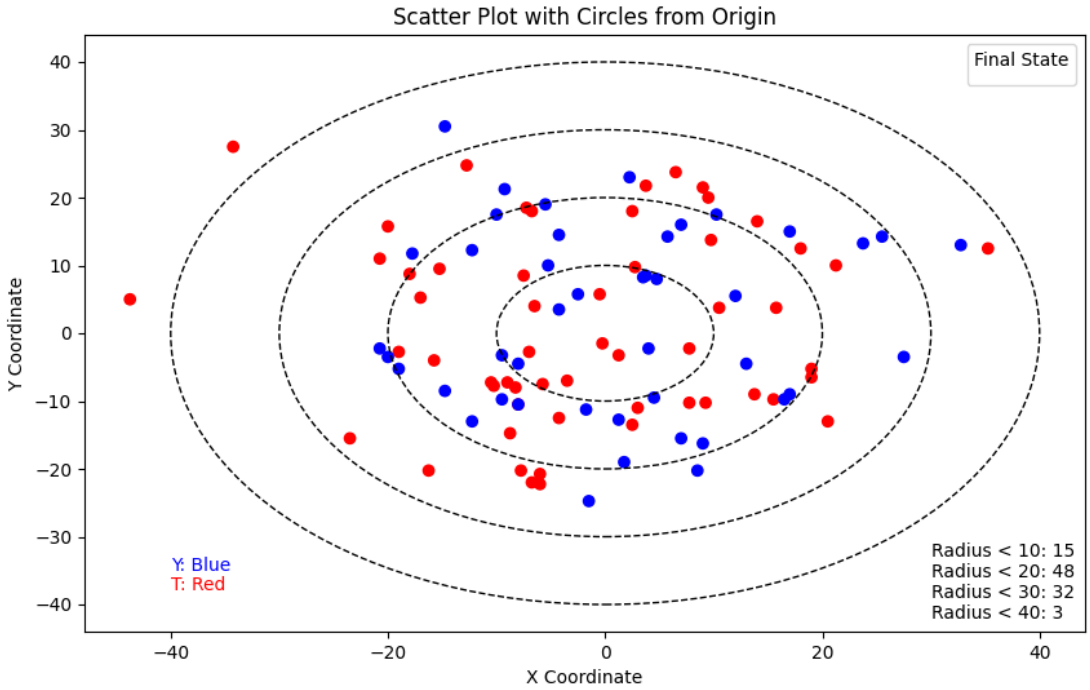
This is my code for visualizing the data, you can check it in person at my Github.

```
import pandas as pd
import matplotlib.pyplot as plt

file_path = r'C:\Users\Demirkaya\Desktop\ANN Final\myData.xlsx'
df = pd.read_excel(file_path)

radius_ranges = [10, 20, 30, 40]
counts = []

for i in range(len(radius_ranges)):
    if i == 0:
        counts.append(len(df[df[df.columns[3]] < radius_ranges[i]]))
    else:
        counts.append(len(df[(radius_ranges[i - 1] <= df[df.columns[3]]) & (df[df.columns[3]] < radius_ranges[i])]))

plt.figure(figsize=(10, 6))
colors = {'Y': 'blue', 'T': 'red'}
scatter = plt.scatter(df[df.columns[1]], df[df.columns[2]], c=df[df.columns[4]].map(colors), label=df[df.columns[4]])

for radius in radius_ranges:
    circle = plt.Circle((0, 0), radius, fill=False, color='black', linestyle='dashed', linewidth=1)
    plt.gca().add_patch(circle)

plt.xlabel('X Coordinate')
plt.ylabel('Y Coordinate')
plt.title('Scatter Plot with Circles from Origin')

handles, labels = scatter.legend_elements()
legend_labels = [colors[label] for label in labels]
plt.legend(handles, labels, title='Final State', labels=legend_labels)

plt.text(-40, -35, 'Y: Blue', color='blue')
plt.text(-40, -38, 'T: Red', color='red')

for i in range(len(radius_ranges)):
    plt.text(30, -33 - 3*i, f'Radius < {radius_ranges[i]}: {counts[i]}', color='black', fontsize=10)

plt.show()
```

To briefly explain what it does, it reads the data from my excel sheet and stores it inside a data frame (df). From here on I am able to manipulate my data through code. For example I created a Cartesian graph to visualize my data points. I assigned labels for the final states of my data and drew circles on the graph with radiuses of 10, 20, 30 and 40. This is done so it can be better understood that where in my surface the coins end up more frequently compared to other intervals of distances.

It uses pandas for the creation of the data frame and matplotlib for the creation of the graph itself. More details about the graph and its image are below.

The Graph:



Above is my graph. As you can observe, heads (Y) is painted blue and tails (T) is painted red for recognition. I also individually counted the distance intervals with the increments of 10s which you can see in the right down corner. Apparently, %80 of the coins is collectively landed between 10-30cm with 20-30cm having the highest density of %48.

3. Data preprocessing and neural network training:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the dataset without assuming the first row as headers
file_path = r'C:\Users\Demirkaya\Desktop\ANN Final\myData.xlsx'  # Replace with the actual path to your Excel file
df = pd.read_excel(file_path, header=None)

# Assign original column names
df.columns = df.iloc[0]

# Drop the first row (which contains the original column names) after assigning them
df = df.drop(0)

# Impute missing values with mean for numerical columns
imputer = SimpleImputer(strategy='mean')
df[df.columns[1:4]] = imputer.fit_transform(df[df.columns[1:4]])

# Shuffle the dataset
df = df.sample(frac=1).reset_index(drop=True)

# Convert categorical variables to numerical representation using Label Encoding
le = LabelEncoder()
df[df.columns[4]] = le.fit_transform(df[df.columns[4]])
```

```
# 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)

# Display the first few rows of the preprocessed data
print("Preprocessed Data:")
print(df)
```

Above is my code for data preprocessing, what I do here basically is restructuring the data for processing initially. Then I replace the char values (heads and tails) in my excel sheet with 0 and 1 for preparation of the neural network. Finally, I split my data into test and training data then print the preprocessed data. The image of it is below.

```
Preprocessed Data:
0  Initial state of the coin  ...  Final state of the coin
0                          T  ...                        1
1                          T  ...                        0
2                          T  ...                        1
3                          Y  ...                        0
4                          T  ...                        1
..                       ...  ...                      ...
95                         T  ...                        1
96                         Y  ...                        0
97                         Y  ...                        0
98                         T  ...                        1
99                         Y  ...                        1

[100 rows x 5 columns]
```

As you can see “Final State of the Coin” values are transitioned to binary values for the network to process it in numerical values.

Neural Network Archetype:

```
# Define the neural network model
model = Sequential()
model.add(Dense(64, input_dim=3, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy * 100:.2f}%')

# Make predictions on the test set
predictions = model.predict(X_test)

# Convert the predictions to binary labels
binary_predictions = (predictions > 0.5).astype(int)

# Calculate accuracy on the test set
accuracy_test = accuracy_score(y_test, binary_predictions)
print(f'Accuracy on Test Set: {accuracy_test * 100:.2f}%')
```

Above is my code for the neural network training of my data. In this code segment, a neural network model is constructed using the Keras library to predict the final state of a coin based on its initial state, X-coordinate, and Y-coordinate. The model consists of three layers: two densely connected layers with rectified linear unit (ReLU) activation functions and a final output layer with a sigmoid activation function for binary classification. The model is compiled using the Adam optimizer and binary crossentropy loss, and then trained on a dataset divided into training and testing sets. The training progress is monitored with a validation set, and after 50 epochs, the model is evaluated on the test set, providing metrics such as loss and accuracy. Predictions are made on the test set, and their accuracy is further verified using scikit-learn's accuracy_score function. The results showcase the effectiveness of the neural network in accurately predicting the final state of the coin.

Here is my final accuracy result concerning my final results:

```
Epoch 48/50
3/3 [==============================] - 0s 16ms/step - loss: 0.6217 - accuracy: 0.6125 - val_loss: 0.6701 - val_accuracy: 0.5500
Epoch 49/50
3/3 [==============================] - 0s 17ms/step - loss: 0.6184 - accuracy: 0.6500 - val_loss: 0.6636 - val_accuracy: 0.5500
Epoch 50/50
3/3 [==============================] - 0s 18ms/step - loss: 0.6141 - accuracy: 0.6250 - val_loss: 0.6708 - val_accuracy: 0.6500
1/1 [==============================] - 0s 32ms/step - loss: 0.6708 - accuracy: 0.6500
Test Loss: 0.6708176732063293, Test Accuracy: 65.00%
1/1 [==============================] - 0s 142ms/step
Accuracy on Test Set: 65.00%

Process finished with exit code 0
```

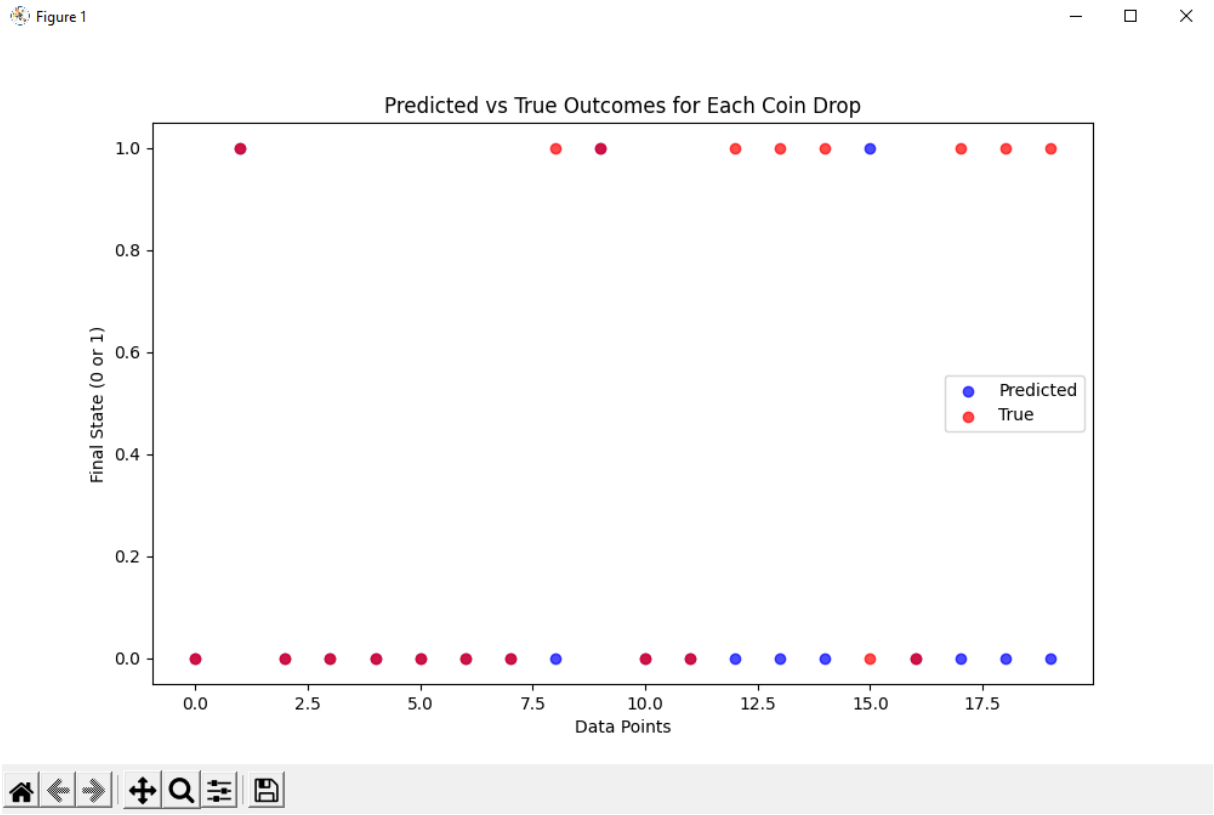
After investing quite a bit of time tweaking and testing, I managed to achieve a test set accuracy of 0.65. It was a bit of a journey, trying out different neural network setups and configurations. While the process took a few hours, reaching this accuracy milestone feels like a significant accomplishment. It shows that the model is doing a decent job predicting the final state of the coin based on its starting conditions and coordinates. Despite the iterative nature of the work and the challenges encountered, hitting 0.65 accuracy is a positive outcome and reflects the dedication to improving the model's predictive abilities.

4. Data Prediction:

Below is my code for data prediction:

```python
# Create a figure and axis
fig, ax = plt.subplots(figsize=(10, 6))

# Plot the predicted outcomes
ax.scatter(range(len(y_test)), binary_predictions, color='blue', label='Predicted', alpha=0.7)

# Plot the true outcomes
ax.scatter(range(len(y_test)), y_test, color='red', label='True', alpha=0.7)

# Set plot labels and title
ax.set_xlabel('Data Points')
ax.set_ylabel('Final State (0 or 1)')
ax.set_title('Predicted vs True Outcomes for Each Coin Drop')

# Show legend
ax.legend()

# Show the plot
plt.show()
```

And my graph (visualization) for comparison of predicted data points and true data points:



5. Conclusion:

- Overall this final project was challenging but also very informative on my behalf.
- Data gathering was physically very challenging for my legs as I had to do a minimum of 100 squats for measuring and data inputting to my excel sheet.
- The mentally challenging part was studying code and neural network archetypes. I had gone over several types however I am happy in my final decision. Although there may be better ones, I am pretty happy with the results overall especially when the effort I put in is taken into consideration.