Intro to Data Science - Assignment 4

Sherwyn Braganza | Isaac McClanahan

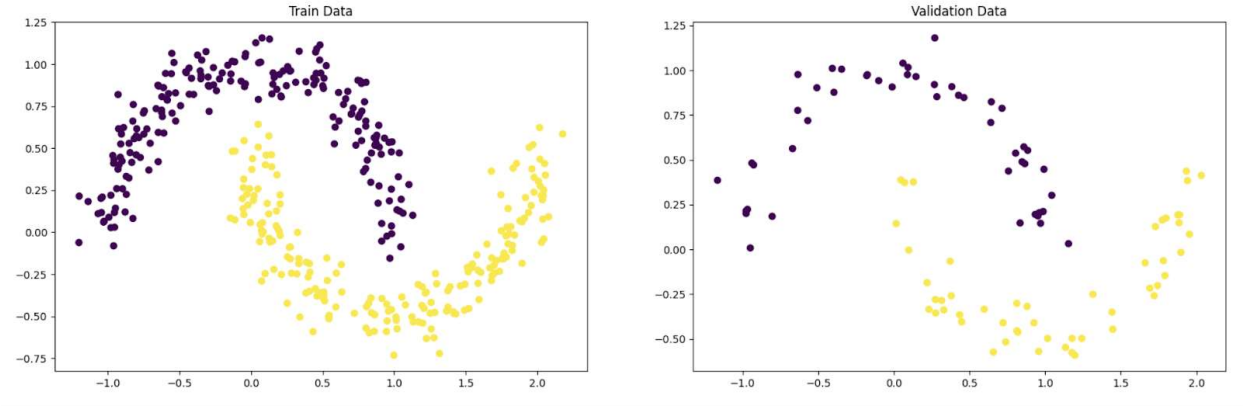
The aim of this project was to showcase our understanding of the Sequential and Functional API of Keras in implementing a deep neural network. The dataset under consideration was Make Moons from the scikit-learn library. 500 samples were chosen from the dataset based on the random seed - 8. The code that implemented this is shown below.

```
from sklearn import datasets
import matplotlib.pyplot as plt

# picking out random samples with noise factor of 0.1
features, targets = datasets.make_moons(n_samples=500, shuffle=True, noise=0.1, random_state=8)
```

From that, 20% of the samples were held back to act as the validation dataset, while the remaining 80% would act as the training data. The code that implements this is shown below along with a graphical visualization of the data.

```
fig, axs = plt.subplots(1,2)
axs[0].scatter(x_train[:,0], x_train[:,1], c=y_train)
axs[0].set_title('Train Data')
axs[1].scatter(x_val[:,0], x_val[:,1], c=y_val)
axs[1].set_title('Validation Data')
fig.set_figwidth(20)
fig.set_figheight(6)
fig.show()
```

```
/var/folders/m2/6gn8k9tn0rz56ryvkbgdrwsr0000gn/T/ipykernel_28552/3112608867.py:8: UserWarning: Matplotlib is curren
tly using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  fig.show()
```



Sequential API

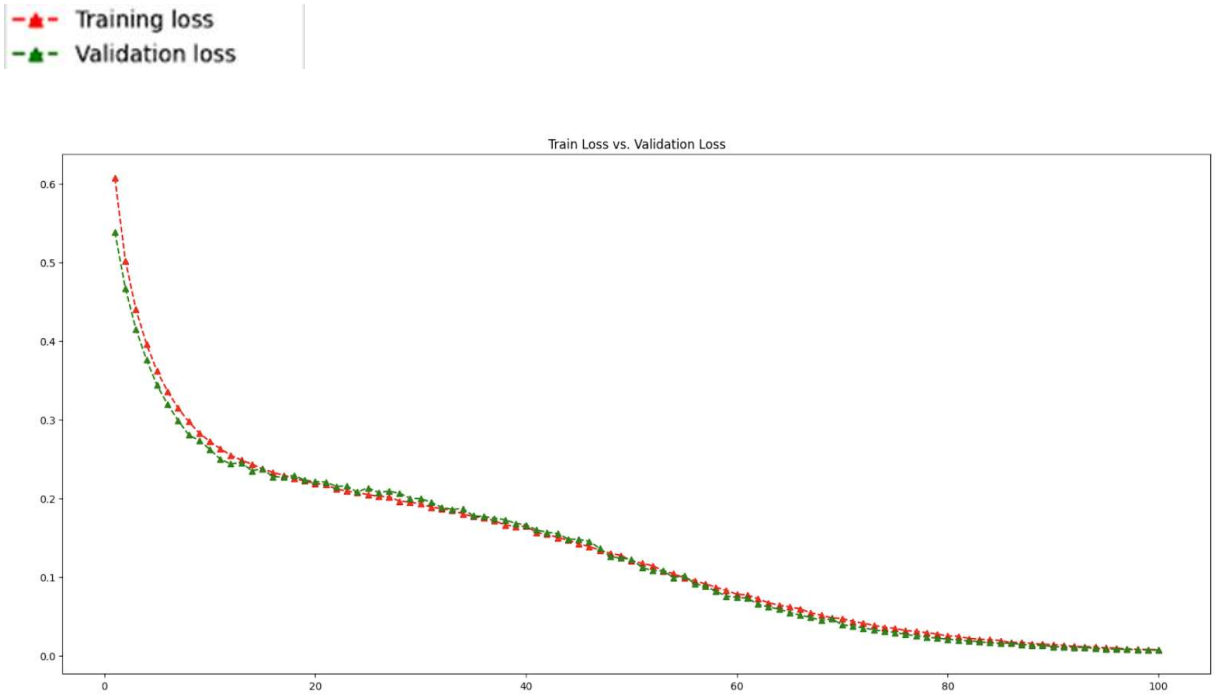
In this section, we implemented a two inner-layer model with 64 and 32 densely connected nodes. The first layer had a **ReLU** activation function, while the second, a **tanH** activation function. The output layer consisted of one node with a **Sigmoid** activation function.

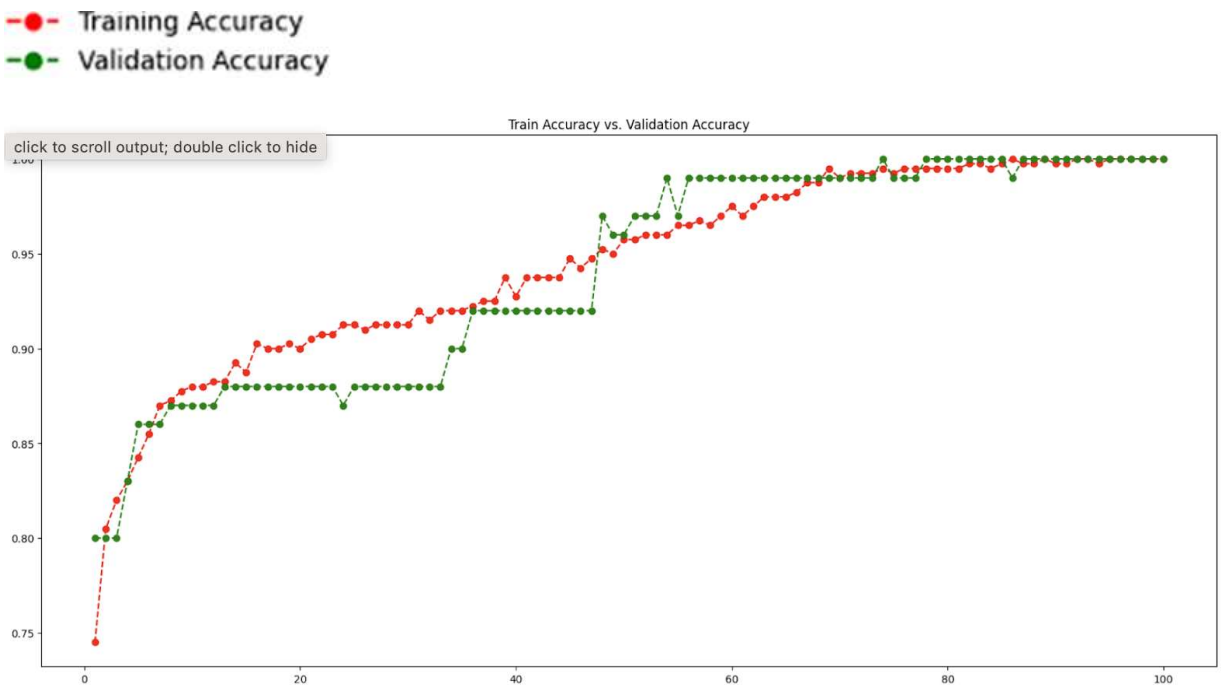
The other parameters of the model included a **Root Mean Squared Propagation** optimizer, a **Binary Cross Entropy** loss function and the ratio of the number of rights to wrongs the model got as a metric.

The model was trained for **100 epochs**, with a **batch size of 50**. The code that implements this is shown below.

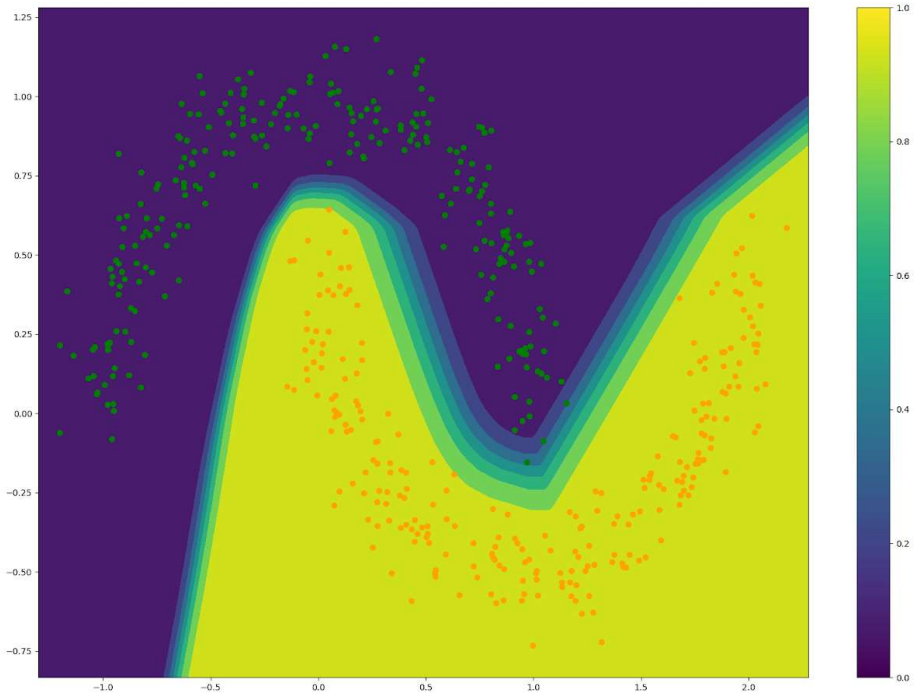
```
from tensorflow import keras
from tensorflow.keras import layers

model_sequential = keras.Sequential(name="my_example_model")
model_sequential.add(layers.Dense(64, activation="relu", name="my_first_layer"))
model_sequential.add(layers.Dense(32, activation="tanh", name="my_second_layer"))
model_sequential.add(layers.Dense(1, activation="sigmoid", name="my_last_layer"))
model_sequential.build(input_shape=(None, 2))
model_sequential.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
model_sequential.fit(x_train, y_train, epochs=100, batch_size=50, validation_data=(x_val, y_val))
```

The following figure shows visual representations of the losses and accuracies across epochs.





The results of the model are shown in the image below. The code that implements this can be found in **Appendix I.**

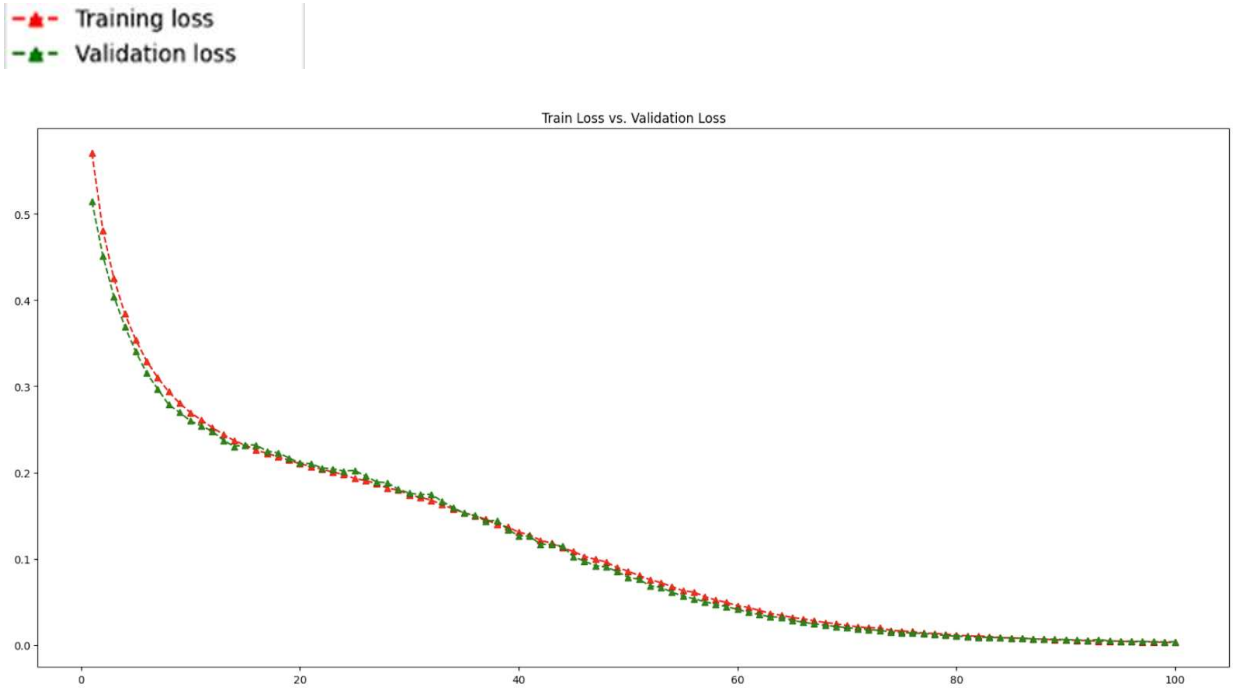


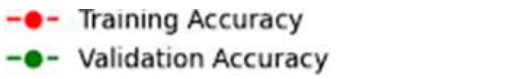
Functional API

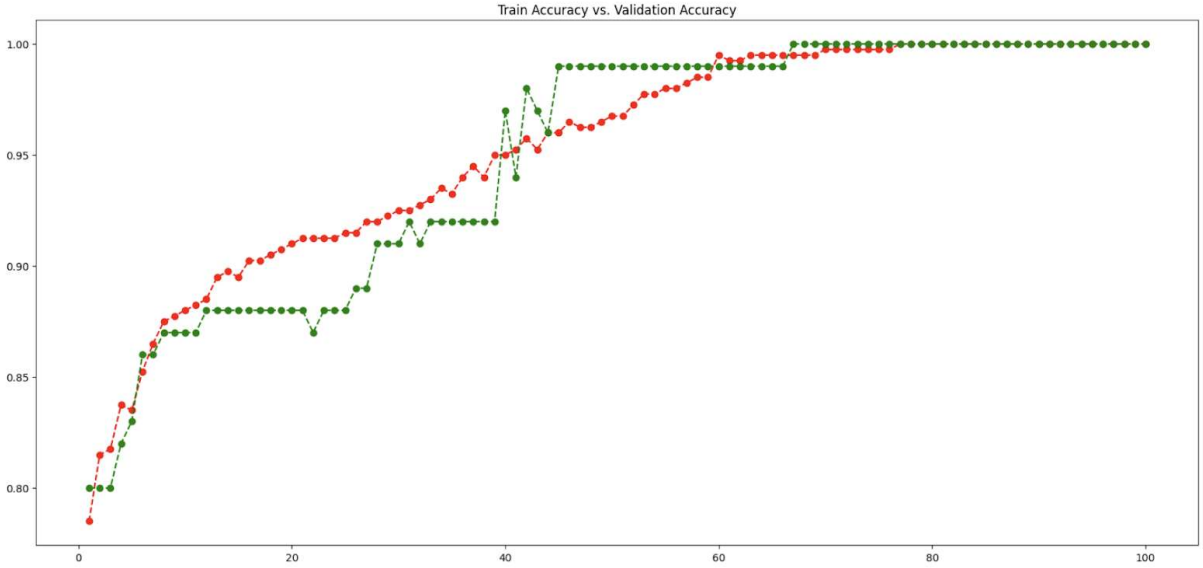
The exact same model implemented in the Sequential API was implemented using Keras' Functional API. The code that achieves this is shown below.

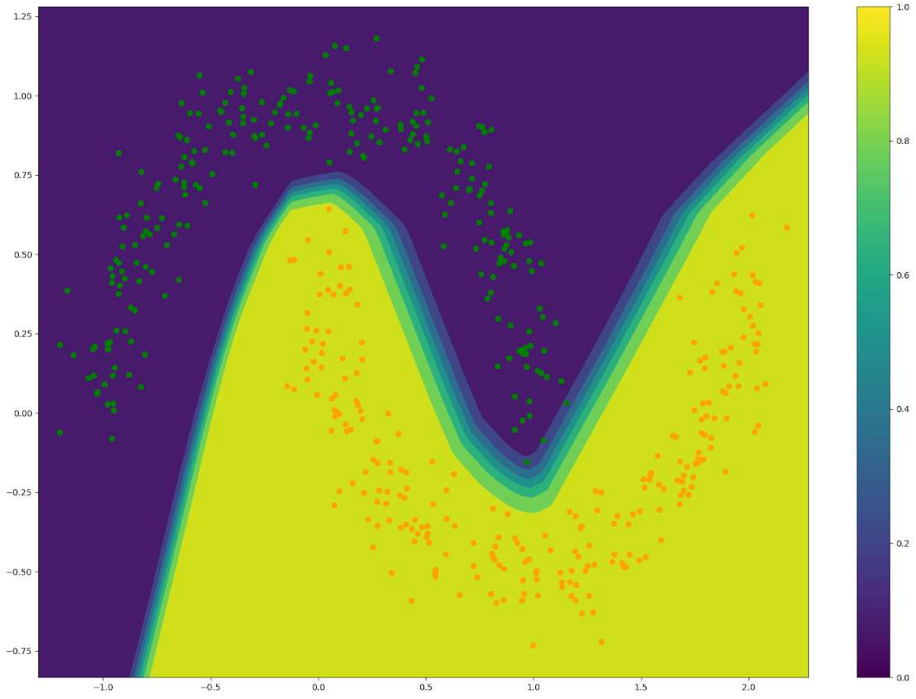
```
inputs = keras.Input(shape=(2,), name='functional_input')
feature = layers.Dense(64, activation="relu")(inputs)
feature = layers.Dense(32, activation='tanh')(feature)
outputs = layers.Dense(1, activation="sigmoid")(feature)
model_functional = keras.Model(inputs=inputs, outputs=outputs)
model_functional.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
model_functional.fit(x_train, y_train, epochs=100, batch_size=50, validation_data=(x_val, y_val))
functional_history = model_functional.history.history
plotMetrics(functional_history)
```

The corresponding loss metrics and results of the model are shown the following figures. The code that generates these plot metrics can be found in **Appendix II.**




Training Accuracy
Validation Accuracy


Train Accuracy vs. Validation Accuracy
1.00
0.95
0.90
0.85
0.80
0
20
40
60
80
100


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1.0
0.8
0.6
0.4
0.2
0.0

Appendix I

```
import numpy as np

def addColor( values ):
    colors = []
    for i in values:
        if i < 0.5:
            colors.append("green")
        else:
            colors.append("orange")
    return colors


res = 500

plt.rcParams["figure.figsize"] = (20, 14)

# Make meshgrid of xy points
xx, yy = np.meshgrid(np.linspace(xMin - 0.1, xMax + 0.1, res),
                     np.linspace(yMin - 0.1, yMax + 0.1, res), indexing="xy")
xy = np.dstack((xx, yy))

# Make meshgrid of equal size containing predictions
zz = np.ndarray((res, res))
for i in range(0, res):
    zz[i] = model_sequential(xy[i])[:, 0]

# Graph decision regions
plt.contourf(xx, yy, zz, levels=8)

# predictions_sequential = np.round(model_sequential.predict(x_val))
xx, yy= np.meshgrid(features[:,0], features[:,1], sparse=True)
plt.scatter(xx, yy, c=addColor( model_sequential.predict(features) ) )
plt.colorbar()
plt.savefig('sequential_contour')
plt.show()
```

Appendix II

```
def plotMetrics(history:dict)->None:
    epochs = range(1, len(history['loss']) + 1)

    fig, axs = plt.subplots(2)
    axs[0].plot(epochs, history['loss'],
                label="Training loss", color = 'red', linestyle='--', marker = '^')
    axs[0].plot(epochs, history['val_loss'],
                label="Validation loss", color = 'green', linestyle='--', marker = '^')
    axs[0].set_title('Train Loss vs. Validation Loss')
    axs[1].plot(epochs, history['accuracy'],
                label="Training Accuracy", color = 'red', linestyle='--', marker = 'o')
    axs[1].plot(epochs, history['val_accuracy'],
                label="Validation Accuracy", color = 'green', linestyle='--', marker = 'o')
    axs[1].set_title('Train Accuracy vs. Validation Accuracy')
    fig.set_figwidth(20)
    fig.set_figheight(20)
    fig.legend()
    fig.show()

sequential_history = model_sequential.history.history
plotMetrics(sequential_history)
```