https://github.com/ahaffne2/64061_ahaffne2/blob/eb9cd654d063b38708ce2e91fd640aad98738c2e/Assignment_1_IMBD.ipynb

1. Table of Results

Method	Test Loss	Test Accuracy
Original	0.2917	0.8872
One Hidden Layer	0.2810	0.8875
3 Hidden Layers	0.2979	0.8835
32 Units	0.3419	0.8704
64 Units	.3284	.8759
8 Units	.2866	.8853
MSE	.0856	0.8844
Tahn Results	.0907	.8804
Regularization	.0.2944	.8845

2. Hidden Layers

a. Original Results

```
[27] model = keras.Sequential([
         layers.Dense(16, activation="relu"),
         layers.Dense(16, activation="relu"),
         layers.Dense(1, activation="sigmoid")
     ])
     model.compile(optimizer="rmsprop",
                   loss="binary_crossentropy",
                   metrics=["accuracy"])
     model.fit(x_train, y_train, epochs=4, batch_size=512)
     results = model.evaluate(x_test, y_test)

     Epoch 1/4
     49/49 [==============================] - 2s 27ms/step - loss: 0.5120 - accuracy: 0.7934
     Epoch 2/4
     49/49 [==============================] - 2s 32ms/step - loss: 0.3030 - accuracy: 0.9056
     Epoch 3/4
     49/49 [==============================] - 1s 30ms/step - loss: 0.2188 - accuracy: 0.9283
     Epoch 4/4
     49/49 [==============================] - 1s 29ms/step - loss: 0.1779 - accuracy: 0.9394
     782/782 [==============================] - 2s 2ms/step - loss: 0.2917 - accuracy: 0.8872

     results

     [0.29174429178237915, 0.8872399926185608]
```

b. One Hidden Layer

```
[22] model = keras.Sequential([
         layers.Dense(16, activation="relu"),
         layers.Dense(1, activation="sigmoid")
     ])
     model.compile(optimizer="rmsprop",
                   loss="binary_crossentropy",
                   metrics=["accuracy"])
     model.fit(x_train, y_train, epochs=4, batch_size=512)
     results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 32ms/step - loss: 0.4382 - accuracy: 0.8312
Epoch 2/4
49/49 [==============================] - 2s 32ms/step - loss: 0.2711 - accuracy: 0.9085
Epoch 3/4
49/49 [==============================] - 2s 32ms/step - loss: 0.2167 - accuracy: 0.9267
Epoch 4/4
49/49 [==============================] - 2s 32ms/step - loss: 0.1853 - accuracy: 0.9360
782/782 [==============================] - 2s 2ms/step - loss: 0.2810 - accuracy: 0.8875
```

```
[23]
     results
```

```
[0.2810378968715668, 0.8875200152397156]
```

c. Three Hidden Layer

```
[25] model = keras.Sequential([
         layers.Dense(16, activation="relu"),
         layers.Dense(16, activation="relu"),
         layers.Dense(16, activation="relu"),
         layers.Dense(1, activation="sigmoid")
     ])
     model.compile(optimizer="rmsprop",
                   loss="binary_crossentropy",
                   metrics=["accuracy"])
     model.fit(x_train, y_train, epochs=4, batch_size=512)
     results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 3s 42ms/step - loss: 0.4859 - accuracy: 0.8079
Epoch 2/4
49/49 [==============================] - 2s 33ms/step - loss: 0.2700 - accuracy: 0.9047
Epoch 3/4
49/49 [==============================] - 2s 33ms/step - loss: 0.2011 - accuracy: 0.9269
Epoch 4/4
49/49 [==============================] - 2s 32ms/step - loss: 0.1679 - accuracy: 0.9405
782/782 [==============================] - 2s 2ms/step - loss: 0.2980 - accuracy: 0.8836
```

```
results
```

```
[0.29798460006713867, 0.8835600018501282]
```

d.

In my results, one hidden layer proved to have the highest test accuracy in the final run of the model and lowest test loss number.

3. Layers with Units

All of the models with higher numbers of units compared to the original were less accurate and had relatively high test loss numbers. Whereas the model with only 8 units actually proved the most accurate and the least amount of test lost. This makes sense when you look at regularization and reducing the network size.

A. Original

```
[27] model = keras.Sequential([
         layers.Dense(16, activation="relu"),
         layers.Dense(16, activation="relu"),
         layers.Dense(1, activation="sigmoid")
     ])
     model.compile(optimizer="rmsprop",
                   loss="binary_crossentropy",
                   metrics=["accuracy"])
     model.fit(x_train, y_train, epochs=4, batch_size=512)
     results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 27ms/step - loss: 0.5120 - accuracy: 0.7934
Epoch 2/4
49/49 [==============================] - 2s 32ms/step - loss: 0.3030 - accuracy: 0.9056
Epoch 3/4
49/49 [==============================] - 1s 30ms/step - loss: 0.2188 - accuracy: 0.9283
Epoch 4/4
49/49 [==============================] - 1s 29ms/step - loss: 0.1779 - accuracy: 0.9394
782/782 [==============================] - 2s 2ms/step - loss: 0.2917 - accuracy: 0.8872
```

```
results
```

```
[0.29174429178237915, 0.8872399926185608]
```

B.32 Units

```
model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 35ms/step - loss: 0.4269 - accuracy: 0.8235
Epoch 2/4
49/49 [==============================] - 2s 37ms/step - loss: 0.2390 - accuracy: 0.9119
Epoch 3/4
49/49 [==============================] - 2s 38ms/step - loss: 0.1880 - accuracy: 0.9308
Epoch 4/4
49/49 [==============================] - 2s 38ms/step - loss: 0.1555 - accuracy: 0.9431
782/782 [==============================] - 2s 3ms/step - loss: 0.3419 - accuracy: 0.8704
```

```
results
```

```
[0.3419124186038971, 0.8704000115394592]
```

C. 64 Units

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 3s 46ms/step - loss: 0.4209 - accuracy: 0.8160
Epoch 2/4
49/49 [==============================] - 2s 42ms/step - loss: 0.2369 - accuracy: 0.9089
Epoch 3/4
49/49 [==============================] - 2s 41ms/step - loss: 0.1894 - accuracy: 0.9275
Epoch 4/4
49/49 [==============================] - 2s 47ms/step - loss: 0.1504 - accuracy: 0.9428
782/782 [==============================] - 3s 3ms/step - loss: 0.3284 - accuracy: 0.8760
```

```
results
```

```
[0.32843777537345886, 0.8759599924087524]
```

D. 8 Units

```
[22] model = keras.Sequential([
         layers.Dense(8, activation="relu"),
         layers.Dense(8, activation="relu"),
         layers.Dense(1, activation="sigmoid")
     ])
     model.compile(optimizer="rmsprop",
                   loss="binary_crossentropy",
                   metrics=["accuracy"])
     model.fit(x_train, y_train, epochs=4, batch_size=512)
     results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 29ms/step - loss: 0.4802 - accuracy: 0.8148
Epoch 2/4
49/49 [==============================] - 1s 26ms/step - loss: 0.2853 - accuracy: 0.9077
Epoch 3/4
49/49 [==============================] - 1s 31ms/step - loss: 0.2178 - accuracy: 0.9254
Epoch 4/4
49/49 [==============================] - 1s 26ms/step - loss: 0.1822 - accuracy: 0.9386
782/782 [==============================] - 3s 4ms/step - loss: 0.2866 - accuracy: 0.8853
```

```
results
```

```
[0.28664538264274597, 0.8853200078010559]
```

4. MSE Function

The MSE function yielded lower test lost results but a much lower accuracy.

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 28ms/step - loss: 0.1438 - accuracy: 0.8200
Epoch 2/4
49/49 [==============================] - 2s 36ms/step - loss: 0.0766 - accuracy: 0.9112
Epoch 3/4
49/49 [==============================] - 2s 35ms/step - loss: 0.0578 - accuracy: 0.9326
Epoch 4/4
49/49 [==============================] - 2s 31ms/step - loss: 0.0480 - accuracy: 0.9445
782/782 [==============================] - 2s 2ms/step - loss: 0.0856 - accuracy: 0.8844
```

```
results
```

```
[0.08563128858804703, 0.8844000101089478]
```

5. Tanh

Tahn yielded a much lower test lost results but a slightly lower accuracy result as well.

```
[29] model = keras.Sequential([
         layers.Dense(16, activation="tanh"),
         layers.Dense(16, activation="tanh"),
         layers.Dense(1, activation="sigmoid")
     ])
     model.compile(optimizer="rmsprop",
                   loss="mse",
                   metrics=["accuracy"])
     model.fit(x_train, y_train, epochs=4, batch_size=512)
     results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 34ms/step - loss: 0.1360 - accuracy: 0.8272
Epoch 2/4
49/49 [==============================] - 2s 39ms/step - loss: 0.0697 - accuracy: 0.9128
Epoch 3/4
49/49 [==============================] - 2s 37ms/step - loss: 0.0507 - accuracy: 0.9371
Epoch 4/4
49/49 [==============================] - 2s 38ms/step - loss: 0.0421 - accuracy: 0.9476
782/782 [==============================] - 2s 2ms/step - loss: 0.0908 - accuracy: 0.8804
```

```
results
```

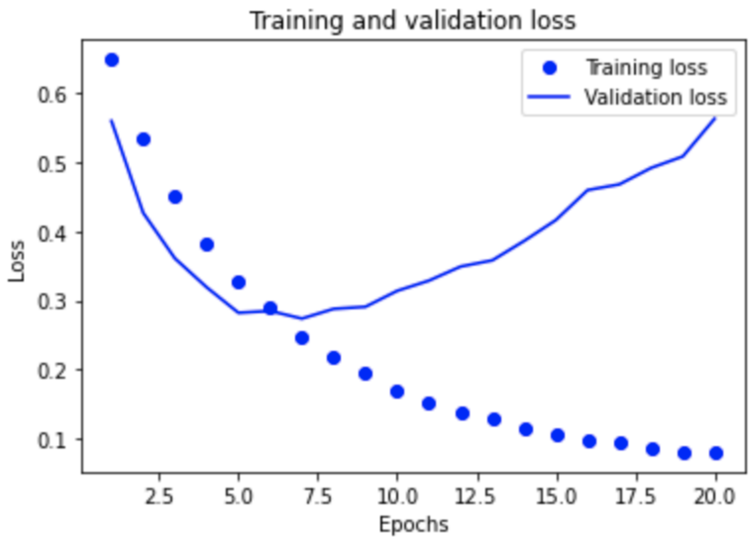
```
[0.09077661484479904, 0.8803600072860718]
```

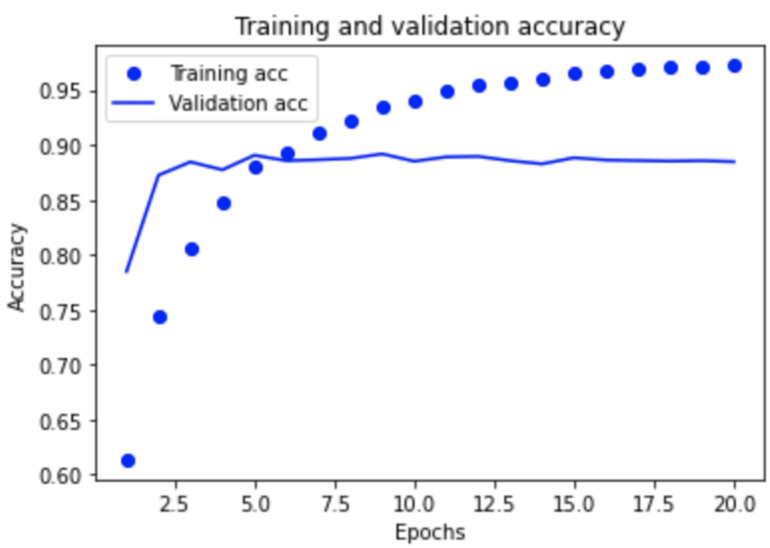
6. Dropout

Using dropout, delayed the overfitting of validation and training loss to the 6th epoch instead of 7th. We could use this technique and then run the model later with 6 epochs instead of 4.

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

model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
```


Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Training and validation accuracy
Training acc
Validation acc
Accuracy
Epochs
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0