Advanced Machine Learning- Assignmnet 1

Submitted by - Riba Khan

Classifying movie reviews: A binary classification example

Double-click (or enter) to edit

The IMDB dataset

Loading the IMDB dataset

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17465344/17464789 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
```

```
train_data[0]
```

```
 88,
 4,
 381,
 15,
 297,
 98,
 32,
 2071,
 56
```

```
36,
26,
141,
6,
194,
7486,
18,
4,
226,
22,
21,
134,
476,
26,
480,
5,
144,
30,
5535,
18,
51,
36,
28,
224,
92,
25,
104,
4,
226,
65,
16,
38,
1334,
88,
12,
16,
283,
5,
16,
4472,
113,
103
```

```
     103,
     32,
     15,
     16,
     5345,
     19,
     178,
     32]
```

```
train_labels[0]
```

```
1
```

```
max([max(sequence) for sequence in train_data])
```

```
9999
```

Decoding reviews back to text

```
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 0s 0us/step
```

Preparing the data

Encoding the integer sequences via multi-hot encoding

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

```
x_train[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

Double-click (or enter) to edit

Building your model

Model definition

Approach 1

Using only one hidden layer with 32 units

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

model = keras.Sequential([
    layers.Dense(32, activation="tanh"), #using tanh function
    layers.Dense(1, activation="sigmoid")
])
```

Compiling the model

```
model.compile(optimizer="adam", #using adam as the optimiser instead of rmsprop
              loss="mean_squared_error", # using mse instead of binary_crossentropy
              metrics=["accuracy"])
```

Validating your approach

Setting aside a validation set

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training your model

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
```

```
30/30 [==============================] - 2s 50ms/step - loss: 0.1518 - accuracy: 0.8077 - val_loss: 0.1059 - val_accu
Epoch 2/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0765 - accuracy: 0.9143 - val_loss: 0.0890 - val_accu
Epoch 3/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0555 - accuracy: 0.9435 - val_loss: 0.0853 - val_accu
Epoch 4/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0432 - accuracy: 0.9589 - val_loss: 0.0842 - val_accu
Epoch 5/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0346 - accuracy: 0.9716 - val_loss: 0.0851 - val_accu
Epoch 6/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0283 - accuracy: 0.9790 - val_loss: 0.0859 - val_accu
Epoch 7/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0232 - accuracy: 0.9843 - val_loss: 0.0879 - val_accu
Epoch 8/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0193 - accuracy: 0.9877 - val_loss: 0.0893 - val_accu
Epoch 9/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0162 - accuracy: 0.9901 - val_loss: 0.0907 - val_accu
Epoch 10/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0141 - accuracy: 0.9921 - val_loss: 0.0926 - val_accu
Epoch 11/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0119 - accuracy: 0.9935 - val_loss: 0.0953 - val_accu
Epoch 12/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0102 - accuracy: 0.9947 - val_loss: 0.0959 - val_accu
Epoch 13/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0089 - accuracy: 0.9956 - val_loss: 0.0971 - val_accu
Epoch 14/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0077 - accuracy: 0.9961 - val_loss: 0.0980 - val_accu
Epoch 15/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0069 - accuracy: 0.9964 - val_loss: 0.0998 - val_accu
Epoch 16/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0062 - accuracy: 0.9968 - val_loss: 0.1009 - val_accu
Epoch 17/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0056 - accuracy: 0.9971 - val_loss: 0.1014 - val_accu
Epoch 18/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0052 - accuracy: 0.9971 - val_loss: 0.1022 - val_accu
Epoch 19/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0048 - accuracy: 0.9971 - val_loss: 0.1029 - val_accu
Epoch 20/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0045 - accuracy: 0.9971 - val_loss: 0.1037 - val_accu
```

```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Here the validation accuracy appears to be 0.8680.

Plotting the training and validation loss

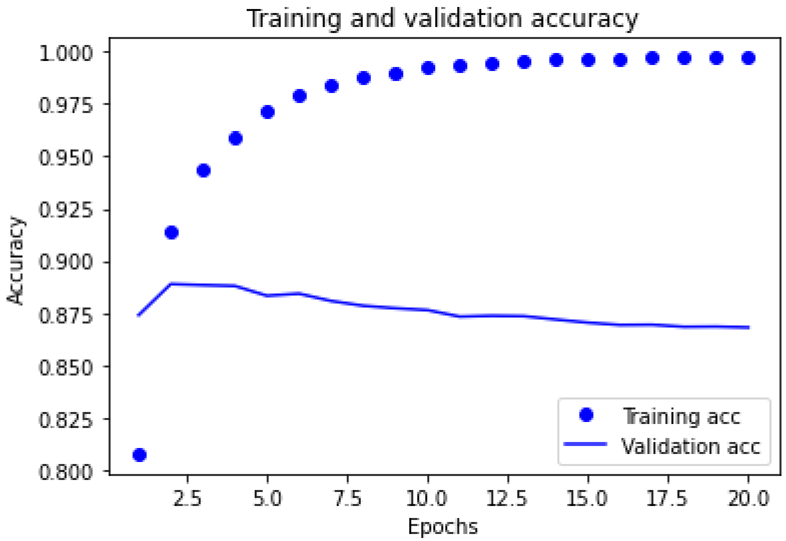
```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Training and validation loss

Plotting the training and validation accuracy

0.12

```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining a model from scratch

```
model = keras.Sequential([
    layers.Dense(32, activation="tanh"),
```

```
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="mean_squared_error",
              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 30ms/step - loss: 0.1349 - accuracy: 0.8368
Epoch 2/4
49/49 [==============================] - 1s 30ms/step - loss: 0.0707 - accuracy: 0.9187
Epoch 3/4
49/49 [==============================] - 1s 29ms/step - loss: 0.0536 - accuracy: 0.9408
Epoch 4/4
49/49 [==============================] - 1s 30ms/step - loss: 0.0442 - accuracy: 0.9526
782/782 [==============================] - 2s 3ms/step - loss: 0.0880 - accuracy: 0.8805
```

```
results
```

```
[0.08802604675292969, 0.8805199861526489]
```

Using a trained model to generate predictions on new data

```
model.predict(x_test)
```

```
array([[0.17096704],
       [0.99962246],
       [0.71258503],
       ...,
       [0.16114956],
       [0.09092519],
       [0.6515388 ]], dtype=float32)
```

Accuracy of 0.8805 is achieved

Approach 2

Using three hidden layers with 64 units each

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

model= keras.Sequential([
    layers.Dense( 64, activation="tanh"),
    layers.Dense(64, activation="tanh"),
    layers.Dense(64, activation= "tanh"),
    layers.Dense(1, activation="sigmoid")
])
```

```
model.compile(optimizer="adam",
              loss="mean_squared_error",
              metrics=["accuracy"])
```

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
    =========================] - 2s 65ms/step - loss: 0.1311 - accuracy: 0.8175 - val_loss: 0.0860 - val_accuracy: 0.8826
```

```
=========================] - 2s 52ms/step - loss: 0.0493 - accuracy: 0.9375 - val_loss: 0.0967 - val_accuracy: 0.8752

=========================] - 2s 52ms/step - loss: 0.0301 - accuracy: 0.9673 - val_loss: 0.0969 - val_accuracy: 0.8772

=========================] - 2s 73ms/step - loss: 0.0205 - accuracy: 0.9790 - val_loss: 0.1048 - val_accuracy: 0.8730

=========================] - 2s 51ms/step - loss: 0.0169 - accuracy: 0.9829 - val_loss: 0.1085 - val_accuracy: 0.8721

=========================] - 2s 52ms/step - loss: 0.0200 - accuracy: 0.9775 - val_loss: 0.1138 - val_accuracy: 0.8688

=========================] - 2s 52ms/step - loss: 0.0183 - accuracy: 0.9801 - val_loss: 0.1162 - val_accuracy: 0.8684

=========================] - 2s 52ms/step - loss: 0.0173 - accuracy: 0.9812 - val_loss: 0.1177 - val_accuracy: 0.8668

=========================] - 2s 52ms/step - loss: 0.0158 - accuracy: 0.9825 - val_loss: 0.1186 - val_accuracy: 0.8674

=========================] - 2s 51ms/step - loss: 0.0145 - accuracy: 0.9849 - val_loss: 0.1207 - val_accuracy: 0.8661

=========================] - 2s 52ms/step - loss: 0.0142 - accuracy: 0.9853 - val_loss: 0.1209 - val_accuracy: 0.8672

=========================] - 2s 51ms/step - loss: 0.0138 - accuracy: 0.9853 - val_loss: 0.1228 - val_accuracy: 0.8647

=========================] - 1s 50ms/step - loss: 0.0133 - accuracy: 0.9862 - val_loss: 0.1268 - val_accuracy: 0.8606

=========================] - 2s 50ms/step - loss: 0.0126 - accuracy: 0.9869 - val_loss: 0.1284 - val_accuracy: 0.8606

=========================] - 2s 52ms/step - loss: 0.0121 - accuracy: 0.9874 - val_loss: 0.1315 - val_accuracy: 0.8586

=========================] - 2s 51ms/step - loss: 0.0150 - accuracy: 0.9833 - val_loss: 0.1240 - val_accuracy: 0.8670

=========================] - 2s 51ms/step - loss: 0.0140 - accuracy: 0.9849 - val_loss: 0.1223 - val_accuracy: 0.8677

=========================] - 2s 52ms/step - loss: 0.0154 - accuracy: 0.9831 - val_loss: 0.1242 - val_accuracy: 0.8675

=========================] - 2s 52ms/step - loss: 0.0137 - accuracy: 0.9855 - val_loss: 0.1267 - val_accuracy: 0.8645

=========================] - 2s 51ms/step - loss: 0.0136 - accuracy: 0.9853 - val_loss: 0.1261 - val_accuracy: 0.8648
```

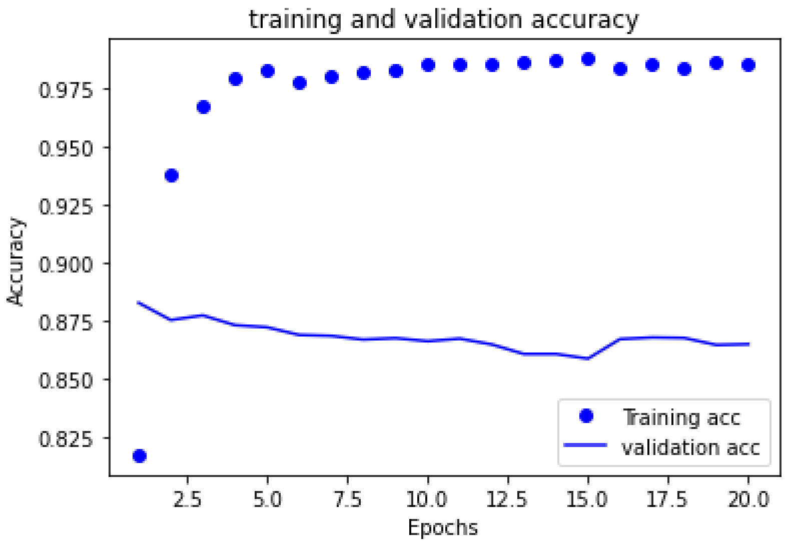
Here the validation accuracy is 0.8648

Plotting the training and validation loss

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("epochs")
plt.ylable("loss")
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-31-ffeea3d99c43> in <module>
      8 plt.title("Training and validation loss")
      9 plt.xlabel("epochs")
---> 10 plt.ylable("loss")
     11 plt.legend()
```

```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="validation acc")
plt.title("training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining model from scratch

```
model = keras.Sequential([
    layers.Dense(64, activation="tanh"),
    layers.Dense(64, activation="tanh"),
    layers.Dense(64, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])
```

```
model.compile(optimizer="adam",
              loss="mean_squared_error",
              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 40ms/step - loss: 0.1129 - accuracy: 0.8444
Epoch 2/4
49/49 [==============================] - 2s 39ms/step - loss: 0.0508 - accuracy: 0.9359
Epoch 3/4
49/49 [==============================] - 2s 41ms/step - loss: 0.0366 - accuracy: 0.9575
Epoch 4/4
49/49 [==============================] - 2s 39ms/step - loss: 0.0303 - accuracy: 0.9652
782/782 [==============================] - 2s 3ms/step - loss: 0.1107 - accuracy: 0.8674
```

```
results
```

```
[0.11068927496671677, 0.8673999905586243]
```

Accuracy of 0.867 achieved

Approach 3

Using three hidden layer with combination of 64, 32 and 64 units and adding dropout layer of 0.2

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

model = keras.Sequential([
    layers.Dense(64, activation ="tanh"),
    layers.Dropout(0.2),
    layers.Dense(32, activation="tanh"),
    layers.Dropout(0.2),
    layers.Dense(64, activation="tanh"),
    layers.Dropout(0.2),
    layers.Dense(1, activation="sigmoid")])
```

Compling the model

```
model.compile(optimizer="adam",
              loss="mean_squared_error",
              metrics=["accuracy"])
```

Validating your approach

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

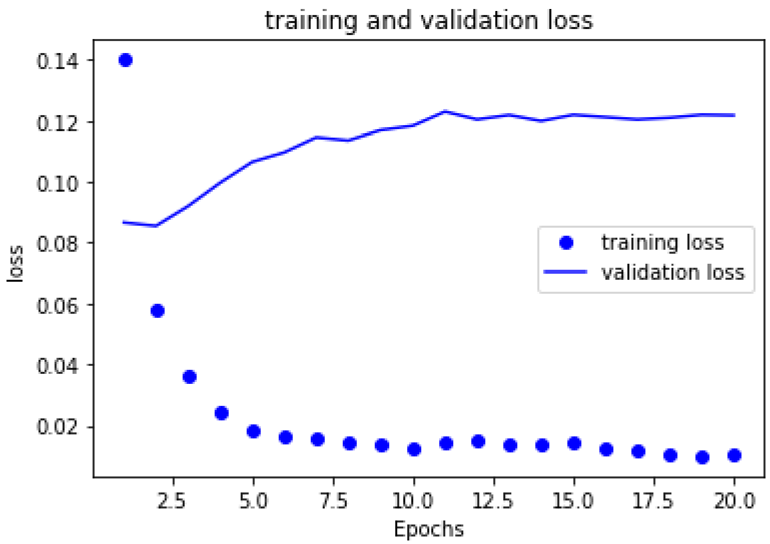
```
Epoch 1/20
```

```
30/30 [==============================] - 2s 57ms/step - loss: 0.1401 - accuracy: 0.8087 - val_loss: 0.0867 - val_acc
Epoch 2/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0580 - accuracy: 0.9247 - val_loss: 0.0855 - val_acc
Epoch 3/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0361 - accuracy: 0.9579 - val_loss: 0.0920 - val_acc
Epoch 4/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0243 - accuracy: 0.9741 - val_loss: 0.0997 - val_acc
Epoch 5/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0187 - accuracy: 0.9803 - val_loss: 0.1066 - val_acc
Epoch 6/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0167 - accuracy: 0.9817 - val_loss: 0.1097 - val_acc
Epoch 7/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0154 - accuracy: 0.9828 - val_loss: 0.1146 - val_acc
Epoch 8/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0141 - accuracy: 0.9847 - val_loss: 0.1136 - val_acc
Epoch 9/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0137 - accuracy: 0.9849 - val_loss: 0.1170 - val_acc
Epoch 10/20
30/30 [==============================] - 2s 50ms/step - loss: 0.0126 - accuracy: 0.9864 - val_loss: 0.1184 - val_acc
Epoch 11/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0144 - accuracy: 0.9836 - val_loss: 0.1230 - val_acc
Epoch 12/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0152 - accuracy: 0.9828 - val_loss: 0.1204 - val_acc
Epoch 13/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0139 - accuracy: 0.9846 - val_loss: 0.1219 - val_acc
Epoch 14/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0140 - accuracy: 0.9841 - val_loss: 0.1199 - val_acc
Epoch 15/20
30/30 [==============================] - 2s 51ms/step - loss: 0.0143 - accuracy: 0.9835 - val_loss: 0.1220 - val_acc
Epoch 16/20
30/30 [==============================] - 2s 51ms/step - loss: 0.0124 - accuracy: 0.9863 - val_loss: 0.1212 - val_acc
Epoch 17/20
30/30 [==============================] - 2s 50ms/step - loss: 0.0115 - accuracy: 0.9875 - val_loss: 0.1204 - val_acc
Epoch 18/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0103 - accuracy: 0.9891 - val_loss: 0.1210 - val_acc
Epoch 19/20
30/30 [==============================] - 2s 51ms/step - loss: 0.0099 - accuracy: 0.9897 - val_loss: 0.1220 - val_acc
Epoch 20/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0103 - accuracy: 0.9889 - val_loss: 0.1218 - val_acc
```

Here the validation accuracy is 0.8690

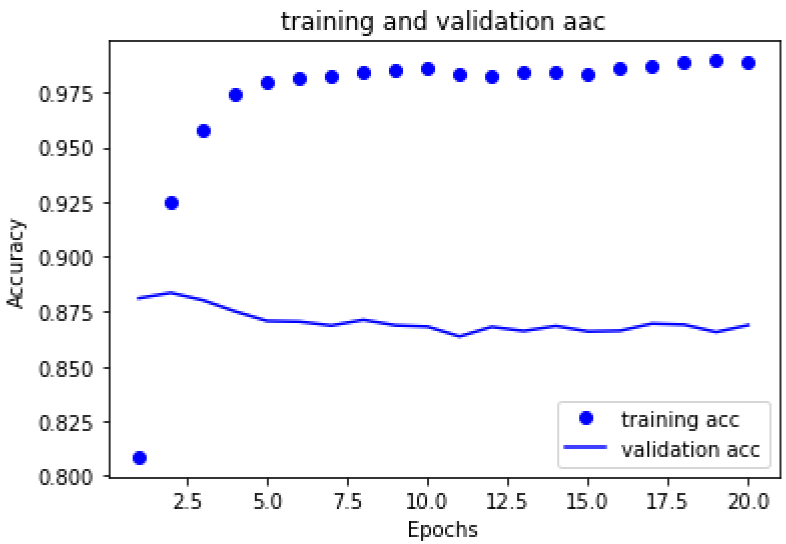
plotting training and validation loss

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1 , len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="training loss")
plt.plot(epochs, val_loss_values , "b", label="validation loss")
plt.title("training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.legend()
plt.show()
```



plotting training and validation accuracy

```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc , "bo", label="training acc")
plt.plot(epochs, val_acc, "b", label="validation acc")
plt.title("training and validation aac")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining model from scratch

```
model = keras.Sequential([
    layers.Dense(64, activation="tanh"),
    layers.Dropout(0.2),
    layers.Dense(64, activation="tanh"),
    layers.Dropout(0.2),
    layers.Dense(64, activation="tanh"),
    layers.Dropout(0.2),
```

```
    layers.Dense(1, activation="sigmoid")
])
```

```
model.compile(optimizer="adam",
              loss="mean_squared_error",
              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 44ms/step - loss: 0.0360 - accuracy: 0.9565
Epoch 2/4
49/49 [==============================] - 2s 43ms/step - loss: 0.0276 - accuracy: 0.9691
Epoch 3/4
49/49 [==============================] - 2s 44ms/step - loss: 0.0246 - accuracy: 0.9723
Epoch 4/4
49/49 [==============================] - 2s 44ms/step - loss: 0.0231 - accuracy: 0.9742
782/782 [==============================] - 2s 3ms/step - loss: 0.1246 - accuracy: 0.8607
```

```
results
```

```
[0.12463366240262985, 0.8607199788093567]
```

Accuracy of 0.8607 achieved

*Summary *

Steps Implemented:

1. Used Tanh activation function instead of relu.

2. Used Mse loss function isntead of binary crossentropy
3. Added dropout layers with a value of 0.2.(implemented only in approach 3)
4. Optimizer adam is used instead of rmsprop

I have used three differnet approaches:

1. Using only one hidden layer with 32 units which gives the validation accuracy of 0.8680.
2. Using three hidden layers with combination of 64, 32 and 64 units and adding dropout layer of 0.2 which gave the accuracy of 0.8690. We can say it touches 87% approximately.
3. Using three hidden layers with each 64 units which gave the accuracy of 0.8648.

As we can see the accuracy remains almost the same in all the three approaches.

Observations

1. when we use one hidden layer with 32 units and implementing the model from scratch, the results of the model give the highest accuracy of 0.8805. In other words 88% accuracy.
2. Despite using different techniques the accuracy can not be increase more than 87%.
3. More hidden units(a higher- dimensional representation space) enable network to learn more complex representations, but this increases computational cost and raises the possibility of learning undesirable patters(patterns that will improve performance on the training data but not on the test data.
4. As we see neural networks get better on their training data, it eventually starts to overfit and therefore ends up performing worse on the unseen data. To prevent overfiting one approach could be to used less number of epochs. Becuase as we can observe from the graphs that training loss decreases with every epoch but on the other hand training accuracy increases with every epoch.