‘’’TITLE: ASSIGNMENT 1
Name: Madhukar, Heerekar
Student ID: 811190131’’’

#Importing the required libraries

```
import matplotlib
import matplotlib.pyplot as plt
import NumPy as np
from keras import models
from keras import layers
```

#Loading the IMDB dataset using tensorflow and Keras

```
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.n
    17465344/17464789 [==============================] - 0s 0us/step
    17473536/17464789 [==============================] - 0s 0us/step
```

```
train_data[0]
```

```
     104,
     88,
     4,
     381,
     15,
     297,
     98,
     32,
     2071,
     56,
     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,
 32,
 15,
 16,
 5345,
 19,
 178,
 32]
```

```
train_labels[0]
```

```
1
```

#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_w
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 0s 0us/step
```

‘’’ We will use a technique called One-hot Encoding to convert our lists into vectors containing only 0s and 1s, which will prepare our data. Each sequence in the list will be multiplied by 10,000, with a value of 1 at every index corresponding to each integer in the sequence. Any indices in the vector that are not present in the sequence will have a value of 0. As a result, each review will be represented by a 10,000-dimensional vector. Each review will be represented by a 10,000-dimensional vector.’’’

#Vectorized the dataset

(len(sequences), dimension) and Sets specific indices of results[i] to 1s

```
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
```

#Vectorizing Training and Test Dataset

```
x_train = vectorize_sequences(train_data)
x_test =vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

Building Model

‘’’We are utilizing the ReLU activation function in two intermediate levels of our model. Each of these levels contains 16 hidden layers, with the purpose of zeroing out any negative values. The third layer of the model will be the output layer, which employs the sigmoid activation function.’’’

```
from tensorflow import keras
 model = keras.Sequential([
layers.Dense(16, activation="relu"),
```

```
layers.Dense(16, activation="relu"),
layers.Dense(1, activation="sigmoid")
```

"'DB-In our input dataset, we needed to convert the vectors into encoder labels consisting of 0s and 1s. We observed that Dense layers with ReLU activation performed well in this transformation process. Hidden layers in neural networks are placed between the input and output of the algorithm, and they apply weights to the inputs and pass them through an activation function to produce the output. These hidden layers essentially perform nonlinear transformations on the network's inputs.'"

Model definition

'''In this case, we will be using the following routines: the binary cross entropy loss function for binary classification (although we could also use Mean Squared Error), the RMSprop optimizer, and accuracy as the metric to evaluate performance.

Binary cross entropy computes the score by comparing each predicted probability with the actual class output, which can be 0 or 1, and penalizing deviations from the predicted value accordingly. It indicates how close or far the predicted value is from the actual value.

The RMSprop optimizer helps limit oscillations in the vertical plane, enabling us to increase the learning rate and take larger horizontal steps, resulting in faster convergence of the algorithm.'''

Model Compilation

```
model.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
```

Setting aside a validation set

'''As our model improves, we will allocate a portion of our training data for validation purposes to ensure the accuracy of the model. By creating a validation set, companies can monitor the development of the model as it advances through different epochs during training, thereby evaluating its performance on previously unseen data.'"

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

Model Training

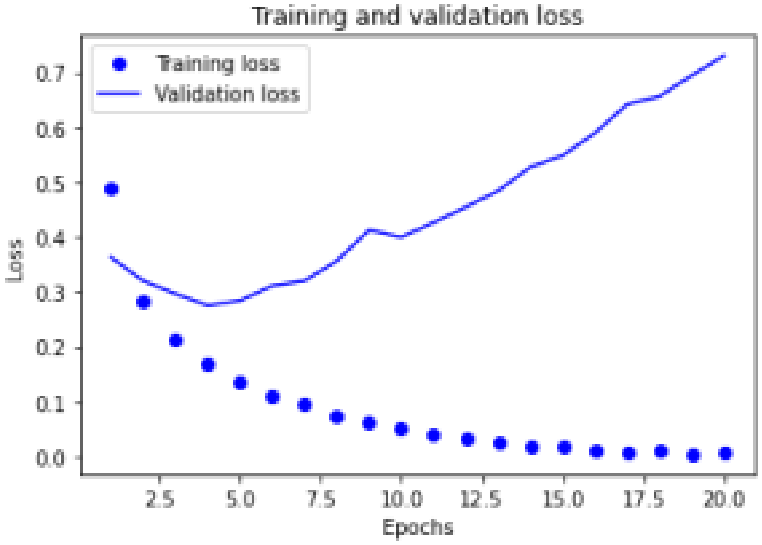
#We are training the model with batch size 512 and epochs=20

```
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 46ms/step - loss: 0.4877 - accuracy: 0.7910
Epoch 2/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2841 - accuracy: 0.9082
Epoch 3/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2143 - accuracy: 0.9277
Epoch 4/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1691 - accuracy: 0.9440
Epoch 5/20
30/30 [==============================] - 1s 34ms/step - loss: 0.1365 - accuracy: 0.9559
Epoch 6/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1112 - accuracy: 0.9657
Epoch 7/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0949 - accuracy: 0.9706
Epoch 8/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0748 - accuracy: 0.9796
Epoch 9/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0608 - accuracy: 0.9844
Epoch 10/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0499 - accuracy: 0.9882
Epoch 11/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0410 - accuracy: 0.9901
Epoch 12/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0335 - accuracy: 0.9930
Epoch 13/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0265 - accuracy: 0.9945
Epoch 14/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0175 - accuracy: 0.9981
Epoch 15/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0173 - accuracy: 0.9969
Epoch 16/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0125 - accuracy: 0.9987
Epoch 17/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0086 - accuracy: 0.9995
Epoch 18/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0096 - accuracy: 0.9985
Epoch 19/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0041 - accuracy: 0.9999
Epoch 20/20
 30/30 [==============================] - 1s 32ms/step - loss: 0.0070 - accuracy: 0.9987
```

#Plotting of Training and Validation Loss

```
history_dict = history.history
loss_values=history_dict["loss"]
3
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()
```

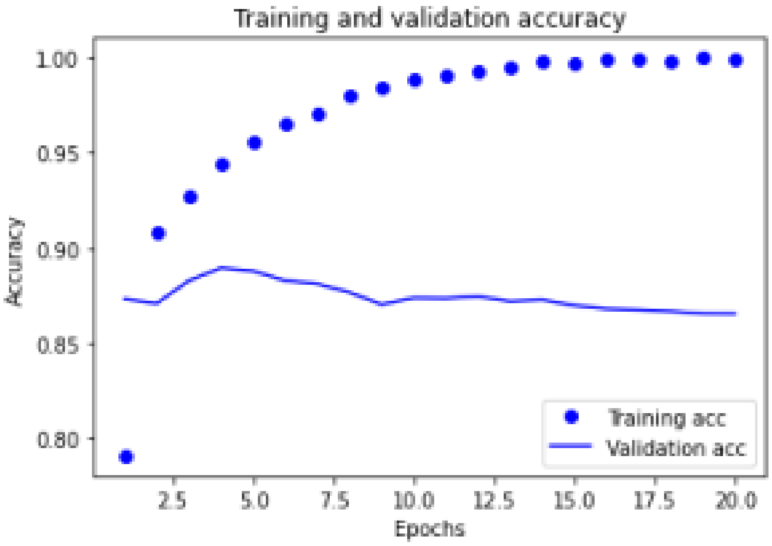


Validation loss started increasing from 3 epochs

#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 accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



‘’’After training a model with 16 nodes and 2 layers using the ReLU activation function and binary cross entropy loss, we observed that the validation loss decreased up to a certain point and then dramatically increased. Similarly, the validation accuracy decreased while the training accuracy increased. This indicates that the model is overfitting and becoming too specialized in classifying the training data, making it less accurate when predicting on new and unseen data. The trend of overfitting became more apparent after the fifth epoch, suggesting that the model is becoming too closely similar to the training data.’’’

#Retraining Model from begining

```
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 26ms/step - loss: 0.4468 - accuracy: 0.8151
Epoch 2/4
49/49 [==============================]  - 1s 26ms/step - loss: 0.2546 - accuracy: 0.9094
Epoch 3/4
49/49 [==============================]  - 1s 27ms/step - loss: 0.1942 - accuracy: 0.9305
Epoch 4/4
49/49 [==============================]  - 1s 26ms/step - loss: 0.1627 - accuracy: 0.9413
782/782 [==============================] - 3s 3ms/step - loss: 0.3160 - accuracy: 0.8759
```

#Here we have used epocs=4 for training the data.

```
results
```

```
[0.31600797176361084, 0.8759199976921082]
```

`’’ASSIGNMENT

1.You used two hidden layers. Try using one or three hidden layers, and see how doing so affects validation and test accuracy.’’’

#model_1 is build with 3 layers of relu activation function and model_2 with 1 layers of relu
#activation function

```
model_1 = keras.Sequential([
layers.Dense(16, activation="relu"),
layers.Dense(16, activation="relu"),
layers.Dense(16, activation="relu"),
layers.Dense(1, activation="sigmoid")
])
model_2 = keras.Sequential([
layers.Dense(16,activation="relu"),
layers.Dense(1, activation="sigmoid")
])
```

Both models were calculated using the binary cross-entropy loss function and

#the RMSprop optimizer. [model_1(3layers), model_2(1layer)]

```
model_1.compile(optimizer="rmsprop",
```

```
loss="binary_crossentropy",
metrics=["accuracy"])
model_2.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
```

Model Training is done with epocs= 20

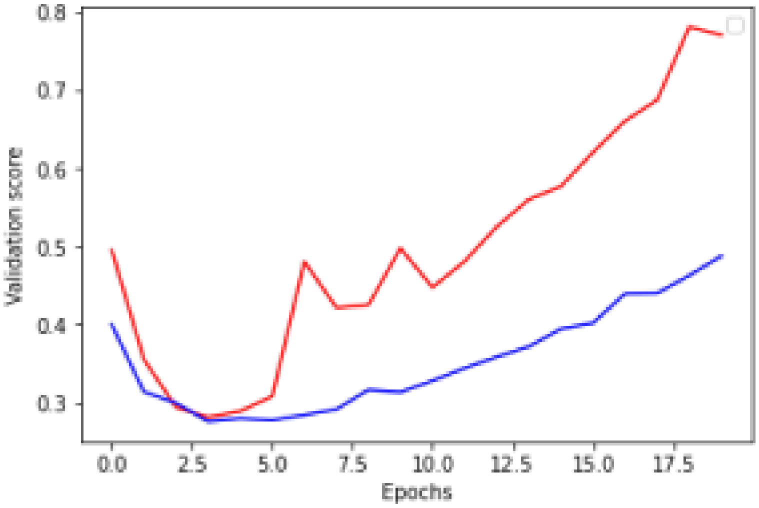
```
history_1 = model_1.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))
history_2 = model_2.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))

model 1 summary()
model_1.summary()
model_2.summary()
```

```
    30/30 [ ] 1s 34ms/step loss: 0.1243 accuracy: 0.96 Epoch 8/20
    30/30 [==============================] - 1s 40ms/step - loss: 0.1100 - accuracy: 0.96
    Epoch 9/20
    30/30 [==============================] - 2s 50ms/step - loss: 0.0955 - accuracy: 0.97
    Epoch 10/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0850 - accuracy: 0.97
    Epoch 11/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0744 - accuracy: 0.98
    Epoch 12/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0662 - accuracy: 0.98
    Epoch 13/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0585 - accuracy: 0.98
    Epoch 14/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0521 - accuracy: 0.98
    Epoch 15/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0453 - accuracy: 0.99
    Epoch 16/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0397 - accuracy: 0.99
    Epoch 17/20
    30/30 [==============================] - 1s 34ms/step - loss: 0.0349 - accuracy: 0.99
    Epoch 18/20
    30/30 [==============================] - 1s 37ms/step - loss: 0.0308 - accuracy: 0.99
    Epoch 19/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0262 - accuracy: 0.99
    Epoch 20/20
    30/30 [==============================] - 1s 35ms/step - loss: 0.0236 - accuracy: 0.99
    Model: "sequential_2"
```

```
 Layer (type) Output Shape Param #
=================================================================
 dense_6 (Dense) (None, 16) 160016

 dense_7 (Dense) (None, 16) 272

 dense_8 (Dense) (None, 16) 272

 dense_9 (Dense) (None, 1) 17

=================================================================
Total params: 160,577
Trainable params: 160,577
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
 Layer (type) Output Shape Param #
=================================================================
 dense_10 (Dense) (None, 16) 160016

 dense_11 (Dense) (None, 1) 17

=================================================================
Total params: 160,033
      p ,
Trainable params: 160,033
Non-trainable params: 0
_________________________________________________________________
```

#Plotting the training and validation loss

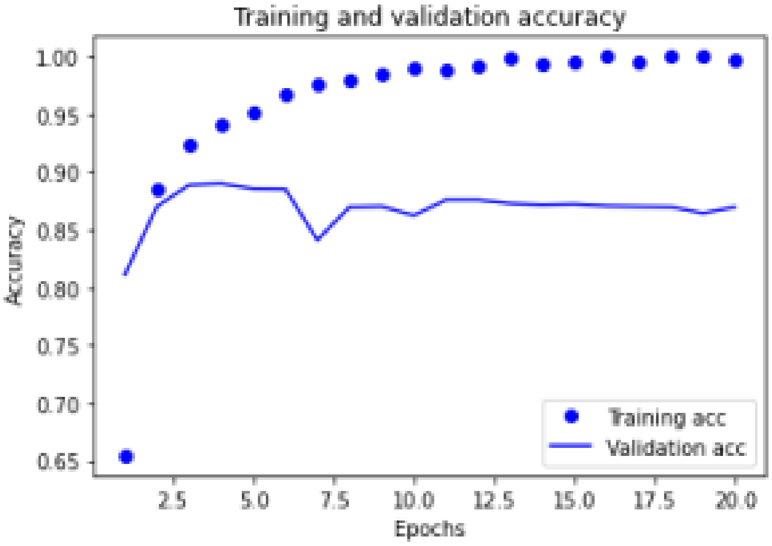
```
history_dict_1 = history_1.history
history_dict_2 = history_2.history
plt.plot(history_1.history['val_loss'], 'r', history_2.history['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.legend()
```



#here the validation loss is at 5 epochs

#Plotting Training and Validation accuracy

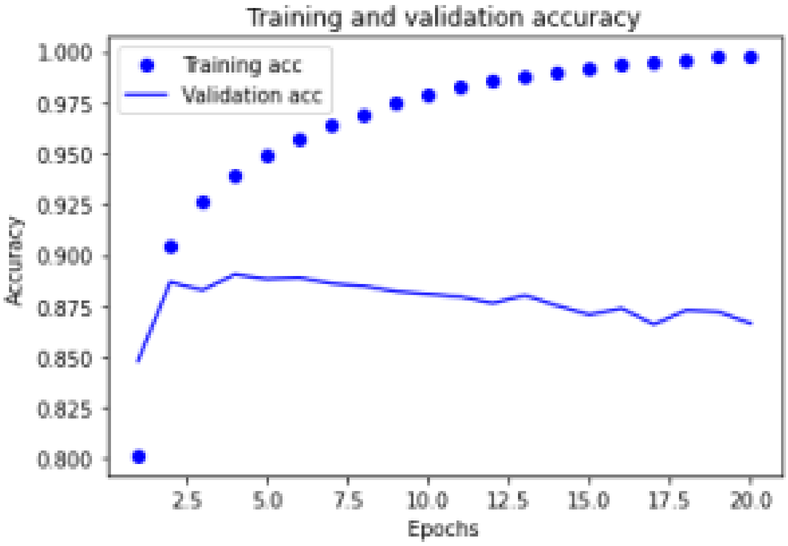
```
plt.clf()
acc = history_dict_1["accuracy"]
val_acc = history_dict_1["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
10
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



#here the validation accuracy is at 5 Epochs

#plot_loss

```
 plt.clf()
acc = history_dict_2["accuracy"]
val_acc = history_dict_2["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()
```



Here, the maximum validation accuracy is observed at 5th epoch.

2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.

```
model_3 = keras.Sequential([
layers.Dense(32, activation="relu"),
layers.Dense(64, activation="relu"),
layers.Dense(1, activation="sigmoid")
])
```

#We are using rmsprop and binary cross-entropy

```
model_3.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
```

#Here we have taken epochs= 20, and batch size=512 to fit the model

```
history_3 = model_3.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))
     Epoch 1/20
     30/30 [==============================] - 2s 54ms/step - loss: 0.5002 - accuracy: 0.7712
     Epoch 2/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.2675 - accuracy: 0.9085
     Epoch 3/20
     30/30 [==============================] - 1s 44ms/step - loss: 0.1993 - accuracy: 0.9291
     Epoch 4/20
     30/30 [==============================] - 1s 43ms/step - loss: 0.1551 - accuracy: 0.9439
     Epoch 5/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.1175 - accuracy: 0.9621
     Epoch 6/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0867 - accuracy: 0.9737
     Epoch 7/20
     30/30 [==============================] - 1s 43ms/step - loss: 0.0730 - accuracy: 0.9765
     Epoch 8/20
     30/30 [==============================] - 1s 41ms/step - loss: 0.0472 - accuracy: 0.9874
     Epoch 9/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0367 - accuracy: 0.9898
     Epoch 10/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0296 - accuracy: 0.9926
     Epoch 11/20
     30/30 [==============================] - 1s 43ms/step - loss: 0.0258 - accuracy: 0.9926
     Epoch 12/20
     30/30 [==============================] - 1s 43ms/step - loss: 0.0088 - accuracy: 0.9993
     Epoch 13/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0124 - accuracy: 0.9970
     Epoch 14/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0234 - accuracy: 0.9928
     Epoch 15/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0026 - accuracy: 0.9999
     Epoch 16/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0024 - accuracy: 0.9997
     Epoch 17/20
     30/30 [==============================] - 1s 42ms/step - loss: 0.0017 - accuracy: 1.0000
     Epoch 18/20
     30/30 [==============================] - 1s 41ms/step - loss: 0.0197 - accuracy: 0.9947
     Epoch 19/20
     30/30 [==============================] - 1s 43ms/step - loss: 4.8248e-04 - accuracy: 1.0
     Epoch 20/20
```

```
30/30 [==============================] - 1s 42ms/step - loss: 3.5065e-04 - accuracy: 1.0
```

```
model_3.summary()
```

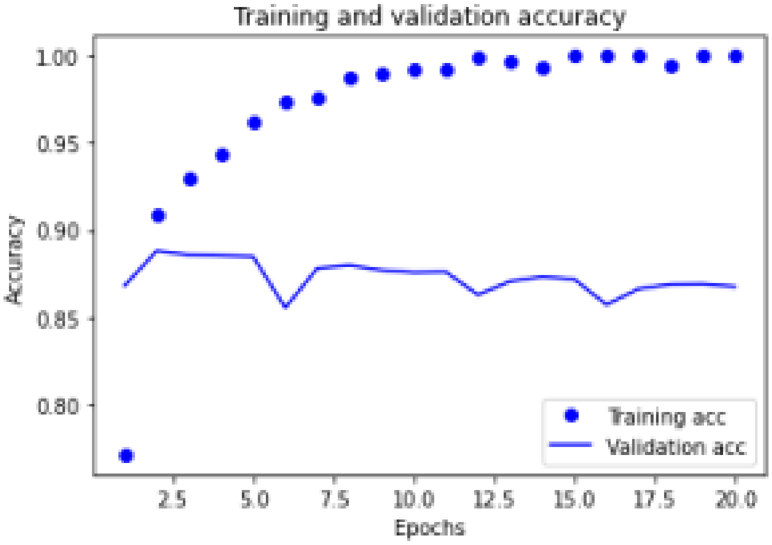
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type) Output Shape Param #
=================================================================
 dense_12 (Dense) (None, 32) 320032

 dense_13 (Dense) (None, 64) 2112

 dense_14 (Dense) (None, 1) 65

=================================================================
Total params: 322,209
Trainable params: 322,209
Non-trainable params: 0
_________________________________________________________________
```

```
history_dict_3 = history_3.history
```

```
loss_values = history_dict_3["loss"]
val_loss_values = history_dict_3["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()
```

```
plt.clf()
acc = history_dict_3["accuracy"]
```

```
val_acc = history_dict_3["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()
```



#The minimum validation loss is observed at the 2.5th epoch and maximum validation accuracy is #observed between the 2.5th and 3rd epochs.

3. Try using the 'mse' loss function instead of 'binary_crossentropy'

```
model_4 = keras.Sequential([
layers.Dense(16, activation="relu"),
layers.Dense(16, activation="relu"),
layers.Dense(1, activation="sigmoid")
])
```

We are using rmsprop and mse

```
model_4.compile(optimizer="rmsprop",
loss="mse",
metrics=["accuracy"])
```

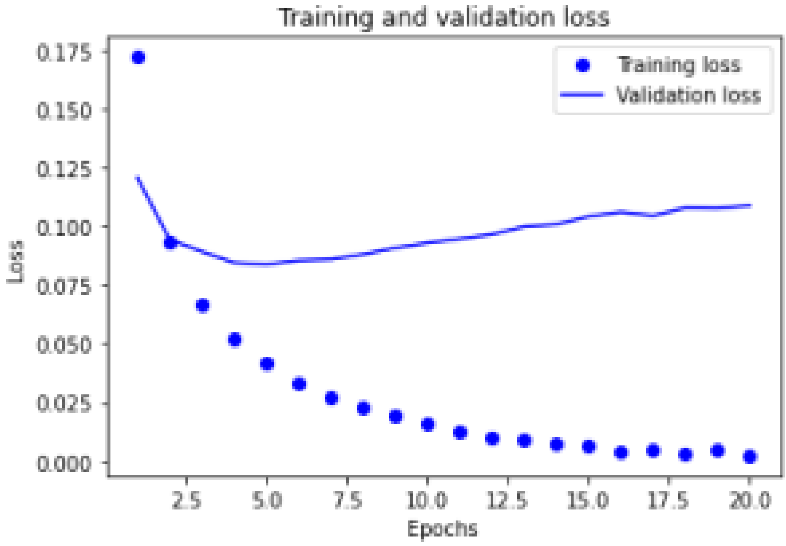
#Training your model

```
history_4 = model_4.fit(partial_x_train,
partial_y_train,
epochs=20,
b t h i 512
batch_size=512,
validation_data=(x_val, y_val))
```

```
     Epoch 1/20
     30/30 [==============================] - 2s 50ms/step - loss: 0.1720 - accuracy: 0.7875
     Epoch 2/20
     30/30 [==============================] - 1s 37ms/step - loss: 0.0931 - accuracy: 0.9063
     Epoch 3/20
     30/30 [==============================] - 1s 39ms/step - loss: 0.0670 - accuracy: 0.9281
     Epoch 4/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0517 - accuracy: 0.9457
     Epoch 5/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0419 - accuracy: 0.9552
     Epoch 6/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0336 - accuracy: 0.9657
     Epoch 7/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0274 - accuracy: 0.9746
     Epoch 8/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0228 - accuracy: 0.9793
     Epoch 9/20
     30/30 [==============================] - 1s 39ms/step - loss: 0.0192 - accuracy: 0.9825
     Epoch 10/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0159 - accuracy: 0.9868
     Epoch 11/20
     30/30 [==============================] - 1s 40ms/step - loss: 0.0126 - accuracy: 0.9907
     Epoch 12/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0099 - accuracy: 0.9929
     Epoch 13/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0089 - accuracy: 0.9931
     Epoch 14/20
     30/30 [==============================] - 1s 35ms/step - loss: 0.0075 - accuracy: 0.9940
     Epoch 15/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0063 - accuracy: 0.9949
     Epoch 16/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0042 - accuracy: 0.9970
     Epoch 17/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0052 - accuracy: 0.9955
     Epoch 18/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0031 - accuracy: 0.9975
     Epoch 19/20
     30/30 [==============================] - 1s 36ms/step - loss: 0.0045 - accuracy: 0.9958
     Epoch 20/20
      30/30 [==============================] - 1s 35ms/step - loss: 0.0024 - accuracy: 0.9979
```

```
history_dict_4 = history_4.history
loss_values = history_dict_4["loss"]
val_loss_values = history_dict_4["val_loss"]
```

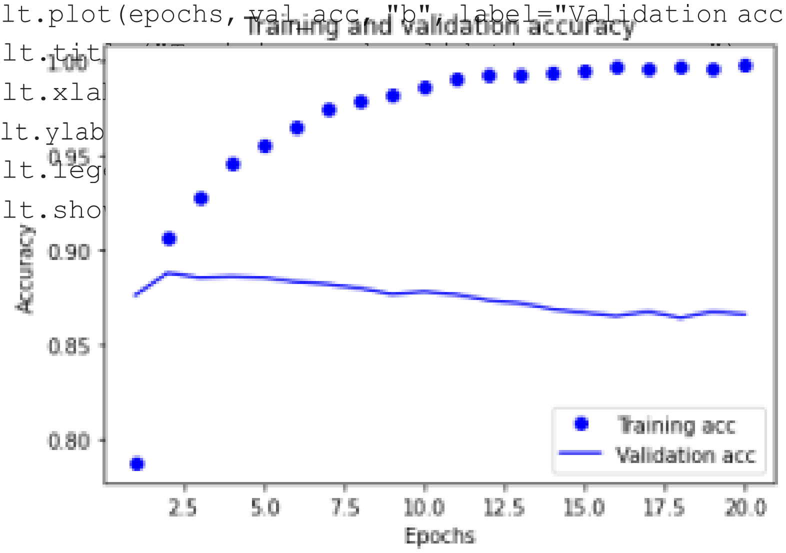
```
epochs = range(1, len(loss_values) +1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Here, the minimum validation loss is observed in 3rd epoch

#Plotting the training and validation accuracy

```
plt.clf()
acc = history_dict_4["accuracy"]
val_acc = history_dict_4["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.tit
plt.xlal
plt.ylab
plt.leg
plt.sho
```



#Here, the maximum accuracy is observed in the 2nd 3rd epochs.

4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of 'relu'

```
model_5 = keras.Sequential([
layers.Dense(16, activation="tanh"),
layers.Dense(16, activation="tanh"),
layers.Dense(1, activation="sigmoid")
])
```

#here we are using rmsprop and mse

```
model_5.compile(optimizer="rmsprop",
loss="mse",
metrics=["accuracy"])
```

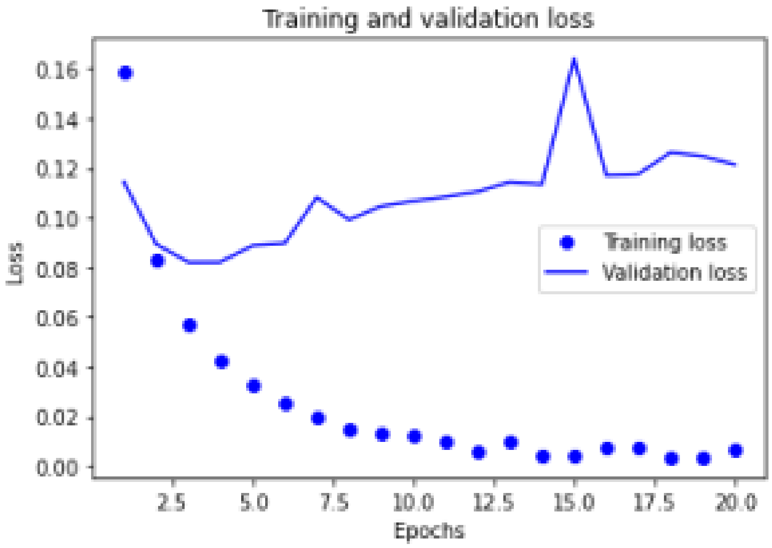
```
history_5 = model_5.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 54ms/step - loss: 0.1589 - accuracy: 0.8005
Epoch 2/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0827 - accuracy: 0.9083
Epoch 3/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0569 - accuracy: 0.9349
Epoch 4/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0424 - accuracy: 0.9511
Epoch 5/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0326 - accuracy: 0.9635
Epoch 6/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0258 - accuracy: 0.9717
Epoch 7/20
```

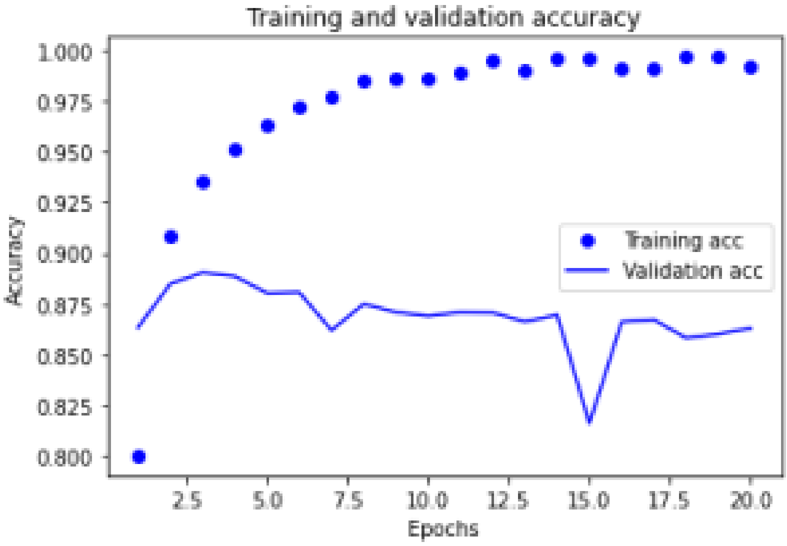
```
30/30 [==============================] - 2s 60ms/step - loss: 0.0200 - accuracy: 0.9772
Epoch 8/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0147 - accuracy: 0.9845
Epoch 9/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0130 - accuracy: 0.9863
Epoch 10/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0124 - accuracy: 0.9859
Epoch 11/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0101 - accuracy: 0.9887
Epoch 12/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0058 - accuracy: 0.9945
Epoch 13/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0097 - accuracy: 0.9894
Epoch 14/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0045 - accuracy: 0.9957
Epoch 15/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0044 - accuracy: 0.9957
Epoch 16/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0078 - accuracy: 0.9907
Epoch 17/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0077 - accuracy: 0.9909
Epoch 18/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0037 - accuracy: 0.9964
Epoch 19/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0035 - accuracy: 0.9965
Epoch 20/20
 30/30 [==============================] - 1s 36ms/step - loss: 0.0069 - accuracy: 0.9919
```

```
history_dict_5 = history_5.history
loss_values = history_dict_5["loss"]
val_loss_values = history_dict_5["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 the training and validation accuracy



```
plt.clf()
acc = history_dict_5["accuracy"]
val_acc = history_dict_5["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()
```



5.‘’’Use any technique we studied in class, and these include regularization,dropout, etc., to get your model to perform better on validation.’’’

```
model_6 = keras.Sequential([
#layers.Dropout(0.2),
```

```
layers.Dense(20, activation="relu"),
layers.Dropout(0.2),
layers.Dense(15, activation="relu"),
layers.Dense(1, activation="sigmoid")
])
```

#Model completion

```
model_6.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
history_6 = model_6.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 77ms/step - loss: 0.5114 - accuracy: 0.7702
Epoch 2/20
30/30 [==============================] - 1s 39ms/step - loss: 0.3204 - accuracy: 0.8874
Epoch 3/20
30/30 [==============================] - 1s 40ms/step - loss: 0.2396 - accuracy: 0.9173
Epoch 4/20
30/30 [==============================] - 1s 39ms/step - loss: 0.1930 - accuracy: 0.9324
Epoch 5/20
30/30 [==============================] - 1s 40ms/step - loss: 0.1575 - accuracy: 0.9451
Epoch 6/20
30/30 [==============================] - 1s 39ms/step - loss: 0.1321 - accuracy: 0.9575
Epoch 7/20
30/30 [==============================] - 1s 39ms/step - loss: 0.1077 - accuracy: 0.9656
Epoch 8/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0908 - accuracy: 0.9699
Epoch 9/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0717 - accuracy: 0.9780
Epoch 10/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0614 - accuracy: 0.9828
Epoch 11/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0483 - accuracy: 0.9861
Epoch 12/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0389 - accuracy: 0.9900
Epoch 13/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0299 - accuracy: 0.9921
Epoch 14/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0265 - accuracy: 0.9931
Epoch 15/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0176 - accuracy: 0.9965
Epoch 16/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0152 - accuracy: 0.9969
Epoch 17/20
```

```
 30/30 [==============================] - 1s 39ms/step - loss: 0.0125 - accuracy: 0.9974
Epoch 18/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0083 - accuracy: 0.9988
Epoch 19/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0101 - accuracy: 0.9977
Epoch 20/20
 30/30 [==============================] - 1s 40ms/step - loss: 0.0053 - accuracy: 0.9995
```

#Compiling the model

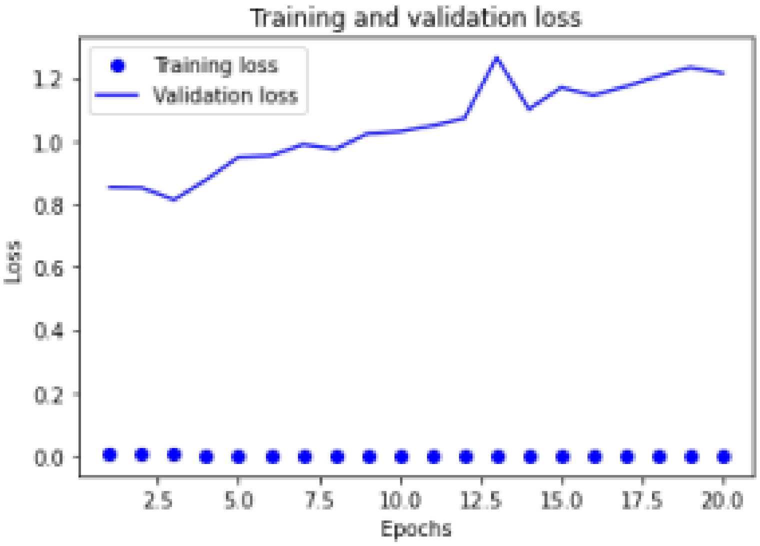
```
model_5.compile(optimizer="rmsprop",
           loss="binary_crossentropy",
           metrics=["accuracy"])
```

```
history_6 = model_6.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================]  - 1s 46ms/step - loss: 0.0054 - accuracy: 0.9989
Epoch 2/20
30/30 [==============================]  - 1s 39ms/step - loss: 0.0060 - accuracy: 0.9986
Epoch 3/20
30/30 [==============================]  - 1s 40ms/step - loss: 0.0053 - accuracy: 0.9988
Epoch 4/20
30/30 [==============================]  - 1s 39ms/step - loss: 0.0029 - accuracy: 0.9994
Epoch 5/20
30/30 [==============================]  - 1s 39ms/step - loss: 0.0025 - accuracy: 0.9997
Epoch 6/20
30/30 [==============================]  - 1s 40ms/step - loss: 0.0031 - accuracy: 0.9993
Epoch 7/20
30/30 [==============================]  - 1s 40ms/step - loss: 0.0033 - accuracy: 0.9993
Epoch 8/20
30/30 [==============================]  - 1s 39ms/step - loss: 0.0033 - accuracy: 0.9995
Epoch 9/20
30/30 [==============================]  - 1s 40ms/step - loss: 0.0029 - accuracy: 0.9991
Epoch 10/20
30/30 [==============================]  - 1s 39ms/step - loss: 0.0038 - accuracy: 0.9994
Epoch 11/20
30/30 [==============================]  - 1s 39ms/step - loss: 0.0032 - accuracy: 0.9994
Epoch 12/20
30/30 [==============================]  - 1s 39ms/step - loss: 0.0019 - accuracy: 0.9997
Epoch 13/20
30/30 [==============================]  - 1s 41ms/step - loss: 0.0021 - accuracy: 0.9995
Epoch 14/20
 30/30 [==============================] - 1s 39ms/step - loss: 0.0026 - accuracy: 0.9995
```

```
Epoch 15/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0015 - accuracy: 0.9997
Epoch 16/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0031 - accuracy: 0.9995
Epoch 17/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0014 - accuracy: 0.9998
Epoch 18/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0029 - accuracy: 0.9995
Epoch 19/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0019 - accuracy: 0.9995
Epoch 20/20
 30/30 [==============================] - 1s 39ms/step - loss: 0.0031 - accuracy: 0.9993
```

```
history_dict_6 = history_6.history
loss_values = history_dict_6["loss"]
val_loss_values = history_dict_6["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 the training and validation accuracy

```
plt.clf()
acc = history_dict_6["accuracy"]
val_acc = history_dict_6["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()
```