AML Assignment

September 25, 2023

[1]:
```
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
```

[2]:
```
train_labels[0]
```

[2]: 1

[3]:
```
max([max(sequence) for sequence in train_data])
```

[3]: 9999

[4]:
```
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
```

[5]:
```
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)
```

[6]:
```
x_train[0]
```

[6]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

[7]:
```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

1 constructing a model with a single hidden layer and 32 hidden units while utilizing the tanh activation function

[8]:
```
from tensorflow import keras
from tensorflow.keras import layers

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

2 Observations & Modifications:

This neural network has 32 hidden units with a tanh activation function and is built as a single layer. The Sigmoid activation units in the output layer employing mse during model compilation rather than binary_crossentropy.

[9]:
```
model.compile(optimizer="adam", #changing optimizer to ADAM
              loss="mean_squared_error",
              metrics=["accuracy"])
```

[ ]:
```
# When choosing an optimizer, I prefer Adam over rmsprop.
Adam is considered as the top optimizer by several sites including recent
↪Google trends.
loss is a binary_crossentrophy change to mse
Validating the approach
```

[10]:
```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[11]:
```
## Model was set up to train over 20 iterations using 256-person batches.
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=256,
                    validation_data=(x_val, y_val))
```

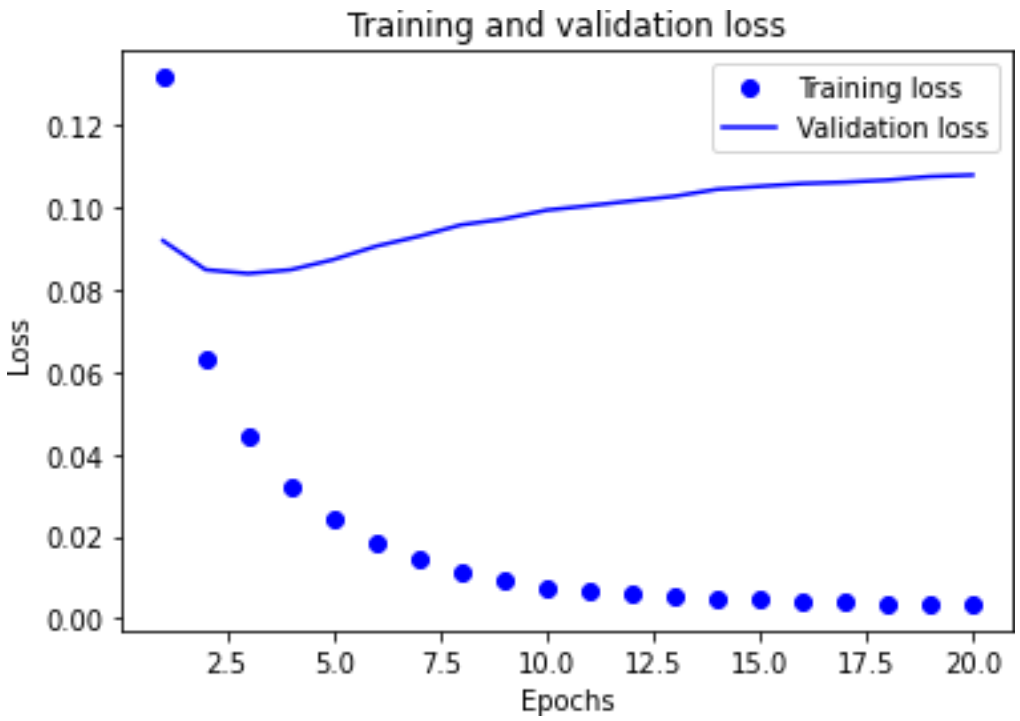
```
Epoch 1/20
59/59 [==============================] - 2s 27ms/step - loss: 0.1318 - accuracy: 0.8370 - val_loss: 0.0920 - val_accuracy: 0.8861
Epoch 2/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0634 - accuracy: 0.9272 - val_loss: 0.0849 - val_accuracy: 0.8877
Epoch 3/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0440 - accuracy: 0.9546 - val_loss: 0.0840 - val_accuracy: 0.8863
Epoch 4/20
59/59 [==============================] - 1s 20ms/step - loss: 0.0322 - accuracy: 0.9709 - val_loss: 0.0849 - val_accuracy: 0.8849
Epoch 5/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0241 - accuracy: 0.9805 - val_loss: 0.0874 - val_accuracy: 0.8801
Epoch 6/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0183 - accuracy: 0.9868 - val_loss: 0.0906 - val_accuracy: 0.8791
Epoch 7/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0142 - accuracy: 0.9909 - val_loss: 0.0930 - val_accuracy: 0.8773
Epoch 8/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0112 - accuracy: 0.9932 - val_loss: 0.0958 - val_accuracy: 0.8730
Epoch 9/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0092 - accuracy: 0.9942 - val_loss: 0.0973 - val_accuracy: 0.8731
Epoch 10/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0076 - accuracy: 0.9953 - val_loss: 0.0994 - val_accuracy: 0.8717
Epoch 11/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0065 - accuracy: 0.9958 - val_loss: 0.1005 - val_accuracy: 0.8693
Epoch 12/20
59/59 [==============================] - 1s 17ms/step - loss: 0.0058 - accuracy: 0.9960 - val_loss: 0.1017 - val_accuracy: 0.8697
Epoch 13/20
59/59 [==============================] - 1s 16ms/step - loss: 0.0052 - accuracy: 0.9964 - val_loss: 0.1028 - val_accuracy: 0.8685
Epoch 14/20
59/59 [==============================] - 1s 20ms/step - loss: 0.0048 - accuracy: 0.9965 - val_loss: 0.1045 - val_accuracy: 0.8679
Epoch 15/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0045 - accuracy: 0.9965 - val_loss: 0.1052 - val_accuracy: 0.8668
Epoch 16/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0042 - accuracy: 0.9967 - val_loss: 0.1059 - val_accuracy: 0.8672
```

```
Epoch 17/20
59/59 [==============================] - 1s 24ms/step - loss: 0.0040 - accuracy:
0.9967 - val_loss: 0.1062 - val_accuracy: 0.8665
Epoch 18/20
59/59 [==============================] - 2s 33ms/step - loss: 0.0037 - accuracy:
0.9969 - val_loss: 0.1067 - val_accuracy: 0.8657
Epoch 19/20
59/59 [==============================] - 1s 18ms/step - loss: 0.0036 - accuracy:
0.9970 - val_loss: 0.1076 - val_accuracy: 0.8661
Epoch 20/20
59/59 [==============================] - 1s 14ms/step - loss: 0.0035 - accuracy:
0.9970 - val_loss: 0.1080 - val_accuracy: 0.8663
```

[12]:
```
history_dict = history.history
history_dict.keys()
```

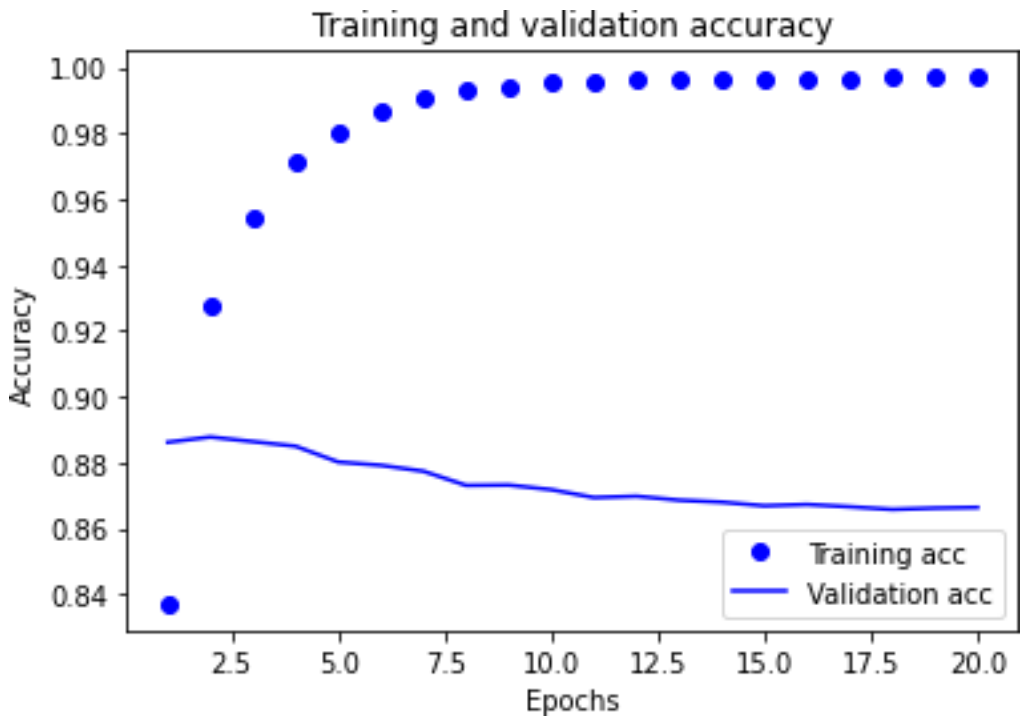
[12]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[13]:
```
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()
```



3 Plotting the training and validation accuracy

[14]:
```
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()
```



[ ]:
```
From the above figure we can notice that training accuracy almost reaches to␣
↪100 percent accurately 99.81 with 20 epochs
if we observe the validation accuracy, initially it tends to decrease and at␣
↪the end it gives a constant 87%
```

[15]:
```
results = model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 1s 703us/step - loss: 0.1186 -
accuracy: 0.8544
```

[16]:
```
results
```

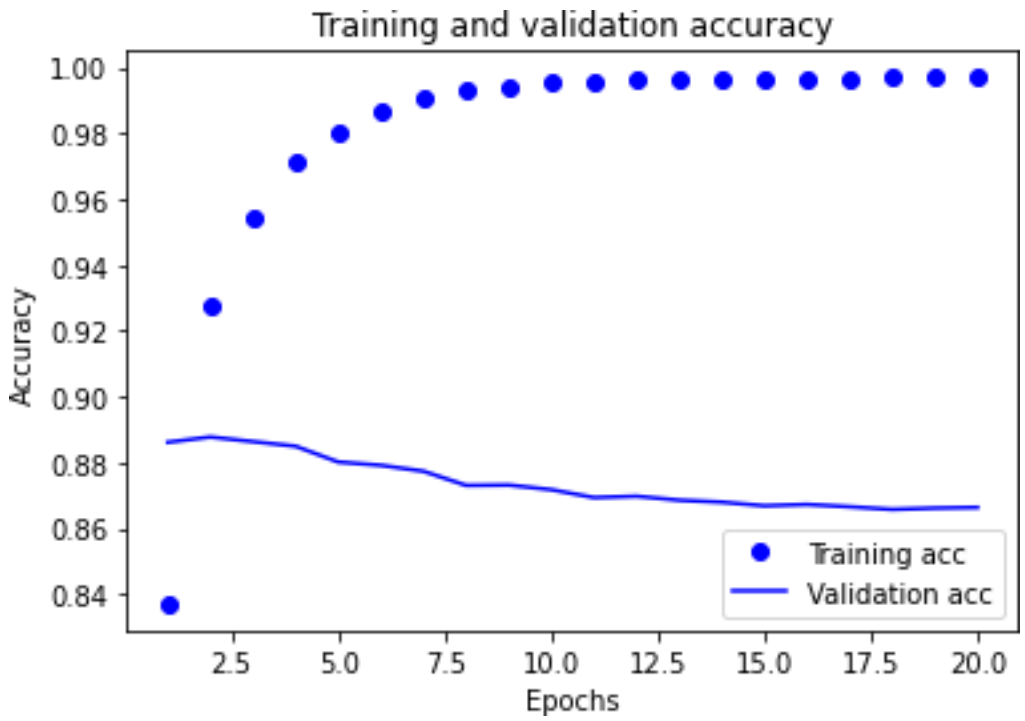
[16]: [0.11861928552389145, 0.8544399738311768]

4 Lets consider adding dropout layer & Regularizers

[17]:
```
from tensorflow import keras
from tensorflow.keras import layers
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.keras import regularizers
```

```
model = keras.Sequential()
model.add(Dense(32,activation='tanh', activity_regularizer=regularizers.L2(0.
↪01)))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer="adam", #changing optimizer to 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=0,
                     batch_size=256,
                     validation_data=(x_val, y_val))
```

[18]:
```
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()
```



[ ]: When I tried to apply the dropout layer, the accuracy of my findings validation only increased by a decimal, to 87%.

[19]:
```
results = model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 1s 666us/step - loss: 0.2594 -
accuracy: 0.5007
```

[20]:
```
results
```

[20]: [0.2594389319419861, 0.5006800293922424]

[ ]:

[21]:
```
from tensorflow import keras
from tensorflow.keras import layers
from keras.layers import Dense
from keras.layers import Dropout


model = keras.Sequential()
model.add(Dense(32,activation="tanh"))
model.add(Dropout(0.5))
```

```
model.add(Dense(32,activation="tanh",kernel_regularizer=regularizers.L1(0.01),
 ↪activity_regularizer=regularizers.L2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(32,activation="tanh"))
model.add(Dense(1, activation="sigmoid"))

model.compile(optimizer="adam", #changing optimizer to 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=256,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
59/59 [==============================] - 2s 29ms/step - loss: 1.5048 - accuracy:
0.7731 - val_loss: 1.1703 - val_accuracy: 0.8679
Epoch 2/20
59/59 [==============================] - 1s 19ms/step - loss: 0.9355 - accuracy:
0.8978 - val_loss: 0.7132 - val_accuracy: 0.8889
Epoch 3/20
59/59 [==============================] - 1s 17ms/step - loss: 0.5324 - accuracy:
0.9254 - val_loss: 0.3937 - val_accuracy: 0.8891
Epoch 4/20
59/59 [==============================] - 1s 19ms/step - loss: 0.2662 - accuracy:
0.9411 - val_loss: 0.2013 - val_accuracy: 0.8901
Epoch 5/20
59/59 [==============================] - 1s 22ms/step - loss: 0.1247 - accuracy:
0.9553 - val_loss: 0.1275 - val_accuracy: 0.8874
Epoch 6/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0844 - accuracy:
0.9621 - val_loss: 0.1169 - val_accuracy: 0.8867
Epoch 7/20
59/59 [==============================] - 1s 21ms/step - loss: 0.0716 - accuracy:
0.9673 - val_loss: 0.1141 - val_accuracy: 0.8872
Epoch 8/20
59/59 [==============================] - 2s 28ms/step - loss: 0.0623 - accuracy:
```

```
0.9726 - val_loss: 0.1110 - val_accuracy: 0.8860
Epoch 9/20
59/59 [==============================] - 2s 33ms/step - loss: 0.0560 - accuracy:
0.9763 - val_loss: 0.1101 - val_accuracy: 0.8845
Epoch 10/20
59/59 [==============================] - 2s 28ms/step - loss: 0.0507 - accuracy:
0.9793 - val_loss: 0.1101 - val_accuracy: 0.8832
Epoch 11/20
59/59 [==============================] - 1s 21ms/step - loss: 0.0464 - accuracy:
0.9822 - val_loss: 0.1103 - val_accuracy: 0.8818
Epoch 12/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0424 - accuracy:
0.9854 - val_loss: 0.1128 - val_accuracy: 0.8778
Epoch 13/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0394 - accuracy:
0.9881 - val_loss: 0.1117 - val_accuracy: 0.8780
Epoch 14/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0363 - accuracy:
0.9885 - val_loss: 0.1117 - val_accuracy: 0.8775
Epoch 15/20
59/59 [==============================] - 2s 26ms/step - loss: 0.0342 - accuracy:
0.9907 - val_loss: 0.1121 - val_accuracy: 0.8779
Epoch 16/20
59/59 [==============================] - 2s 26ms/step - loss: 0.0320 - accuracy:
0.9919 - val_loss: 0.1129 - val_accuracy: 0.8764
Epoch 17/20
59/59 [==============================] - 2s 26ms/step - loss: 0.0302 - accuracy:
0.9923 - val_loss: 0.1119 - val_accuracy: 0.8751
Epoch 18/20
59/59 [==============================] - 1s 20ms/step - loss: 0.0291 - accuracy:
0.9930 - val_loss: 0.1140 - val_accuracy: 0.8752
Epoch 19/20
59/59 [==============================] - 1s 19ms/step - loss: 0.0275 - accuracy:
0.9923 - val_loss: 0.1137 - val_accuracy: 0.8742
Epoch 20/20
59/59 [==============================] - 1s 20ms/step - loss: 0.0265 - accuracy:
0.9941 - val_loss: 0.1141 - val_accuracy: 0.8741
```

5 Summary

Here is the brief summary about my assignment.

–> Firstly we imported the keras library from tensorflow module.

from tensorflow import keras from tensorflow.keras import layers from keras.layers import Dense from keras.layers import Dropout

–> For Implementing the neural networks, we need the layers

1. input layer – using keras we try to create a model that starts with input represented using "Keras.Sequential" model = keras.Sequential()
2. hidden layer – we will add layers using the format "model.add(Dense(32,activation='tanh'))" model.add(Dense(32,activation='tanh')) model.add(Dense(32,activation='tanh')) model.add(Dense(32,activation='tanh'))
3. Output layer – The ouput layer will have the 1 units which produces the unit, generally represent using "model.add(Dense(1, activation='sigmoid'))" –> Here I would like briefly explain "model.add(Dense(32,activation='tanh'))".

Basically, we are adding a dense layer that contains 32 hidden units. It contains "tanh activation function".

The Activation Function's main responsibility is to convert the node's summed weighted input into an output value that may either be passed to the following hidden layer or used as output.

Activation function Reference: www.v7labs.com/blog/neural-networks-activation-functions

—> when we say 2 or 3 hidden layers, it contains above definition 2 or 3 times. —> Familar activation function are relu, tanh, sigmoid —> Preferably output layer will have 1 units that products the result and mostly people use sigmoid.

model.compile(optimizer="adam", #changing optimizer to ADAM loss="mean_squared_error", metrics=["accuracy"])

–> The above statement says that it is going to compile the network using adam optimizer with loss (mse) and accuracy metric.

Optimizer: Optimizers are Classes or methods used to change the attributes of your machine/deep learning model such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster. Available optimizers are listed below

SGD RMSprop Adam Adadelta Adagrad Adamax Nadam Ftrl Optimizers Reference : https://keras.io/api/optimizers/ analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/

Loss: The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

Available losses: we have a bunch of probabilist loss and regression loss. In the below reference it clearly explains.

Reference: https://keras.io/api/losses/

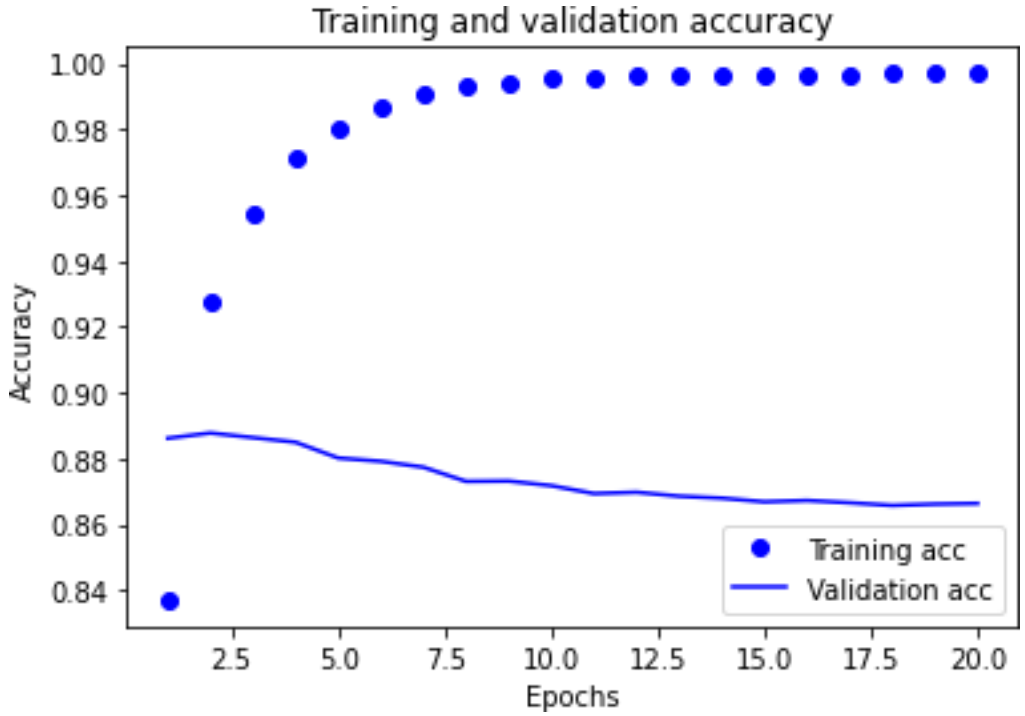
Below piece are code are splitting the data for vaidation from traininig x_val = x_train[:10000] partial_x_train = x_train[10000:] y_val = y_train[:10000] partial_y_train = y_train[10000:]

—> Lastly we train the model using the model.fit. That takes training data, also it check with validation during every eopoch and results will be plotted. history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=256, validation_data=(x_val, y_val))

—> The below graphs displays the plots for accuracy and loss.

[22]:
```
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()
```



[23]:
```
results = model.evaluate(x_test, y_test)
results
```

```
782/782 [==============================] - 1s 922us/step - loss: 0.1240 -
accuracy: 0.8619
```

[23]: [0.12399397790431976, 0.8619199991226196]

6 Conclusion:

1.A single layer and three layers are used to create neural networks. 2. Tanh is used for activation functions rather than relu 3. Rmsprop is substituted with the optimizer adam. 4. In the single layer and three layers models, the dropout layer is inserted with 0.4 and 0.5, respectively. 5. Regularizers L1 and L2 are utilized Finally, I can say that I obtained a training accuracy of 99% for this IMDB, and when comparing the two validation approaches, the single layer approach's accuracy touches

86.8, while the three layered approach's accuracy touches 86.9. We may expect that this can be improved with the addition of new data. At first, I chose overfitting, but after adding dropout layers, the accuracy increased to 87 when I attempted using dropouts and regularizers L1 and L2 with a three-layered technique.

Approach	**Training Accuracy**	**Validation Accuracy**	**Observations**
Single layer, Activation-tanh Optimizer-Adam Loss-mse	99.80	86.8	
Single Layer, Same as above with dropout and regularization	99.83	86.9	Here, I noticed slight change in training accuracy but adding validation accuracy drop. I assume this single layer model holds good originally, when we try to increase by adding dropouts and regularizers it diminishes
Three layer, Activation-tanh Optimizer-Adam Loss-mse With dropouts and regularizers	99.76	87	This approach holds good with training and validation accuracy.