Assignment 1

Name: Priyanka

Advance Machine Learning

In [38]:
```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

In [39]:
```
train_labels[0]
```

Out[39]:
```
1
```

In [40]:
```
max([max(sequence) for sequence in train_data])
```

Out[40]:
```
9999
```

In [41]:
```
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]])
```

In [42]:
```
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)
```

In [43]:
```
x_train[0]
```

Out[43]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [44]:
```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

In [45]:
```
from tensorflow import keras
from tensorflow.keras import layers

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

In [49]:
```
model.compile(optimizer="adam",
              loss="mean_squared_error",
              metrics=["accuracy"])
```

Validating the approach

In [50]:
```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [51]:
```
## model planned to train with 20 epoch with batch size of 256

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 [==============================] - 1s 10ms/step - loss: 0.1068 - accurac
y: 0.8577 - val_loss: 0.0846 - val_accuracy: 0.8854
Epoch 2/20
59/59 [==============================] - 0s 7ms/step - loss: 0.0468 - accuracy:
0.9401 - val_loss: 0.0927 - val_accuracy: 0.8776
Epoch 3/20
59/59 [==============================] - 0s 7ms/step - loss: 0.0288 - accuracy:
0.9678 - val_loss: 0.0995 - val_accuracy: 0.8752
Epoch 4/20
59/59 [==============================] - 0s 7ms/step - loss: 0.0212 - accuracy:
0.9768 - val_loss: 0.1122 - val_accuracy: 0.8645
Epoch 5/20
59/59 [==============================] - 0s 7ms/step - loss: 0.0201 - accuracy:
0.9777 - val_loss: 0.1130 - val_accuracy: 0.8686
Epoch 6/20
59/59 [==============================] - 0s 7ms/step - loss: 0.0204 - accuracy:
0.9768 - val_loss: 0.1159 - val_accuracy: 0.8652
Epoch 7/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0194 - accuracy:
0.9772 - val_loss: 0.1155 - val_accuracy: 0.8681
Epoch 8/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0168 - accuracy:
0.9809 - val_loss: 0.1223 - val_accuracy: 0.8605
Epoch 9/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0147 - accuracy:
0.9840 - val_loss: 0.1253 - val_accuracy: 0.8584
Epoch 10/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0122 - accuracy:
0.9872 - val_loss: 0.1219 - val_accuracy: 0.8636
Epoch 11/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0114 - accuracy:
0.9879 - val_loss: 0.1229 - val_accuracy: 0.8637
Epoch 12/20
59/59 [==============================] - 1s 10ms/step - loss: 0.0111 - accurac
y: 0.9880 - val_loss: 0.1270 - val_accuracy: 0.8611
Epoch 13/20
59/59 [==============================] - 1s 10ms/step - loss: 0.0116 - accurac
y: 0.9878 - val_loss: 0.1270 - val_accuracy: 0.8609
Epoch 14/20
59/59 [==============================] - 1s 10ms/step - loss: 0.0125 - accurac
y: 0.9865 - val_loss: 0.1284 - val_accuracy: 0.8601
Epoch 15/20
59/59 [==============================] - 1s 9ms/step - loss: 0.0109 - accuracy:
0.9887 - val_loss: 0.1347 - val_accuracy: 0.8534
Epoch 16/20
59/59 [==============================] - 1s 9ms/step - loss: 0.0117 - accuracy:
0.9873 - val_loss: 0.1308 - val_accuracy: 0.8587
Epoch 17/20
```

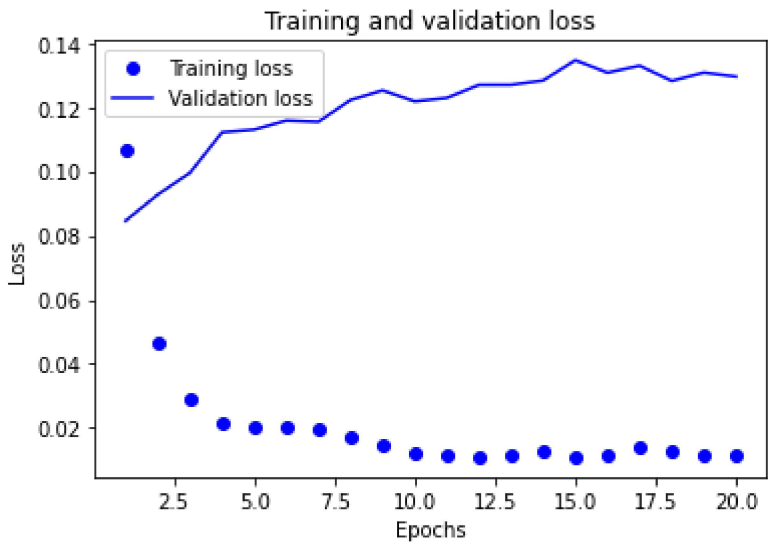
```
59/59 [==============================] - 1s 9ms/step - loss: 0.0138 - accuracy:
0.9848 - val_loss: 0.1330 - val_accuracy: 0.8570
Epoch 18/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0129 - accuracy:
0.9859 - val_loss: 0.1282 - val_accuracy: 0.8607
Epoch 19/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0118 - accuracy:
0.9873 - val_loss: 0.1308 - val_accuracy: 0.8584
Epoch 20/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0115 - accuracy:
0.9877 - val_loss: 0.1296 - val_accuracy: 0.8601
```

In [52]:
```
history_dict = history.history
history_dict.keys()
```

Out[52]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

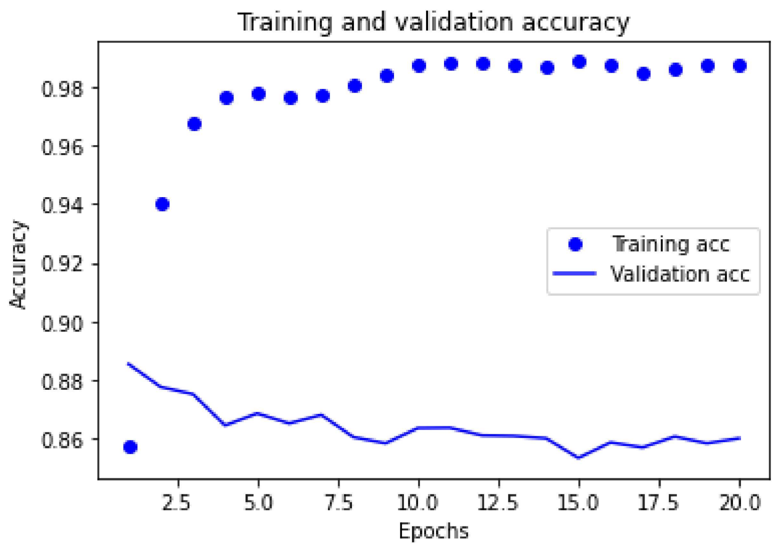
Plotting the train & Validation loss

In [53]:
```
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 the training and validation accuracy

In [54]:
```
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()
```



In [55]:
```
results = model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 1s 960us/step - loss: 0.1417 - accur
acy: 0.8470
```

In [56]:
```
results
```

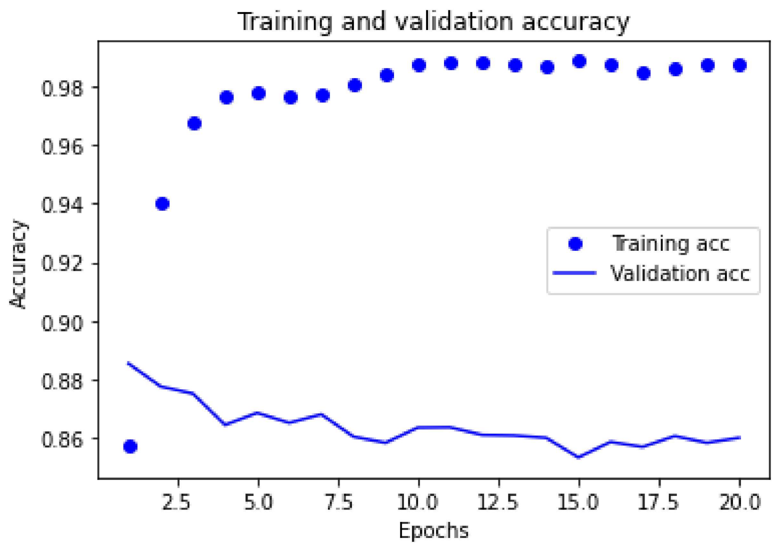
Out[56]:
```
[0.14169913530349731, 0.847000002861023]
```

Combining all code together along with dropout layer

In [61]:
```
## Libraries required for setting up an environment

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

# Neural network implementation using 3 layered approach with a single dropout la
#######################################
model = keras.Sequential()
model.add(Dense(32,activation='tanh'))
model.add(Dropout(0.5))
#kernel_regularizer=regularizers.L1(0.01), activity_regularizer=regularizers.L2(0
model.add(Dense(32,activation='tanh',kernel_regularizer=regularizers.L1(0.01), ac
model.add(Dropout(0.5))
model.add(Dense(32,activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
########################################

# Here for compilation we used optimizer "adagrad", mean squared error loss and a
########################################
model.compile(optimizer="adam",
              loss="mean_squared_error",
              metrics=["accuracy"])
########################################

## splitting the data
#######################################
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
#######################################

# Train a neural network
#####################################################
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=256,
                    validation_data=(x_val, y_val))
#####################################################

# plotting the 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()
####################################################


# Evaluating the results
results = model.evaluate(x_test, y_test)
results
```

```
Epoch 1/20
59/59 [==============================] - 1s 13ms/step - loss: 1.4738 - accura
cy: 0.7827 - val_loss: 1.1340 - val_accuracy: 0.8830
Epoch 2/20
59/59 [==============================] - 0s 8ms/step - loss: 0.9096 - accurac
y: 0.9031 - val_loss: 0.6972 - val_accuracy: 0.8869
Epoch 3/20
59/59 [==============================] - 0s 7ms/step - loss: 0.5177 - accurac
y: 0.9311 - val_loss: 0.3846 - val_accuracy: 0.8884
Epoch 4/20
59/59 [==============================] - 0s 7ms/step - loss: 0.2560 - accurac
y: 0.9467 - val_loss: 0.1964 - val_accuracy: 0.8881
Epoch 5/20
59/59 [==============================] - 0s 7ms/step - loss: 0.1225 - accurac
y: 0.9577 - val_loss: 0.1300 - val_accuracy: 0.8887
Epoch 6/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0852 - accurac
y: 0.9636 - val_loss: 0.1183 - val_accuracy: 0.8879
Epoch 7/20
59/59 [==============================] - 0s 7ms/step - loss: 0.0731 - accurac
y: 0.9673 - val_loss: 0.1133 - val_accuracy: 0.8865
Epoch 8/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0637 - accurac
y: 0.9733 - val_loss: 0.1112 - val_accuracy: 0.8853
Epoch 9/20
59/59 [==============================] - 1s 9ms/step - loss: 0.0577 - accurac
y: 0.9754 - val_loss: 0.1095 - val_accuracy: 0.8844
Epoch 10/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0519 - accurac
y: 0.9809 - val_loss: 0.1095 - val_accuracy: 0.8817
Epoch 11/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0471 - accurac
y: 0.9834 - val_loss: 0.1115 - val_accuracy: 0.8808
Epoch 12/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0434 - accurac
y: 0.9854 - val_loss: 0.1123 - val_accuracy: 0.8782
Epoch 13/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0398 - accurac
y: 0.9881 - val_loss: 0.1121 - val_accuracy: 0.8788
Epoch 14/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0378 - accurac
y: 0.9885 - val_loss: 0.1112 - val_accuracy: 0.8776
Epoch 15/20
59/59 [==============================] - 1s 9ms/step - loss: 0.0346 - accurac
y: 0.9902 - val_loss: 0.1128 - val_accuracy: 0.8777
Epoch 16/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0325 - accurac
y: 0.9915 - val_loss: 0.1133 - val_accuracy: 0.8762
```

```
Epoch 17/20
59/59 [==============================] - 0s 7ms/step - loss: 0.0305 - accurac
y: 0.9928 - val_loss: 0.1131 - val_accuracy: 0.8769
Epoch 18/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0286 - accurac
y: 0.9941 - val_loss: 0.1120 - val_accuracy: 0.8761
Epoch 19/20
59/59 [==============================] - 0s 8ms/step - loss: 0.0277 - accurac
y: 0.9943 - val_loss: 0.1133 - val_accuracy: 0.8737
Epoch 20/20
59/59 [==============================] - 1s 9ms/step - loss: 0.0263 - accurac
y: 0.9939 - val_loss: 0.1147 - val_accuracy: 0.8726
```



```
782/782 [==============================] - 1s 1ms/step - loss: 0.1240 - accurac
y: 0.8623
```

Out[61]:
```
[0.12398882955312729, 0.862280011177063]
```

Summary about the three-layered neural network for IMDB data:

• Initially we gathered required libraries for our neural network to be on track. During my study and little research, I can infer that TensorFlow has good support and implementation among other deep learning libraries like pytorch.

List of Imports are:

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

• we imported keras, keras.layers, Dense and Dropouts. Each of its own has significant importance in its implementation. Keras is the high-level API of TensorFlow 2: an approachable, highly productive interface for solving machine learning problems, with a focus on modern deep learning. The core data structures of Keras are layers and models. The simplest type of model is the Sequential model, a linear stack of layers. Dense represent the number of hidden units in the neural network. Dropout: The significance of dropout is taking a bunch of inputs or hidden layer input, random remove the connections. Now let's talk about the designing the neural network layers. model = keras.Sequential() ----- Sequential model is the simplest mode of keras, which is stack up the layers in the sequences. model.add(Dense(32,activation='tanh')) Stacking layers is easy using the. add function. Also 32 represents the number of hidden units and we use the activation function tanh. Before going to the next topic, I will describe the contents of a neural network.

1. Input layer -- where we provide our input to it. – here we provide vector representation of IMDB data
2. Hidden layers – it contains the number of dense units, and we can stack up as many layers as we want depending on the requirement.
3. Output layer – output layer, Preferably the output layer has 1 dense unit. Here in this task I tried to implement three layered approach as per the requirement given in the assignment. model = keras.Sequential([layers.Dense(32, activation="tanh"), layers.Dense(32, activation="tanh"), layers.Dense(32, activation="tanh"), layers.Dense(1, activation="sigmoid")])

The above code model initialized as sequential. And we stack up three layers with 32 dense units and tanh activation function. In the task, I implemented tanh instead of relu as suggested in the assignment. model.compile(optimizer="adagrad", loss="mean_squared_error", metrics= ["accuracy"]) The above piece of code uses an optimizer adagrad with mse loss. I still have a doubt here initially IMBD data uses a loss of binary_crossentrophy which is a probabilistic loss and what if we changed the regression loss. More information will be available in 2nd reference link. Optimizers are very important to minimize the error and we have different techniques/optimizers. For example, adam is considered as good optimizers among the different approaches. In this task, I used adagrad. More details about optimizers will be explained in the 3nd reference link. We split the data into training and validation part and the code below shows the split

x_val = x_train[:10000] partial_x_train = x_train[10000:] y_val = y_train[:10000] partial_y_train = y_train[10000:] Training the data history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=256, validation_data=(x_val, y_val)) The above line of code represent it will train the neural network with 20 epoch and batch size of 256 and parallely it compare with validation data. I used L1 and L2 regularizers but it does not gives much impact on the total validation accuracy.

Reference:

1. https://keras.io/about (https://keras.io/about)
2. https://keras.io/api/losses/ (https://keras.io/api/losses/)
3. https://keras.io/api/optimizers/ (https://keras.io/api/optimizers/)

In [ ]:

```
```

Conclusions

1. neural network designed with 3 layers

2. Activation functions tanh is used instead of relu

3. Optimizer adam is used instead of rmsprop

4. L1 & L2 regularizers are used

4. Dropout layer with 0.5 is used. That means we are dropping 50 percent of inputs during the training.

Final accuracy of 99.39 and validation accuracy of 87.2 is achieved using the above changes..

In [ ]:

```
```

In [ ]:

```
```