Assignment 1

Name: Pranitha Kasireddy

Student Id : 811319207

Advance Machine Learning

In [1]:
```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

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

Out[2]:
```
1
```

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

Out[3]:
```
9999
```

In [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]])
```

In [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)
```

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

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

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

In [8]:
```
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 [9]:
```
model.compile(optimizer="adam",
              loss="mean_squared_error",
              metrics=["accuracy"])
```

Validating your approach Setting aside a validation set

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

In [11]:
```
## 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 ━━━━━━━━━━━━━━━━━━━━ 6s 54ms/step - accuracy: 0.7683 - loss: 0.1596 - val_accu
racy: 0.8854 - val_loss: 0.0853
Epoch 2/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9402 - loss: 0.0480 - val_accu
racy: 0.8827 - val_loss: 0.0890
Epoch 3/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9671 - loss: 0.0300 - val_accu
racy: 0.8717 - val_loss: 0.1030
Epoch 4/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9768 - loss: 0.0221 - val_accu
racy: 0.8717 - val_loss: 0.1061
Epoch 5/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9815 - loss: 0.0173 - val_accu
racy: 0.8679 - val_loss: 0.1130
Epoch 6/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9842 - loss: 0.0149 - val_accu
racy: 0.8685 - val_loss: 0.1136
Epoch 7/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9810 - loss: 0.0164 - val_accu
racy: 0.8689 - val_loss: 0.1143
Epoch 8/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9844 - loss: 0.0147 - val_accu
racy: 0.8619 - val_loss: 0.1237
Epoch 9/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9830 - loss: 0.0157 - val_accu
racy: 0.8632 - val_loss: 0.1223
Epoch 10/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9817 - loss: 0.0162 - val_accu
racy: 0.8657 - val_loss: 0.1214
Epoch 11/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9856 - loss: 0.0142 - val_accu
racy: 0.8645 - val_loss: 0.1232
Epoch 12/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9885 - loss: 0.0112 - val_accu
racy: 0.8631 - val_loss: 0.1247
Epoch 13/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9895 - loss: 0.0103 - val_accu
racy: 0.8635 - val_loss: 0.1241
Epoch 14/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9908 - loss: 0.0093 - val_accu
racy: 0.8629 - val_loss: 0.1259
Epoch 15/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9898 - loss: 0.0100 - val_accu
racy: 0.8663 - val_loss: 0.1237
Epoch 16/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9893 - loss: 0.0108 - val_accu
racy: 0.8651 - val_loss: 0.1246
Epoch 17/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9898 - loss: 0.0101 - val_accu
racy: 0.8651 - val_loss: 0.1256
Epoch 18/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9893 - loss: 0.0099 - val_accu
racy: 0.8649 - val_loss: 0.1256
Epoch 19/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9844 - loss: 0.0143 - val_accu
```

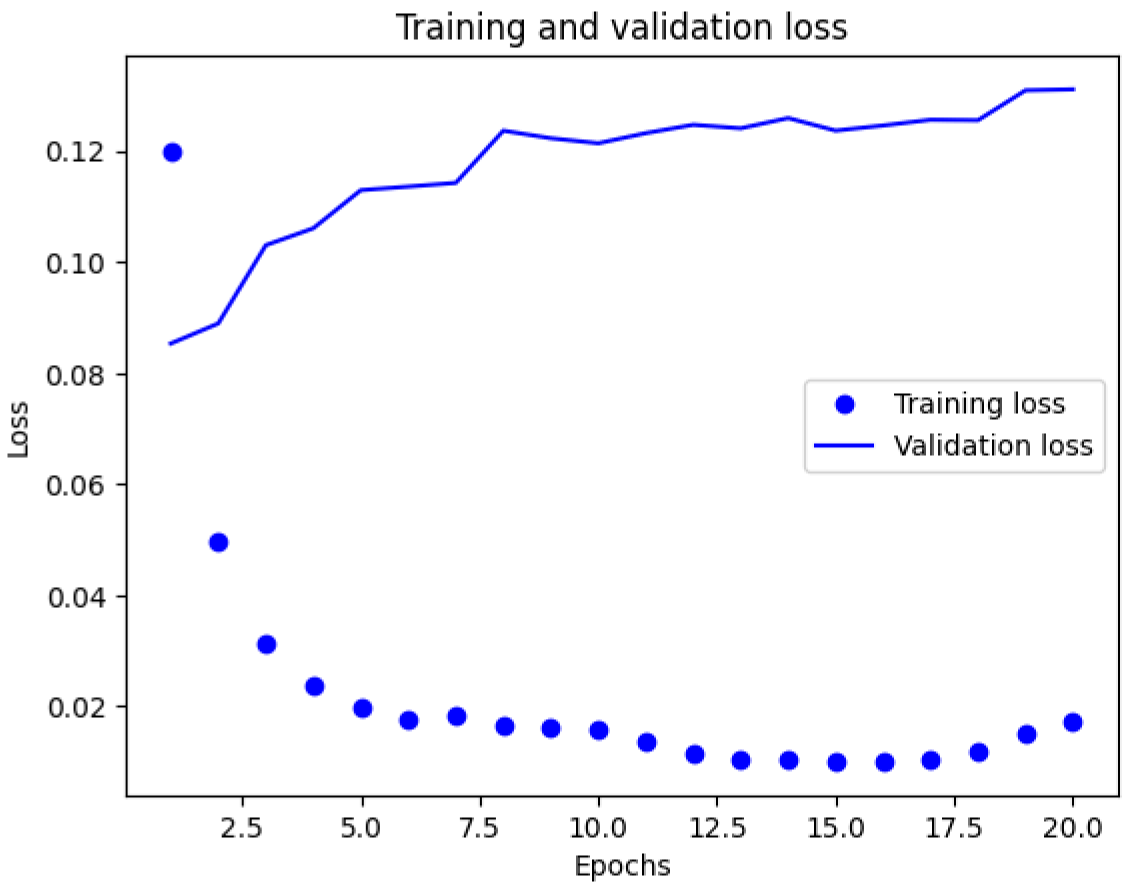
```
racy: 0.8593 - val_loss: 0.1309
Epoch 20/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9827 - loss: 0.0161 - val_accu
racy: 0.8590 - val_loss: 0.1311
```

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

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

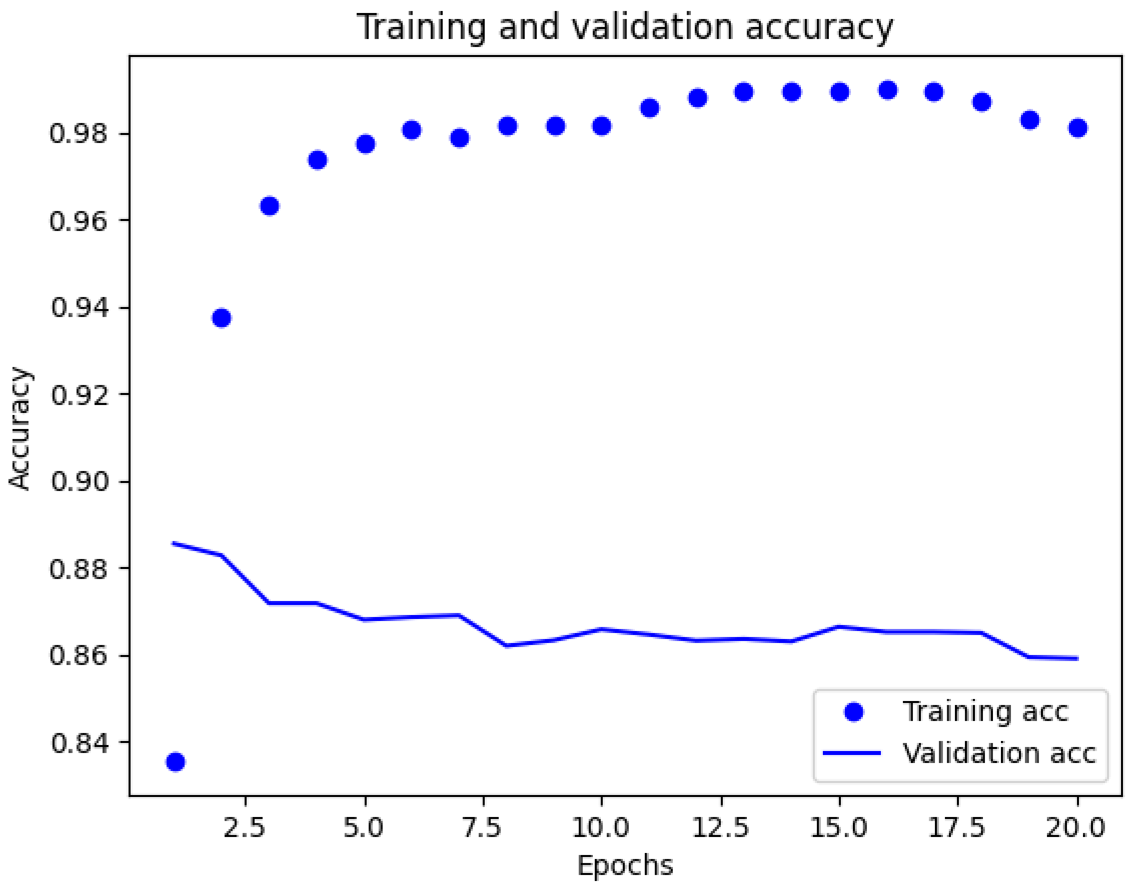
Plotting the train & Validation loss

In [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()
```



Plotting the training and validation accuracy

In [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()
```



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

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8439 - loss: 0.1461
```

In [16]:
```
results
```

Out[16]:
```
[0.1443871706724167, 0.8457599878311157]
```

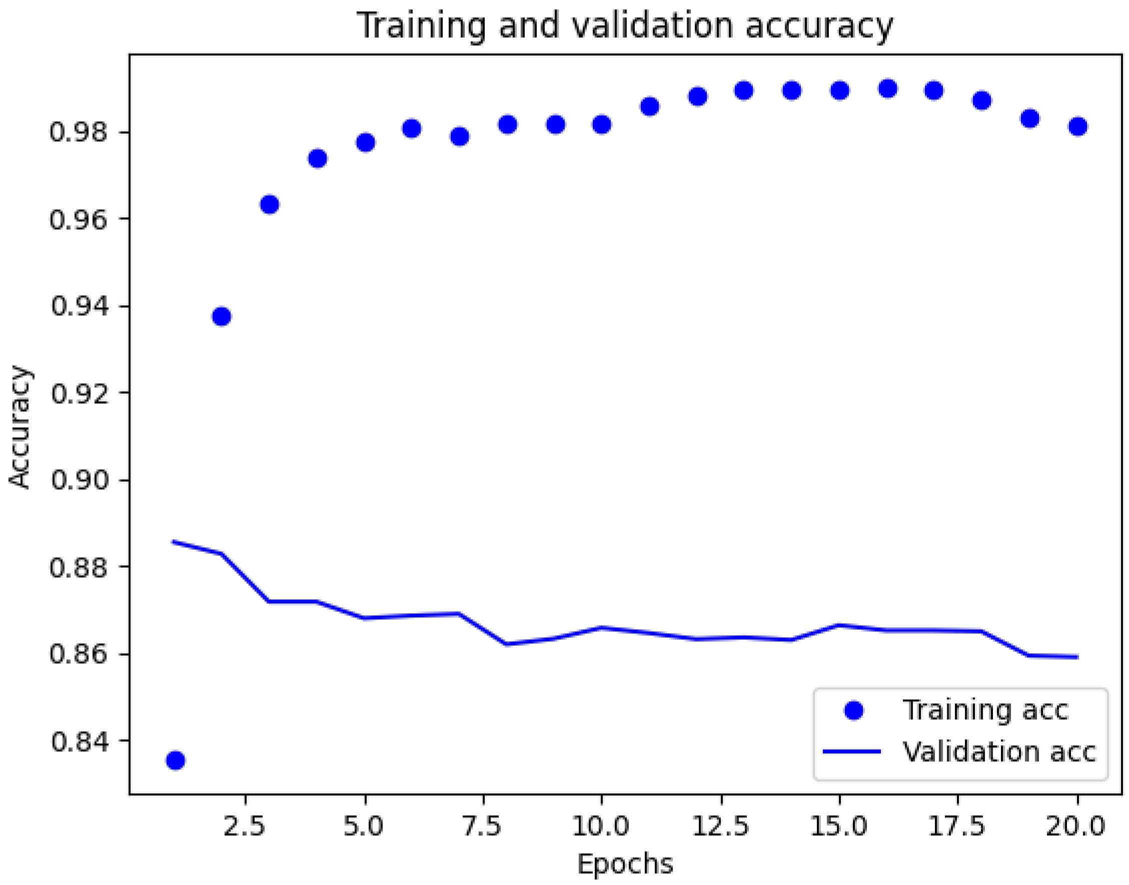
Combining all code together along with dropout layer

In [17]:
```
## The libraries needed to configure 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
#####################################

# Three-layered neural network implementation with a single dropout layer
#######################################
model = keras.Sequential()
model.add(Dense(32,activation='tanh'))
model.add(Dropout(0.5))
#kernel_regularizer=regularizers.L1(0.01), activity_regularizer=regularizers.L2(0.0
model.add(Dense(32,activation='tanh',kernel_regularizer=regularizers.L1(0.01), acti
model.add(Dropout(0.5))
model.add(Dense(32,activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
########################################

# Here, we employed accuracy measurements, mean squared error loss, and the optimiz
########################################
model.compile(optimizer="adam",
              loss="mean_squared_error",
              metrics=["accuracy"])
########################################

## dividing 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 Accuracy of Training and Validation
####################################################
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 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - accuracy: 0.5144 - loss: 4.2437 - val_accu
racy: 0.5604 - val_loss: 1.8258
Epoch 2/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5500 - loss: 1.9579 - val_accu
racy: 0.6117 - val_loss: 1.2528
Epoch 3/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.5820 - loss: 1.2517 - val_accu
racy: 0.6622 - val_loss: 0.9234
Epoch 4/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6813 - loss: 0.8926 - val_accu
racy: 0.8173 - val_loss: 0.6850
Epoch 5/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.8183 - loss: 0.6419 - val_accu
racy: 0.8638 - val_loss: 0.4787
Epoch 6/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8915 - loss: 0.4401 - val_accu
racy: 0.8774 - val_loss: 0.3246
Epoch 7/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9233 - loss: 0.2872 - val_accu
racy: 0.8790 - val_loss: 0.2227
Epoch 8/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9404 - loss: 0.1882 - val_accu
racy: 0.8815 - val_loss: 0.1656
Epoch 9/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9549 - loss: 0.1312 - val_accu
racy: 0.8837 - val_loss: 0.1404
Epoch 10/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9566 - loss: 0.1070 - val_accu
racy: 0.8815 - val_loss: 0.1302
Epoch 11/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9642 - loss: 0.0925 - val_accu
racy: 0.8794 - val_loss: 0.1266
Epoch 12/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9697 - loss: 0.0831 - val_accu
racy: 0.8793 - val_loss: 0.1219
Epoch 13/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9733 - loss: 0.0758 - val_accu
racy: 0.8790 - val_loss: 0.1185
Epoch 14/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9755 - loss: 0.0696 - val_accu
racy: 0.8771 - val_loss: 0.1193
Epoch 15/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9803 - loss: 0.0634 - val_accu
racy: 0.8775 - val_loss: 0.1192
Epoch 16/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9817 - loss: 0.0600 - val_accu
racy: 0.8721 - val_loss: 0.1192
Epoch 17/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9855 - loss: 0.0556 - val_accu
racy: 0.8706 - val_loss: 0.1188
Epoch 18/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9873 - loss: 0.0513 - val_accu
racy: 0.8690 - val_loss: 0.1190
Epoch 19/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9881 - loss: 0.0491 - val_accu
```

```
racy: 0.8679 - val_loss: 0.1195
Epoch 20/20
59/59 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9896 - loss: 0.0470 - val_accu
racy: 0.8658 - val_loss: 0.1180
```



```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8496 - loss: 0.1239
```

Out[17]: `[0.12222220748662949, 0.8534799814224243]`

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

• In order to get our neural network operating properly, we first acquired the necessary libraries. From my studies and limited investigation, I've concluded that, in comparison to other deep learning libraries like Pytorch, TensorFlow offers good implementation and support.

The Imports List is:

from keras import tensorflow

from import layers of tensorflow.keras

import from keras.layers Compact

import from keras.layers Dismissal

• 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
2. https://keras.io/api/losses/

3. https://keras.io/api/optimizers/

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..