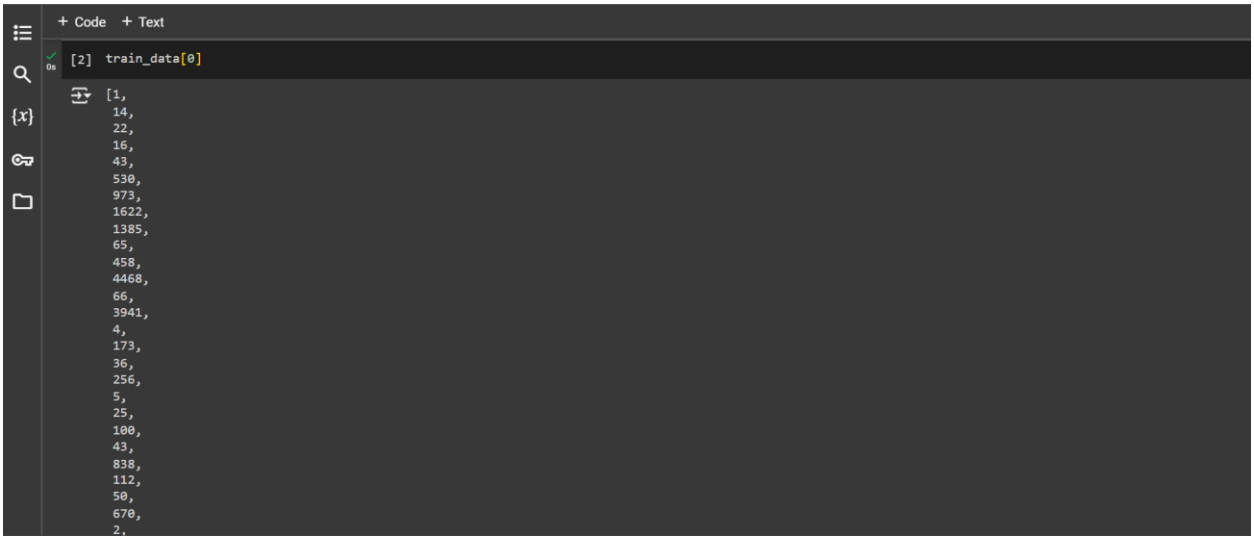
ASSIGNMENT -01
REPORT
NEURAL NETWORKS

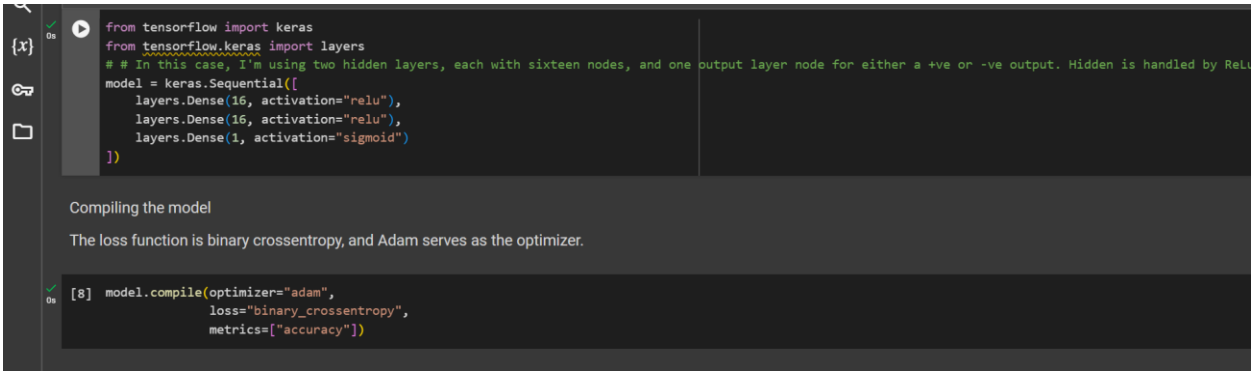
The given code has to import TensorFlow Keras and load the IMDB movie review dataset with only the 10,000 most frequent words. This then divides the data into 70% of training data with labels and 30% of test data with labels.



The line of code `train_data[0]` passes the first movie review from the training set. This review, as the previous ones, is represented as a sequence of integers, where each integer corresponds to a specific word in the IMDB dataset's vocabulary. The actual words have been transformed into integers according to a count vector, derived from the obtained data.

The code below gets the label of the first movie review from the training data and gets the maximum word index in the dataset. It then uses two operations of this dictionary to convert the first review back into text; hence, it gets the word-to-index dictionary from the IMDB dataset and reverses it to get the index-to word dictionary since it encoded the review using indices and not the actual words.

The provided recurrence turns the IMDB movie review sequences into the binary vectors of the length 10,000, where each word in a review sets the appropriate index. This vectorization is applied to the training set and then to the testing set and yields `x_training` structured as a matrix of size `(number of samples, 10000)`. Also the training and the testing labels are converted in float32 arrays, which loading the review data and the corresponding labels in a neural network.



```
from tensorflow import keras
from tensorflow.keras import layers
# # In this case, I'm using two hidden layers, each with sixteen nodes, and one output layer node for either a +ve or -ve output. Hidden is handled by ReLu
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

Compiling the model

The loss function is binary crossentropy, and Adam serves as the optimizer.

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

The code above first imports TensorFlow and from Keras builds and compiles a neural network model. It forms a sequence model with two hidden layers, each containing of 16 nodes with ReLU activation functions, and the final output layer with a single node, a sigmoid activation function due to the binary classification. The model is then defined to compile it with Adam optimizer, binary cross entropy loss and accuracy as the evaluating metrics before it can be trained for a task like determining whether a review is positive or negative about a movie.

+ Code + Text

```
[10] x_value = x_training[:10000]
     partial_x_training = x_training[10000:]
     y_value = y_training[:10000]
     partial_y_training = y_training[10000:]
```

Training the Model

We use 512 batches and 20 epochs to train the model.

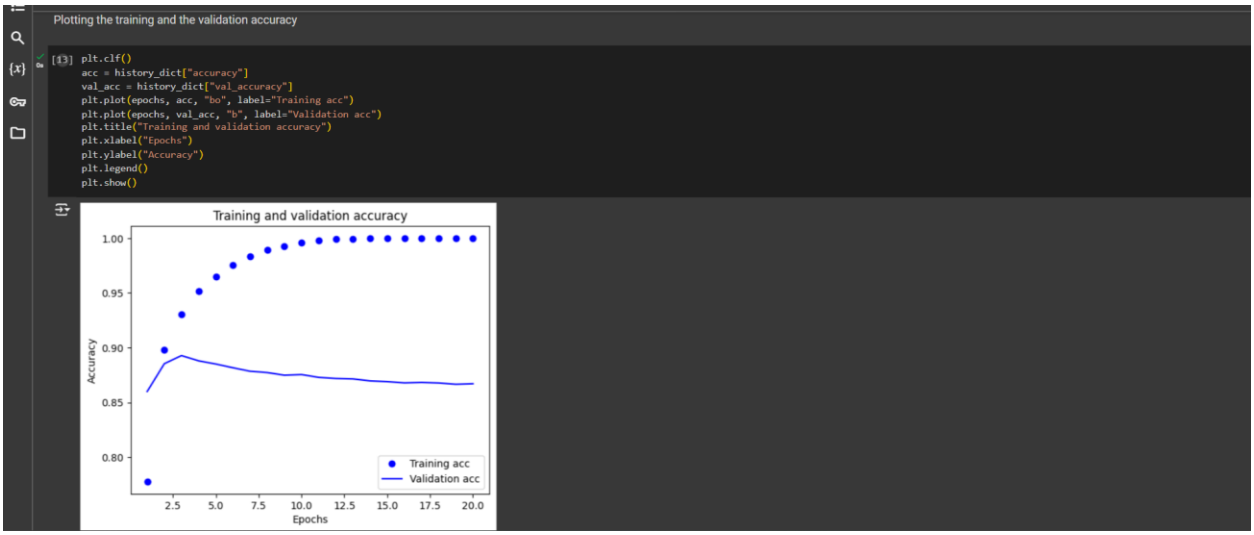
```
history = model.fit(partial_x_training,
                    partial_y_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))

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

```
Epoch 1/20
30/30 [==============================] - 7s 180ms/step - loss: 0.5503 - accuracy: 0.7778 - val_loss: 0.4046 - val_accuracy: 0.8600
Epoch 2/20
30/30 [==============================] - 1s 40ms/step - loss: 0.3076 - accuracy: 0.8979 - val_loss: 0.2990 - val_accuracy: 0.8854
Epoch 3/20
30/30 [==============================] - 1s 37ms/step - loss: 0.2093 - accuracy: 0.9307 - val_loss: 0.2767 - val_accuracy: 0.8928
Epoch 4/20
30/30 [==============================] - 1s 49ms/step - loss: 0.1570 - accuracy: 0.9515 - val_loss: 0.2790 - val_accuracy: 0.8880
```

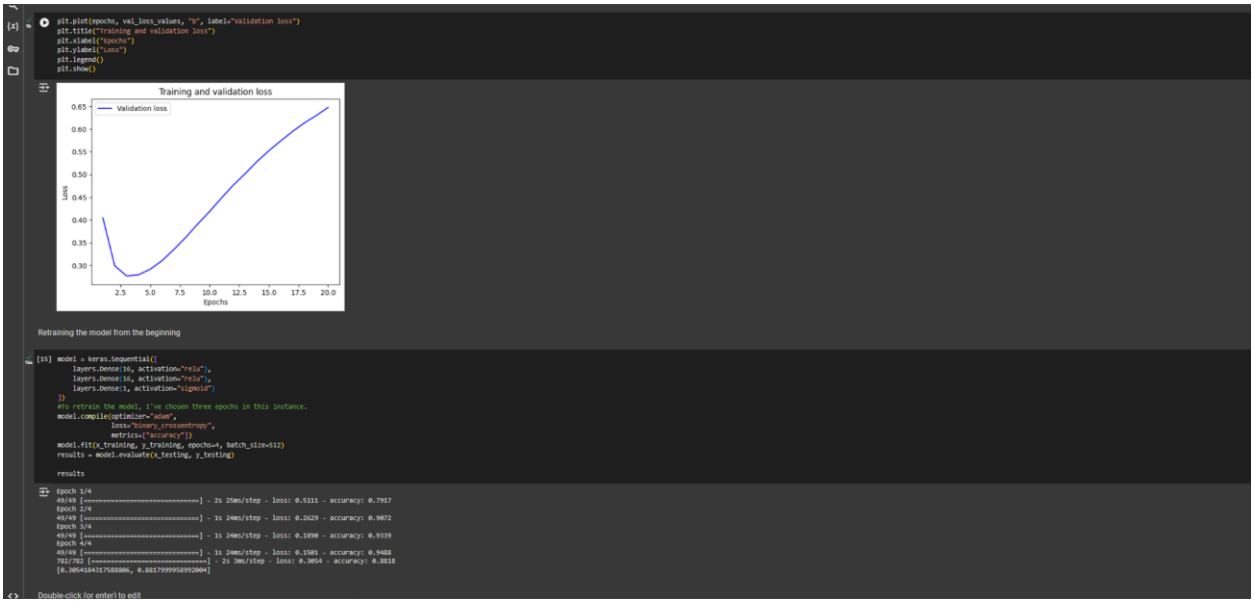
The given code is used to shuffle the IMDB movie review dataset and divide it into training set and validation set. It employs the first 10 000 samples for that purpose a validation set labelled as

`x_value , y_value`, whereas the rest of the samples serve as a training set labelled `partial_x_training , partial_y_training`. The trained neural network model is `model` and the training data is used to train the model for 20 epochs having a batch size of 512. While training, the accuracy and loss of the obtained model together with the results on both the training and validation datasets are recorded in the `history. history`. After training completes, `history_dict. Using the `keys()` function, the names of the dictionary containing these metrics are obtained so that later the training process of the model and the achieved performance can be analyzed and visualized.

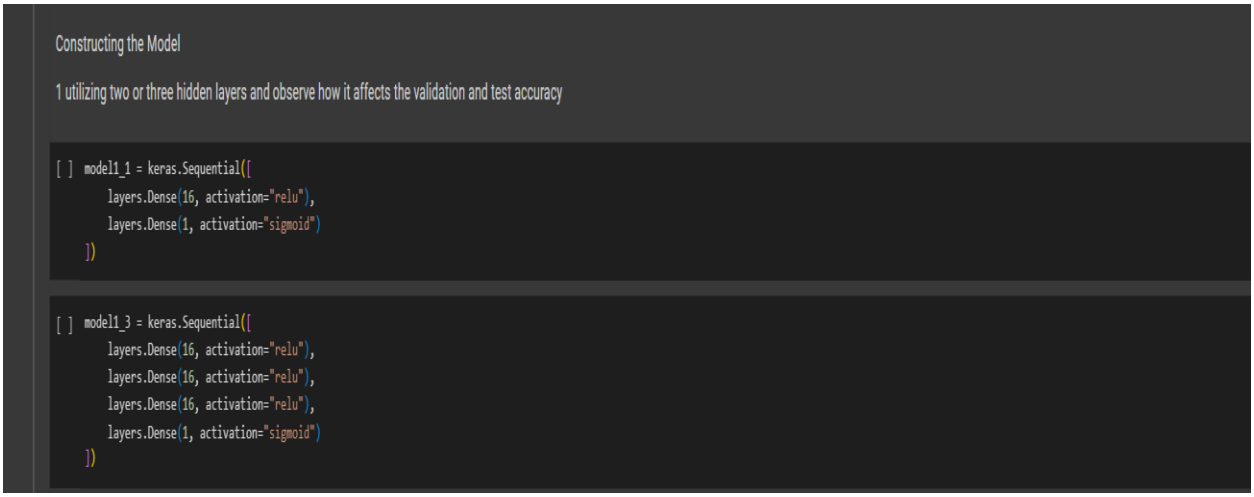
The attached code part employs Matplotlib to plot two arrays containing the training and validation loss of a given neural network trained on the IMDB movie review data. It first gets the training history as defined in (`history. history`) and extracts the training loss as defined in `loss_values` and the validation loss defined in `val_loss_values. `epochs` is the variable ranging over the all epochs starting from 1 to the number of epochs. Then, it uses blue circles (`"bo"`), to plot the values of training loss against the epochs in the plot and puts the label “Training loss” to the line in the plot. This sort of visualization assists in identifying the rate at which the model is learning during successive epochs and the trends of the training loss along with validation loss.



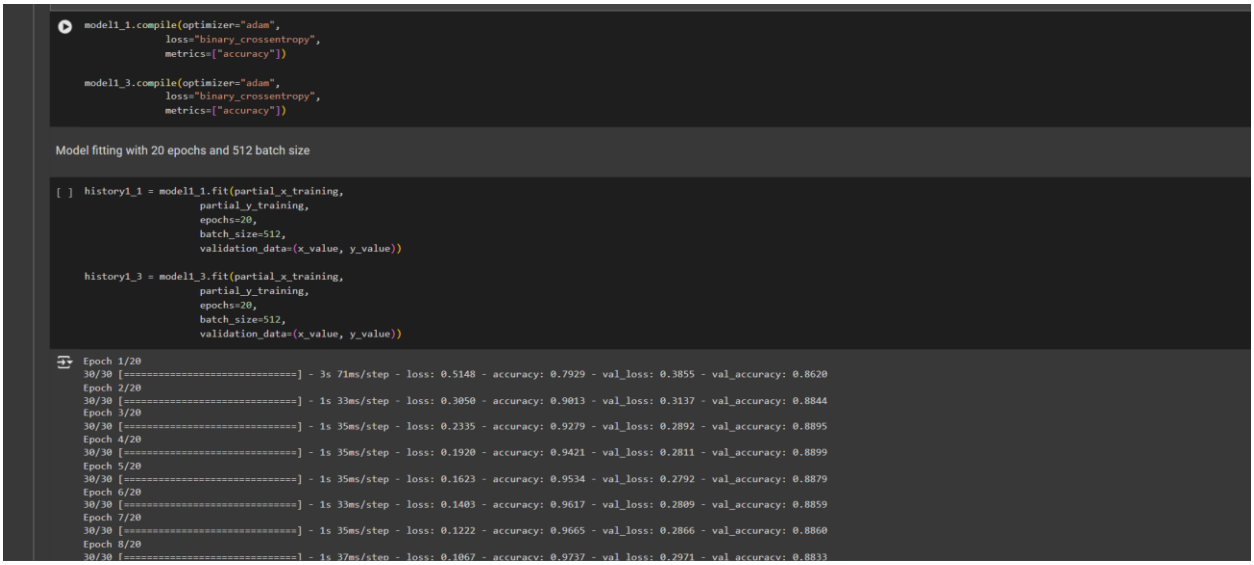
This code historical plot the training and validation accuracy while training a neural network model on the IMDB movie review dataset using Matplotlib. It first blanks all existing plots using the `plt. clf()`. This is then followed by the value in the training history which contains the training accuracy denoted as ‘acc’ and the validation accuracy denoted as ‘val_acc’. Next, it graphs the training accuracy against the number of epochs (`epochs`) using blue circles (`"bo"`), and labels this line as “Training acc” in legends. Similarly, it draws the validation accuracy against epochs with a solid blue color using the code “‘b”’ and is titled as ‘Validation accuracy’ in the legend. Fields in the plot itself include the plot title as ‘Training/validation accuracy,’ axes labelled as `xlabel` for the epochs and `ylabel` for the accuracy. The legend is used to unique the individual lines representing the training and validation accuracy. The following visualization enables to evaluate the quality of the model on the training and validation datasets in terms of epochs.



Certainly! The provided code snippet accomplishes the following tasks: First, it employs Matplotlib to create a plot of `val_loss_values against epochs for the model in the way it fits the validation data during the epochs of training. The plot's title is 'Training and validation loss,' and on the axes, the x-axis is epochs while the y-axis is loss; the legend on the right of the plot states validation loss. Next, it establishes a new neural network model (`model`) with two hidden layers each consisting of 16 nodes with ReLU activation function and an output layer having a single node with sigmoid activation function as it is binary classification problem. The model is compiled with the Adam optimizer, binary cross entropy as loss function and accuracy as the metric function. It then refines the model on the same dataset used for training it a before over the complete training data consisting of `x_training`, `y_training` for another 4 epochs with a batch of size 512 with an attempt to correct the identified anomalies. Last but not the least, the performance of the model is checked on a test data set x_testing, and y_testing through the command model. A forward pass of the model's computations is done by `forward()`, and the outputs, generally, loss and/or accuracy, are held in "results".



model1_1 and model1_3 are two neural network models with different architectures: The first model, which is model1_1, has one hidden layer and the last model that is model1_3 has three hidden layers. As the last layer, both models provide an output layer for binary classification tasks, which uses activation through the sigmoid function to output probabilities. It's possible to assemble these models, train them on data sets and compare them in their effectiveness in specific tasks, like sentiment prediction from movies.



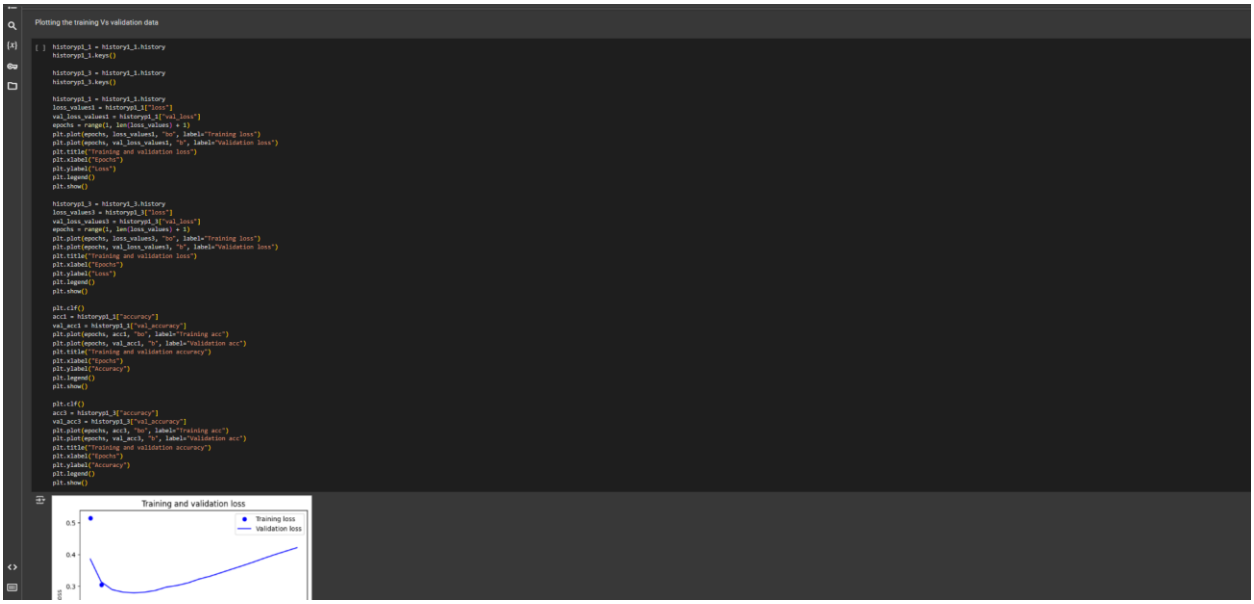
```
model1_1.compile(optimizer="adam",
                 loss="binary_crossentropy",
                 metrics=["accuracy"])

model1_3.compile(optimizer="adam",
                 loss="binary_crossentropy",
                 metrics=["accuracy"])
```

Model fitting with 20 epochs and 512 batch size

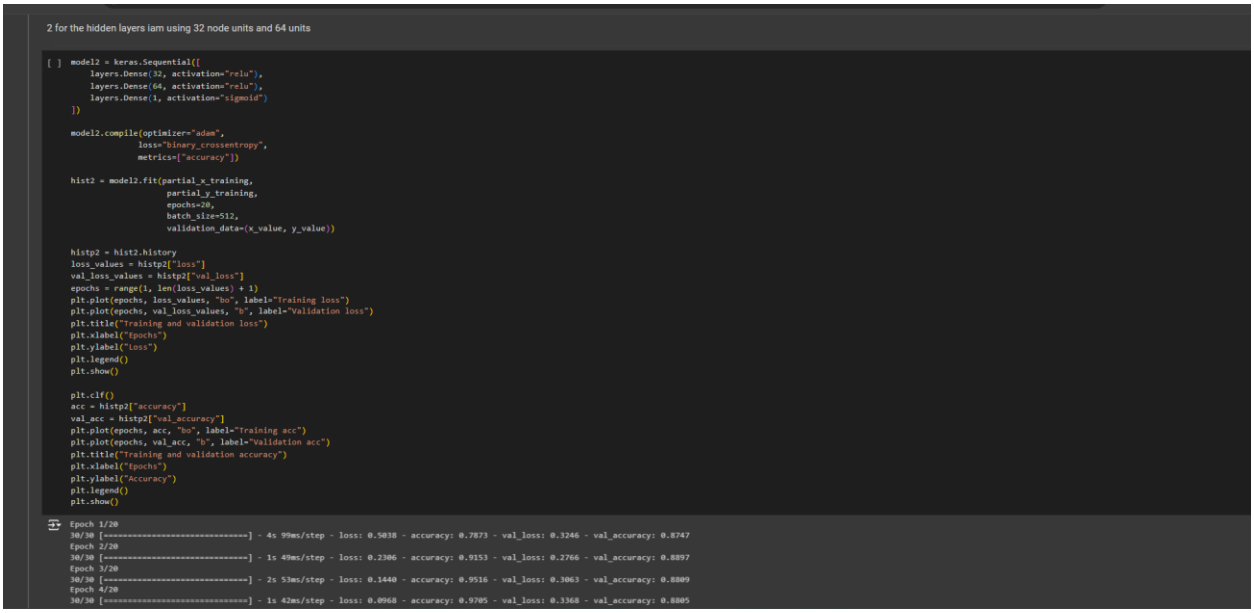
```
[ ] history1_1 = model1_1.fit(partial_x_training,
                      partial_y_training,
                      epochs=20,
                      batch_size=512,
                      validation_data=(x_value, y_value))

    history1_3 = model1_3.fit(partial_x_training,
                      partial_y_training,
                      epochs=20,
                      batch_size=512,
                      validation_data=(x_value, y_value))
```

```
Epoch 1/20
30/30 [==============================] - 3s 71ms/step - loss: 0.5148 - accuracy: 0.7929 - val_loss: 0.3855 - val_accuracy: 0.8620
Epoch 2/20
30/30 [==============================] - 1s 33ms/step - loss: 0.3050 - accuracy: 0.9013 - val_loss: 0.3137 - val_accuracy: 0.8844
Epoch 3/20
30/30 [==============================] - 1s 35ms/step - loss: 0.2335 - accuracy: 0.9279 - val_loss: 0.2892 - val_accuracy: 0.8895
Epoch 4/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1920 - accuracy: 0.9421 - val_loss: 0.2811 - val_accuracy: 0.8899
Epoch 5/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1623 - accuracy: 0.9534 - val_loss: 0.2792 - val_accuracy: 0.8879
Epoch 6/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1403 - accuracy: 0.9617 - val_loss: 0.2809 - val_accuracy: 0.8859
Epoch 7/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1222 - accuracy: 0.9665 - val_loss: 0.2866 - val_accuracy: 0.8860
Epoch 8/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1067 - accuracy: 0.9737 - val_loss: 0.2971 - val_accuracy: 0.8833
```

Let us review these two code snippets Although both of them create and train two distinct neural network models of binary classification (model1_1 and model1_3), they both work on the IMDB movie review dataset based on TensorFlow Keras. Both models are trained using Adam optimizer for gradient based optimization, with binary cross entropy as loss and accuracy as the measure of the model's performance during the training session. Each of the models is trained for 20 epochs and the size of the batch is 512 samples. Here, the training set which is partial_x_training, partial_y_training is used to train the model while the test set namely x_value, y_value is used to test the model on unseen data after each epoch. Information regarding the training progress as well as the validation loss and accuracy such as history1_1 and history1_3 for model 1. 1 and model 1. 3 respectively will allow the operator to compare which of the models is better able to learn from the data of the training set and generalize from it.

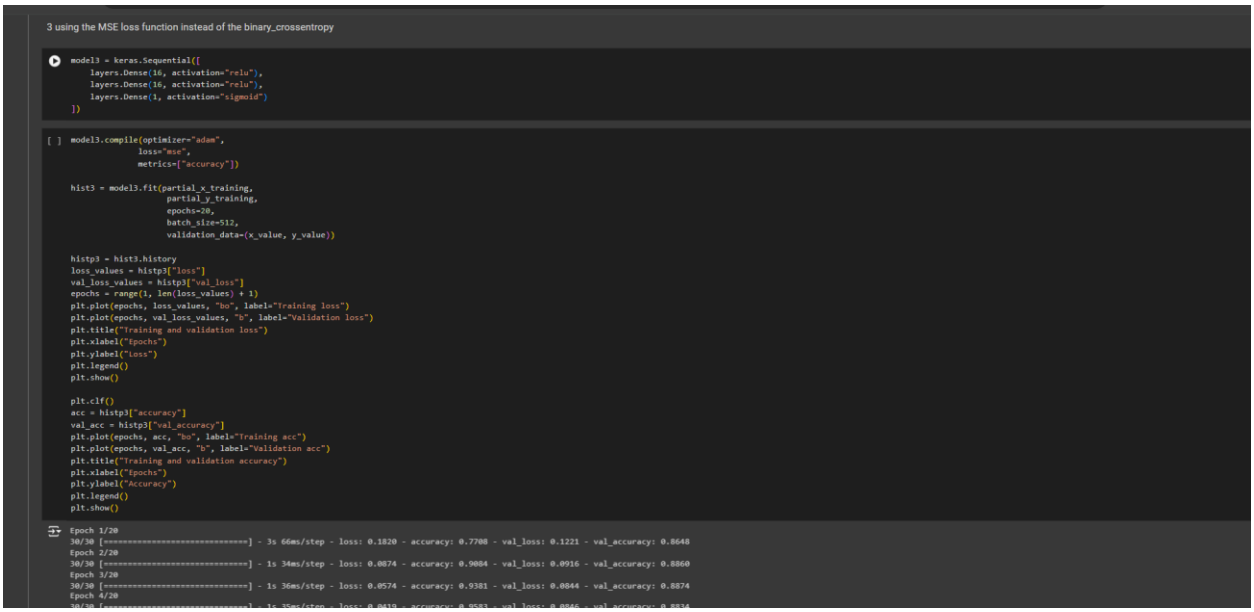


This code sequence implements the use of Matplotlib to plot the training and validation of the performance metrics namely, loss and accuracy for two different models namely `model1_1` and `model1_3` with the IMDB movie review training and validation dataset using TensorFlow Keras. , it performs the learning curves of the train and validation loss for `model1_1` and `model1_3` to show that model improved over epochs. The consequent plots are concerned with the training and the validation accuracy which shows in what extent each model predicts the outcomes as the training goes on. Every plot is provided with titles informing about the type of data as loss or accuracy, and legends help to distinguish the data type as the training or the validation one. These visualization are important to analyse the performance of the two models and decide about their generalization ability as well as the trends to be looked into as part of the future enhancements in the model.

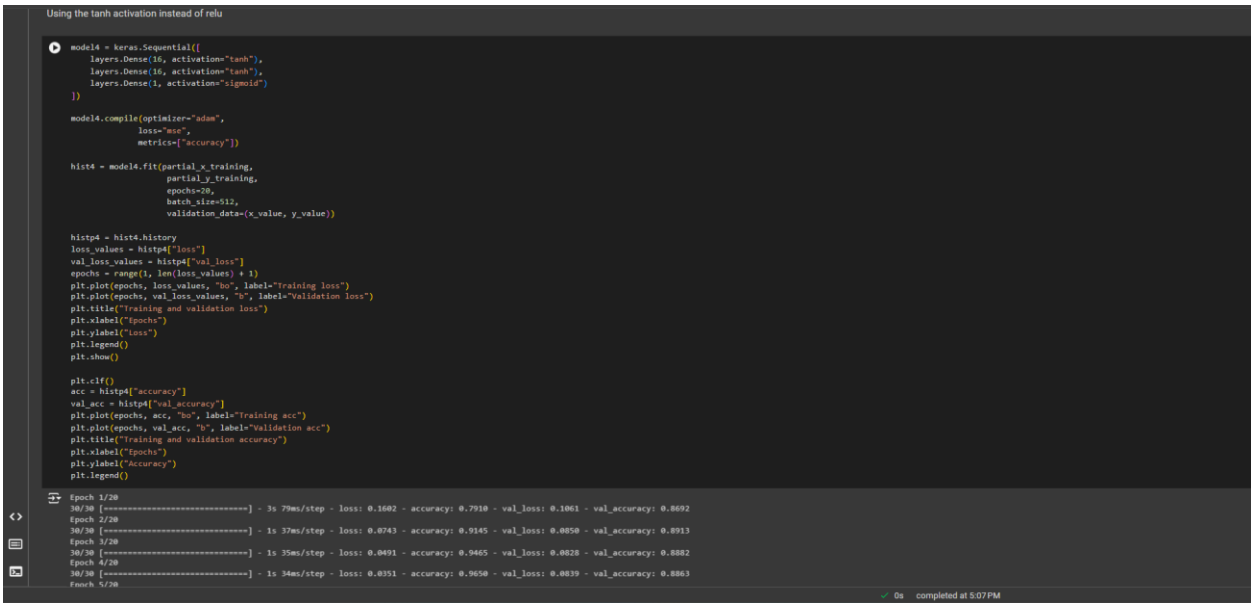


```
2 for the hidden layers iam using 32 node units and 64 units

[ ] model2 = keras.Sequential([
        layers.Dense(32, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(1, activation="sigmoid")
    ])

    model2.compile(optimizer="adam",
                   loss="binary_crossentropy",
                  metrics=["accuracy"])

    hist2 = model2.fit(partial_x_training,
                       partial_y_training,
                       epochs=20,
                       batch_size=512,
                       validation_data=(x_value, y_value))

    histp2 = hist2.history
    loss_values = histp2["loss"]
    val_loss_values = histp2["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 = histp2["accuracy"]
    val_acc = histp2["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()

    Epoch 1/20
    30/30 [==============================] - 4s 99ms/step - loss: 0.5038 - accuracy: 0.7873 - val_loss: 0.3246 - val_accuracy: 0.8747
    Epoch 2/20
    30/30 [==============================] - 1s 49ms/step - loss: 0.2306 - accuracy: 0.9153 - val_loss: 0.2766 - val_accuracy: 0.8897
    Epoch 3/20
    30/30 [==============================] - 2s 53ms/step - loss: 0.1440 - accuracy: 0.9516 - val_loss: 0.3063 - val_accuracy: 0.8809
    Epoch 4/20
    30/30 [==============================] - 1s 42ms/step - loss: 0.0968 - accuracy: 0.9705 - val_loss: 0.3368 - val_accuracy: 0.8805
```

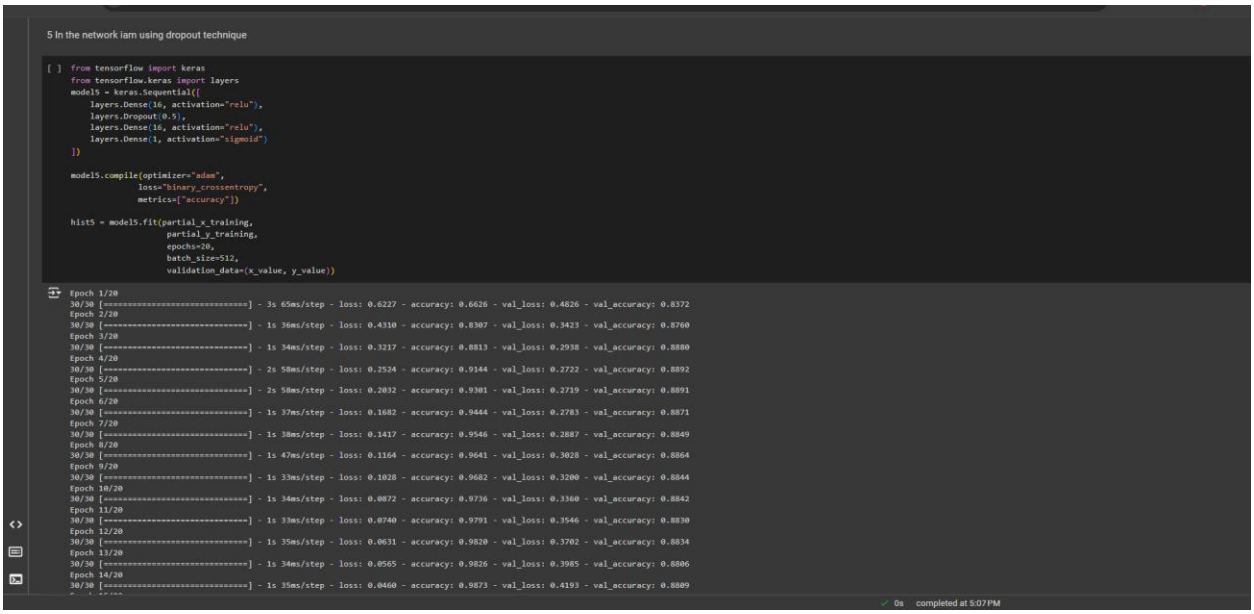
This is the code implementation, model training, and model testing for the binary classification on the IMDB movie review dataset of a neural network model named `model2` built using TensorFlow Keras. `model2` is structured with two hidden layers: the first with 32 nodes and the second with 64 nodes, thereof ReLU activation function are used to map the input data into high level features. L2 vector of 50 nodes Receive input from L1 vector To enables the maximum value to be 1- output node of sigmoid activation fine for binary classification. The final model is first optimized using the Adam optimizer, and the binar cross entropy as the loss function with accuracy as the measure of the model performance. Schooling is done through a 20-epoch, 512-samples batch, both training and validation data sets are used (`partial_x_training`, `partial_y_training`, `x_value`, `y_value`) to measure loss and accuracy during and after training. Then, using Matplotlib, plots of loss values with respect to epochs and trends of the accuracies with respect to epochs of training are created. They help in the evaluation of the model's learning and generalization capabilities from the sample data, and might be useful in improving the model if necessary.



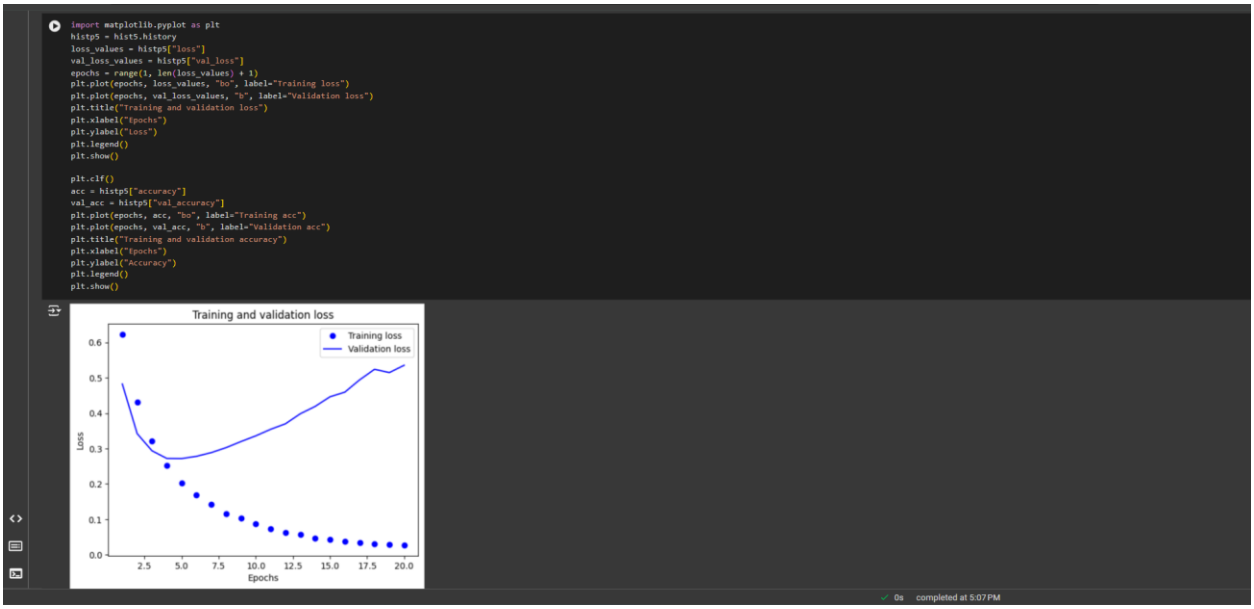
Here, the proposed model is shown as the neural network model named `model3` which is developed, trained, and tested based on TensorFlow Keras, and its tasks are focused on the binary classification for the IMDB movie review datasets. `model3` is set to have two layers with 16 neurons in each layer and ReLU activation function in the hidden layers, and an output layer with a sigmoid activation function for the binary classification task. To apply gradient descent the model is compiled using the Adam optimizer, the loss function is set to mean squared error (`loss="mse"`) and in addition to that, accuracy is chosen as the parameter by means of which model's performance during training can be evaluated. Training takes place in 20 epochs with a batch size of 512 samples; validation data include `x_value` and `y_value` datasets which help to comprehend the model's performance to the unreached information samples. For this purpose, the usage of Matplotlib for the creation of the visuality of training and validation loss as well as of the accuracy of `model3` is introduced and used to assess the performance of training for sentiment from movie reviews.



This code chunk deploys and assesses `model4` that is a neural network for binary classification on IMDB movie review dataset in TensorFlow Keras. The `model4` is configured with hidden layers of 16 units in each layer and applying the activation function that may help the model to learn from the data more profoundly as it can scale the data between -1 and 1, and that is a hyperbolic tangent (tanh). Since this is a binary classification problem, the output layer includes a single node with the sigmoid activation function. To achieve the efficient gradient descent the model is compiled with the Adam optimizer, `loss="mse"` as the loss function, and accuracy is used to observe the training progress. The training is performed 20 times with 512 samples in a batch, and the data of validation are used to evaluate the model's performance. Matplotlib is then used to plot train and validation curve which shows the training or validation loss and the corresponding accuracy of `model4` over the epochs to explain its learning ability as well as the accuracy of the model based on the movie review dataset.



This piece of code creates and compiles `model5`, a neural network for binary classification on the IMDB movie review dataset using TensorFlow Keras. `model5` consists of three layers: two hidden layers, each with 16 nodes and ReLU activation and the last layer with a single node as sigmoid activation since the problem is 2-class classification. The Dropout layer with dropout rate equal to 0. 5 is used after the first hidden layer as drop out which randomly set half of the neurons to zero during back propagation which help in reducing the over fitting problem by introducing noise into training data. This model is further trained with the Adam optimizer for the gradient descent with binary cross entropy being used as the loss function for evaluating the disparity between the forecasted and actual results. Batch size is 512 samples and training is done for 20 epochs, also validation data used here are (`x_value`, `y_value`). This can be observed in the training and validation loss plot, as well as the training and validation accuracy as obtained using Matplotlib for `model5`, the architecture proposed in the paper; this shows how well the model learns sentiment from the movie reviews during the training epochs.



It is a piece of code that plots might indicate the training of `model5, which is a binary classifier neural network` that has been trained on the IMDB movie review dataset using TensorFlow Keras. The first plot describes Training & Validation Loss over Epochs where the training loss is shown as the blue dots and the line through the blue color line is the validation loss. Such kind of visualization assists in the evaluation of the performance of the model in terms of ability to learn from the training data and was able to perform the same on the validation data that the model has never seen. The second plot is training and validation accuracy against epochs, where the dot is the training accuracy in blue and line is validation accuracy in blue. This graph provides a picture showing how the model is learning the data set used in training and how effective it is in generalizing in validation data set during training. These two types of visualizations give the training metrics into `model5` and some understanding of how the training phase of the model is learning from the movies reviews dataset and how well the classifier can differentiate between sentiment in the movies revises.

Conclusion:-

These sequence of code snippets gave a detailed tutorial on the construction, training, assessment and visualizations of neural network models for sentiment analysis on the IMDB movie review dataset under TensorFlow Keras. Based on data loading IMDB dataset and its preprocessing, the codes covered different model architectures with Dense layers and different activations functions ReLU,tanh,sigmoid. Even for overfitting issues, techniques like dropout regularization were implemented. The models were trained with Adam optimizer and binary cross-entropy loss was used for classification tasks and the values of accuracy were tracked while training the models. In training, the model was trained through epochs and batches of new data, where the validation data was used to evaluate the model on examples which was unseen to it. A tool known as Matplotlib was used to plot training and validation loss as well as accuracy to assess for model

convergence, overfitting, and its generality. It was found that each of the model variants and visualizations helped in comparing the effects of various architectures, activation functions, regularization, training dynamics, and accuracy. To summarise, these exercises provided an experience of deep learning model building and evaluation while demonstrating ideas for increasing model resistance and effectiveness for sentiment analysis tasks.