Assignment 2: Neural Network Model

Building a neural network model to predict the sentiment of movie reviews by using the IMDB movie review dataset.

For this assignment I began with installing the necessary packages 'tensorflow' and 'keras' using the pip package manager.

Then loaded dataset using the `tensorflow.keras.datasets import imdb` and splitting it into training and testing sets , following that I used the 'vectorize_sequences()' function to encode the reviews as binary word vectors(results of the vectors are stored in X_train and X_test).

Step 1: At first I started with training the neural network with two hidden layers , each containing 16 neurons and is trained for 20 epochs with a batch size of 512, loss function used here is Mean square error , metric used is Accuracy and optimizer used is Adam.

Activation function 'tanh' was used in the hidden layers instead of 'relu' in the hidden layers and for the output layer used the 'sigmoid' activation function.

Step 2: In step 2, implemented the dropouts and Regularizers, by checking the performance by changing the dense layers to 32 and 64 hidden units respectively. For this I used the following code

```
from tensorflow.keras.layers import Dropout

from tensorflow.keras import regularizers
```

Then next "adam" optimizer, "mean squared error" loss function, and "accuracy" metric are used again to compile the model. The model is then trained for 30 epochs with a batch size of 512 on the training data. The validation data is used to evaluate the model's performance during training.

Additionally, dropout layers with a rate of 0.5 are added after the first two dense layers, and a L2 activity regularizer with a rate of 0.01 is added to the output layer.

Activation function and hidden layers

```
model.add(Dense(32, activation="tanh"))

model.add(Dense(64, activation="tanh"))
```

These two lines from the code add a new hidden 32 and 64 dense units that use the tanh activation function, enabling the layer's 32 and 64 neurons to acquire the vector data in a non-linear manner.

Drop out function.

```
model.add(Dropout(0.5))
```

The purpose of using Dropout is to prevent overfitting,and the line "model.add(Dropout(0.5))" states that during training, 50% of the neurons in the previous layer will be randomly dropped out during to improve the model's ability to perform well on new, unseen data.

Regularizers.

```
model.add(Dense(1, activation="sigmoid",activity_regularizer=regularizers.
L2(0.01)))

model.add(Dense(64, activation="tanh",activity_regularizer=regularizers.L2
(0.01)))
```

The above code uses L2 regularization with a coefficient of 0.01. L2 regularization helps to prevent overfitting by adding a penalty term to the loss function that encourages the weights to be small. This regularization helps in making the model less complex and better generalization performance on the validation set. Perhaps the model here is saturated and the best validation accuracy we can get a range of 86 or 87 percent.

mse loss function instead of binary_crossentropy

By replacing binary cross-entropy with mean squared error as the loss function, the model's validation loss improved significantly from 0.5 to 0.1.

Below is the Table to indicate my conclusions:

Step	Combinations	Optimizer	Loss Function	Metric	Batch Size	Epochs	Train Acc	Validation Acc
1	Two hidden layers with 16 neurons each with tanh activation function	Adam	Mean square error	Accuracy	512	20	0.9964	0.8670
2	Three hidden layers with 32 neurons each with dropouts (0.5) with tanh activation function	Adam	Mean square error	Accuracy	512	30	0.9896	0.8658
3	Three hidden layers with 64 neurons each with dropouts (0.5) with tanh activation function	Adam	Mean square error	Accuracy	512	30	0.9854	0.8624

Concluding points:

Evaluating of the neural network combinations with varying numbers of hidden layers and neurons, trained on the same dataset with Adam optimizer, mean squared error loss function, accuracy metric, and dropout layers, showed that simpler models tend to have better validation accuracy.

Addition of dropout layers helps to reduce overfitting and improves the models ability to perform well.

More tuning of hyperparameters such as learning rate, number of epochs, and architecture could potentially improve the model performance.