Neural Networks Assignment-2

1. **Modify an existing neural network model to improve performance?**

There are several ways to modify an existing neural network model to improve its performance. Here are a few:

1. Modify the structure: The number of layers, the number of neurons in each layer, and the types of layers make up a neural network's architecture. Find a configuration that works better for your data by experimenting with varying the number of layers, neurons, or layers.
2. Modify the activation mode: Each neuron's output in the network is controlled by the activation function. To see if it improves the model's performance, try employing a different activation function, such as the hyperbolic tangent or the rectified linear unit (ReLU).
3. Adapt the loss function as needed: The difference in error between the expected and actual output is determined using the loss function. Use a different loss function, such as mean squared error (MSE) or categorical cross-entropy, to see if it improves the accuracy of your model if it is not performing well.
4. Make the model consistent: Regularization methods can be utilized to forestall overfitting and work on the speculation of the model. To boost the model's performance, you can try adding dropout layers, L1 or L2 regularization, or early stopping.
5. Make more use of the data: If your model isn't working as well as it should, try training it with more data. The model may be able to learn more patterns and better adapt to new data because of this.
6. Modify the model: You can fine-tune a pre-trained model for your task by training it for a few more epochs on your data if you are using one. The model may be able to learn to recognize patterns in your data more effectively because of this.

The performance of an existing neural network model can be enhanced in several ways, some of which are listed here. Other modifications might be better for your job.

2. **Explain how different approaches affect the performance of the model?**

1. Increasing the complexity of the model: Depending on the kind of model being used, "increasing the intricacy or complexity of a model" can refer to several different methods and approaches. In general, increasing the complexity of a model can be beneficial when the model needs to capture more complex data relationships or patterns, but it can also result in overfitting or inefficient computation.

2. Modifying the loss function: In machine learning, one important method for improving a model's performance is modifying the loss function. The model's performance on a particular task, like predicting the right output for a given input, is measured by the loss function. We can alter the way the model learns from the data and alter its behavior to better match our desired outcomes by modifying the loss function.

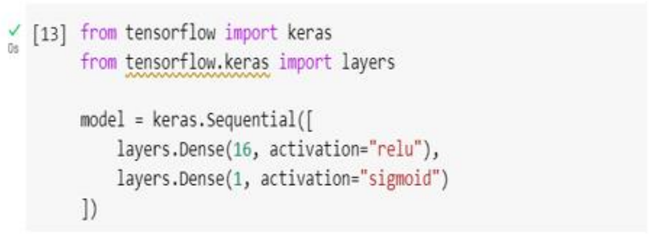
3. Regularizing the model: - In machine learning, regularization is a method for preventing the model from overfitting to the training data. When a model is too complex and fits the noise in the training data, it overfits, resulting in poor performance on new data that has not been seen before.

4. Increasing the amount of data: - One of the most efficient ways to boost a machine learning model's performance is to add more data. The model's ability to adapt to new, previously unknown data improves with training data. This is because more data provides the model with a sample that is more diverse and representative of the data's underlying distribution.

5. Changing the activation function: - A neural network's activation function is a crucial component that contributes to the model's non-linearity. Neural networks need non-linearity because it enables them to learn complex data patterns and relationships.

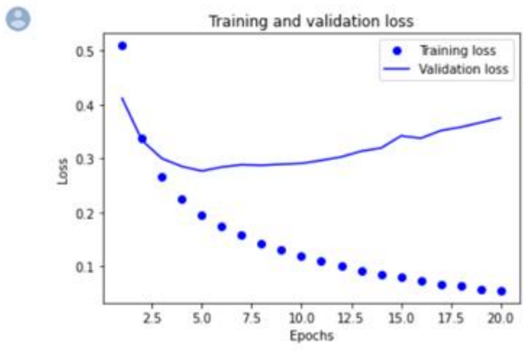
For the IMDB example that we discussed in class, do the following:

1. You used two hidden layers. Try using one or three hidden layers and see how doing so affects validation and test accuracy. Model.
   # One hidden layer the number of hidden layers is an important hyperparameter that can affect the performance of the



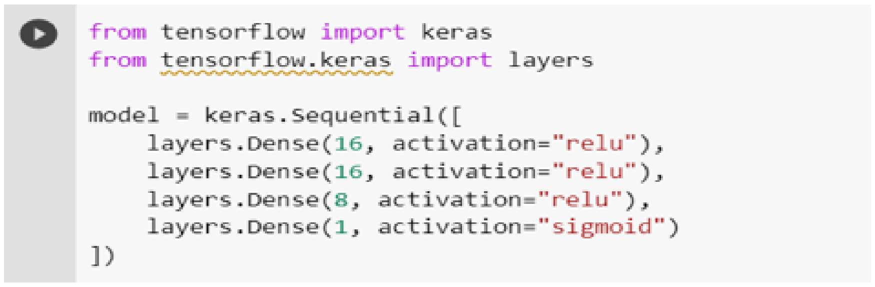
```
[13] from tensorflow import keras
     from tensorflow.keras import layers

     model = keras.Sequential([
         layers.Dense(16, activation="relu"),
         layers.Dense(1, activation="sigmoid")
     ])
```



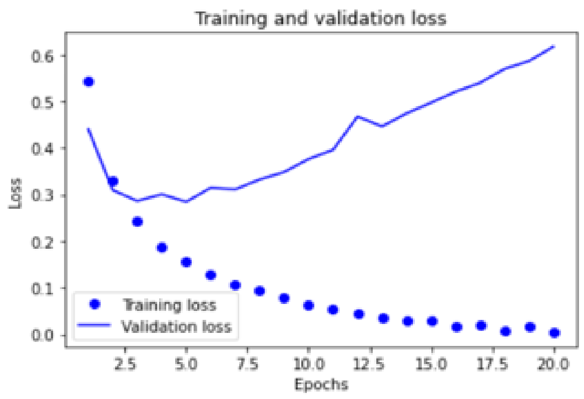
In one hidden layer the validation loss: 0.2767 and validation accuracy: 0.8893 which is highest in Epoch 5.

#Three hidden layers



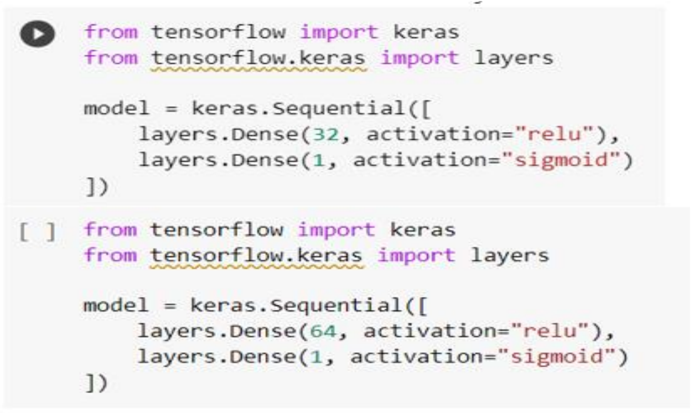
```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(8, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```



Z

2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.?



```
from tensorflow import keras
from tensorflow.keras import layers

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

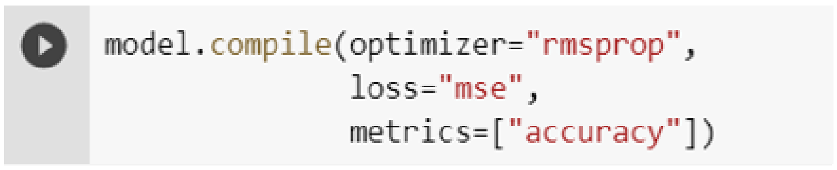
```
[ ] from tensorflow import keras
    from tensorflow.keras import layers

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

Validation loss: 0.2782 validation accuracy: 0.8899 (for 32 units)

Validation loss: 0.2754-Validation accuracy: 0.8887(for 64 units)

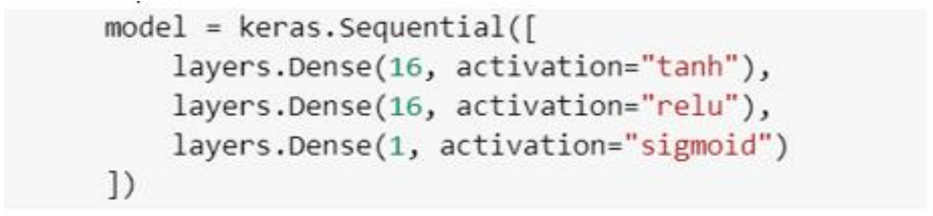
3. Try using the mse loss function instead of binary_crossentropy.



```
model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
```

Validation loss: 0.0838 validation accuracy: 0.8863

4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.



```
model = keras.Sequential([
    layers.Dense(16, activation="tanh"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.

some techniques to improve model performance on validation data:

1. Regularization: Overfitting can be avoided by employing regularization methods like L1, L2, or Elastic Net regularization. Regularization helps the model generalize better and reduces its complexity. The model can have regularization applied to various layers.
2. Dropout: During training, a regularization method called "dropout" randomly eliminates some neurons. This aids in generalization and prevents overfitting. Dropout can be added after each network layer.
3. Normalization by Batch: A neural network's inputs are normalized using a method called batch normalization. It can speed up the training process and aid in stabilizing the distribution of inputs. After every layer in the network, batch normalization can be added.

4. Early termination: By keeping an eye on the validation loss during training, early stopping can be used to avoid overfitting. When the validation loss stops growing, the model training stops.
5. Data Enhancement: Data augmentation is a method that creates new synthetic examples to make the training dataset larger. Overfitting is reduced and generalization of the model is improved as a result.
6. Tuning the Hyperparameters: Adjusting the hyperparameters, such as the batch size, activation functions, learning rate, and number of epochs, can help in working on the exhibition of the model on approval information.

The model's performance on the validation data can be improved by combining or using these methods separately. It's important to play around with these methods to find the best combination for the problem and dataset at hand.

we will be using Dropout technique to improve the model performance.

```
from keras.models import Sequential

from keras.layers import Dense, Dropout

# Define your model architecture

model = Sequential()

model.add(Dense(64, activation='relu', input_dim=100))

model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid')
```