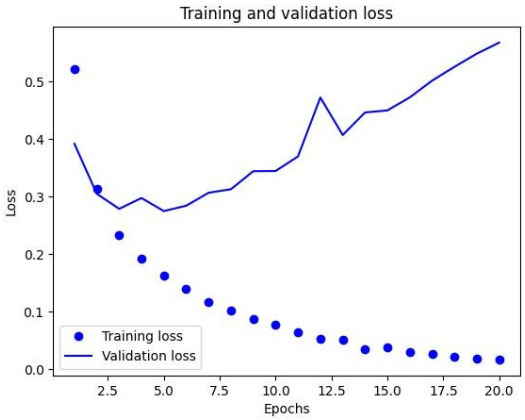
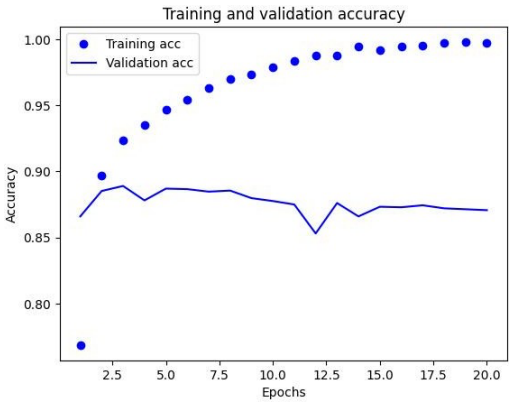
Assignment 1: Summary Report

Q1. You used two hidden layers. Try using one or three hidden layers, and see how doing so affects validation and test accuracy.

The original code used two hidden layers, here is the outcome of training accuracy and loss & validation accuracy and loss:

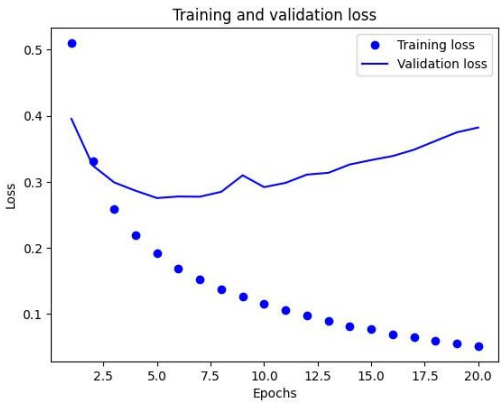


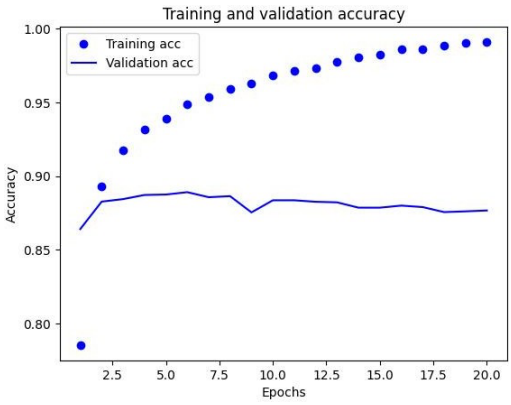


Then, I tried using one hidden layer:

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

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

Here is the training and validation loss & accuracy:





Then I tuned to epoch to 7 according to the result and compared with the two hidden layer model:

```
Epoch 1/4
49/49 [==============================] - 5s 79ms/step - loss: 0.4692 - accuracy: 0.8088
Epoch 2/4
49/49 [==============================] - 2s 31ms/step - loss: 0.2741 - accuracy: 0.9019
Epoch 3/4
49/49 [==============================] - 1s 25ms/step - loss: 0.2143 - accuracy: 0.9218
Epoch 4/4
49/49 [==============================] - 1s 25ms/step - loss: 0.1848 - accuracy: 0.9327
782/782 [==============================] - 2s 2ms/step - loss: 0.2983 - accuracy: 0.8799
```

```
Epoch 1/7
49/49 [==============================] - 4s 51ms/step - loss: 0.4745 - accuracy: 0.8195
Epoch 2/7
49/49 [==============================] - 2s 40ms/step - loss: 0.3031 - accuracy: 0.8979
Epoch 3/7
49/49 [==============================] - 2s 32ms/step - loss: 0.2448 - accuracy: 0.9139
Epoch 4/7
49/49 [==============================] - 1s 28ms/step - loss: 0.2121 - accuracy: 0.9259
Epoch 5/7
49/49 [==============================] - 2s 38ms/step - loss: 0.1916 - accuracy: 0.9324
Epoch 6/7
49/49 [==============================] - 1s 28ms/step - loss: 0.1750 - accuracy: 0.9399
Epoch 7/7
49/49 [==============================] - 1s 22ms/step - loss: 0.1621 - accuracy: 0.9442
782/782 [==============================] - 3s 4ms/step - loss: 0.2869 - accuracy: 0.8851
```

Comparison:
The validation accuracy of the model with two hidden layers was somewhat higher (87.07%) than that of the model with one hidden layer (86.41%).
On the test set, both models demonstrated great accuracy; however, the one-hidden-layer model outperformed the two-hidden-layers model by a little margin. In comparison to the two-hidden-layers model, the one-hidden-layer model had a somewhat lower test loss (0.2869 vs. 0.2983) and a marginally higher test accuracy (88.51% vs. 87.99%). Both models demonstrated successful learning from the training data by increasing accuracy and decreasing loss over the epochs.
In contrast to the two-hidden-layers model, which trained over four epochs, the one-hidden-layer model trained over seven. Nonetheless, both models obtained great accuracy on the test set, and the performance difference between them is negligible.

Q2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.

As we know, model with 16 Units:

Training Accuracy: Achieved an accuracy of approximately 93.27%.

Test Accuracy: Achieved an accuracy of approximately 87.99%.

Model with 32 hidden units:

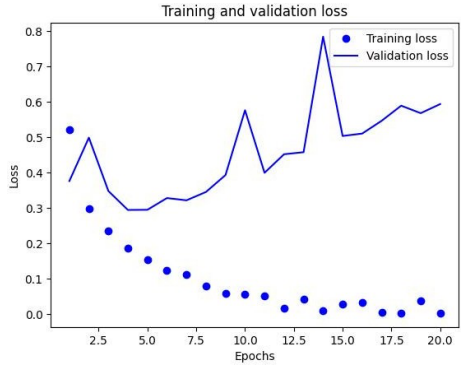
```python
from tensorflow import keras
from tensorflow.keras import layers

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

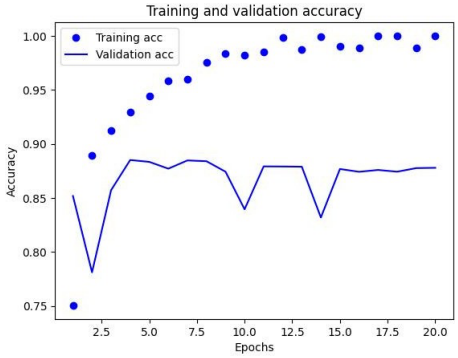
```
Epoch 1/4
49/49 [==============================] - 2s 28ms/step - loss: 0.4821 - accuracy: 0.7964
Epoch 2/4
49/49 [==============================] - 1s 28ms/step - loss: 0.2751 - accuracy: 0.9005
Epoch 3/4
49/49 [==============================] - 1s 28ms/step - loss: 0.2154 - accuracy: 0.9212
Epoch 4/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1833 - accuracy: 0.9330
782/782 [==============================] - 2s 2ms/step - loss: 0.3058 - accuracy: 0.8785
```

Training Accuracy: Achieved an accuracy of approximately 93.30%.

Test Accuracy: Achieved an accuracy of approximately 87.85%.

Model with 64 hidden units:





Based on the observed trend, I tried training for 11 epochs

```
Epoch 1/11
49/49 [==============================] - 3s 53ms/step - loss: 0.4501 - accuracy: 0.7935
Epoch 2/11
49/49 [==============================] - 2s 48ms/step - loss: 0.2650 - accuracy: 0.8965
Epoch 3/11
49/49 [==============================] - 2s 42ms/step - loss: 0.2124 - accuracy: 0.9184
Epoch 4/11
49/49 [==============================] - 3s 61ms/step - loss: 0.1770 - accuracy: 0.9340
Epoch 5/11
49/49 [==============================] - 2s 42ms/step - loss: 0.1537 - accuracy: 0.9435
Epoch 6/11
49/49 [==============================] - 2s 41ms/step - loss: 0.1314 - accuracy: 0.9504
Epoch 7/11
49/49 [==============================] - 2s 41ms/step - loss: 0.1050 - accuracy: 0.9636
Epoch 8/11
49/49 [==============================] - 2s 41ms/step - loss: 0.0915 - accuracy: 0.9667
Epoch 9/11
49/49 [==============================] - 2s 48ms/step - loss: 0.0662 - accuracy: 0.9786
Epoch 10/11
49/49 [==============================] - 3s 56ms/step - loss: 0.0521 - accuracy: 0.9840
Epoch 11/11
49/49 [==============================] - 2s 39ms/step - loss: 0.0389 - accuracy: 0.9891
782/782 [==============================] - 3s 3ms/step - loss: 0.4619 - accuracy: 0.8679
```

Training Accuracy: Achieved an accuracy of approximately 98.91%.

Test Accuracy: Achieved an accuracy of approximately 86.79%.

Comparison:

Out of the three models, the one with 64 hidden units had the highest training accuracy, suggesting a superior fit to the training set. Its test accuracy was marginally worse than that of the models with less hidden units, though. Overfitting is more likely when model capacity is increased.

Similar test accuracies were shown by the models with 16 and 32 hidden units, with the 16-unit model marginally beating the 32-unit model.

The better training accuracy in the 64-unit model suggests that increasing the number of hidden units enhances the model's capacity to match the training data. The drop in test accuracy as compared to models with fewer hidden units, however, indicates that overfitting resulted from this increase in complexity.

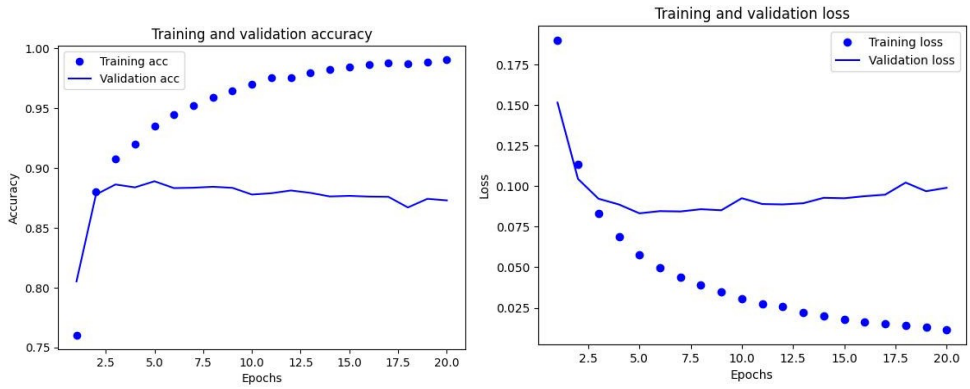
However, as demonstrated by the marginally improved test accuracy of the model with 16 units in comparison to the others, fewer hidden units lead to simpler models that generalize better to unseen data.

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

Q3. Try using the mse loss function instead of binary_crossentropy.

Validation performance:

```
Epoch 1/20
30/30 [==============================] - 4s 103ms/step - loss: 0.1898 - accuracy: 0.7601 - val_loss: 0.1514 - val_accuracy: 0.8051
Epoch 2/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1136 - accuracy: 0.8800 - val_loss: 0.1044 - val_accuracy: 0.8778
Epoch 3/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0829 - accuracy: 0.9078 - val_loss: 0.0922 - val_accuracy: 0.8862
Epoch 4/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0685 - accuracy: 0.9199 - val_loss: 0.0886 - val_accuracy: 0.8838
Epoch 5/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0579 - accuracy: 0.9351 - val_loss: 0.0832 - val_accuracy: 0.8889
Epoch 6/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0498 - accuracy: 0.9446 - val_loss: 0.0846 - val_accuracy: 0.8832
Epoch 7/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0440 - accuracy: 0.9523 - val_loss: 0.0843 - val_accuracy: 0.8835
Epoch 8/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0390 - accuracy: 0.9588 - val_loss: 0.0857 - val_accuracy: 0.8843
Epoch 9/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0349 - accuracy: 0.9643 - val_loss: 0.0851 - val_accuracy: 0.8834
Epoch 10/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0306 - accuracy: 0.9698 - val_loss: 0.0925 - val_accuracy: 0.8778
Epoch 11/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0275 - accuracy: 0.9754 - val_loss: 0.0889 - val_accuracy: 0.8789
Epoch 12/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0258 - accuracy: 0.9757 - val_loss: 0.0887 - val_accuracy: 0.8812
Epoch 13/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0219 - accuracy: 0.9795 - val_loss: 0.0894 - val_accuracy: 0.8792
Epoch 14/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0198 - accuracy: 0.9825 - val_loss: 0.0928 - val_accuracy: 0.8762
Epoch 15/20
30/30 [==============================] - 2s 56ms/step - loss: 0.0180 - accuracy: 0.9845 - val_loss: 0.0925 - val_accuracy: 0.8767
Epoch 16/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0162 - accuracy: 0.9864 - val_loss: 0.0937 - val_accuracy: 0.8761
Epoch 17/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0152 - accuracy: 0.9875 - val_loss: 0.0947 - val_accuracy: 0.8759
Epoch 18/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0143 - accuracy: 0.9874 - val_loss: 0.1021 - val_accuracy: 0.8670
Epoch 19/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0129 - accuracy: 0.9889 - val_loss: 0.0968 - val_accuracy: 0.8742
Epoch 20/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0115 - accuracy: 0.9907 - val_loss: 0.0989 - val_accuracy: 0.8729
```



Then I tuned the epochs to 7:

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=7, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/7
49/49 [==============================] - 2s 24ms/step - loss: 0.1702 - accuracy: 0.7804
Epoch 2/7
49/49 [==============================] - 1s 23ms/step - loss: 0.0929 - accuracy: 0.8914
Epoch 3/7
49/49 [==============================] - 1s 22ms/step - loss: 0.0700 - accuracy: 0.9141
Epoch 4/7
49/49 [==============================] - 1s 25ms/step - loss: 0.0585 - accuracy: 0.9287
Epoch 5/7
49/49 [==============================] - 1s 23ms/step - loss: 0.0528 - accuracy: 0.9360
Epoch 6/7
49/49 [==============================] - 1s 25ms/step - loss: 0.0461 - accuracy: 0.9449
Epoch 7/7
49/49 [==============================] - 1s 24ms/step - loss: 0.0415 - accuracy: 0.9516
782/782 [==============================] - 2s 2ms/step - loss: 0.0897 - accuracy: 0.8790
```

The model appears to be overfitting to some degree based on the performance. While the training accuracy keeps getting better, the validation accuracy appears to peak and then significantly decline after a specific number of epochs. This implies that the model is not generalizing effectively to new data and is beginning to overfit the training set. MSE loss may not be the ideal option for classification issues such as binary sentiment analysis, but it is appropriate for regression tasks.

Q4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.

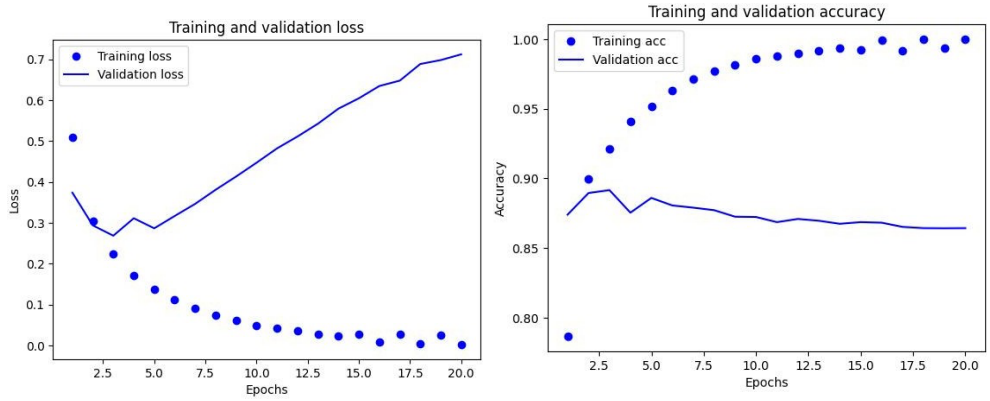
```
[9]  from tensorflow import keras
     from tensorflow.keras import layers

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

compliling the model

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

Training and validation performance:

```
Epoch 1/20
30/30 [==============================] - 3s 85ms/step - loss: 0.5083 - accuracy: 0.7867 - val_loss: 0.3737 - val_accuracy: 0.8740
Epoch 2/20
30/30 [==============================] - 1s 49ms/step - loss: 0.3033 - accuracy: 0.8998 - val_loss: 0.2936 - val_accuracy: 0.8895
Epoch 3/20
30/30 [==============================] - 1s 44ms/step - loss: 0.2244 - accuracy: 0.9210 - val_loss: 0.2685 - val_accuracy: 0.8916
Epoch 4/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1712 - accuracy: 0.9409 - val_loss: 0.3111 - val_accuracy: 0.8754
Epoch 5/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1382 - accuracy: 0.9519 - val_loss: 0.2865 - val_accuracy: 0.8860
Epoch 6/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1122 - accuracy: 0.9632 - val_loss: 0.3167 - val_accuracy: 0.8806
Epoch 7/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0911 - accuracy: 0.9714 - val_loss: 0.3464 - val_accuracy: 0.8790
Epoch 8/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0738 - accuracy: 0.9768 - val_loss: 0.3809 - val_accuracy: 0.8772
Epoch 9/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0622 - accuracy: 0.9815 - val_loss: 0.4132 - val_accuracy: 0.8725
Epoch 10/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0494 - accuracy: 0.9858 - val_loss: 0.4471 - val_accuracy: 0.8723
Epoch 11/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0431 - accuracy: 0.9879 - val_loss: 0.4821 - val_accuracy: 0.8686
Epoch 12/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0357 - accuracy: 0.9899 - val_loss: 0.5111 - val_accuracy: 0.8709
Epoch 13/20
30/30 [==============================] - 2s 59ms/step - loss: 0.0272 - accuracy: 0.9919 - val_loss: 0.5425 - val_accuracy: 0.8696
Epoch 14/20
30/30 [==============================] - 2s 58ms/step - loss: 0.0230 - accuracy: 0.9933 - val_loss: 0.5793 - val_accuracy: 0.8674
Epoch 15/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0280 - accuracy: 0.9925 - val_loss: 0.6046 - val_accuracy: 0.8686
Epoch 16/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0076 - accuracy: 0.9993 - val_loss: 0.6347 - val_accuracy: 0.8682
Epoch 17/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0281 - accuracy: 0.9916 - val_loss: 0.6478 - val_accuracy: 0.8652
Epoch 18/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0038 - accuracy: 0.9997 - val_loss: 0.6885 - val_accuracy: 0.8643
Epoch 19/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0249 - accuracy: 0.9935 - val_loss: 0.6981 - val_accuracy: 0.8642
Epoch 20/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.7122 - val_accuracy: 0.8643
```



Then based on the observation i adjust the epochs number to 3

```
[16] model = keras.Sequential([
        layers.Dense(16, activation="tanh"),
        layers.Dense(16, activation="tanh"),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer="rmsprop",
                  loss="binary_crossentropy",
                  metrics=["accuracy"])
    model.fit(x_train, y_train, epochs=3, batch_size=512)
    results = model.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 [==============================] - 2s 27ms/step - loss: 0.4385 - accuracy: 0.8243
Epoch 2/3
49/49 [==============================] - 1s 27ms/step - loss: 0.2519 - accuracy: 0.9062
Epoch 3/3
49/49 [==============================] - 1s 26ms/step - loss: 0.1947 - accuracy: 0.9271
782/782 [==============================] - 3s 3ms/step - loss: 0.2893 - accuracy: 0.8829
```

There are signs of overfitting as seen in the increasing gap between training and validation performance, especially towards the later epochs.

Comparing these two models:

Both models achieved similar validation accuracy. The model with ReLU activation converged faster, reaching a higher training accuracy after fewer epochs. The model with Tanh activation required fewer epochs but achieved a slightly lower training accuracy compared to the ReLU model. The choice of activation function seems to have influenced the convergence rate and the final training accuracy.

Q5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.

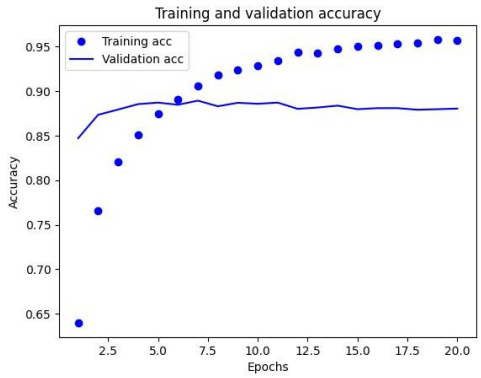
To improve the performance of the model on the validation set, I modify the model to include dropout in it:

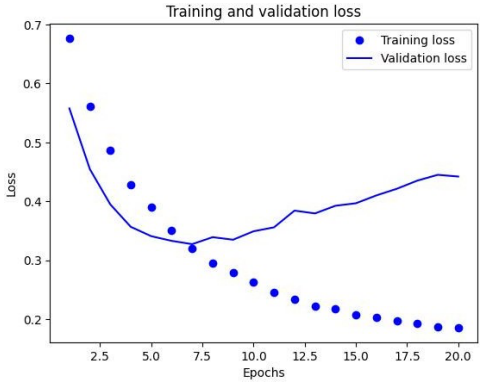
```
from tensorflow import keras
from tensorflow.keras import layers, regularizers


model = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
```

compliling the model

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

Here is the validation performance:






```
Epoch 1/20
30/30 [==============================] - 5s 143ms/step - loss: 0.6766 - accuracy: 0.6397 - val_loss: 0.5576 - val_accuracy: 0.8471
Epoch 2/20
30/30 [==============================] - 4s 122ms/step - loss: 0.5613 - accuracy: 0.7657 - val_loss: 0.4546 - val_accuracy: 0.8734
Epoch 3/20
30/30 [==============================] - 3s 98ms/step - loss: 0.4860 - accuracy: 0.8209 - val_loss: 0.3948 - val_accuracy: 0.8793
Epoch 4/20
30/30 [==============================] - 2s 58ms/step - loss: 0.4283 - accuracy: 0.8509 - val_loss: 0.3568 - val_accuracy: 0.8854
Epoch 5/20
30/30 [==============================] - 1s 39ms/step - loss: 0.3895 - accuracy: 0.8744 - val_loss: 0.3409 - val_accuracy: 0.8871
Epoch 6/20
30/30 [==============================] - 1s 37ms/step - loss: 0.3504 - accuracy: 0.8911 - val_loss: 0.3330 - val_accuracy: 0.8847
Epoch 7/20
30/30 [==============================] - 1s 41ms/step - loss: 0.3195 - accuracy: 0.9054 - val_loss: 0.3274 - val_accuracy: 0.8893
Epoch 8/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2953 - accuracy: 0.9185 - val_loss: 0.3393 - val_accuracy: 0.8830
Epoch 9/20
30/30 [==============================] - 1s 48ms/step - loss: 0.2786 - accuracy: 0.9234 - val_loss: 0.3349 - val_accuracy: 0.8869
Epoch 10/20
30/30 [==============================] - 1s 37ms/step - loss: 0.2633 - accuracy: 0.9282 - val_loss: 0.3493 - val_accuracy: 0.8858
Epoch 11/20
30/30 [==============================] - 1s 49ms/step - loss: 0.2457 - accuracy: 0.9341 - val_loss: 0.3559 - val_accuracy: 0.8871
Epoch 12/20
30/30 [==============================] - 2s 54ms/step - loss: 0.2333 - accuracy: 0.9437 - val_loss: 0.3843 - val_accuracy: 0.8801
Epoch 13/20
30/30 [==============================] - 2s 60ms/step - loss: 0.2221 - accuracy: 0.9425 - val_loss: 0.3795 - val_accuracy: 0.8816
Epoch 14/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2185 - accuracy: 0.9471 - val_loss: 0.3925 - val_accuracy: 0.8837
Epoch 15/20
30/30 [==============================] - 1s 38ms/step - loss: 0.2076 - accuracy: 0.9505 - val_loss: 0.3967 - val_accuracy: 0.8797
Epoch 16/20
30/30 [==============================] - 1s 37ms/step - loss: 0.2033 - accuracy: 0.9514 - val_loss: 0.4100 - val_accuracy: 0.8809
Epoch 17/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1980 - accuracy: 0.9529 - val_loss: 0.4213 - val_accuracy: 0.8809
Epoch 18/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1928 - accuracy: 0.9539 - val_loss: 0.4351 - val_accuracy: 0.8791
Epoch 19/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1878 - accuracy: 0.9580 - val_loss: 0.4450 - val_accuracy: 0.8797
Epoch 20/20
30/30 [==============================] - 1s 34ms/step - loss: 0.1858 - accuracy: 0.9569 - val_loss: 0.4421 - val_accuracy: 0.8804
```

Examining the results prior to and following the implementation of dropout regularization: Prior to regularization of dropouts: The training accuracy varies between 97-98%.
The validation accuracy varies from 87-89%.
Following regularization of dropouts: Training Accuracy: Variables between 94 and 95 percent, lower than previously.
Validation Accuracy: Varying between 88 and 89%, slightly better than before dropout.
As demonstrated by the gain in validation accuracy and the decrease in training accuracy, dropout regularization has assisted in reducing overfitting of the original model. Better generalization performance is the outcome of the updated model's decreased reliance on particular features during training. Additionally, dropout has enhanced performance on unseen data (validation set) by preventing the model from memorizing the training data.

```
model = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=7, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/7
49/49 [==============================] - 2s 26ms/step - loss: 0.6343 - accuracy: 0.6890
Epoch 2/7
49/49 [==============================] - 1s 25ms/step - loss: 0.4924 - accuracy: 0.8152
Epoch 3/7
49/49 [==============================] - 2s 33ms/step - loss: 0.4150 - accuracy: 0.8623
Epoch 4/7
49/49 [==============================] - 2s 36ms/step - loss: 0.3625 - accuracy: 0.8864
Epoch 5/7
49/49 [==============================] - 1s 26ms/step - loss: 0.3299 - accuracy: 0.9060
Epoch 6/7
49/49 [==============================] - 1s 26ms/step - loss: 0.3031 - accuracy: 0.9149
Epoch 7/7
49/49 [==============================] - 1s 25ms/step - loss: 0.2889 - accuracy: 0.9224
782/782 [==============================] - 2s 2ms/step - loss: 0.3381 - accuracy: 0.8861
```

Following the implementation of dropout regularization and training on the testing dataset, the model's accuracy was 88.61% and its performance was 0.3381.
It appears that the model has maintained a comparable level of performance on unseen data when compared to the validation set results following the application of dropout regularization. This further suggests that the dropout regularization technique has successfully prevented overfitting, enabling the model to generalize well to new data.