Summary Report on IMDB Neural Network Model With different Approaches to improve Performance

Professor **Submitted By**

CJ Wu **Pavan Chaitanya Bommadevara**

Summary:

Modify an existing IMDB neural network model to improve performance and explain how different approaches affect the performance of the model.

Procedure:

They are total of 6 steps involved in this Procedure. They are

- Importing the Libraries and IMDB Dataset.
- Preparing the Required Data for Building Network.
- Building the Network Model.
- Validation of our Network Model.
- Plotting the Results.
- Retrain the model for Predictions on New Data.

Importing the Libraries and the IMDB Dataset:

My approach towards the problem is initially learned the importance of NumPy ,Pandas, warnings, tensorflow, models, layers, optimizers, losses matplotlib libraries.

Then I have imported the imdb dataset that is present in keras.datasets.

These are the required imports for our problem

```
import os
from operator import itemgetter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
get_ipython().magic(u'matplotlib inline')
plt.style.use('ggplot')

import tensorflow as tf

from keras import models, regularizers, layers, optimizers, losses, metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils, to_categorical

from keras.datasets import imdb
```

```
# IMDB Dataset
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

IMDB dataset has 50,000 reviews. We are now using just the top 10,000 reviews only which will be loaded via imdb.load_data(num_words=10000) variable.

Train_data ,Test_data are the data reviews that we will be using train_labels, test_labels are just the Positive and Negative Labels.

Preparing the Required Data for Building Network:

We cannot feed the integers into the neural Network. We must turn our lists to tensors.

1) Pad out lists so that they all have same length, turn them into the integer tensor via (samples,word_indices) and use them as the first layer of our network.
2) By One-Hot encoding we can turn our lists to vectors of 0's and 1's.

The Padding and the One-hot Encoding looks like the below code:

Vectorizing the Labels:

```
# Vectorized Labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
print("y_train ", y_train.shape)
print("y_test ", y_test.shape)
```

```
y_train  (25000,)
y_test  (25000,)
```

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

print("x_val ", x_val.shape)
print("partial_x_train ", partial_x_train.shape)
print("y_val ", y_val.shape)
print("partial_y_train ", partial_y_train.shape)
```

```
x_val  (10000, 10000)
partial_x_train  (15000, 10000)
y_val  (10000,)
partial_y_train  (15000,)
```

Vectorizing the Data:

```
#Vectorizing the data
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

```
x_train[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

So, by now we have what the sample data for our problem looks like, we are ready to build our Network Model and to feed our Prepared Data to our Network.

Building the Network Model:

For our Network Model the input data is vector and our input labels are scalar. So this type of the input data we can go with the (Dense layers, Relu Activation Function, rmsprop optimizer, binary_crossentropy) and then finally with the sigmoid activation function.

This would be the very effective way to build the neural network model.

The code for building the model looks like the below figure:

```
# Building our Network
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
layers.Dense(16, activation="relu"),
layers.Dense(16, activation="relu"),
layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

Validation of our Network Model:

To know the accuracy of the model that we built we will use a validation set data which is a part of original training data which contains 10,000 sample data which the model has never seen before.

The code for validation looks like:

```
#Validation of our Network
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
tf.random.set_seed(8985)

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

We have used 20 epochs (Iterations) with each a batch size of 512 of validation set data splitting into training with help of model.fit() function. The results for validation look like

```
30/30 [==============================] - 1s 37ms/step - loss: 0.0420 - accuracy: 0.9910 - val_loss: 0.4016 - val_accuracy: 0.8766
Epoch 14/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0384 - accuracy: 0.9912 - val_loss: 0.4253 - val_accuracy: 0.8737
Epoch 15/20
30/30 [==============================] - 2s 60ms/step - loss: 0.0276 - accuracy: 0.9957 - val_loss: 0.4466 - val_accuracy: 0.8741
Epoch 16/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0262 - accuracy: 0.9948 - val_loss: 0.4711 - val_accuracy: 0.8720
Epoch 17/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0187 - accuracy: 0.9979 - val_loss: 0.4969 - val_accuracy: 0.8703
Epoch 18/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0143 - accuracy: 0.9987 - val_loss: 0.5791 - val_accuracy: 0.8600
Epoch 19/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0107 - accuracy: 0.9995 - val_loss: 0.5614 - val_accuracy: 0.8694
Epoch 20/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0122 - accuracy: 0.9981 - val_loss: 0.5672 - val_accuracy: 0.8715
```

It will iterate for 20 epochs.

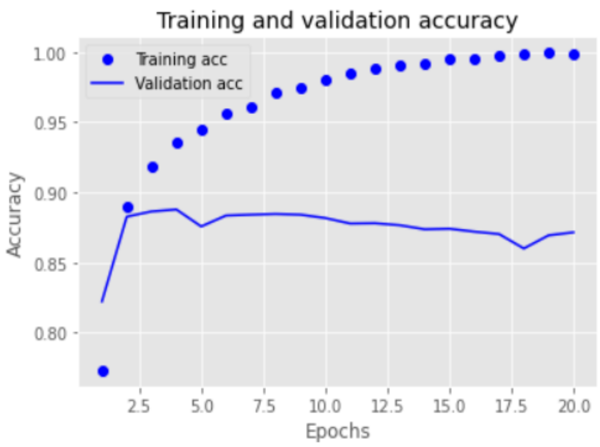
Plotting the Results:

We will now use the Matplotlib to plot the training, validation loss as well as the training, validation accuracy. The code for Plotting will be:

```python
# Training/Validation Loss
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss'
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```python
# Training/Validation Accuracy

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(1, len(acc) + 1)

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()
```

The Graphs looks like:



Retrain the model for Predictions on New Data:

Let us retrain our neural network model to predict on the new data that it hasn't seen. For that we will use just the 4 epochs.

The code for this will look like:

```
tf.random.set_seed(8985)

model = models.Sequential()
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history1 = model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

The generated results will look similar like this:

```
Epoch 1/4
49/49 [==============================] - 2s 28ms/step - loss: 0.4894 - accuracy: 0.8052
Epoch 2/4
49/49 [==============================] - 1s 27ms/step - loss: 0.2915 - accuracy: 0.8994
Epoch 3/4
49/49 [==============================] - 1s 27ms/step - loss: 0.2259 - accuracy: 0.9188
Epoch 4/4
49/49 [==============================] - 1s 26ms/step - loss: 0.1920 - accuracy: 0.9314
782/782 [==============================] - 3s 3ms/step - loss: 0.2802 - accuracy: 0.8885
```

Up to now we have built the model and make predictions on new data. We now take this to one step ahead by doing this Scenarios. The Scenarios are

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

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

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

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

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

So, let us do every scenario and try to identify the results for Accuracy and the Loss for each Scenario.

1 Layer,16 Hidden Units, Relu Activation, binary cross entropy:

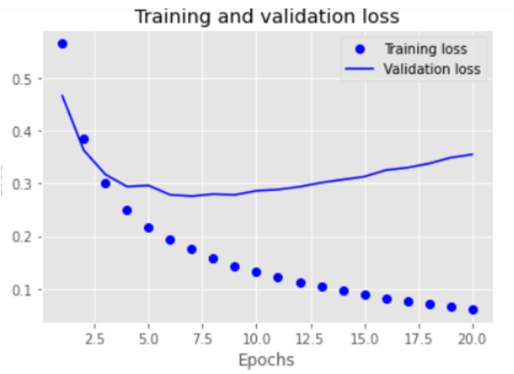
In this scenario we will be using 1 hidden Layer, 16 Hidden Units, Relu Activation Function and binary cross entropy as the loss function.

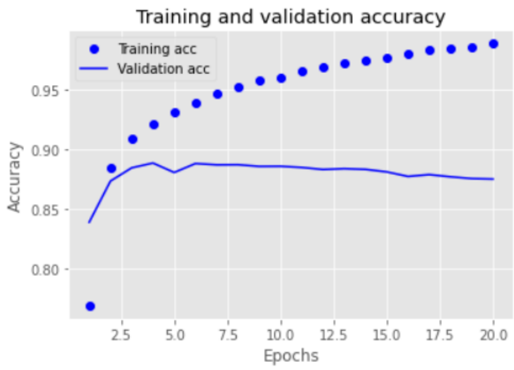
```
# One Hidden Layer
tf.random.set_seed(8985)

model = models.Sequential()
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0715 - accuracy: 0.9839 - val_loss: 0.3379 - val_accuracy: 0.8770
Epoch 19/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0656 - accuracy: 0.9855 - val_loss: 0.3491 - val_accuracy: 0.8755
Epoch 20/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0611 - accuracy: 0.9885 - val_loss: 0.3552 - val_accuracy: 0.8751
```

The Graph will be plotted like this:





The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1754 - accuracy: 0.9444
Epoch 2/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1448 - accuracy: 0.9528
Epoch 3/4
49/49 [==============================] - 1s 27ms/step - loss: 0.1279 - accuracy: 0.9599
Epoch 4/4
49/49 [==============================] - 1s 27ms/step - loss: 0.1155 - accuracy: 0.9636
782/782 [==============================] - 2s 3ms/step - loss: 0.3657 - accuracy: 0.8692
```

3 Layer,16 Hidden Units, Relu Activation, binary cross entropy:

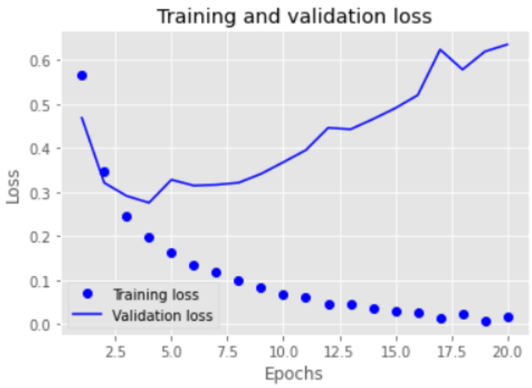
In this scenario we will be using 3 hidden Layer, 16 Hidden Units, Relu Activation Function and binary cross entropy as the loss function.

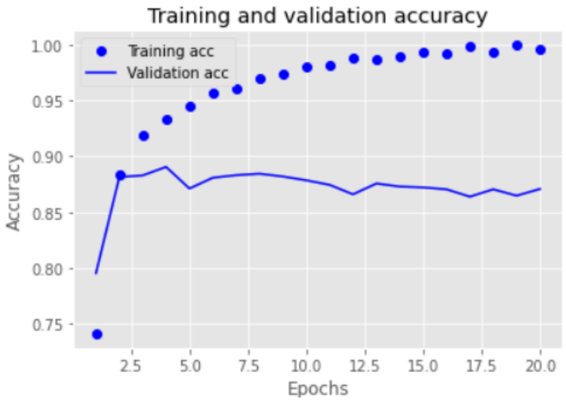
```
# Three Hidden Layer
tf.random.set_seed(8985)

model = models.Sequential()
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0232 - accuracy: 0.9942 - val_loss: 0.5779 - val_accuracy: 0.8704
Epoch 19/20
30/30 [==============================] - 2s 62ms/step - loss: 0.0081 - accuracy: 0.9998 - val_loss: 0.6189 - val_accuracy: 0.8647
Epoch 20/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0166 - accuracy: 0.9963 - val_loss: 0.6347 - val_accuracy: 0.8707
```

The Graph will be plotted like this:





The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 1s 29ms/step - loss: 0.2263 - accuracy: 0.9464
Epoch 2/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1312 - accuracy: 0.9611
Epoch 3/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1037 - accuracy: 0.9698
Epoch 4/4
49/49 [==============================] - 2s 37ms/step - loss: 0.0847 - accuracy: 0.9745
782/782 [==============================] - 2s 3ms/step - loss: 0.4600 - accuracy: 0.8634
```

2 Layer,32 Hidden Units, Relu Activation, binary cross entropy:

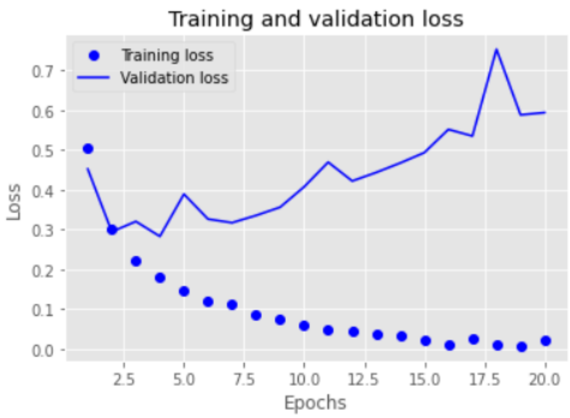
In this scenario we will be using 2 hidden Layer, 32 Hidden Units, Relu Activation Function and binary cross entropy as the loss function.

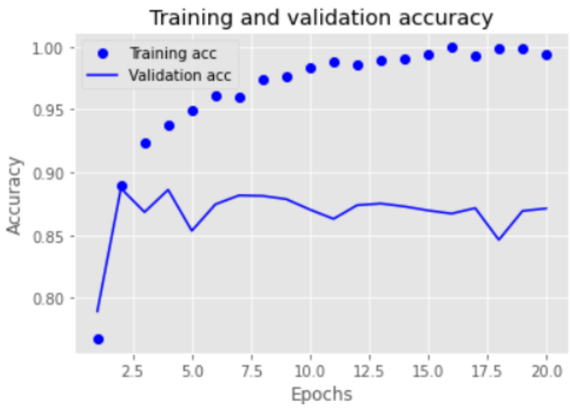
```
# 32 Hidden Layers
tf.random.set_seed(8985)

model = models.Sequential()
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0097 - accuracy: 0.9989 - val_loss: 0.7532 - val_accuracy: 0.8463
Epoch 19/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0074 - accuracy: 0.9992 - val_loss: 0.5883 - val_accuracy: 0.8693
Epoch 20/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0219 - accuracy: 0.9939 - val_loss: 0.5940 - val_accuracy: 0.8713
```

The Graph will be plotted like this





The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 2s 34ms/step - loss: 0.1991 - accuracy: 0.9464
Epoch 2/4
49/49 [==============================] - 2s 39ms/step - loss: 0.1179 - accuracy: 0.9628
Epoch 3/4
49/49 [==============================] - 2s 47ms/step - loss: 0.0800 - accuracy: 0.9767
Epoch 4/4
49/49 [==============================] - 2s 32ms/step - loss: 0.0563 - accuracy: 0.9838
782/782 [==============================] - 2s 3ms/step - loss: 0.4639 - accuracy: 0.8663
```

2 Layer,64 Hidden Units, Relu Activation, binary cross entropy:

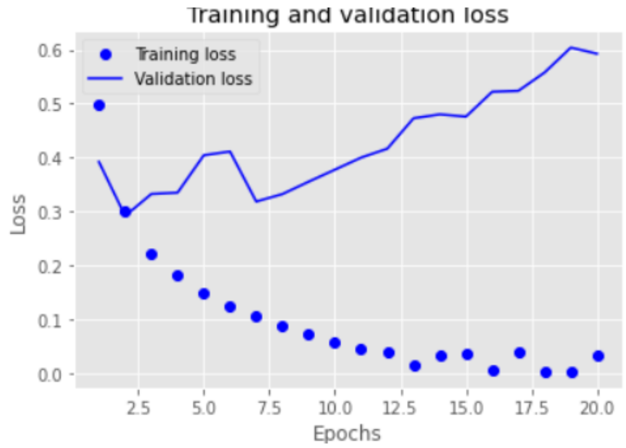
In this scenario we will be using 2 hidden Layer, 64 Hidden Units, Relu Activation Function and binary cross entropy as the loss function.

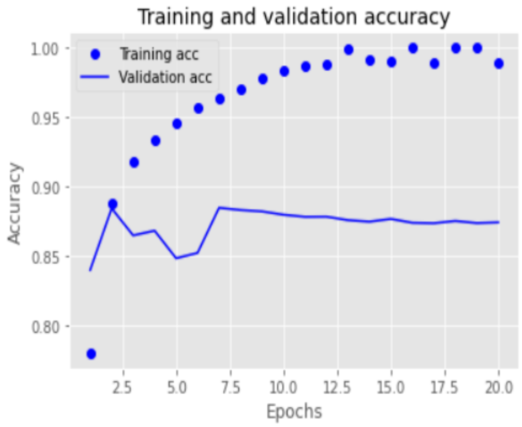
```
# 64 Layers
tf.random.set_seed(8985)

model  = models.Sequential()
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 2s 67ms/step - loss: 0.0039 - accuracy: 0.9999 - val_loss: 0.5578 - val_accuracy: 0.8750
Epoch 19/20
30/30 [==============================] - 4s 120ms/step - loss: 0.0028 - accuracy: 0.9999 - val_loss: 0.6040 - val_accuracy: 0.8735
Epoch 20/20
30/30 [==============================] - 2s 63ms/step - loss: 0.0339 - accuracy: 0.9893 - val_loss: 0.5926 - val_accuracy: 0.8740
```

The Graph will be plotted like this:





The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 2s 47ms/step - loss: 0.1783 - accuracy: 0.9472
Epoch 2/4
49/49 [==============================] - 2s 46ms/step - loss: 0.0988 - accuracy: 0.9691
Epoch 3/4
49/49 [==============================] - 2s 49ms/step - loss: 0.0607 - accuracy: 0.9823
Epoch 4/4
49/49 [==============================] - 3s 66ms/step - loss: 0.0389 - accuracy: 0.9897
782/782 [==============================] - 3s 4ms/step - loss: 0.4531 - accuracy: 0.8706
```

2 Layer,128 Hidden Units, Relu Activation, binary cross entropy:

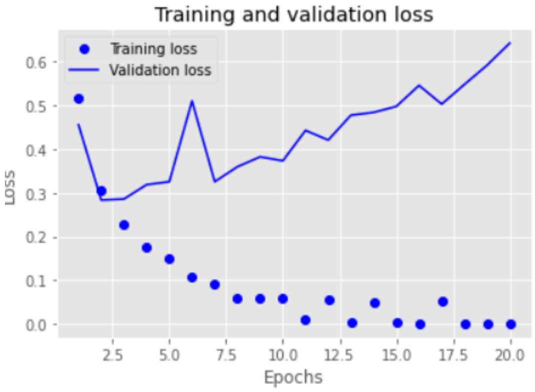
In this scenario we will be using 2 hidden Layer, 128 Hidden Units, Relu Activation Function and binary cross entropy as the loss function.

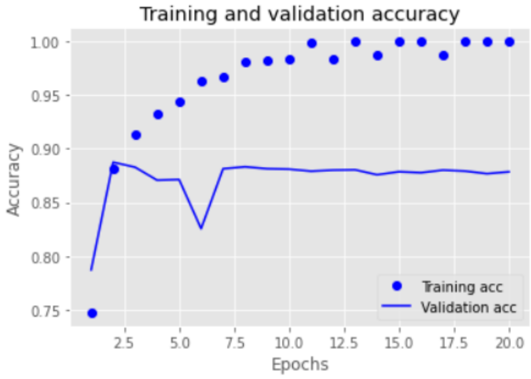
```
# 128 Layers
tf.random.set_seed(8985)

model  = models.Sequential()
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 3s 89ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.5477 - val_accuracy: 0.8792
Epoch 19/20
30/30 [==============================] - 3s 116ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.5919 - val_accuracy: 0.8768
Epoch 20/20
30/30 [==============================] - 3s 113ms/step - loss: 7.3886e-04 - accuracy: 1.0000 - val_loss: 0.6424 - val_accuracy: 0.8785
```

The Graph will be plotted like this:





The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 5s 96ms/step - loss: 0.1887 - accuracy: 0.9460
Epoch 2/4
49/49 [==============================] - 3s 70ms/step - loss: 0.0854 - accuracy: 0.9739
Epoch 3/4
49/49 [==============================] - 3s 69ms/step - loss: 0.0483 - accuracy: 0.9864
Epoch 4/4
49/49 [==============================] - 7s 134ms/step - loss: 0.0294 - accuracy: 0.9914
782/782 [==============================] - 4s 5ms/step - loss: 0.4423 - accuracy: 0.8751
```

2 Layer,16 Hidden Units, Relu Activation, MSE:

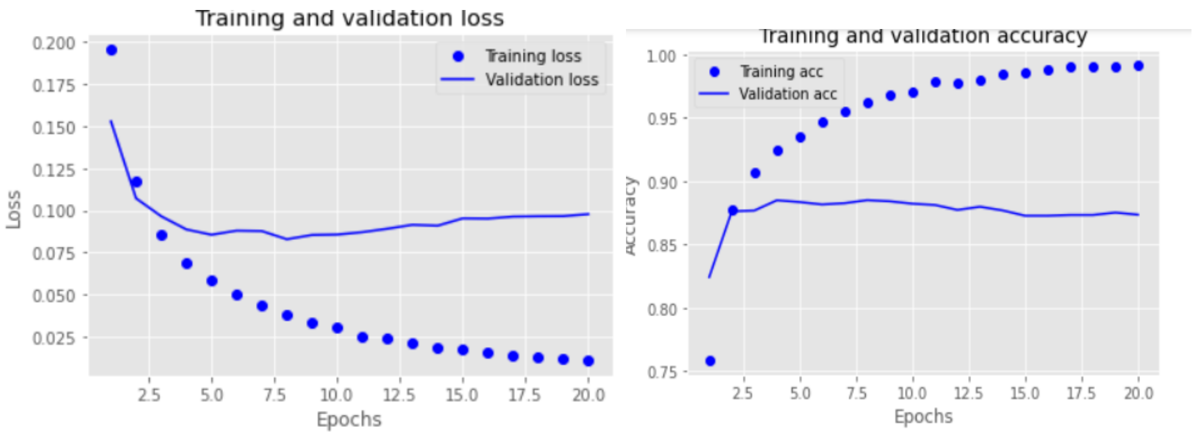
In this scenario we will be using 2 hidden Layer, 16 Hidden Units, Relu Activation Function and MSE as the loss function.

```
# MSE Loass Function
tf.random.set_seed(8985)

model = models.Sequential()
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0128 - accuracy: 0.9903 - val_loss: 0.0965 - val_accuracy: 0.8732
Epoch 19/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0120 - accuracy: 0.9903 - val_loss: 0.0965 - val_accuracy: 0.8751
Epoch 20/20
30/30 [==============================] - 2s 62ms/step - loss: 0.0109 - accuracy: 0.9911 - val_loss: 0.0977 - val_accuracy: 0.8734
```

The Graph will be plotted like this:



The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 1s 28ms/step - loss: 0.0447 - accuracy: 0.9466
Epoch 2/4
49/49 [==============================] - 1s 28ms/step - loss: 0.0357 - accuracy: 0.9594
Epoch 3/4
49/49 [==============================] - 1s 27ms/step - loss: 0.0312 - accuracy: 0.9657
Epoch 4/4
49/49 [==============================] - 1s 28ms/step - loss: 0.0271 - accuracy: 0.9718
782/782 [==============================] - 2s 3ms/step - loss: 0.1027 - accuracy: 0.8704
```

2 Layer,16 Hidden Units, Tanh Activation, binary cross entropy:

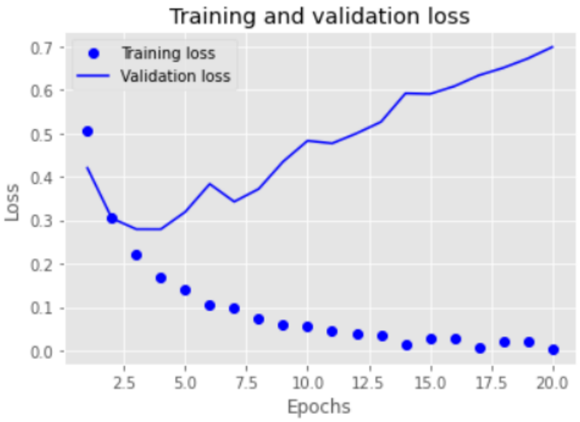
In this scenario we will be using 2 hidden Layer, 16 Hidden Units, Tanh Activation Function and binary cross entropy as the loss function.

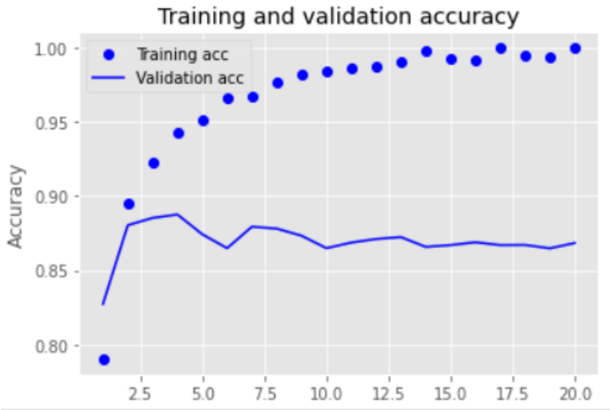
```
# Tanh Activation Function
tf.random.set_seed(8985)

model  = models.Sequential()
model.add(layers.Dense(16, activation='tanh'))
model.add(layers.Dense(16, activation='tanh'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0203 - accuracy: 0.9944 - val_loss: 0.6507 - val_accuracy: 0.8671
Epoch 19/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0197 - accuracy: 0.9939 - val_loss: 0.6722 - val_accuracy: 0.8648
Epoch 20/20
30/30 [==============================] - 2s 63ms/step - loss: 0.0038 - accuracy: 0.9998 - val_loss: 0.6986 - val_accuracy: 0.8684
```

The Graph will be plotted like this:





The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 1s 29ms/step - loss: 0.2617 - accuracy: 0.9418
Epoch 2/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1433 - accuracy: 0.9579
Epoch 3/4
49/49 [==============================] - 1s 26ms/step - loss: 0.1153 - accuracy: 0.9661
Epoch 4/4
49/49 [==============================] - 3s 62ms/step - loss: 0.0994 - accuracy: 0.9696
782/782 [==============================] - 2s 3ms/step - loss: 0.4720 - accuracy: 0.8601
```

2 Layer,16 Hidden Units, Relu Activation, binary cross entropy, dropout:

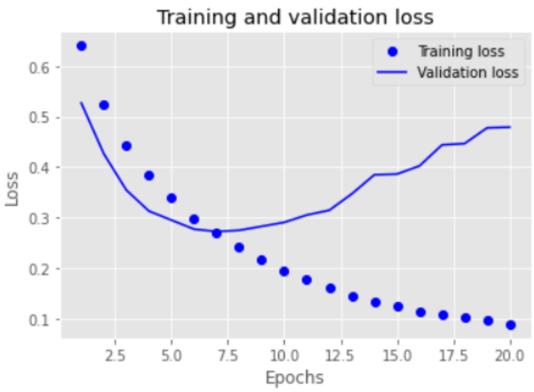
In this scenario we will be using 2 hidden Layer, 16 Hidden Units, Relu Activation Function and binary cross entropy as the loss function, dropout technique.

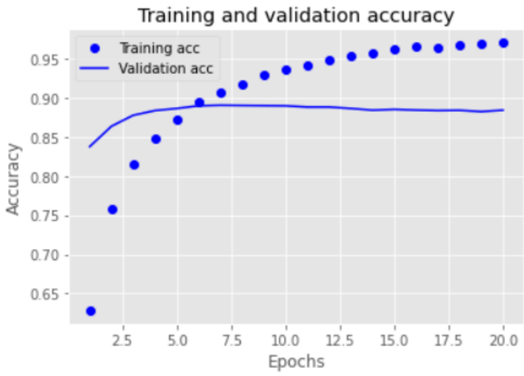
```
# Dropout Technique
tf.random.set_seed(8985)

model  = models.Sequential()
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 2s 67ms/step - loss: 0.1025 - accuracy: 0.9673 - val_loss: 0.4465 - val_accuracy: 0.8839
Epoch 19/20
30/30 [==============================] - 2s 50ms/step - loss: 0.0972 - accuracy: 0.9699 - val_loss: 0.4776 - val_accuracy: 0.8823
Epoch 20/20
30/30 [==============================] - 2s 71ms/step - loss: 0.0893 - accuracy: 0.9703 - val_loss: 0.4790 - val_accuracy: 0.8841
```

The Graph will be plotted like this:





The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 1s 30ms/step - loss: 0.2492 - accuracy: 0.9246
Epoch 2/4
49/49 [==============================] - 1s 29ms/step - loss: 0.2085 - accuracy: 0.9355
Epoch 3/4
49/49 [==============================] - 1s 29ms/step - loss: 0.1898 - accuracy: 0.9419
Epoch 4/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1760 - accuracy: 0.9466
782/782 [==============================] - 2s 3ms/step - loss: 0.4179 - accuracy: 0.8774
```

2 Layer,16 Hidden Units, Relu Activation, binary cross entropy, Regularization:

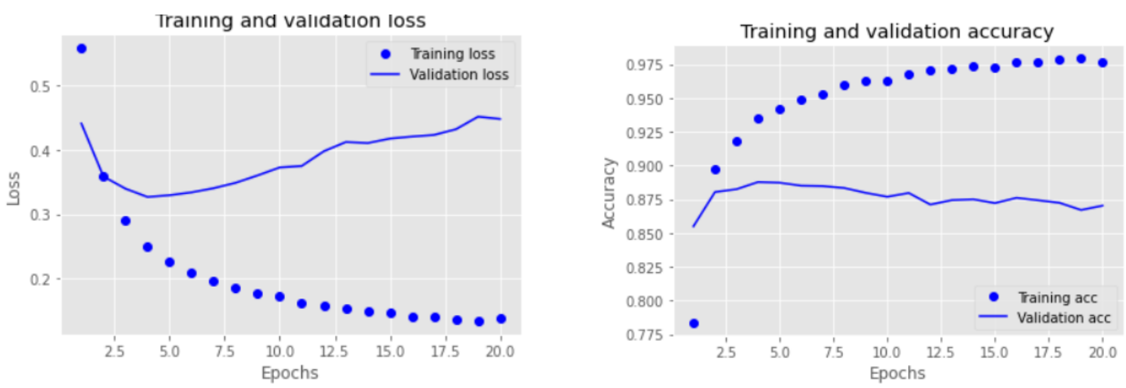
In this scenario we will be using 2 hidden Layer, 16 Hidden Units, Relu Activation Function and binary cross entropy as the loss function, Regularization technique.

```
# Regularization Technique
from keras import regularizers

tf.random.set_seed(1234)

model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),activation='relu'))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))


model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

This is generating the Results like this for 20 iterations:

```
Epoch 18/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1360 - accuracy: 0.9786 - val_loss: 0.4323 - val_accuracy: 0.8723
Epoch 19/20
30/30 [==============================] - 1s 41ms/step - loss: 0.1336 - accuracy: 0.9793 - val_loss: 0.4518 - val_accuracy: 0.8670
Epoch 20/20
30/30 [==============================] - 1s 47ms/step - loss: 0.1380 - accuracy: 0.9765 - val_loss: 0.4483 - val_accuracy: 0.8702
```

The Graph will be plotted like this:



The results for 4 epochs for predicting will be like:

```
Epoch 1/4
49/49 [==============================] - 1s 30ms/step - loss: 0.2516 - accuracy: 0.9373
Epoch 2/4
49/49 [==============================] - 1s 30ms/step - loss: 0.2098 - accuracy: 0.9457
Epoch 3/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1984 - accuracy: 0.9481
Epoch 4/4
49/49 [==============================] - 1s 28ms/step - loss: 0.1860 - accuracy: 0.9535
782/782 [==============================] - 3s 3ms/step - loss: 0.4162 - accuracy: 0.8675
```

Summary:

Now we will summarize all the results of accuracy, loss in the tabular Form for clear understanding.

Scenarios Used	Loss		Accuracy	
	Validation	Test	Validation	Test
1 Layer,16 Hidden Units, Relu Activation, binary cross entropy	0.3552	0.3657	87.51%	86.92%
2 Layer,16 Hidden Units, Relu Activation, binary cross entropy	0.5672	0.2802	87.15%	88.85%
3 Layer,16 Hidden Units, Relu Activation, binary cross entropy	0.6347	0.4600	87.07%	86.34%
2 Layer,32 Hidden Units, Relu Activation, binary cross entropy	0.5940	0.4639	87.13%	86.63%
2 Layer,64 Hidden Units, Relu Activation, binary cross entropy	0.5926	0.4531	87.40%	87.06%
2 Layer,128 Hidden Units, Relu Activation, binary cross entropy	0.6424	0.4423	87.85%	87.51%
2 Layer,16 Hidden Units, Relu Activation, MSE	0.0977	0.1027	87.34%	87.04%
2 Layer,16 Hidden Units, Tanh Activation, binary cross entropy	0.6986	0.4720	86.84%	86.01%
2 Layer,16 Hidden Units, Relu Activation, binary cross entropy, dropout	0.4790	0.4179	88.41%	87.74%
2 Layer,16 Hidden Units, Relu Activation, binary cross entropy, Regularization	0.4483	0.4162	87.02%	86.75%

Conclusions:

1 Hidden layer Validation Accuracy is best compared to 2 and 3 Hidden Layers.

The greater number of hidden Units the better the validation Accuracy.

The MSE loss function performs better than the binary_crossentropy loss function.

When we are using the relu or tanh functions, there is a slightly difference in the Validation Accuracy

On both the test and validation datasets, the model with 2 Dense Layers, 16 Hidden Units, Relu Activation, binary_crossentropy loss function, and dropout (0.5) approach has the maximum accuracy.

On the test dataset, the model with 2 dense layers, 16 hidden units, Tanh Activation, and binary_crossentropy loss function has the lowest accuracy.

On the validation dataset, the model with 2 Dense Layers, 16 Hidden Units, Tanh Activation, and binary_crossentropy loss function, Regularization has the lowest accuracy.

References:

Lecture 4: Examples of NN 2:

https://kent.instructure.com/courses/57954/pages/lecture-4-examples-of-nn-2?module_item_id=2966279

IMDB - Sentiment analysis Keras and TensorFlow:

https://www.kaggle.com/code/drscarlat/imdb-sentiment-analysis-keras-and-tensorflow/notebook

How to Build a Neural Network With Keras Using the IMDB Dataset:

https://builtin.com/data-science/how-build-neural-network-keras