Assignment 5.1 - The IMDB Dataset

Loading Data Set

In [1]:
```
import numpy as np
from tensorflow.keras.datasets import imdb
```

In [2]:
```
# splitting into training and test dataset.
# each review is list of word indices. train labels: 0 = negative, 1 = positiv
e.
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_word
s=10000)
```

In [3]:
```
max([max(sequence) for sequence in train_data])
```

Out[3]: 9999

In [4]:
```
# maps words to an integer index
word_index = imdb.get_word_index()
#word_index
```

In [5]:
```
# maps integer indices to words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items
()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_d
ata[0]])
decoded_review
```

Out[5]:
```
"? this film was just brilliant casting location scenery story direction ever
yone's really suited the part they played and you could just imagine being th
ere robert ? is an amazing actor and now the same being director ? father cam
e from the same scottish island as myself so i loved the fact there was a rea
l connection with this film the witty remarks throughout the film were great
it was just brilliant so much that i bought the film as soon as it was releas
ed for ? and would recommend it to everyone to watch and the fly fishing was
amazing really cried at the end it was so sad and you know what they say if y
ou cry at a film it must have been good and this definitely was also ? to the
two little boy's that played the ? of norman and paul they were just brillian
t children are often left out of the ? list i think because the stars that pl
ay them all grown up are such a big profile for the whole film but these chil
dren are amazing and should be praised for what they have done don't you thin
k the whole story was so lovely because it was true and was someone's life af
ter all that was shared with us all"
```

Preparing Dataset

In [6]:
```
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)
```

In [7]:
```
x_train[0]
```

Out[7]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [8]:
```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Building your network

In [9]:
```
# Import keras libraries
from keras import models, layers, losses, metrics, optimizers
```

```
Using TensorFlow backend.
```

In [10]:
```
# Creating a neural net with shape (1000,16,16,1)
model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input_shape = (10000,)))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
```

In [11]:
```
model.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])
```

In [12]:
```
model.compile(optimizer = optimizers.RMSprop(lr = 0.001),
             loss = 'binary_crossentropy',
             metrics = ['accuracy'])
```

In [13]:
```
# using custom loses and metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

Validating the model

In [14]:
```
# Splitting the data into validation and train set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training the model

In [15]:
```
model.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

In [16]:
```
# Training the neural network with partial_x_train and partial_y_train
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data = (x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 197us/step - loss: 0.5084 -
acc: 0.7813 - val_loss: 0.3797 - val_acc: 0.8684
Epoch 2/20
15000/15000 [==============================] - 2s 153us/step - loss: 0.3004 -
acc: 0.9047 - val_loss: 0.3004 - val_acc: 0.8897
Epoch 3/20
15000/15000 [==============================] - 2s 152us/step - loss: 0.2179 -
acc: 0.9285 - val_loss: 0.3085 - val_acc: 0.8711
Epoch 4/20
15000/15000 [==============================] - 2s 157us/step - loss: 0.1750 -
acc: 0.9437 - val_loss: 0.2840 - val_acc: 0.8832
Epoch 5/20
15000/15000 [==============================] - 2s 145us/step - loss: 0.1427 -
acc: 0.9543 - val_loss: 0.2841 - val_acc: 0.8872
Epoch 6/20
15000/15000 [==============================] - 2s 149us/step - loss: 0.1150 -
acc: 0.9650 - val_loss: 0.3166 - val_acc: 0.8772
Epoch 7/20
15000/15000 [==============================] - 2s 152us/step - loss: 0.0980 -
acc: 0.9705 - val_loss: 0.3127 - val_acc: 0.8846
Epoch 8/20
15000/15000 [==============================] - 2s 155us/step - loss: 0.0807 -
acc: 0.9763 - val_loss: 0.3861 - val_acc: 0.8649
Epoch 9/20
15000/15000 [==============================] - 2s 154us/step - loss: 0.0661 -
acc: 0.9821 - val_loss: 0.3637 - val_acc: 0.8782
Epoch 10/20
15000/15000 [==============================] - 2s 148us/step - loss: 0.0561 -
acc: 0.9853 - val_loss: 0.3850 - val_acc: 0.8792
Epoch 11/20
15000/15000 [==============================] - 2s 155us/step - loss: 0.0439 -
acc: 0.9893 - val_loss: 0.4168 - val_acc: 0.8779
Epoch 12/20
15000/15000 [==============================] - 2s 154us/step - loss: 0.0381 -
acc: 0.9921 - val_loss: 0.4552 - val_acc: 0.8690
Epoch 13/20
15000/15000 [==============================] - 2s 152us/step - loss: 0.0300 -
acc: 0.9928 - val_loss: 0.4733 - val_acc: 0.8729
Epoch 14/20
15000/15000 [==============================] - 2s 157us/step - loss: 0.0247 -
acc: 0.9945 - val_loss: 0.5068 - val_acc: 0.8726
Epoch 15/20
15000/15000 [==============================] - 2s 153us/step - loss: 0.0175 -
acc: 0.9979 - val_loss: 0.5412 - val_acc: 0.8693
Epoch 16/20
15000/15000 [==============================] - 2s 153us/step - loss: 0.0149 -
acc: 0.9983 - val_loss: 0.5802 - val_acc: 0.8699
Epoch 17/20
15000/15000 [==============================] - 2s 155us/step - loss: 0.0151 -
acc: 0.9971 - val_loss: 0.6151 - val_acc: 0.8697
Epoch 18/20
15000/15000 [==============================] - 2s 156us/step - loss: 0.0075 -
acc: 0.9996 - val_loss: 0.6948 - val_acc: 0.8632
Epoch 19/20
15000/15000 [==============================] - 2s 155us/step - loss: 0.0121 -
```

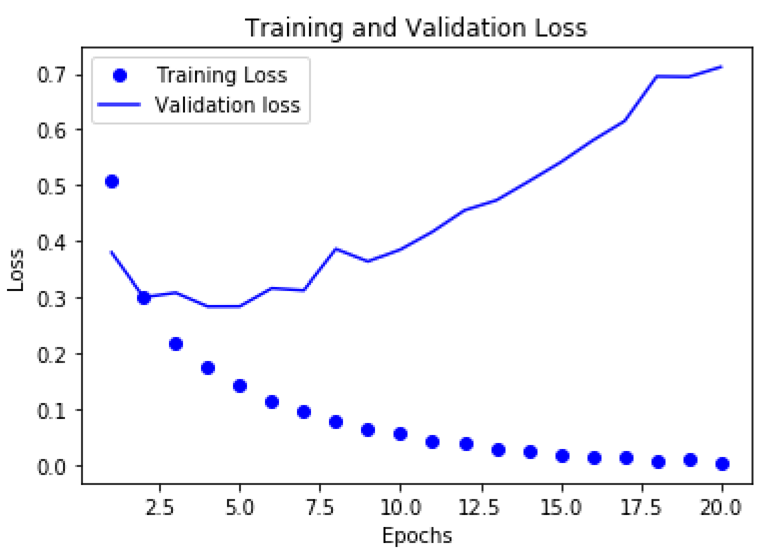
```
acc: 0.9971 - val_loss: 0.6941 - val_acc: 0.8652
Epoch 20/20
15000/15000 [==============================] - 2s 152us/step - loss: 0.0041 -
acc: 0.9999 - val_loss: 0.7117 - val_acc: 0.8657
```

In [17]:
```
history_dict = history.history
history_dict.keys()
```

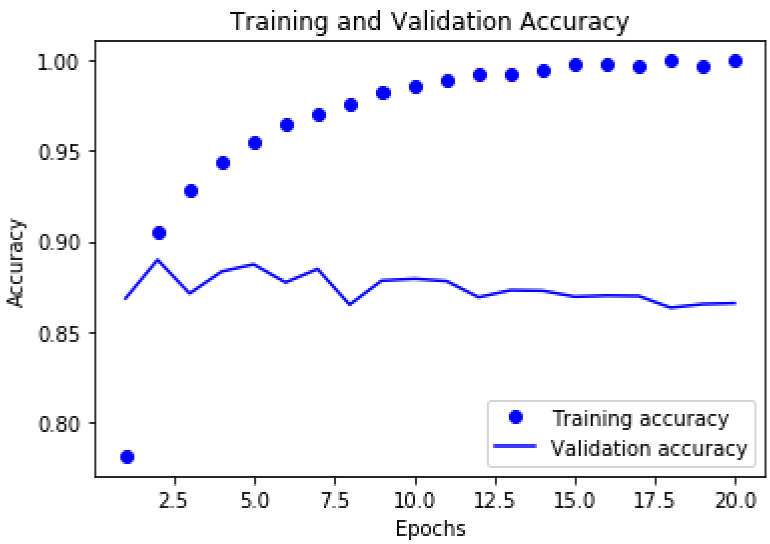
Out[17]:
```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

Plotting the training and validation loss

In [18]:
```
import matplotlib.pyplot as plt
acc = history_dict['acc']
val_acc = history_dict['val_acc']

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

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



In [19]:
```
plt.plot(epochs, acc,  'bo', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [20]:
```
# Retraining a model from scratch
model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input_shape = (10000,)))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))

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

model.fit(partial_x_train,
          partial_y_train,
          epochs = 4,
          batch_size = 512)

results = model.evaluate(x_test, y_test)
results
```

```
Epoch 1/4
15000/15000 [==============================] - 2s 109us/step - loss: 0.5326 -
accuracy: 0.7917
Epoch 2/4
15000/15000 [==============================] - 2s 101us/step - loss: 0.3258 -
accuracy: 0.8987
Epoch 3/4
15000/15000 [==============================] - 1s 97us/step - loss: 0.2357 -
accuracy: 0.9245
Epoch 4/4
15000/15000 [==============================] - 2s 102us/step - loss: 0.1866 -
accuracy: 0.9397
25000/25000 [==============================] - 4s 156us/step
```

Out[20]:
```
[0.30134918537139893, 0.8772000074386597]
```

Use trained model to generate predictions

In [21]:
```
model.predict(x_test)
```

Out[21]:
```
array([[0.33491266],
       [0.9995926 ],
       [0.9345059 ],
       ...,
       [0.14445111],
       [0.19112667],
       [0.65519667]], dtype=float32)
```

In [ ]: