```
from keras.datasets import imdb
%matplotlib inline
import numpy as np
import pandas as pd
from matplotlib import cm
import matplotlib.pyplot as plt
import seaborn as sns
import os
import time
```

```
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.layers import Embedding
from keras.layers import Conv1D, GlobalMaxPooling1D
from keras.callbacks import EarlyStopping
from keras import models
```

```
(X_train, y_train), (X_test, y_test) = imdb.load_data()
X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
```

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

```
##training data shape review
print("Training data: ")
print(X.shape)
print(y.shape)
print("Classes: ")
print(np.unique(y))
```

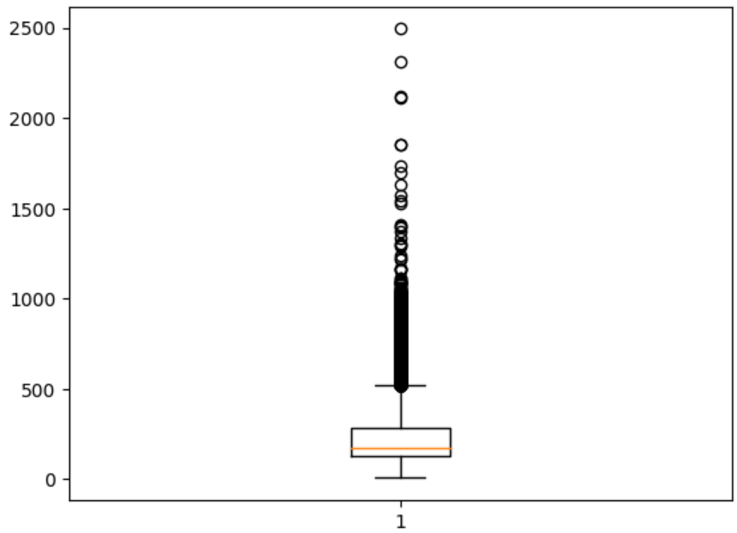
```
Training data:
(50000,)
(50000,)
Classes:
[0 1]
```

```
print("Number of words: ")
print(len(np.unique(np.hstack(X))))
```

```
Number of words:
88585
```

```
print("Review length: ")
result = [len(x) for x in X]
print("Mean %.2f words (%f)" % (np.mean(result), np.std(result)))
# plot review length
plt.boxplot(result)
plt.show()
```

```
Review length:
Mean 234.76 words (172.911495)
```




```
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=5000)
```

```
def vectorize_sequences(sequences, dimension=5000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results
```

```
# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)
```

```
# Our vectorized labels one-hot encoder
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(5000,)))
model.add(layers.Dense(32, activation='relu',))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
#Set validation set aside

x_val = x_train[:10000]
partial_x_train = x_train[10000:]

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

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
```

```
start_time_m1 = time.time()
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
total_time_m1 = time.time() - start_time_m1
```

```
Epoch 1/20
30/30 [==============================] - 3s 78ms/step - loss: 0.5351 - acc: 0.7623 - val_loss: 0.3614 - val_acc: 0.8574
Epoch 2/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2779 - acc: 0.8964 - val_loss: 0.2862 - val_acc: 0.8855
Epoch 3/20
30/30 [==============================] - 1s 30ms/step - loss: 0.2011 - acc: 0.9265 - val_loss: 0.2935 - val_acc: 0.8833
Epoch 4/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1624 - acc: 0.9409 - val_loss: 0.3109 - val_acc: 0.8782
Epoch 5/20
30/30 [==============================] - 1s 46ms/step - loss: 0.1350 - acc: 0.9536 - val_loss: 0.3385 - val_acc: 0.8736
Epoch 6/20
30/30 [==============================] - 1s 47ms/step - loss: 0.1106 - acc: 0.9637 - val_loss: 0.3689 - val_acc: 0.8701
Epoch 7/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0885 - acc: 0.9737 - val_loss: 0.4085 - val_acc: 0.8654
Epoch 8/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0693 - acc: 0.9810 - val_loss: 0.4534 - val_acc: 0.8643
Epoch 9/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0520 - acc: 0.9876 - val_loss: 0.4986 - val_acc: 0.8616
Epoch 10/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0385 - acc: 0.9935 - val_loss: 0.5402 - val_acc: 0.8613
Epoch 11/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0276 - acc: 0.9964 - val_loss: 0.5897 - val_acc: 0.8581
Epoch 12/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0200 - acc: 0.9986 - val_loss: 0.6346 - val_acc: 0.8564
Epoch 13/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0143 - acc: 0.9994 - val_loss: 0.6705 - val_acc: 0.8558
Epoch 14/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0103 - acc: 0.9998 - val_loss: 0.7077 - val_acc: 0.8572
Epoch 15/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0077 - acc: 0.9999 - val_loss: 0.7414 - val_acc: 0.8578
Epoch 16/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0059 - acc: 0.9999 - val_loss: 0.7690 - val_acc: 0.8551
Epoch 17/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0047 - acc: 1.0000 - val_loss: 0.7988 - val_acc: 0.8568
Epoch 18/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0039 - acc: 1.0000 - val_loss: 0.8231 - val_acc: 0.8567
Epoch 19/20
```

```
30/30 [==============================] - 2s 58ms/step - loss: 0.0032 - acc: 1.0000 - val_loss: 0.8450 - val_acc: 0.8556
Epoch 20/20
30/30 [==============================] - 1s 29ms/step - loss: 0.0028 - acc: 1.0000 - val_loss: 0.8674 - val_acc: 0.8553
```

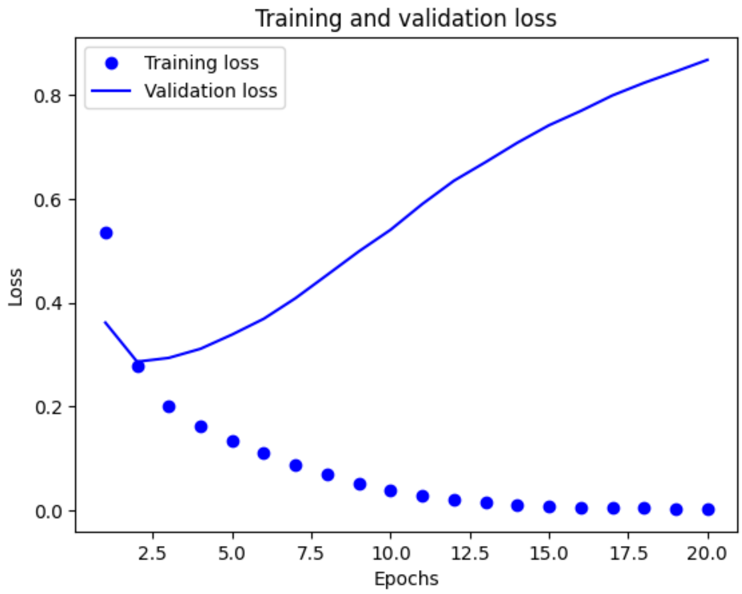
```
print("The Dense Convolutional Neural Network 1 layer took %.4f seconds to train." % (total_time_m1))
```

```
The Dense Convolutional Neural Network 1 layer took 42.7470 seconds to train.
```

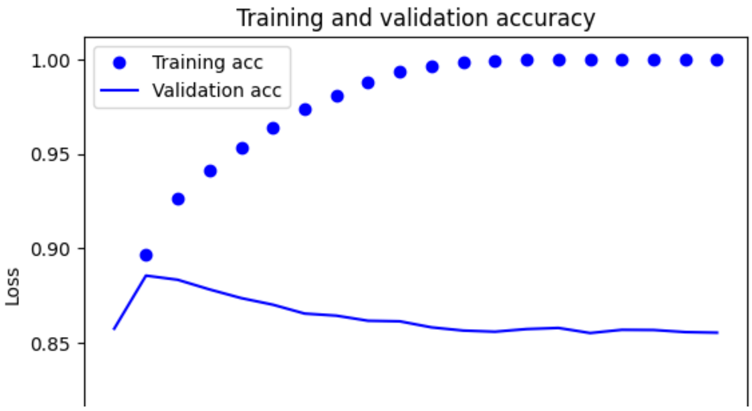
```
history_dict = history.history
history_dict.keys()
```

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

```
import matplotlib.pyplot as plt
%matplotlib inline

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

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




```
plt.clf()   # clear figure
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

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('Loss')
plt.legend()

plt.show()
```




```
model.summary()
```

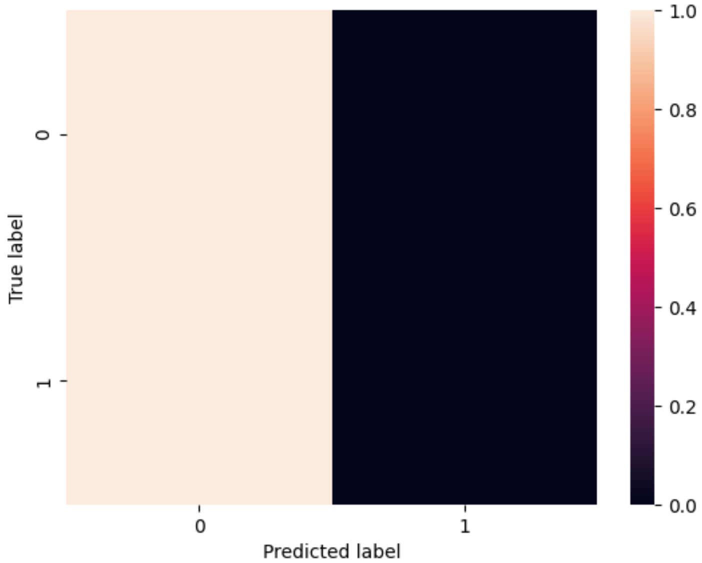
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                160032

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 161,121
Trainable params: 161,121
Non-trainable params: 0
_________________________________________________________________
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, auc
#predictions
pred = model.predict(x_test)
classes_x=np.argmax(pred,axis=1)

#accuracy
accuracy_score(y_test,classes_x)
```

```
782/782 [==============================] - 3s 3ms/step
0.5
```

```
#Confusion Matrix
conf_mat = confusion_matrix(y_test, classes_x)
print(conf_mat)

conf_mat_normalized = conf_mat.astype('float') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat_normalized)
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
[[12500     0]
 [12500     0]]
Text(0.5, 23.52222222222222, 'Predicted label')
```



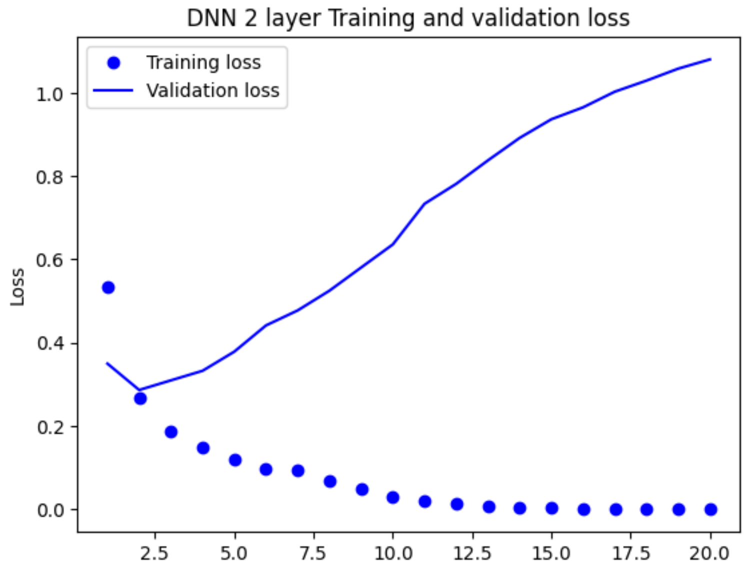
```
#Dense with Two Layer
model2 = models.Sequential()
model2.add(layers.Dense(32, activation='relu', input_shape=(5000,)))
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(1, activation='sigmoid'))
```

```
model2.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
```

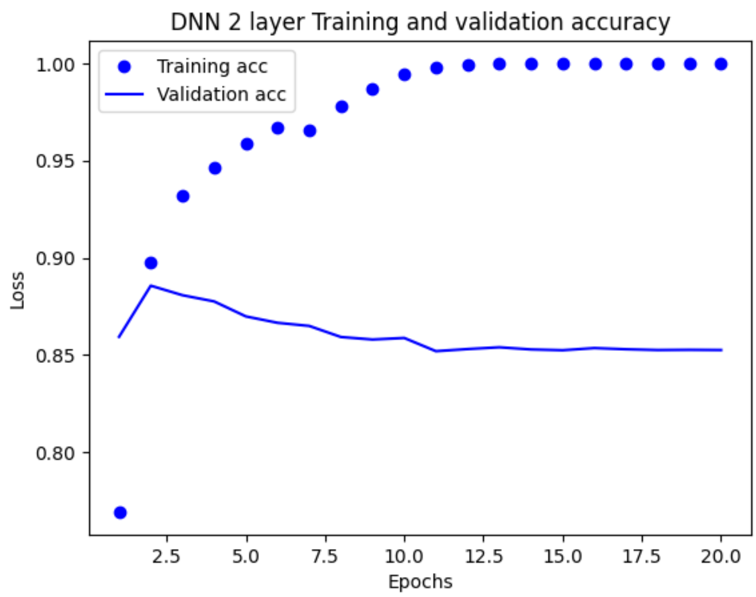
```
start_time_m2 = time.time()
history= model2.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
total_time_m2 = time.time() - start_time_m2

print("The Dense Convolutional Neural Network 2 layers took %.4f seconds to train." % (total_time_m2))
```

```
Epoch 1/20
30/30 [==============================] - 3s 53ms/step - loss: 0.5337 - acc: 0.7692 - val_loss: 0.3494 - val_acc: 0.8593
Epoch 2/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2657 - acc: 0.8977 - val_loss: 0.2862 - val_acc: 0.8856
Epoch 3/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1884 - acc: 0.9320 - val_loss: 0.3093 - val_acc: 0.8807
Epoch 4/20
30/30 [==============================] - 1s 31ms/step - loss: 0.1493 - acc: 0.9462 - val_loss: 0.3323 - val_acc: 0.8775
Epoch 5/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1197 - acc: 0.9589 - val_loss: 0.3782 - val_acc: 0.8698
Epoch 6/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0973 - acc: 0.9667 - val_loss: 0.4412 - val_acc: 0.8665
Epoch 7/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0929 - acc: 0.9653 - val_loss: 0.4773 - val_acc: 0.8649
Epoch 8/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0677 - acc: 0.9779 - val_loss: 0.5246 - val_acc: 0.8592
Epoch 9/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0473 - acc: 0.9867 - val_loss: 0.5802 - val_acc: 0.8579
Epoch 10/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0309 - acc: 0.9946 - val_loss: 0.6357 - val_acc: 0.8587
Epoch 11/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0195 - acc: 0.9977 - val_loss: 0.7333 - val_acc: 0.8519
Epoch 12/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0118 - acc: 0.9990 - val_loss: 0.7813 - val_acc: 0.8530
Epoch 13/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0066 - acc: 0.9998 - val_loss: 0.8375 - val_acc: 0.8539
Epoch 14/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0040 - acc: 1.0000 - val_loss: 0.8915 - val_acc: 0.8528
Epoch 15/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0027 - acc: 1.0000 - val_loss: 0.9363 - val_acc: 0.8524
Epoch 16/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0020 - acc: 1.0000 - val_loss: 0.9649 - val_acc: 0.8535
Epoch 17/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0015 - acc: 1.0000 - val_loss: 1.0025 - val_acc: 0.8529
Epoch 18/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0012 - acc: 1.0000 - val_loss: 1.0291 - val_acc: 0.8525
Epoch 19/20
30/30 [==============================] - 2s 75ms/step - loss: 9.9880e-04 - acc: 1.0000 - val_loss: 1.0578 - val_acc: 0.8
Epoch 20/20
30/30 [==============================] - 1s 36ms/step - loss: 8.3502e-04 - acc: 1.0000 - val_loss: 1.0800 - val_acc: 0.8
The Dense Convolutional Neural Network 2 layers took 24.6517 seconds to train.
```

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 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('DNN 2 layer Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
plt.clf()   # clear figure
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('DNN 2 layer Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_3 (Dense)             (None, 32)                160032    
                                                                 
 dense_4 (Dense)             (None, 32)                1056      
                                                                 
 dense_5 (Dense)             (None, 32)                1056      
                                                                 
 dense_6 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 162,177
Trainable params: 162,177
Non-trainable params: 0
_________________________________________________________________
```

```
from numpy.ma.core import argmax
pred = model2.predict(x_test)
classes_x=argmax(pred,axis=-1)
#accuracy
accuracy_score(y_test,classes_x)
```

```
782/782 [==============================] - 2s 2ms/step
0.5
```

✓ 2s completed at 1:06 PM