In [ ]:

```
# DL_p2 : Binary classification using Deep Neural Networks Example:
# Classify movie reviews into positive" reviews and "negative" reviews, jus
# Use IMDB dataset
```

In [1]:

```
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
```

In [2]:

```
(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-d
atasets/imdb.npz (https://storage.googleapis.com/tensorflow/tf-keras-datas
ets/imdb.npz)
17464789/17464789 [==============================] - 4s 0us/step
```

In [3]:

```
##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]
```

In [4]:

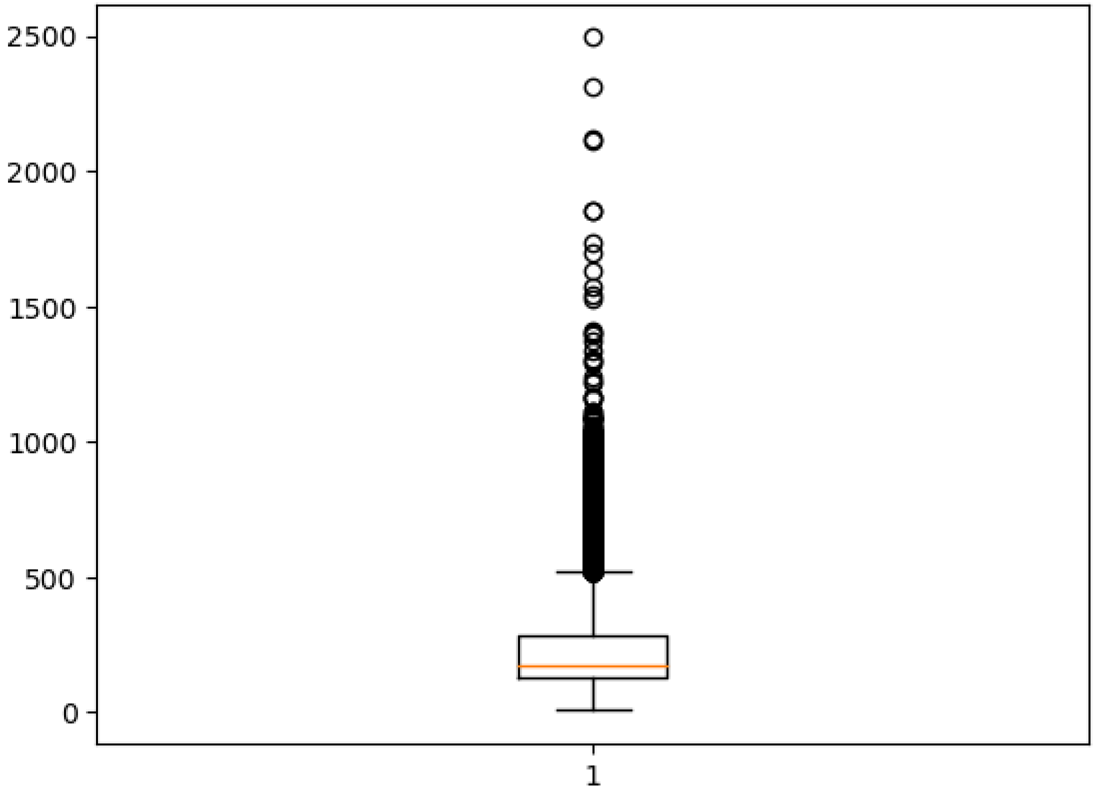
```
print("Number of words: ")
print(len(np.unique(np.hstack(X))))
```

```
Number of words: 
88585
```

In [5]:

```
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)
```



In [7]:

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

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
     return results
```

In [8]:

```
# 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')
```

In [9]:

```
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'))
```

In [10]:

```
#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 [==============================] - 12s 203ms/step - loss: 0.5383 - a
cc: 0.7623 - val_loss: 0.3618 - val_acc: 0.8545
Epoch 2/20
30/30 [==============================] - 1s 37ms/step - loss: 0.2752 - ac
c: 0.8975 - val_loss: 0.2922 - val_acc: 0.8828
Epoch 3/20
30/30 [==============================] - 1s 42ms/step - loss: 0.2009 - ac
c: 0.9271 - val_loss: 0.2943 - val_acc: 0.8821
Epoch 4/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1640 - ac
c: 0.9419 - val_loss: 0.3206 - val_acc: 0.8765
Epoch 5/20
30/30 [==============================] - 1s 32ms/step - loss: 0.1367 - ac
c: 0.9539 - val_loss: 0.3340 - val_acc: 0.8731
Epoch 6/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1118 - ac
c: 0.9619 - val_loss: 0.3632 - val_acc: 0.8717
Epoch 7/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0896 - ac
c: 0.9734 - val_loss: 0.3961 - val_acc: 0.8654
Epoch 8/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0707 - ac
c: 0.9803 - val_loss: 0.4328 - val_acc: 0.8636
Epoch 9/20
30/30 [==============================] - 1s 28ms/step - loss: 0.0508 - ac
c: 0.9887 - val_loss: 0.4717 - val_acc: 0.8612
Epoch 10/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0369 - ac
c: 0.9939 - val_loss: 0.5074 - val_acc: 0.8608
Epoch 11/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0242 - ac
c: 0.9972 - val_loss: 0.5504 - val_acc: 0.8610
Epoch 12/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0160 - ac
c: 0.9992 - val_loss: 0.5919 - val_acc: 0.8617
Epoch 13/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0108 - ac
c: 0.9999 - val_loss: 0.6167 - val_acc: 0.8587
Epoch 14/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0073 - ac
c: 0.9999 - val_loss: 0.6456 - val_acc: 0.8601
Epoch 15/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0053 - ac
c: 0.9999 - val_loss: 0.6644 - val_acc: 0.8602
Epoch 16/20
30/30 [==============================] - 2s 51ms/step - loss: 0.0040 - ac
c: 0.9999 - val_loss: 0.6851 - val_acc: 0.8598
Epoch 17/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0031 - ac
c: 1.0000 - val_loss: 0.7027 - val_acc: 0.8596
Epoch 18/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0026 - ac
c: 1.0000 - val_loss: 0.7194 - val_acc: 0.8608
Epoch 19/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0021 - ac
c: 1.0000 - val_loss: 0.7336 - val_acc: 0.8608
Epoch 20/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0018 - ac
c: 1.0000 - val_loss: 0.7467 - val_acc: 0.8605
```

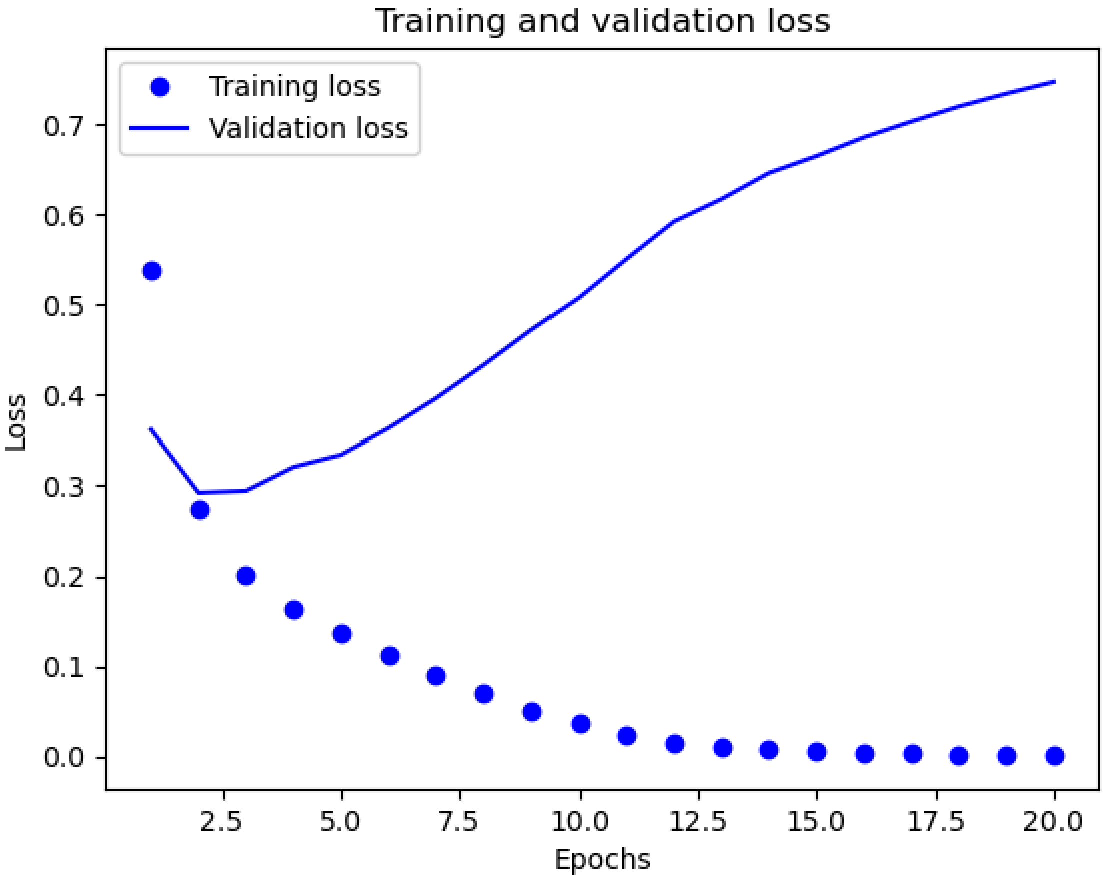
In [25]:
```
print("The Dense Convolutional Neural Network 1 layer took %.4f seconds to
      % (total_time_m1))
```

```
The Dense Convolutional Neural Network 1 layer took 45.8890 seconds to tra
in.
```

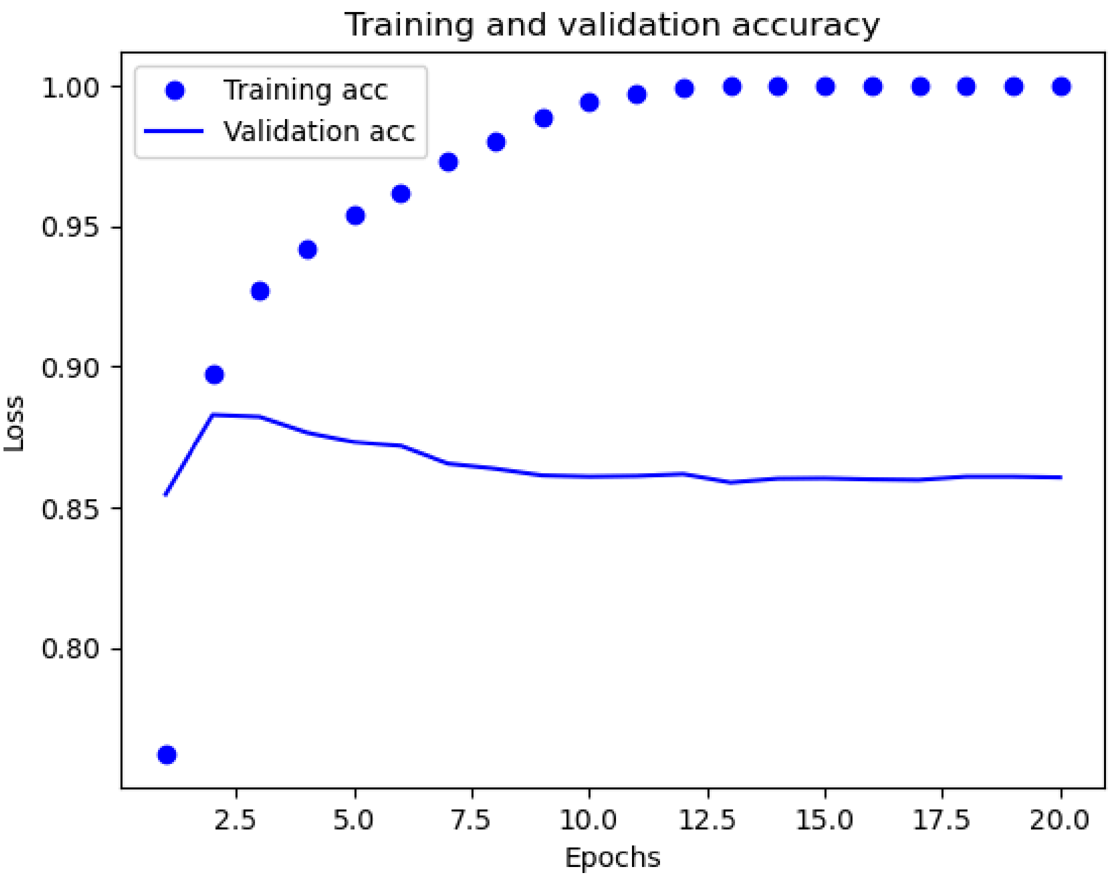
In [12]:
```
history_dict = history.history
history_dict.keys()
```

Out[12]:
```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

In [14]:
```
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()
```



In [15]:
```
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()
```



In [16]:
```
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: 161121 (629.38 KB)
Trainable params: 161121 (629.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [17]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, auc
#predictions
pred = model.predict(x_test)
classes_x=np.argmax(pred,axis=1)
```

```
782/782 [==============================] - 5s 5ms/step
```

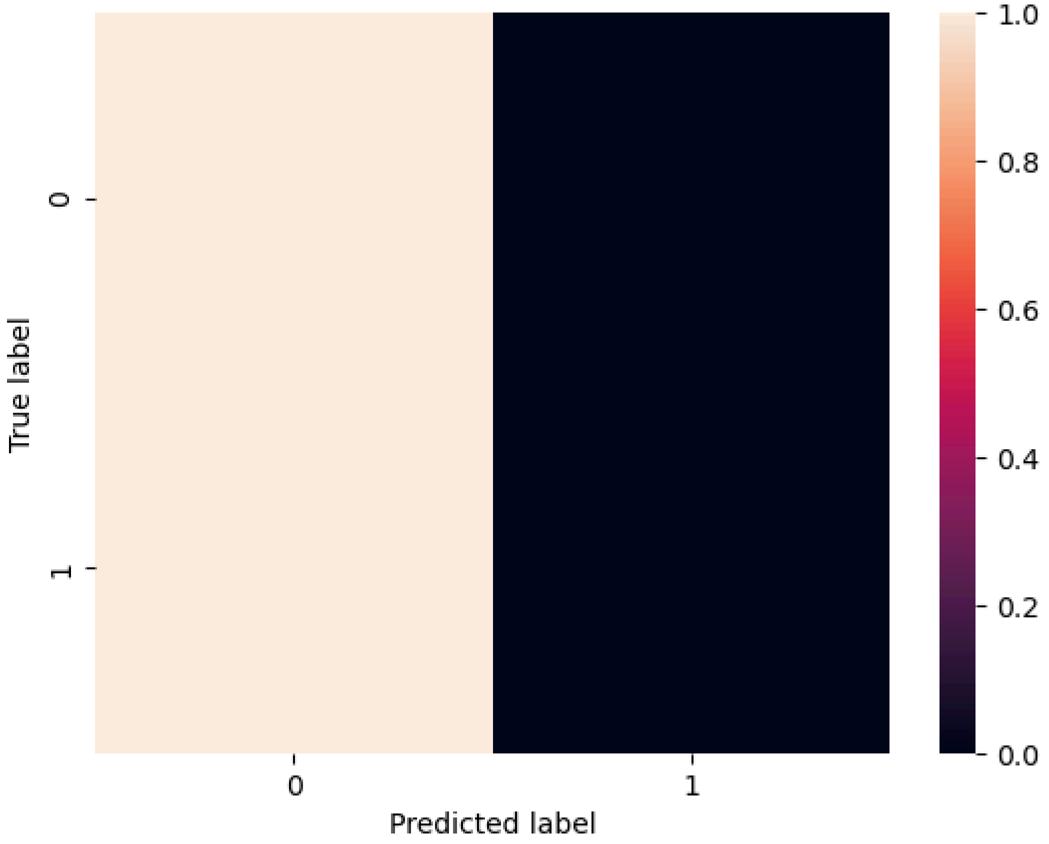
In [18]:
```
#accuracy
accuracy_score(y_test,classes_x)
```

Out[18]:
```
0.5
```

In [19]:
```
#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]]
```

Out[19]:
```
Text(0.5, 23.52222222222222, 'Predicted label')
```



In [26]:

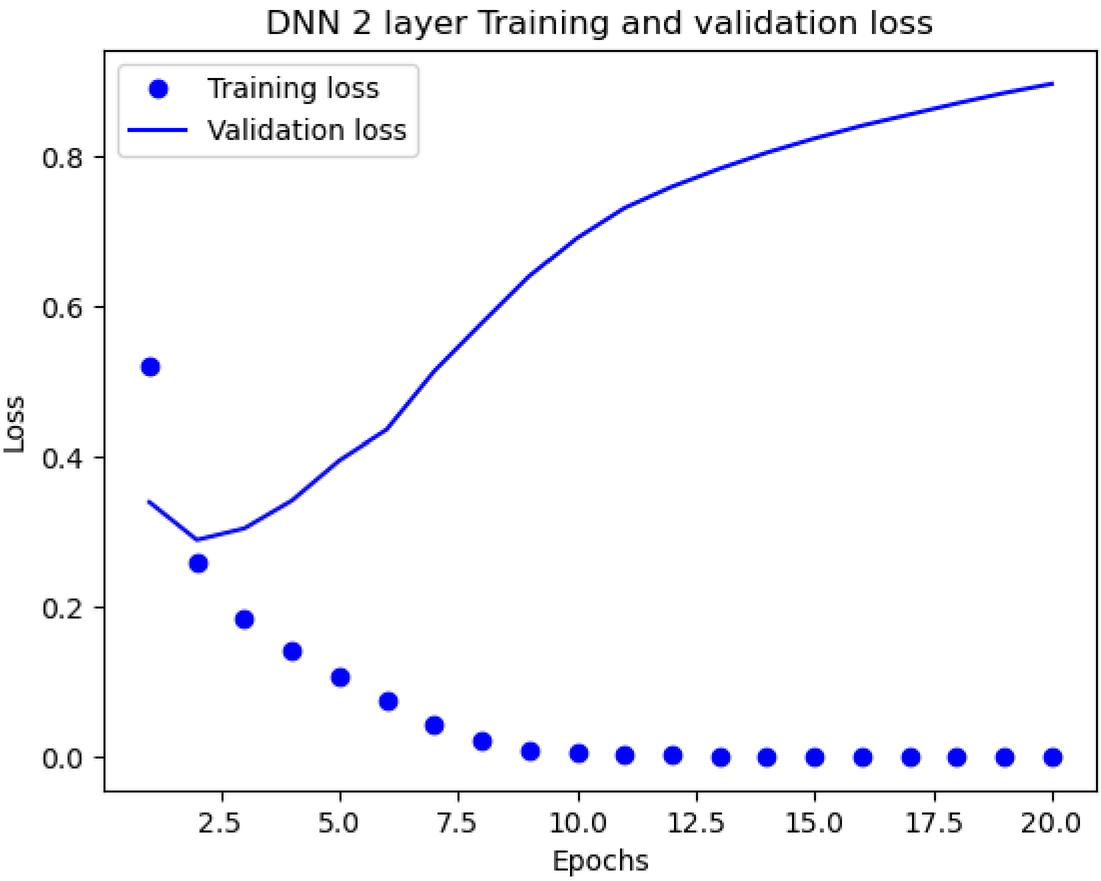
```
#Dense with Two Layer
model2 = models.Sequential()
model2.add(layers.Dense(32, activation='relu', input_shape=(5000,)))
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(1, activation='sigmoid'))

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
      % (total_time_m2))
```

```
Epoch 1/20
30/30 [==============================] - 10s 239ms/step - loss: 0.5568 - a
cc: 0.7617 - val_loss: 0.3759 - val_acc: 0.8593
Epoch 2/20
30/30 [==============================] - 1s 46ms/step - loss: 0.2871 - ac
c: 0.8920 - val_loss: 0.2885 - val_acc: 0.8843
Epoch 3/20
30/30 [==============================] - 2s 54ms/step - loss: 0.1991 - ac
c: 0.9257 - val_loss: 0.2981 - val_acc: 0.8817
Epoch 4/20
30/30 [==============================] - 1s 26ms/step - loss: 0.1556 - ac
c: 0.9433 - val_loss: 0.3302 - val_acc: 0.8730
Epoch 5/20
30/30 [==============================] - 1s 40ms/step - loss: 0.1262 - ac
c: 0.9559 - val_loss: 0.3686 - val_acc: 0.8716
Epoch 6/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0995 - ac
c: 0.9666 - val_loss: 0.4148 - val_acc: 0.8680
Epoch 7/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0751 - ac
c: 0.9771 - val_loss: 0.4631 - val_acc: 0.8643
Epoch 8/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0548 - ac
c: 0.9844 - val_loss: 0.5244 - val_acc: 0.8611
Epoch 9/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0390 - ac
c: 0.9906 - val_loss: 0.5956 - val_acc: 0.8575
Epoch 10/20
30/30 [==============================] - 2s 57ms/step - loss: 0.0244 - ac
c: 0.9957 - val_loss: 0.6459 - val_acc: 0.8581
Epoch 11/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0143 - ac
c: 0.9984 - val_loss: 0.7006 - val_acc: 0.8587
Epoch 12/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0076 - ac
c: 0.9994 - val_loss: 0.7545 - val_acc: 0.8545
Epoch 13/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0046 - ac
c: 0.9997 - val_loss: 0.7954 - val_acc: 0.8543
Epoch 14/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0027 - ac
c: 0.9999 - val_loss: 0.8320 - val_acc: 0.8563
Epoch 15/20
30/30 [==============================] - 1s 39ms/step - loss: 0.0019 - ac
c: 1.0000 - val_loss: 0.8628 - val_acc: 0.8555
Epoch 16/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0014 - ac
c: 1.0000 - val_loss: 0.8906 - val_acc: 0.8561
Epoch 17/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0011 - ac
c: 1.0000 - val_loss: 0.9142 - val_acc: 0.8549
Epoch 18/20
30/30 [==============================] - 1s 36ms/step - loss: 9.3472e-04 -
acc: 1.0000 - val_loss: 0.9359 - val_acc: 0.8550
Epoch 19/20
30/30 [==============================] - 1s 40ms/step - loss: 7.7895e-04 -
acc: 1.0000 - val_loss: 0.9581 - val_acc: 0.8554
Epoch 20/20
30/30 [==============================] - 1s 29ms/step - loss: 6.2868e-04 -
acc: 1.0000 - val_loss: 0.9914 - val_acc: 0.8555
```

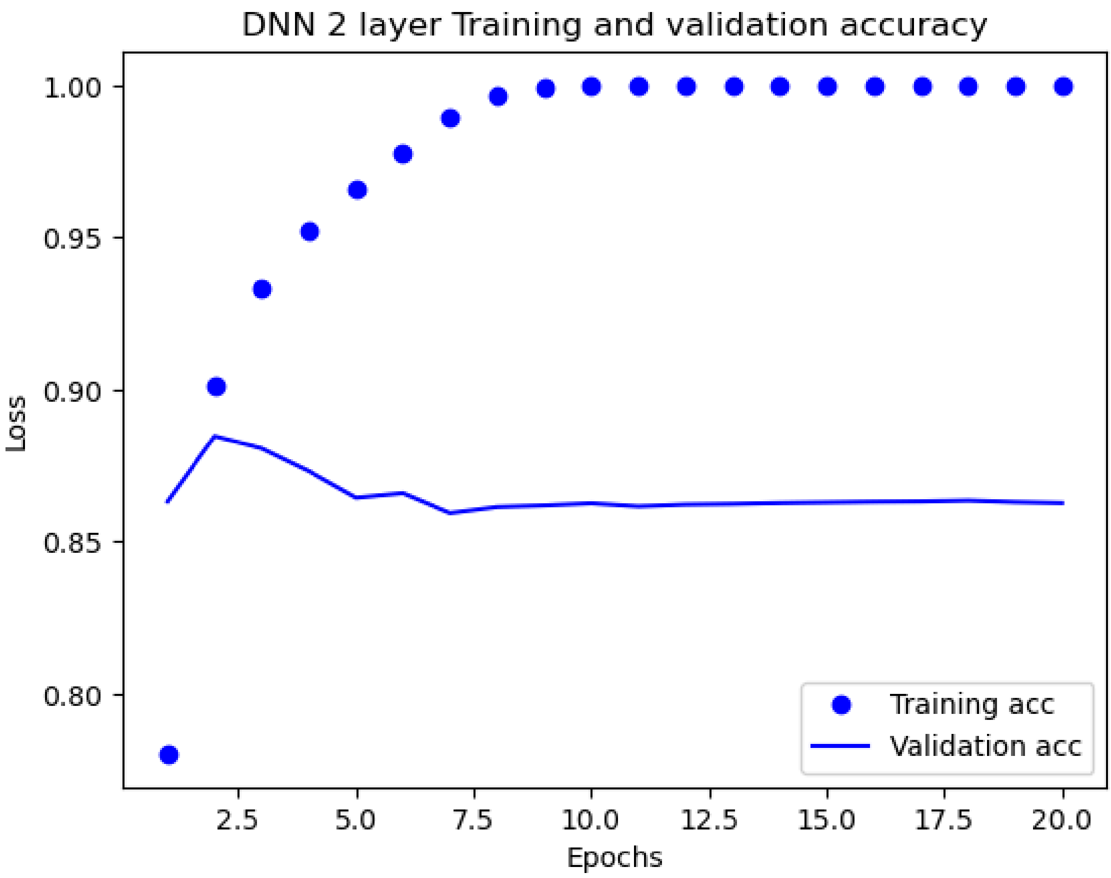
```
The Dense Convolutional Neural Network 2 layers took 35.3684 seconds to tr
ain.
```

In [21]:

```python
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()
```



In [22]:
```
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()
```



In [23]:
```
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: 162177 (633.50 KB)
Trainable params: 162177 (633.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [24]:

```
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 [==============================] - 6s 6ms/step
```

Out[24]:

```
0.5
```

In [ ]: