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
```

In [2]:
```
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 [3]:
```
(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)
```

In [4]:
```
print("Training data: ")
print(X.shape)
print(y.shape)
print("Classes: ")
print(np.unique(y))
```

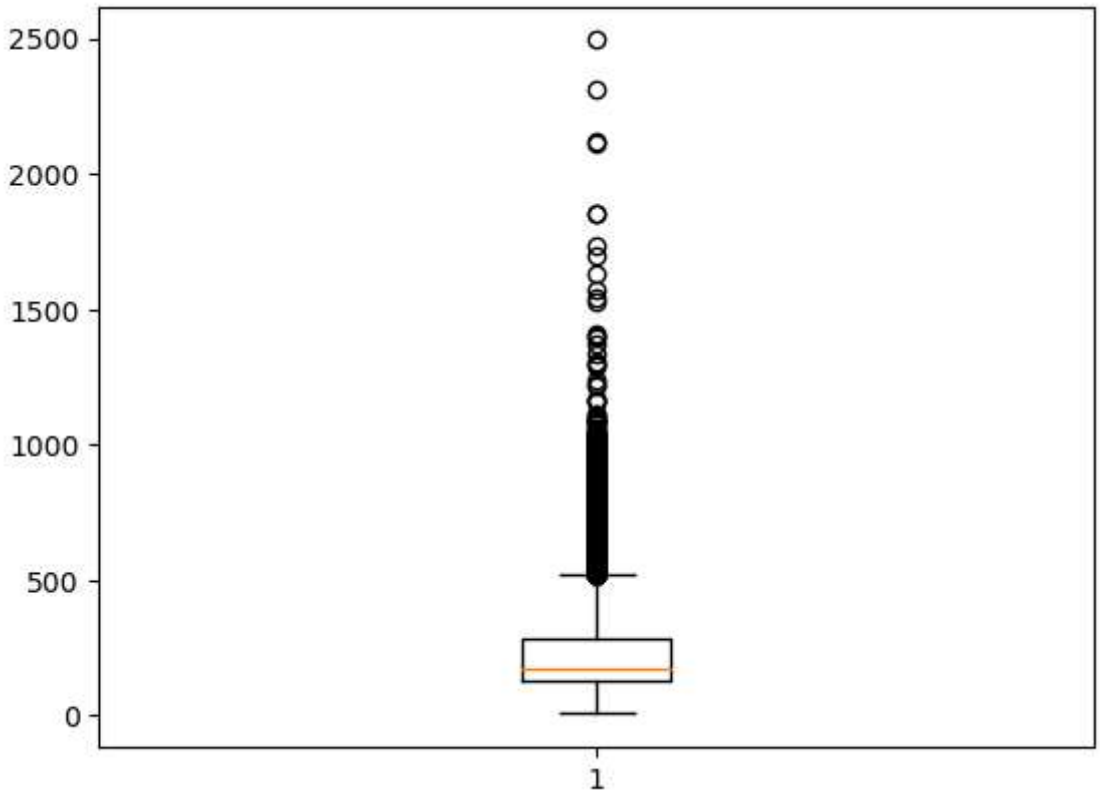
```
Training data:
(50000,)
(50000,)
Classes:
[0 1]
```

In [5]:
```
print("Number of words: ")
print(len(np.unique(np.hstack(X))))
```

```
Number of words:
88585
```

In [6]:
```
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_words=5000)
```

In [8]:
```
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
```

In [9]:
```
# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)
```

In [10]:
```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

In [11]:
```
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 [12]:
```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

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

In [13]:
```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
```

In [14]:
```
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 [==============================] - 1s 19ms/step - loss: 0.5421 - acc: 0.7588 -
val_loss: 0.3710 - val_acc: 0.8569
Epoch 2/20
30/30 [==============================] - 0s 11ms/step - loss: 0.2822 - acc: 0.8953 -
val_loss: 0.2883 - val_acc: 0.8863
Epoch 3/20
30/30 [==============================] - 0s 10ms/step - loss: 0.2035 - acc: 0.9247 -
val_loss: 0.2948 - val_acc: 0.8806
Epoch 4/20
30/30 [==============================] - 0s 10ms/step - loss: 0.1659 - acc: 0.9403 -
val_loss: 0.3073 - val_acc: 0.8815
Epoch 5/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1394 - acc: 0.9517 -
val_loss: 0.3317 - val_acc: 0.8724
Epoch 6/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1140 - acc: 0.9605 -
val_loss: 0.3637 - val_acc: 0.8658
Epoch 7/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0919 - acc: 0.9721 -
val_loss: 0.3975 - val_acc: 0.8632
Epoch 8/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0703 - acc: 0.9796 -
val_loss: 0.4382 - val_acc: 0.8663
Epoch 9/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0522 - acc: 0.9875 -
val_loss: 0.4729 - val_acc: 0.8632
Epoch 10/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0362 - acc: 0.9941 -
val_loss: 0.5111 - val_acc: 0.8631
Epoch 11/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0244 - acc: 0.9978 -
val_loss: 0.5571 - val_acc: 0.8624
Epoch 12/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0167 - acc: 0.9989 - v
al_loss: 0.5891 - val_acc: 0.8600
Epoch 13/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0115 - acc: 0.9995 -
val_loss: 0.6227 - val_acc: 0.8597
Epoch 14/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0085 - acc: 0.9999 -
val_loss: 0.6509 - val_acc: 0.8611
Epoch 15/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0064 - acc: 0.9999 -
val_loss: 0.6784 - val_acc: 0.8593
Epoch 16/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0050 - acc: 0.9999 -
val_loss: 0.6992 - val_acc: 0.8610
Epoch 17/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0041 - acc: 0.9999 -
val_loss: 0.7204 - val_acc: 0.8600
Epoch 18/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0034 - acc: 1.0000 -
val_loss: 0.7414 - val_acc: 0.8591
Epoch 19/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0028 - acc: 1.0000 -
val_loss: 0.7580 - val_acc: 0.8604
Epoch 20/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0024 - acc: 1.0000 -
val_loss: 0.7742 - val_acc: 0.8602
```

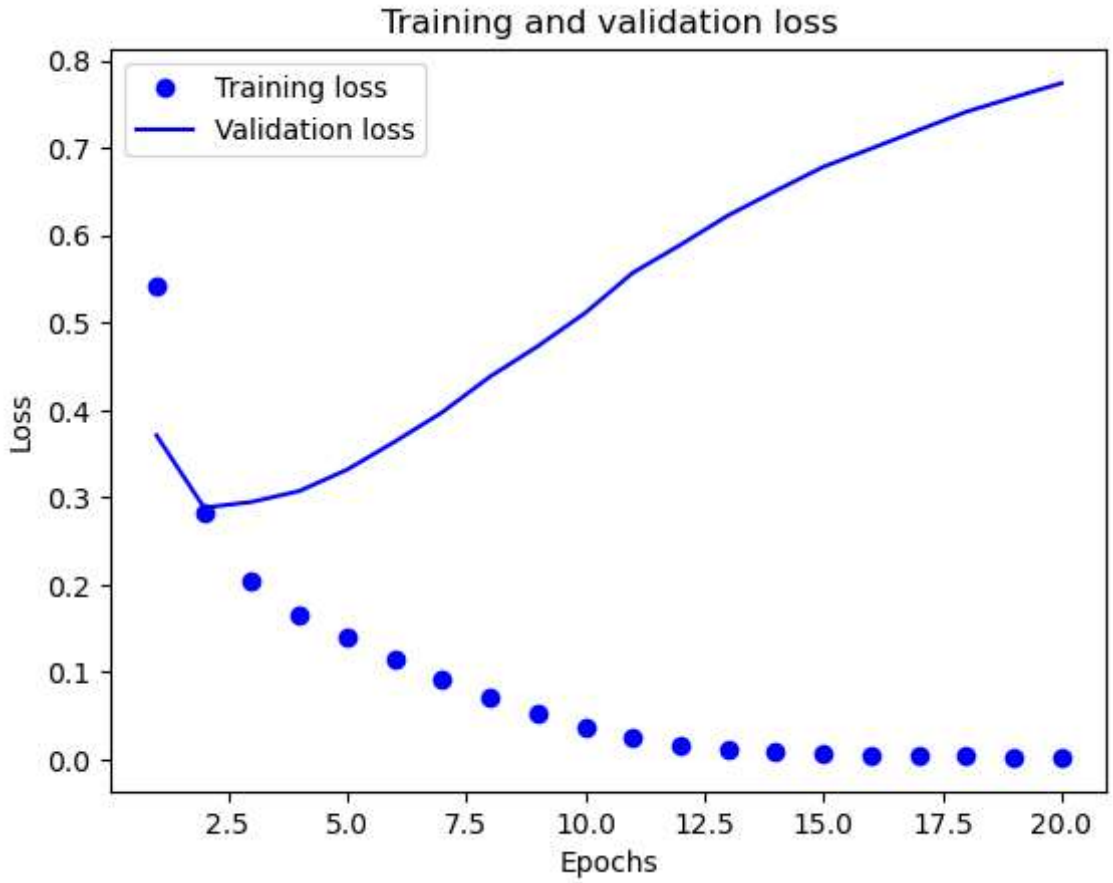
In [15]:
```
print("The Dense Convolutional Neural Network 1 layer took %.4f seconds to train." % (
```

```
The Dense Convolutional Neural Network 1 layer took 7.5016 seconds to train.
```

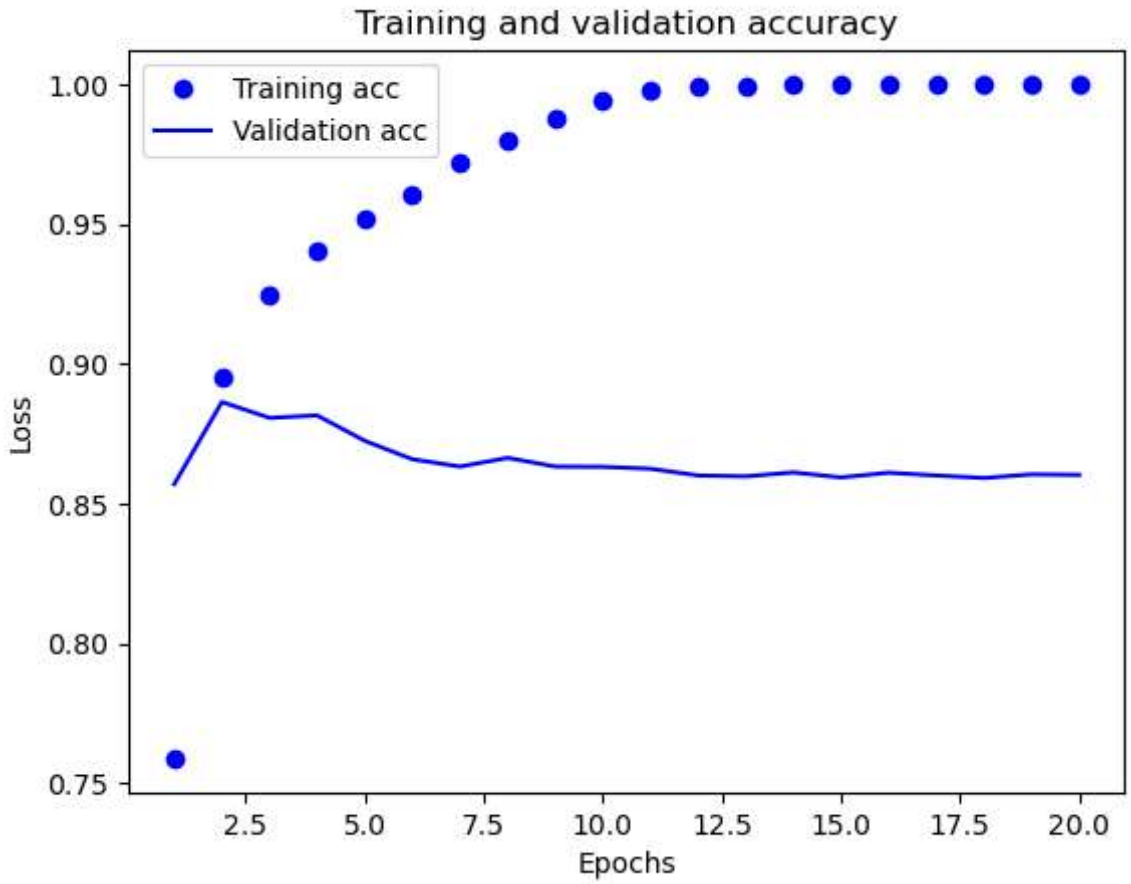
In [16]:
```
history_dict = history.history
history_dict.keys()
```

Out[16]:
```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

In [17]:
```
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 [18]:
```
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 [19]:
```
model.summary()
```

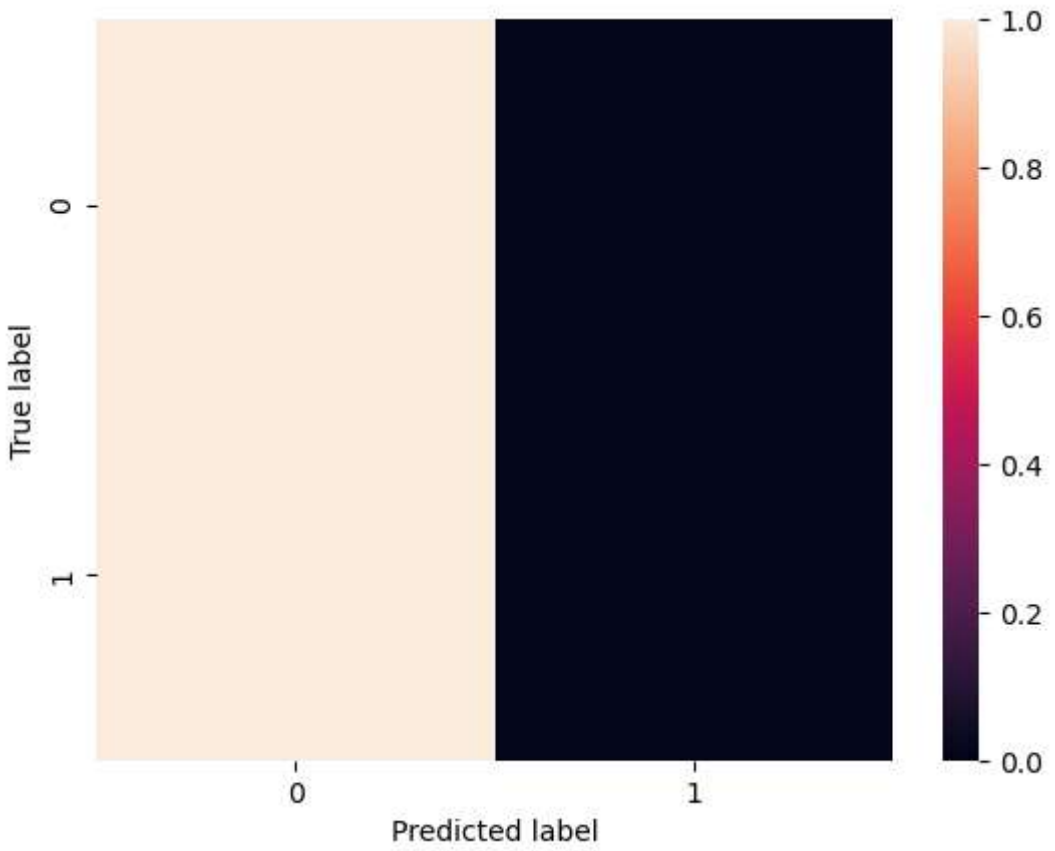
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                160032

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 1)                 33

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

In [20]:
```
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)
```

Out[20]:
```
782/782 [==============================] - 1s 714us/step
0.5
```

In [21]:
```
#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')
```

Out[21]:
```
[[12500     0]
 [12500     0]]
Text(0.5, 23.52222222222222, 'Predicted label')
```



In [22]:
```
#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'))
```

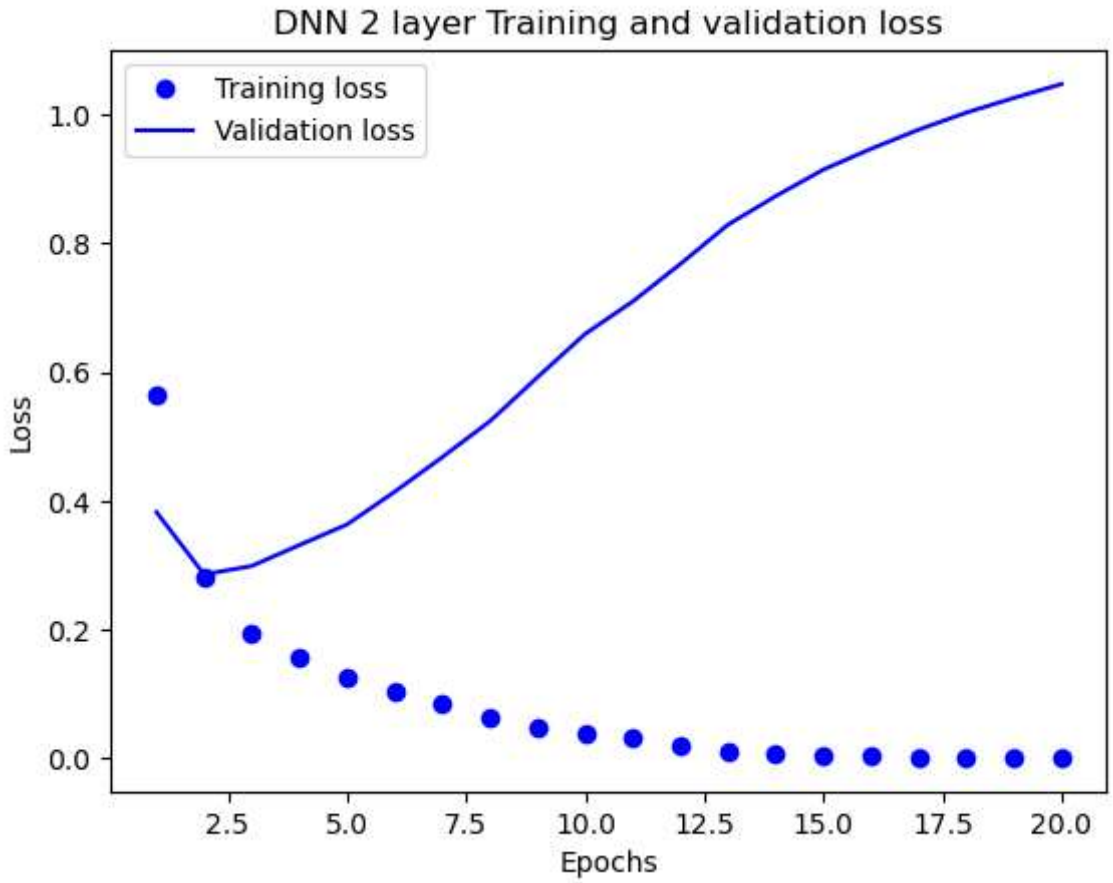
In [23]:
```
model2.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['acc'])
```

In [24]:
```
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." %
```

```
Epoch 1/20
30/30 [==============================] - 1s 23ms/step - loss: 0.5654 - acc: 0.7490 -
val_loss: 0.3823 - val_acc: 0.8627
Epoch 2/20
30/30 [==============================] - 0s 14ms/step - loss: 0.2808 - acc: 0.8957 -
val_loss: 0.2856 - val_acc: 0.8829
Epoch 3/20
30/30 [==============================] - 0s 12ms/step - loss: 0.1939 - acc: 0.9282 -
val_loss: 0.2987 - val_acc: 0.8816
Epoch 4/20
30/30 [==============================] - 0s 10ms/step - loss: 0.1553 - acc: 0.9453 -
val_loss: 0.3312 - val_acc: 0.8768
Epoch 5/20
30/30 [==============================] - 0s 10ms/step - loss: 0.1268 - acc: 0.9581 -
val_loss: 0.3631 - val_acc: 0.8712
Epoch 6/20
30/30 [==============================] - 0s 10ms/step - loss: 0.1029 - acc: 0.9661 -
val_loss: 0.4139 - val_acc: 0.8645
Epoch 7/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0843 - acc: 0.9723 - v
al_loss: 0.4677 - val_acc: 0.8599
Epoch 8/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0642 - acc: 0.9807 - v
al_loss: 0.5235 - val_acc: 0.8591
Epoch 9/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0488 - acc: 0.9864 - v
al_loss: 0.5919 - val_acc: 0.8564
Epoch 10/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0384 - acc: 0.9897 - v
al_loss: 0.6592 - val_acc: 0.8559
Epoch 11/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0327 - acc: 0.9918 -
val_loss: 0.7096 - val_acc: 0.8562
Epoch 12/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0208 - acc: 0.9961 -
val_loss: 0.7678 - val_acc: 0.8536
Epoch 13/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0115 - acc: 0.9986 - v
al_loss: 0.8289 - val_acc: 0.8529
Epoch 14/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0061 - acc: 0.9998 - v
al_loss: 0.8731 - val_acc: 0.8530
Epoch 15/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0038 - acc: 0.9999 - v
al_loss: 0.9144 - val_acc: 0.8525
Epoch 16/20
30/30 [==============================] - 0s 9ms/step - loss: 0.0027 - acc: 0.9999 - v
al_loss: 0.9466 - val_acc: 0.8522
Epoch 17/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0021 - acc: 1.0000 -
val_loss: 0.9764 - val_acc: 0.8522
Epoch 18/20
30/30 [==============================] - 2s 60ms/step - loss: 0.0017 - acc: 1.0000 -
val_loss: 1.0028 - val_acc: 0.8522
Epoch 19/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0014 - acc: 1.0000 -
val_loss: 1.0257 - val_acc: 0.8525
Epoch 20/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0012 - acc: 1.0000 -
```

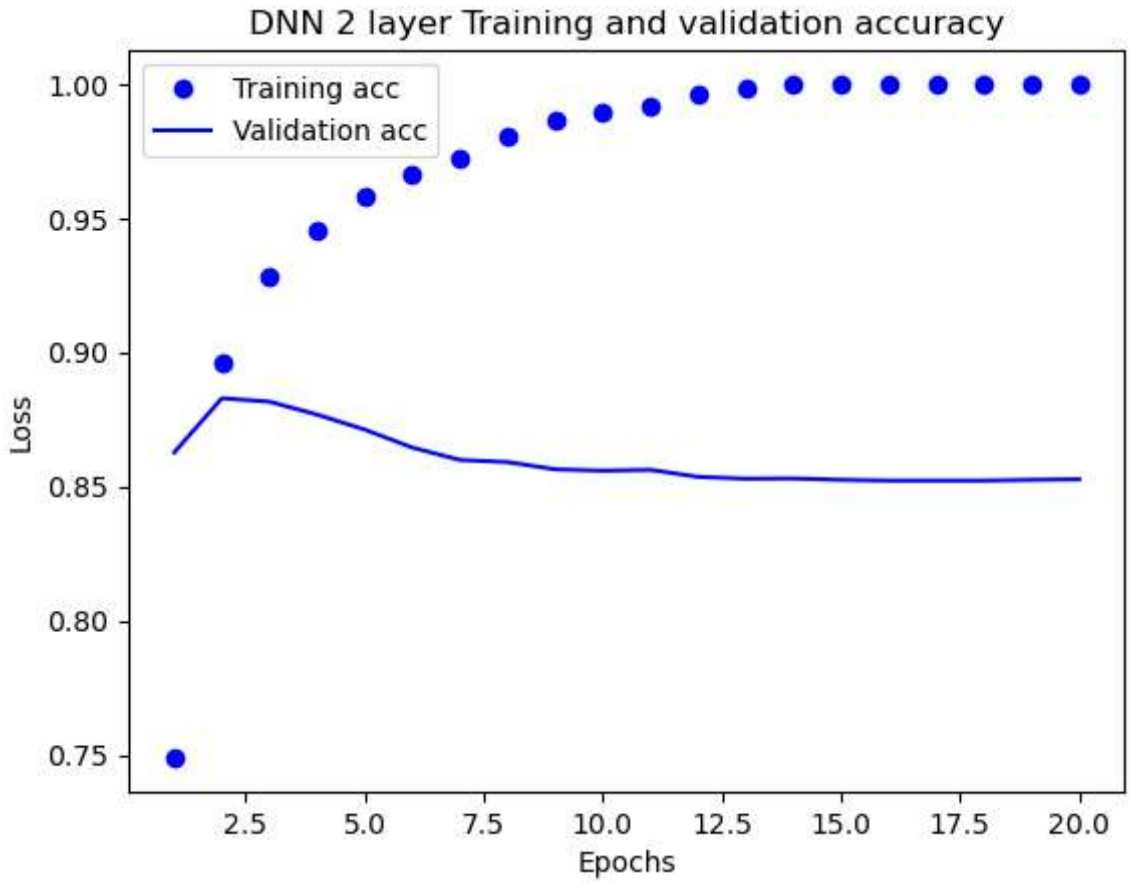
```
val_loss: 1.0473 - val_acc: 0.8527
The Dense Convolutional Neural Network 2 layers took 8.7560 seconds to train.
```

In [25]:
```
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 [26]:
```
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 [27]:
```
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
_________________________________________________________________
```

In [28]:
```
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 [==============================] - 1s 704us/step
```

Out[28]:
```
0.5
```

In [ ]: