In [1]:

```
import os

imdb_dir = './aclImdb'
train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

for label_type in ['neg','pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding='utf8')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

In [2]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

maxlen = 100
training_samples = 200
validation_samples = 10000
max_words = 10000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('%s개의 고유한 토큰을 찾았습니다.' % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('데이터 텐서의 크기:', data.shape)
print('레이블 텐서의 크기:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Using TensorFlow backend.
```

```
88582개의 고유한 토큰을 찾았습니다.
데이터 텐서의 크기: (25000, 100)
레이블 텐서의 크기: (25000,)
```

In [3]:

```
glove_dir = './datasets/glove.6B/'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'), encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('%s개의 단어 벡터를 찾았습니다.' % len(embeddings_index))
```

```
400000개의 단어 벡터를 찾았습니다.
```

In [4]:

```
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

In [5]:

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:From C:WProgramDataWAnaconda3WlibWsite-packagesWtensorflowWpyth
onWframeworkWop_def_library.py:263: colocate_with (from tensorflow.python.framewor
k.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_1 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

사전 훈련된 단어 임베딩을 Embedding 층에 로드하기

In [6]:

```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

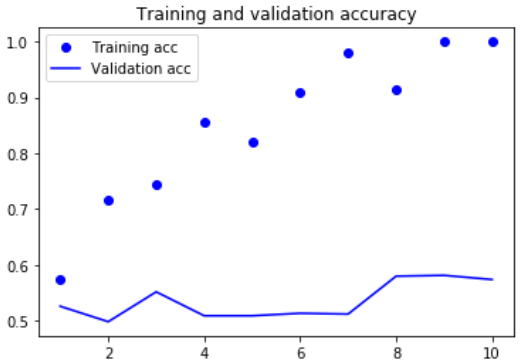
In [7]:

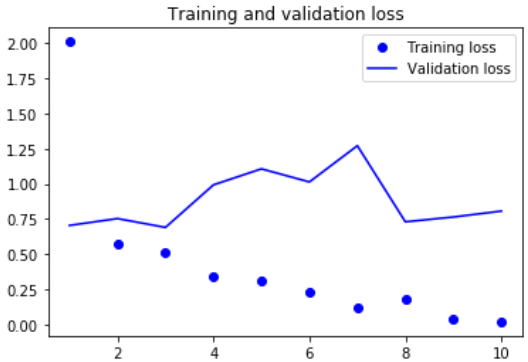
```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
WARNING:tensorflow:From C:WProgramDataWAnaconda3WlibWsite-packagesWtensorflowWpyth
onWopsWmath_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is depreca
ted and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 1s 6ms/step - loss: 2.0096 - acc: 0.575
0 - val_loss: 0.7049 - val_acc: 0.5258
Epoch 2/10
200/200 [==============================] - 0s 2ms/step - loss: 0.5698 - acc: 0.715
0 - val_loss: 0.7536 - val_acc: 0.4981
Epoch 3/10
200/200 [==============================] - 0s 2ms/step - loss: 0.5111 - acc: 0.745
0 - val_loss: 0.6908 - val_acc: 0.5516
Epoch 4/10
200/200 [==============================] - 0s 2ms/step - loss: 0.3424 - acc: 0.855
0 - val_loss: 0.9927 - val_acc: 0.5088
Epoch 5/10
200/200 [==============================] - 0s 2ms/step - loss: 0.3168 - acc: 0.820
0 - val_loss: 1.1066 - val_acc: 0.5088
Epoch 6/10
200/200 [==============================] - 0s 2ms/step - loss: 0.2275 - acc: 0.910
0 - val_loss: 1.0135 - val_acc: 0.5132
Epoch 7/10
200/200 [==============================] - 0s 2ms/step - loss: 0.1174 - acc: 0.980
0 - val_loss: 1.2708 - val_acc: 0.5118
Epoch 8/10
200/200 [==============================] - 0s 2ms/step - loss: 0.1833 - acc: 0.915
0 - val_loss: 0.7305 - val_acc: 0.5797
Epoch 9/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0374 - acc: 1.000
0 - val_loss: 0.7646 - val_acc: 0.5812
Epoch 10/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0212 - acc: 1.000
0 - val_loss: 0.8066 - val_acc: 0.5737
```

In [9]:

```
import matplotlib.pyplot as plt

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)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





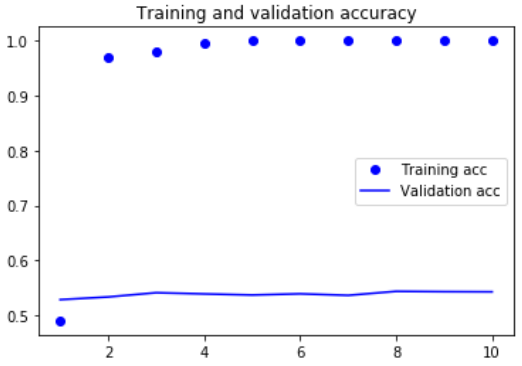
In [10]:

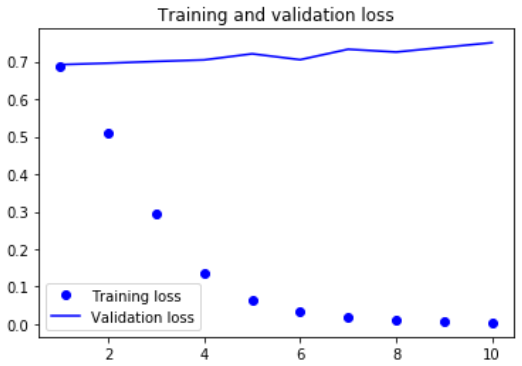
```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_2 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 1s 4ms/step - loss: 0.6887 - acc: 0.490
0 - val_loss: 0.6921 - val_acc: 0.5284
Epoch 2/10
200/200 [==============================] - 0s 2ms/step - loss: 0.5087 - acc: 0.970
0 - val_loss: 0.6960 - val_acc: 0.5333
Epoch 3/10
200/200 [==============================] - 0s 2ms/step - loss: 0.2935 - acc: 0.980
0 - val_loss: 0.7006 - val_acc: 0.5410
Epoch 4/10
200/200 [==============================] - 0s 2ms/step - loss: 0.1374 - acc: 0.995
0 - val_loss: 0.7047 - val_acc: 0.5388
Epoch 5/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0653 - acc: 1.000
0 - val_loss: 0.7209 - val_acc: 0.5368
Epoch 6/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0343 - acc: 1.000
0 - val_loss: 0.7052 - val_acc: 0.5390
Epoch 7/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0186 - acc: 1.000
0 - val_loss: 0.7330 - val_acc: 0.5362
Epoch 8/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0107 - acc: 1.000
0 - val_loss: 0.7256 - val_acc: 0.5435
Epoch 9/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0063 - acc: 1.000
0 - val_loss: 0.7382 - val_acc: 0.5429
Epoch 10/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0038 - acc: 1.000
0 - val_loss: 0.7506 - val_acc: 0.5426
```

In [11]:

```
import matplotlib.pyplot as plt

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)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





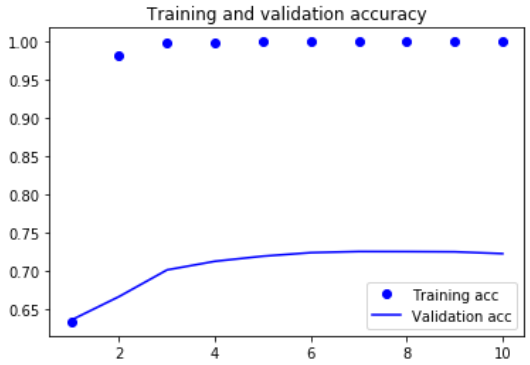
In [12]:

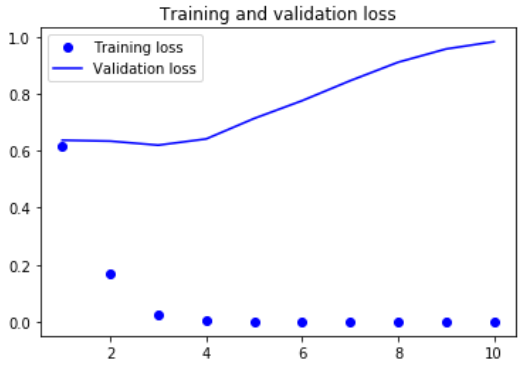
```
training_samples = 2000
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]

history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_val, y_val))
```

```
Train on 2000 samples, validate on 10000 samples
Epoch 1/10
2000/2000 [==============================] - 1s 329us/step - loss: 0.6160 - acc:
0.6330 - val_loss: 0.6369 - val_acc: 0.6351
Epoch 2/10
2000/2000 [==============================] - 1s 329us/step - loss: 0.1683 - acc:
0.9820 - val_loss: 0.6336 - val_acc: 0.6660
Epoch 3/10
2000/2000 [==============================] - 1s 323us/step - loss: 0.0243 - acc:
0.9985 - val_loss: 0.6195 - val_acc: 0.7009
Epoch 4/10
2000/2000 [==============================] - 1s 322us/step - loss: 0.0031 - acc:
0.9995 - val_loss: 0.6413 - val_acc: 0.7122
Epoch 5/10
2000/2000 [==============================] - 1s 319us/step - loss: 2.6340e-04 - ac
c: 1.0000 - val_loss: 0.7132 - val_acc: 0.7189
Epoch 6/10
2000/2000 [==============================] - 1s 328us/step - loss: 2.0136e-05 - ac
c: 1.0000 - val_loss: 0.7755 - val_acc: 0.7235
Epoch 7/10
2000/2000 [==============================] - 1s 320us/step - loss: 1.9653e-06 - ac
c: 1.0000 - val_loss: 0.8455 - val_acc: 0.7250
Epoch 8/10
2000/2000 [==============================] - 1s 324us/step - loss: 3.4668e-07 - ac
c: 1.0000 - val_loss: 0.9104 - val_acc: 0.7249
Epoch 9/10
2000/2000 [==============================] - 1s 322us/step - loss: 1.3345e-07 - ac
c: 1.0000 - val_loss: 0.9570 - val_acc: 0.7246
Epoch 10/10
2000/2000 [==============================] - 1s 318us/step - loss: 1.1359e-07 - ac
c: 1.0000 - val_loss: 0.9826 - val_acc: 0.7222
```

In [13]:

```
import matplotlib.pyplot as plt

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)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





In [17]:

```
test_dir = os.path.join(imdb_dir, 'test')

labels = []
texts = []

for label_type in ['neg','pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding='utf8')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)

sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

In [18]:

```
model.load_weights('pre_trained_glove_model.h5')
model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 1s 43us/step
```

Out[18]:

```
[0.7510663447570801, 0.53856]
```