ADVANCE MACHINE LEARNING, ASSIGNMENT 4- TEXT AND SEQUENCE

In [1]:
```
import os
from operator import itemgetter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
get_ipython().magic(u'matplotlib inline')
plt.style.use('ggplot')

import tensorflow as tf

from keras import models, regularizers, layers, optimizers, losses, metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

In [2]:
```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# The number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing
from keras.utils import pad_sequences

# Number of words to consider as features
max_features = 10000
# After this amount of words, cut the texts
#(among top max_features most common words)
maxlen = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:100]
y_train = y_train[:100]

# This turns our lists of integers into a 2D integer tensor of shape
#`(samples, maxlen)`
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We provide our Embedding layer a maximum input length specification
# in order to flatten the embedded inputs later
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer, our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings into a 2D tensor of shape
#`(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

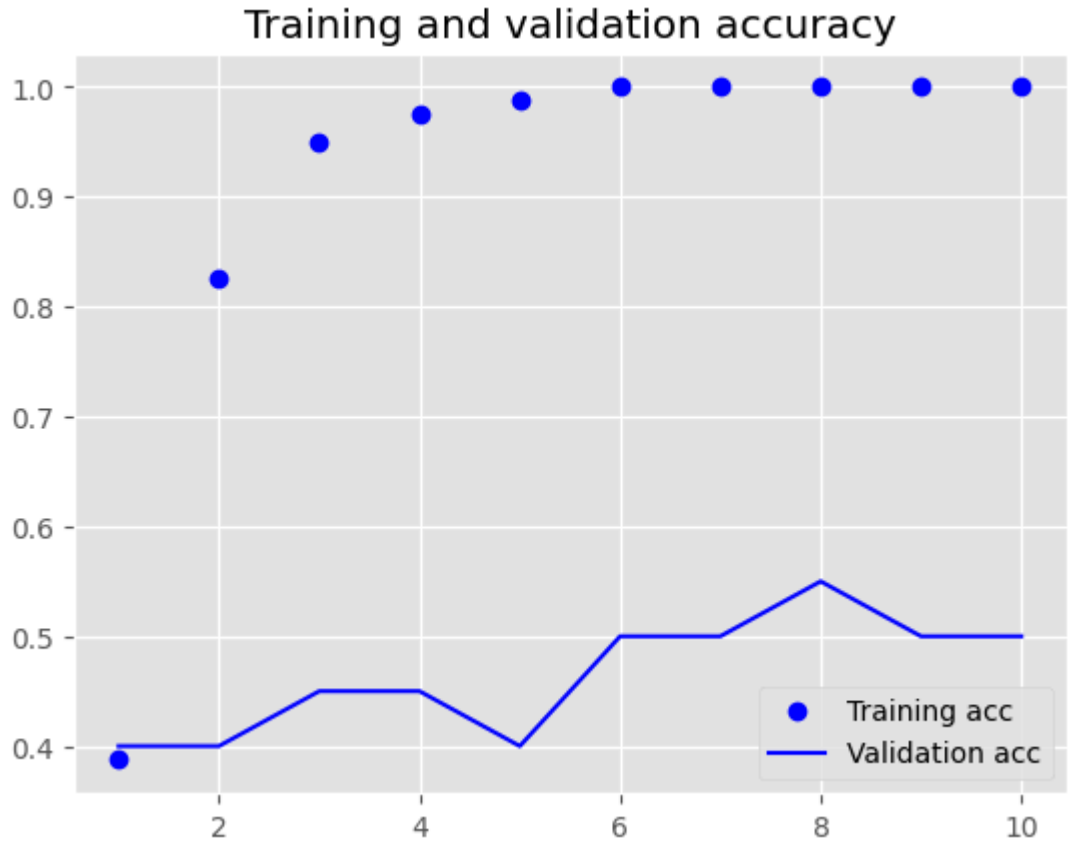
```
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

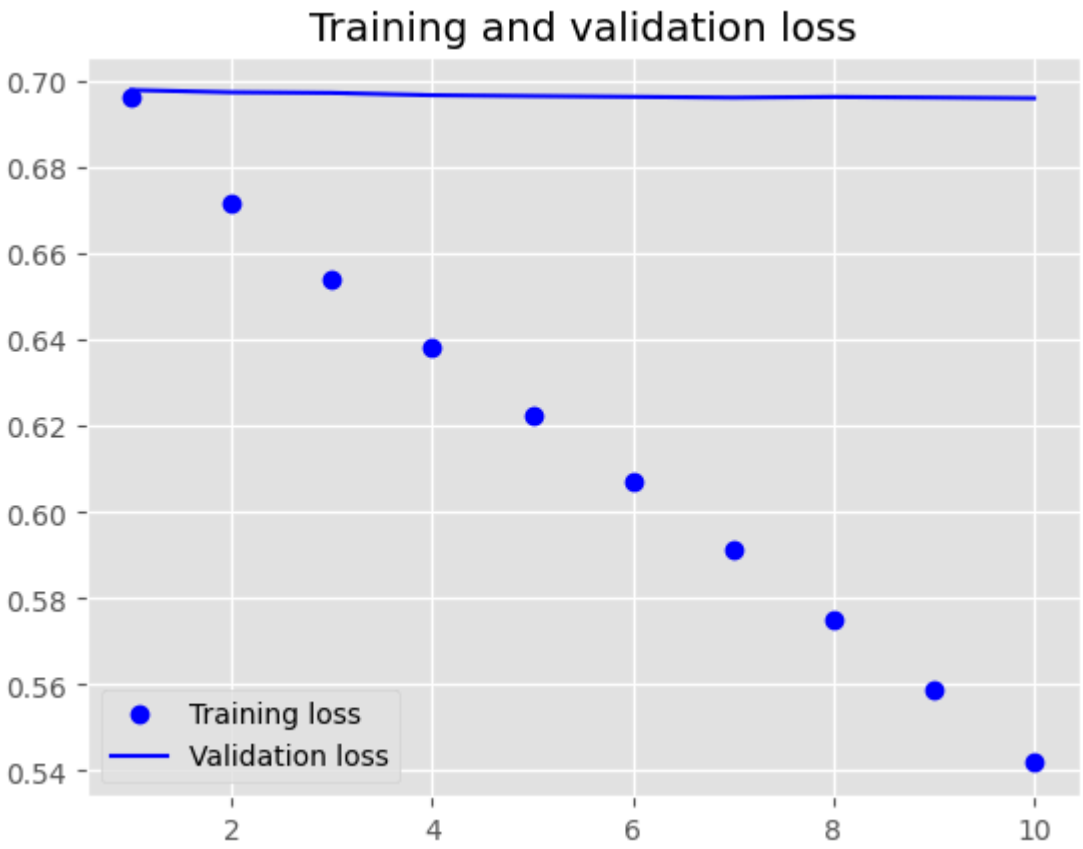
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 150, 8)            80000

 flatten (Flatten)           (None, 1200)              0

 dense (Dense)               (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
3/3 [==============================] - 3s 221ms/step - loss: 0.6965 - acc: 0.3875
- val_loss: 0.6980 - val_acc: 0.4000
Epoch 2/10
3/3 [==============================] - 0s 123ms/step - loss: 0.6719 - acc: 0.8250
- val_loss: 0.6975 - val_acc: 0.4000
Epoch 3/10
3/3 [==============================] - 0s 125ms/step - loss: 0.6541 - acc: 0.9500
- val_loss: 0.6974 - val_acc: 0.4500
Epoch 4/10
3/3 [==============================] - 0s 122ms/step - loss: 0.6381 - acc: 0.9750
- val_loss: 0.6968 - val_acc: 0.4500
Epoch 5/10
3/3 [==============================] - 0s 76ms/step - loss: 0.6225 - acc: 0.9875 -
val_loss: 0.6966 - val_acc: 0.4000
Epoch 6/10
3/3 [==============================] - 0s 199ms/step - loss: 0.6072 - acc: 1.0000
- val_loss: 0.6965 - val_acc: 0.5000
Epoch 7/10
3/3 [==============================] - 1s 191ms/step - loss: 0.5914 - acc: 1.0000
- val_loss: 0.6963 - val_acc: 0.5000
Epoch 8/10
3/3 [==============================] - 1s 196ms/step - loss: 0.5752 - acc: 1.0000
- val_loss: 0.6964 - val_acc: 0.5500
Epoch 9/10
3/3 [==============================] - 0s 118ms/step - loss: 0.5587 - acc: 1.0000
- val_loss: 0.6963 - val_acc: 0.5000
Epoch 10/10
3/3 [==============================] - 0s 160ms/step - loss: 0.5418 - acc: 1.0000
- val_loss: 0.6961 - val_acc: 0.5000
```

In [3]:

```
import matplotlib.pyplot as plt

# Training accuracy
acc = history.history["acc"]
# Validation accuracy
val_acc = history.history["val_acc"]
# Training loss
loss = history.history["loss"]
```

```
# Validation loss
val_loss = history.history["val_loss"]

#plots every epoch, here 10
epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, "bo", label = "Training acc") # "bo" gives dot plot
plt.plot(epochs, val_acc, "b", label = "Validation acc") # "b" gives line plot
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()
```

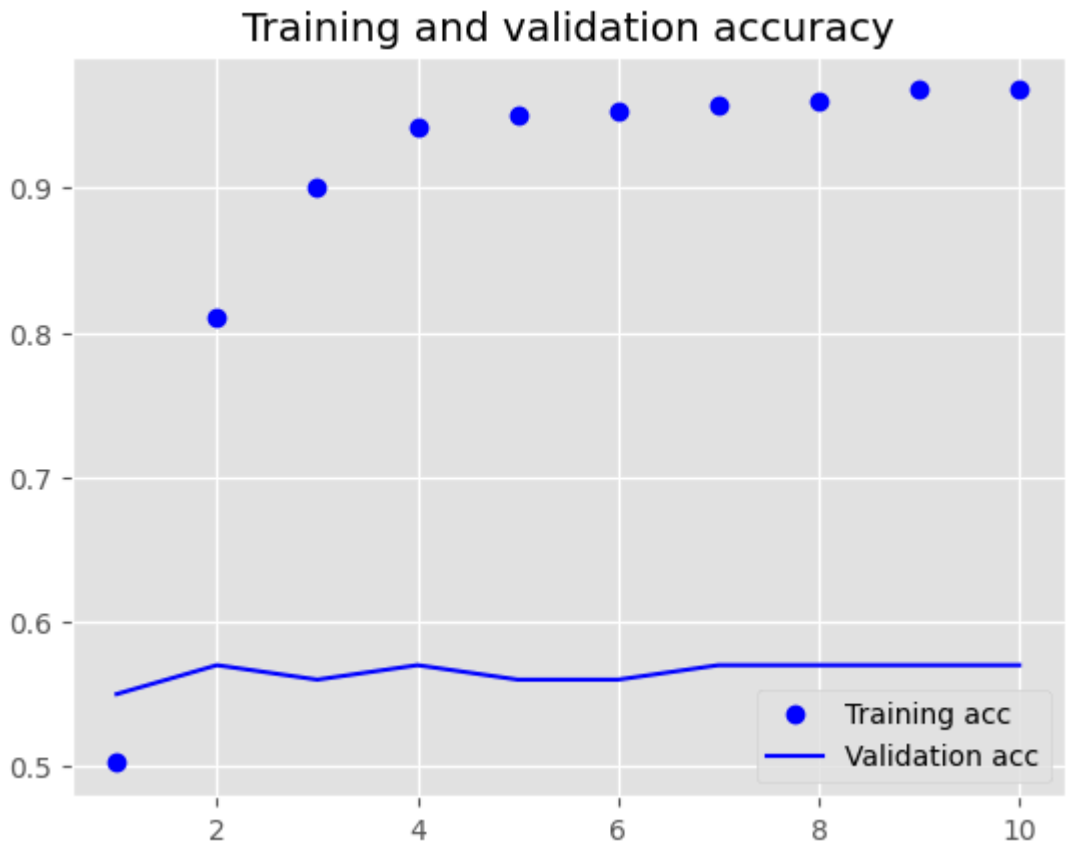


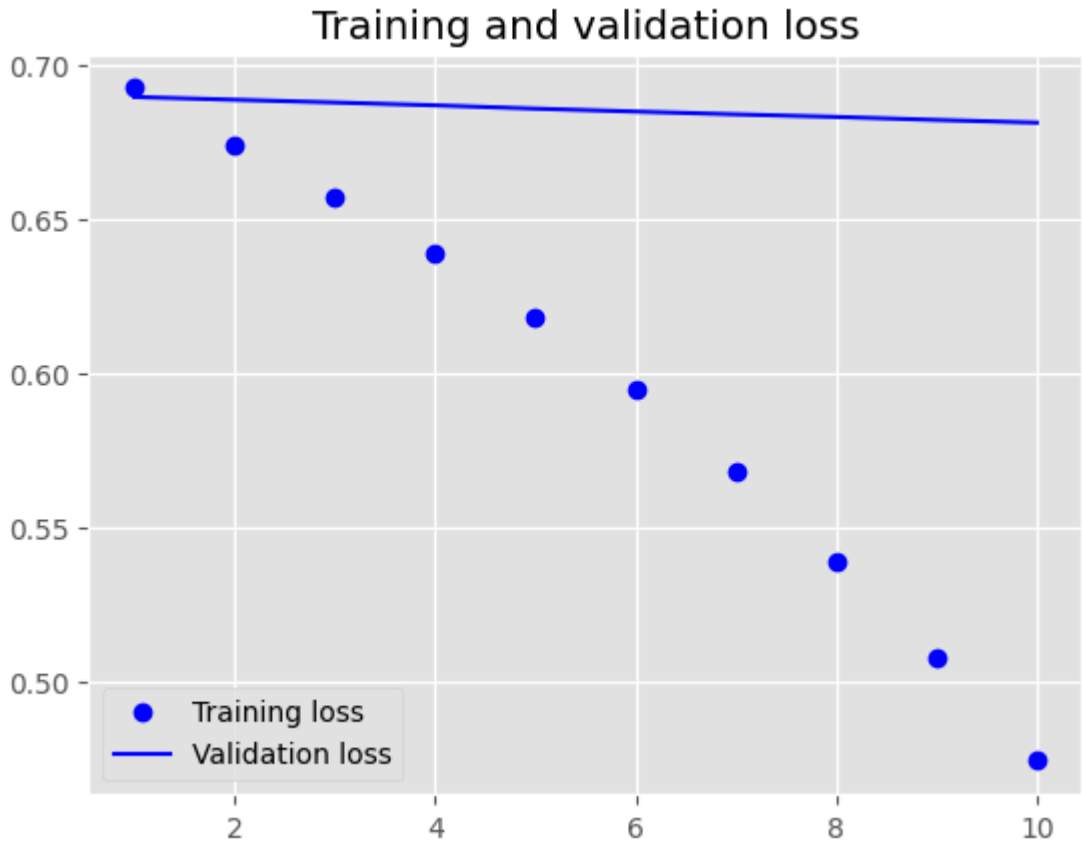


In [4]:
```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing

# Number of words to consider as features
max_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
maxlen = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:500]
y_train = y_train[:500]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We provide our Embedding layer a maximum input length specification
# in order to flatten the embedded inputs later
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
```

```
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
acc = history.history["acc"] # Training accuracy
val_acc = history.history["val_acc"] # Validation accuracy
loss = history.history["loss"] # Training loss
val_loss = history.history["val_loss"] # Validation loss

epochs = range(1, len(acc) + 1) #plots every epoch, here 10

plt.plot(epochs, acc, "bo", label = "Training acc") # "bo" gives dot plot
plt.plot(epochs, val_acc, "b", label = "Validation acc") # "b" gives line plot
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()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 150, 8)            80000

 flatten_1 (Flatten)         (None, 1200)              0

 dense_1 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
13/13 [==============================] - 2s 119ms/step - loss: 0.6922 - acc: 0.502
5 - val_loss: 0.6893 - val_acc: 0.5500
Epoch 2/10
13/13 [==============================] - 1s 110ms/step - loss: 0.6734 - acc: 0.810
0 - val_loss: 0.6884 - val_acc: 0.5700
Epoch 3/10
13/13 [==============================] - 1s 106ms/step - loss: 0.6570 - acc: 0.900
0 - val_loss: 0.6876 - val_acc: 0.5600
Epoch 4/10
13/13 [==============================] - 1s 92ms/step - loss: 0.6389 - acc: 0.9425
- val_loss: 0.6866 - val_acc: 0.5700
Epoch 5/10
13/13 [==============================] - 1s 82ms/step - loss: 0.6182 - acc: 0.9500
- val_loss: 0.6855 - val_acc: 0.5600
Epoch 6/10
13/13 [==============================] - 1s 81ms/step - loss: 0.5945 - acc: 0.9525
- val_loss: 0.6846 - val_acc: 0.5600
Epoch 7/10
13/13 [==============================] - 1s 54ms/step - loss: 0.5680 - acc: 0.9575
- val_loss: 0.6837 - val_acc: 0.5700
Epoch 8/10
13/13 [==============================] - 1s 83ms/step - loss: 0.5389 - acc: 0.9600
- val_loss: 0.6829 - val_acc: 0.5700
Epoch 9/10
13/13 [==============================] - 2s 118ms/step - loss: 0.5074 - acc: 0.967
5 - val_loss: 0.6819 - val_acc: 0.5700
Epoch 10/10
13/13 [==============================] - 1s 43ms/step - loss: 0.4741 - acc: 0.9675
- val_loss: 0.6810 - val_acc: 0.5700
```

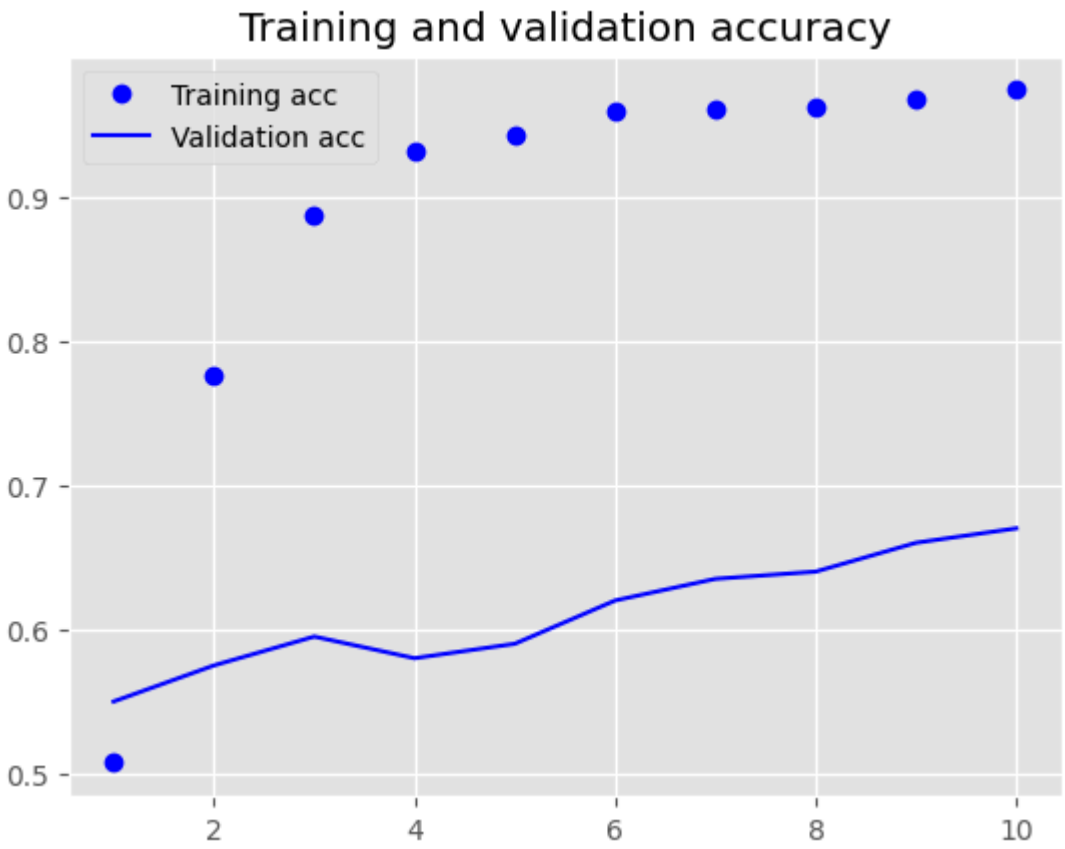


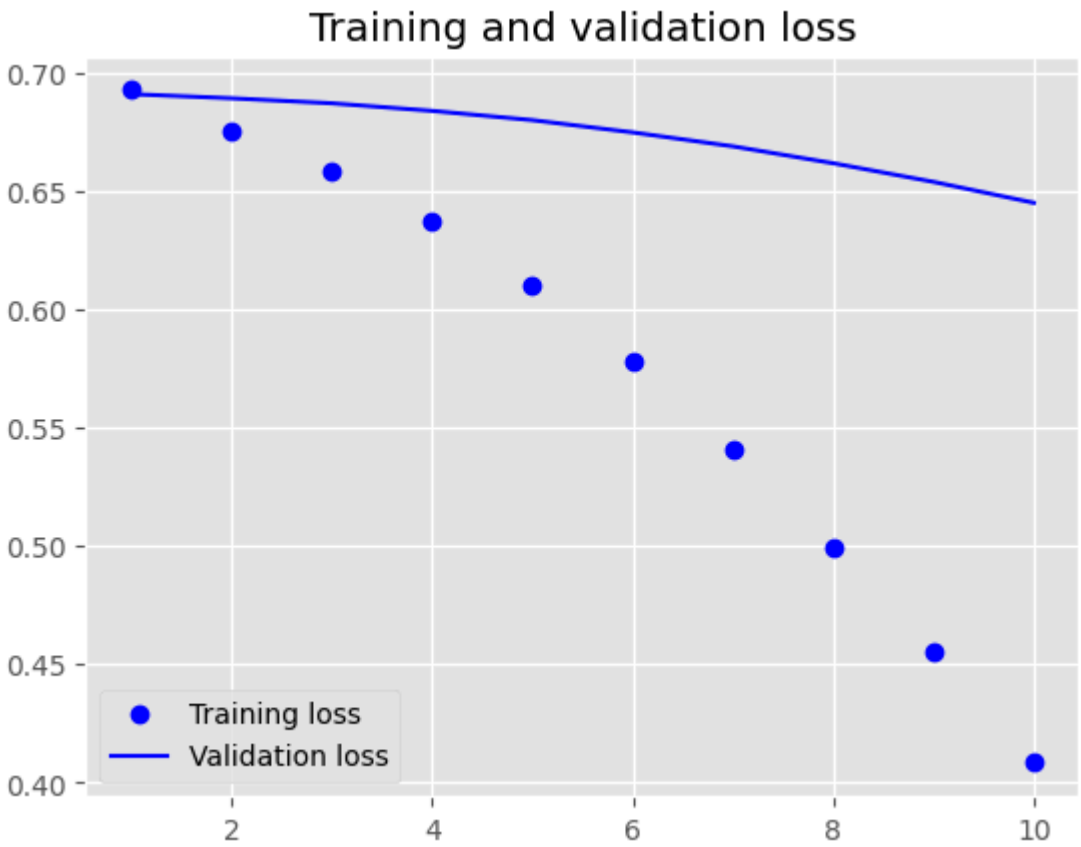


In [5]:
```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing
```

```
# Number of words to consider as features
max_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
maxlen = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:1000]
y_train = y_train[:1000]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We provide our Embedding layer a maximum input length specification
# in order to flatten the embedded inputs later
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
acc = history.history["acc"] # Training accuracy
val_acc = history.history["val_acc"] # Validation accuracy
loss = history.history["loss"] # Training loss
val_loss = history.history["val_loss"] # Validation loss

epochs = range(1, len(acc) + 1) #plots every epoch, here 10

plt.plot(epochs, acc, "bo", label = "Training acc") # "bo" gives dot plot
plt.plot(epochs, val_acc, "b", label = "Validation acc") # "b" gives line plot
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()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 150, 8)            80000

 flatten_2 (Flatten)         (None, 1200)              0

 dense_2 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
25/25 [==============================] - 3s 90ms/step - loss: 0.6929 - acc: 0.5075
- val_loss: 0.6912 - val_acc: 0.5500
Epoch 2/10
25/25 [==============================] - 2s 66ms/step - loss: 0.6758 - acc: 0.7763
- val_loss: 0.6894 - val_acc: 0.5750
Epoch 3/10
25/25 [==============================] - 3s 121ms/step - loss: 0.6587 - acc: 0.887
5 - val_loss: 0.6873 - val_acc: 0.5950
Epoch 4/10
25/25 [==============================] - 2s 77ms/step - loss: 0.6370 - acc: 0.9312
- val_loss: 0.6841 - val_acc: 0.5800
Epoch 5/10
25/25 [==============================] - 1s 43ms/step - loss: 0.6105 - acc: 0.9425
- val_loss: 0.6802 - val_acc: 0.5900
Epoch 6/10
25/25 [==============================] - 1s 55ms/step - loss: 0.5783 - acc: 0.9588
- val_loss: 0.6750 - val_acc: 0.6200
Epoch 7/10
25/25 [==============================] - 1s 45ms/step - loss: 0.5411 - acc: 0.9600
- val_loss: 0.6691 - val_acc: 0.6350
Epoch 8/10
25/25 [==============================] - 1s 49ms/step - loss: 0.4995 - acc: 0.9625
- val_loss: 0.6619 - val_acc: 0.6400
Epoch 9/10
25/25 [==============================] - 1s 34ms/step - loss: 0.4549 - acc: 0.9675
- val_loss: 0.6541 - val_acc: 0.6600
Epoch 10/10
25/25 [==============================] - 1s 30ms/step - loss: 0.4084 - acc: 0.9737
- val_loss: 0.6452 - val_acc: 0.6700
```

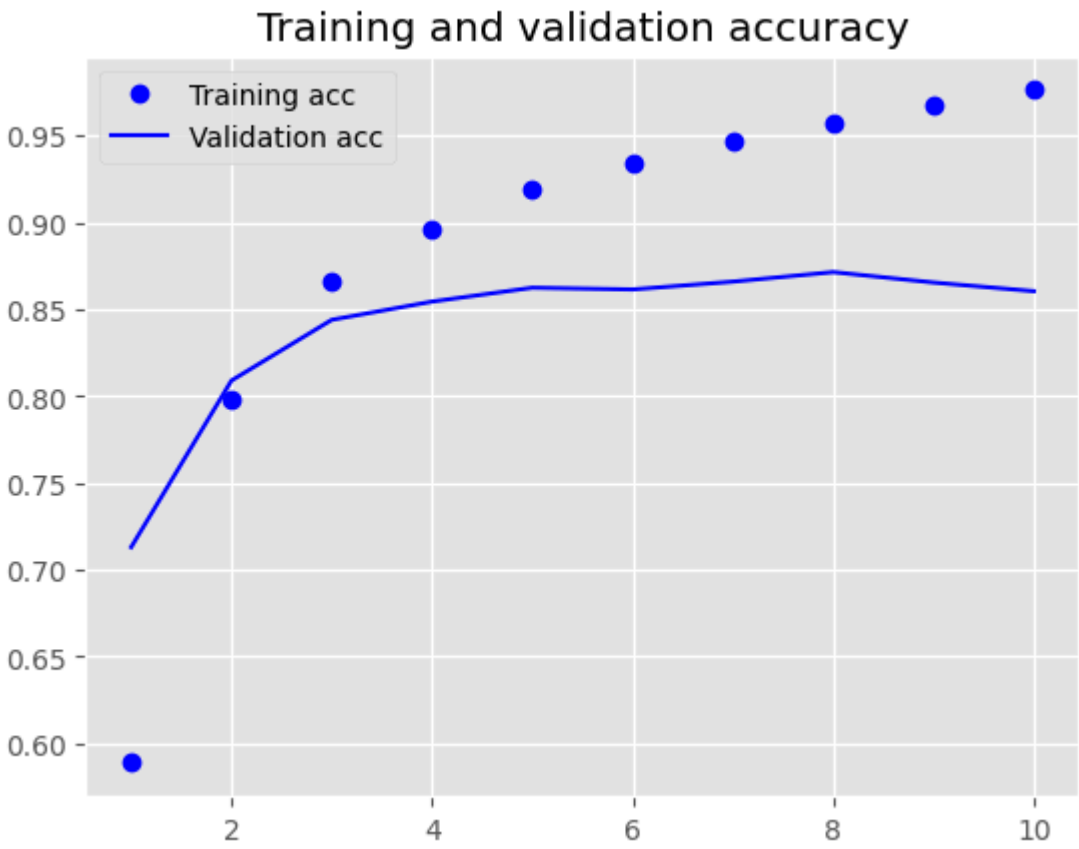


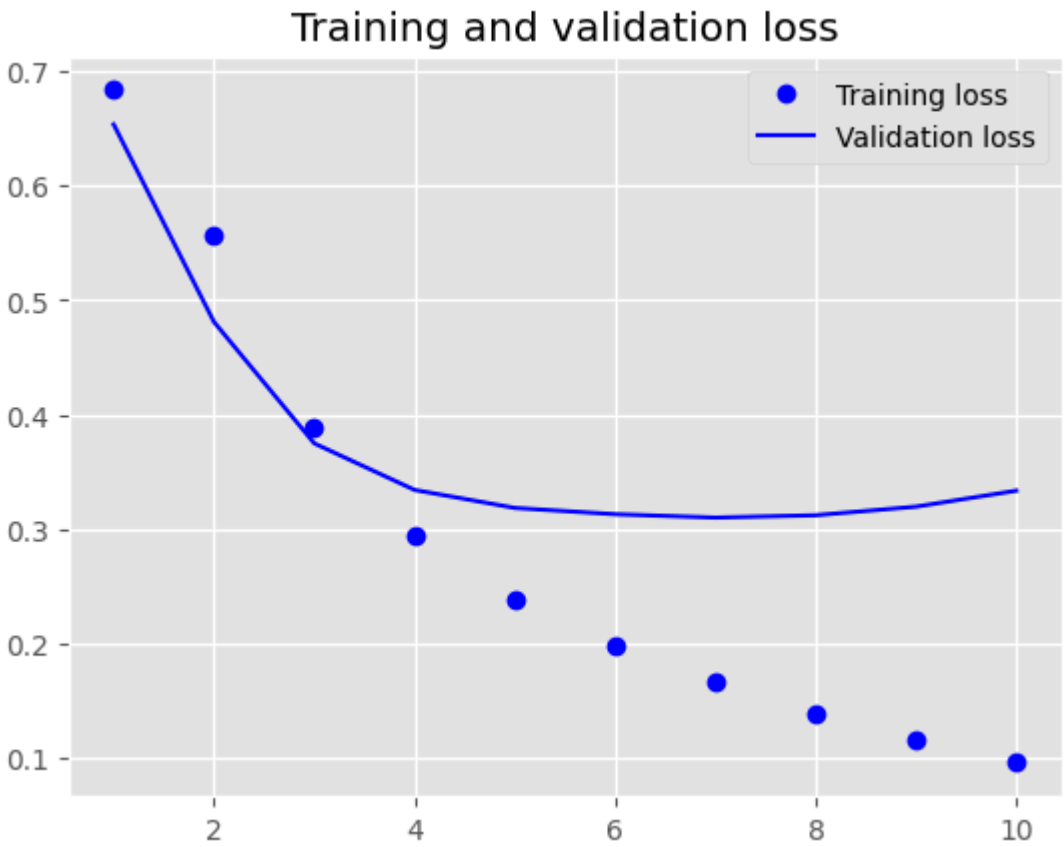


In [6]:
```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing
```

```
# Number of words to consider as features
max_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
maxlen = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:10000]
y_train = y_train[:10000]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We provide our Embedding layer a maximum input length specification
# in order to flatten the embedded inputs later
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
acc = history.history["acc"] # Training accuracy
val_acc = history.history["val_acc"] # Validation accuracy
loss = history.history["loss"] # Training loss
val_loss = history.history["val_loss"] # Validation loss

epochs = range(1, len(acc) + 1) #plots every epoch, here 10

plt.plot(epochs, acc, "bo", label = "Training acc") # "bo" gives dot plot
plt.plot(epochs, val_acc, "b", label = "Validation acc") # "b" gives line plot
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()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, 150, 8)            80000

 flatten_3 (Flatten)         (None, 1200)              0

 dense_3 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
250/250 [==============================] - 12s 46ms/step - loss: 0.6836 - acc: 0.5
888 - val_loss: 0.6537 - val_acc: 0.7130
Epoch 2/10
250/250 [==============================] - 5s 22ms/step - loss: 0.5571 - acc: 0.79
77 - val_loss: 0.4812 - val_acc: 0.8090
Epoch 3/10
250/250 [==============================] - 3s 14ms/step - loss: 0.3882 - acc: 0.86
56 - val_loss: 0.3750 - val_acc: 0.8440
Epoch 4/10
250/250 [==============================] - 2s 8ms/step - loss: 0.2942 - acc: 0.896
3 - val_loss: 0.3344 - val_acc: 0.8545
Epoch 5/10
250/250 [==============================] - 3s 11ms/step - loss: 0.2383 - acc: 0.91
90 - val_loss: 0.3186 - val_acc: 0.8625
Epoch 6/10
250/250 [==============================] - 2s 10ms/step - loss: 0.1974 - acc: 0.93
37 - val_loss: 0.3131 - val_acc: 0.8615
Epoch 7/10
250/250 [==============================] - 2s 7ms/step - loss: 0.1659 - acc: 0.947
3 - val_loss: 0.3102 - val_acc: 0.8660
Epoch 8/10
250/250 [==============================] - 1s 4ms/step - loss: 0.1384 - acc: 0.957
8 - val_loss: 0.3122 - val_acc: 0.8715
Epoch 9/10
250/250 [==============================] - 1s 5ms/step - loss: 0.1158 - acc: 0.967
1 - val_loss: 0.3197 - val_acc: 0.8655
Epoch 10/10
250/250 [==============================] - 1s 4ms/step - loss: 0.0959 - acc: 0.976
4 - val_loss: 0.3337 - val_acc: 0.8605
```





In [7]:
```
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  10.8M      0  0:00:07  0:00:07 --:--:-- 14.1M
```

In [8]:
```
import os
import shutil

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='utf-8')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

You can use pretrained word embeddings if there is insufficient training data to learn word embeddings together with the problem you wish to address. Each individual training review is gathered into a list of strings, with one string representing each review. Additionally, the labels of the reviews—positive or negative—are gathered into a list of labels.

Tokenizing the data

In [9]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

maxlen = 150 # cuts off review after 150 words
training_samples = 100 # Trains on 100 samples
validation_samples = 10000 # Validates 10000 samples
max_words = 10000 # Considers only the top 10000 words in the dataset

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index    # Length: 88582
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print("Shape of data tensor:", data.shape)
print("Shape of label tensor:", labels.shape)

indices = np.arange(data.shape[0]) # Splits data into training and validation set,
# all negatives first, then all positive
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples] # (200, 100)
y_train = labels[:training_samples] # shape (200,)
x_val = data[training_samples:training_samples+validation_samples] # shape (10000,
y_val = labels[training_samples:training_samples+validation_samples] # shape (10000
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
```

Downloading and Preprocessing the GloVe word embedding

In [10]:
```
import numpy as np
import requests
from io import BytesIO
import zipfile  # importing zipfile module

glove_url = 'https://nlp.stanford.edu/data/glove.6B.zip'  # URL to download GloVe e
glove_zip = requests.get(glove_url)

# Unzip the contents
with zipfile.ZipFile(BytesIO(glove_zip.content)) as z:
    z.extractall('/content/glove')

# Loading GloVe embeddings into memory
embeddings_index = {}
with open('/content/glove/glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

After that, an embedding matrix that can be placed into an embedding layer is required. The shape of the matrix has to be (max words, embedding dim), which is a 10000 x 100 matrix. The GloVe is 100 x 400000.

Preparing the GloVe word embeddings matrix

In [11]:
```
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
```

In [12]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

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: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_8 (Embedding)     (None, 150, 100)          1000000

 flatten_4 (Flatten)         (None, 15000)             0

 dense_4 (Dense)             (None, 32)                480032

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [13]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

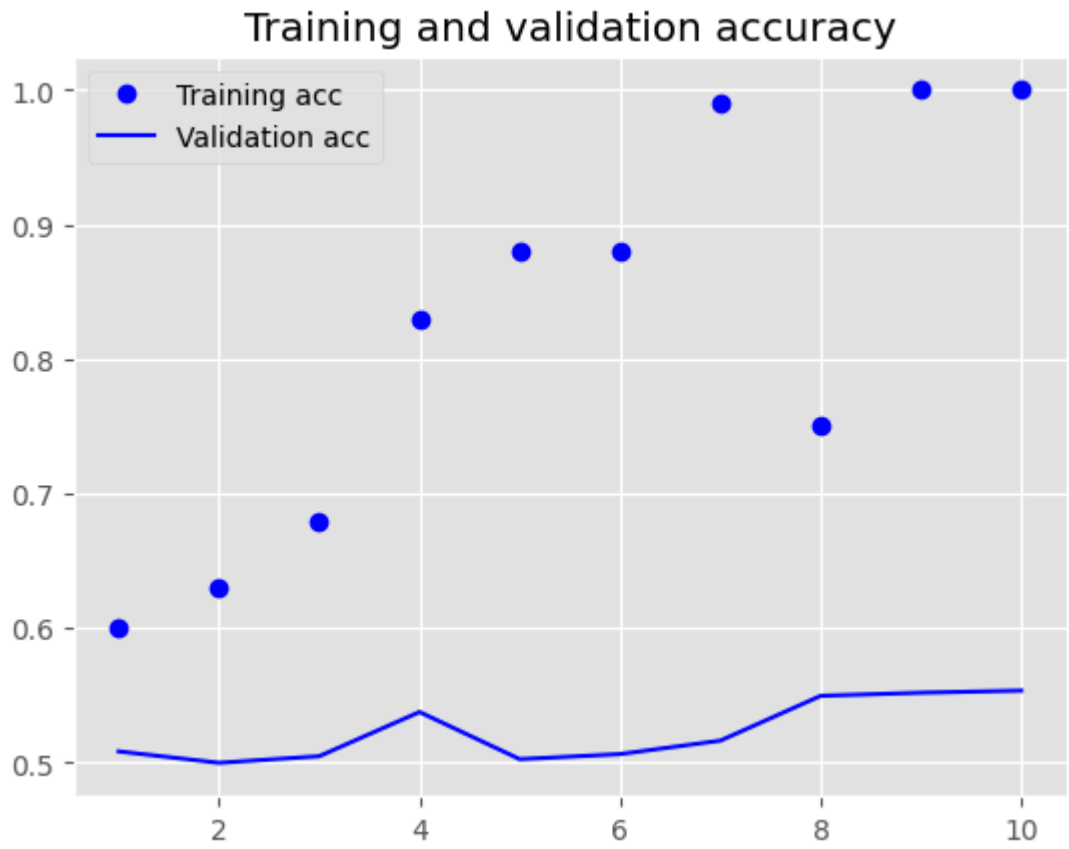
Loading pretrained word embedding into the Embedding layer We can make sure that the Embedding layer is not trainable when you call it by setting this to False. If you set trainable = True, the optimization algorithm will be able to change the word embedding settings. To keep pretrained sections from forgetting what they already "know," it is advisable to avoid updating them while they are still being trained.

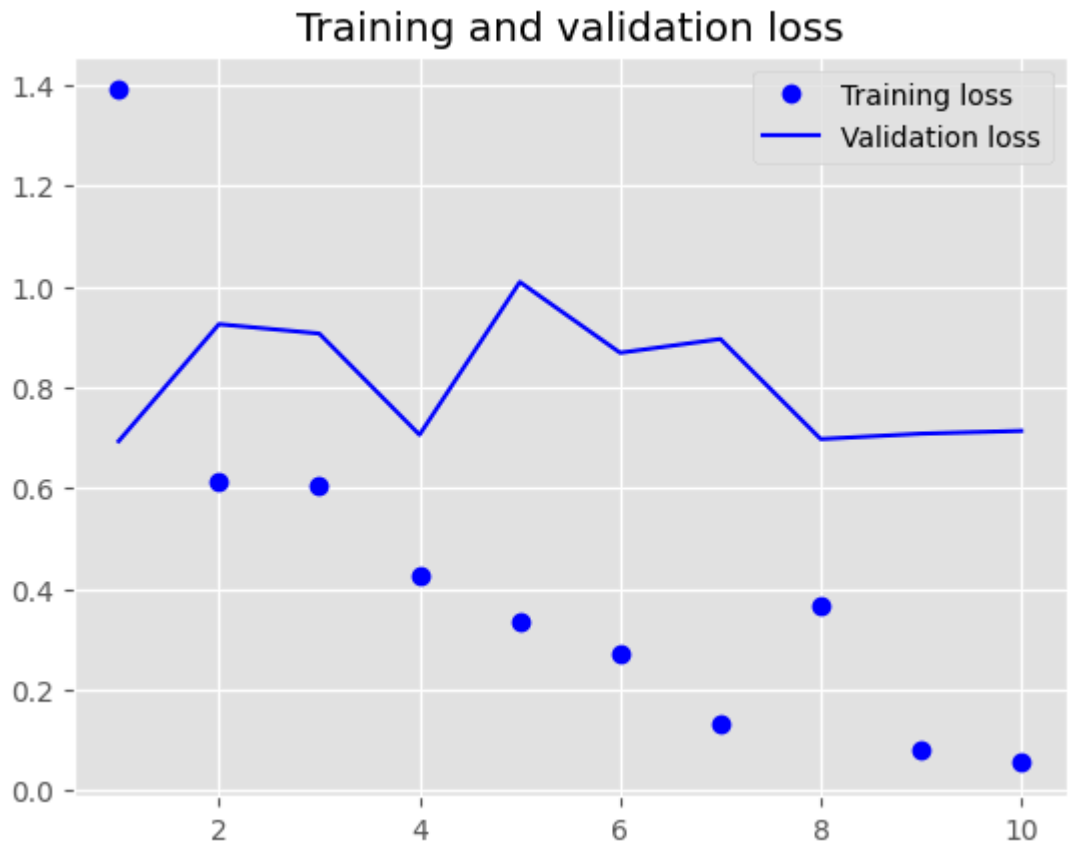
In [14]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
Epoch 1/10
4/4 [==============================] - 3s 520ms/step - loss: 1.3899 - acc: 0.6000
- val_loss: 0.6928 - val_acc: 0.5085
Epoch 2/10
4/4 [==============================] - 1s 245ms/step - loss: 0.6117 - acc: 0.6300
- val_loss: 0.9252 - val_acc: 0.5000
Epoch 3/10
4/4 [==============================] - 1s 220ms/step - loss: 0.6030 - acc: 0.6800
- val_loss: 0.9065 - val_acc: 0.5049
Epoch 4/10
4/4 [==============================] - 1s 219ms/step - loss: 0.4245 - acc: 0.8300
- val_loss: 0.7056 - val_acc: 0.5378
Epoch 5/10
4/4 [==============================] - 1s 200ms/step - loss: 0.3337 - acc: 0.8800
- val_loss: 1.0088 - val_acc: 0.5027
Epoch 6/10
4/4 [==============================] - 1s 203ms/step - loss: 0.2697 - acc: 0.8800
- val_loss: 0.8684 - val_acc: 0.5065
Epoch 7/10
4/4 [==============================] - 1s 218ms/step - loss: 0.1315 - acc: 0.9900
- val_loss: 0.8956 - val_acc: 0.5165
Epoch 8/10
4/4 [==============================] - 1s 197ms/step - loss: 0.3676 - acc: 0.7500
- val_loss: 0.6971 - val_acc: 0.5498
Epoch 9/10
4/4 [==============================] - 1s 195ms/step - loss: 0.0803 - acc: 1.0000
- val_loss: 0.7078 - val_acc: 0.5521
Epoch 10/10
4/4 [==============================] - 1s 219ms/step - loss: 0.0543 - acc: 1.0000
- val_loss: 0.7133 - val_acc: 0.5537
```

The model clearly overfits really quickly, which is to be anticipated given the small number of training examples. The reason for the significant variance in validation accuracy is the same.

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



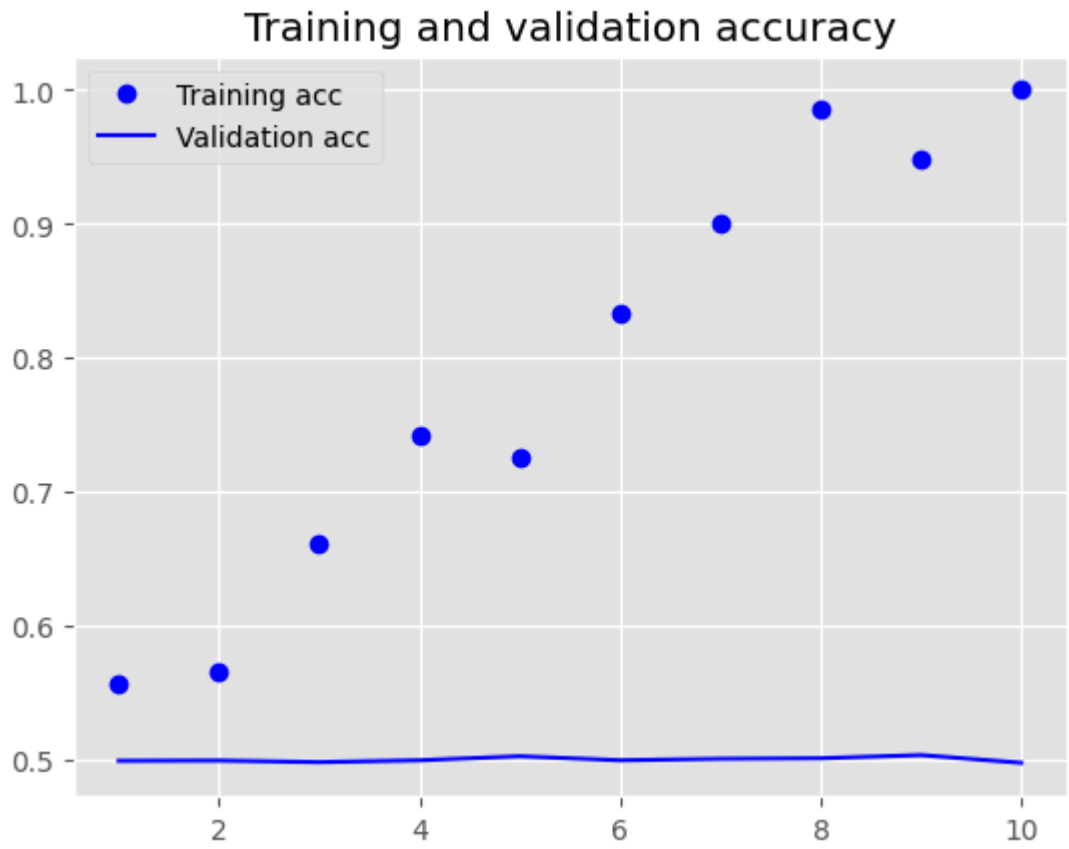


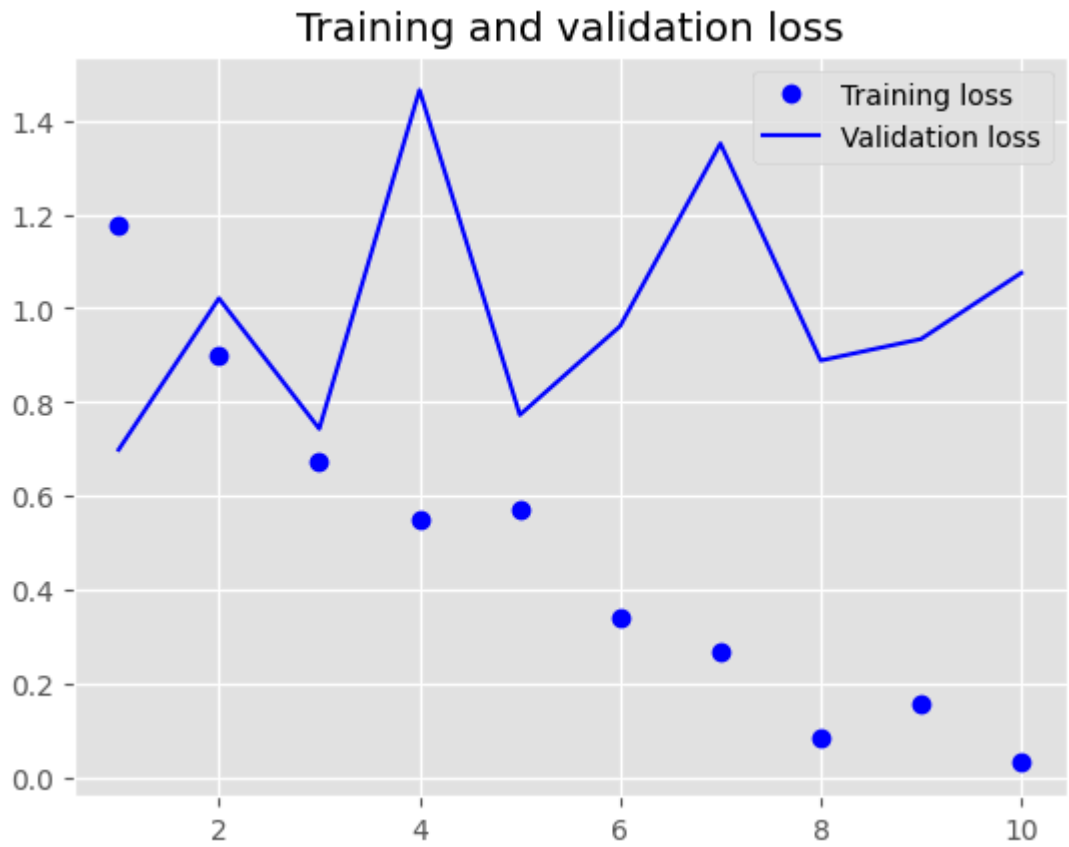
In [16]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

maxlen = 150 # cuts off review after 150 words
training_samples = 500 # Trains on 500 samples
validation_samples = 10000 # Validates 10000 samples
max_words = 10000 # Considers only the top 10000 words in the dataset
```

```
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index    # Length: 88582
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print("Shape of data tensor:", data.shape)
print("Shape of label tensor:", labels.shape)

indices = np.arange(data.shape[0]) # splits data into training and validation sets,
# however since the samples are arranged, it shuffles the data: all negatives first
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples] # (200, 100)
y_train = labels[:training_samples] # shape (200,)
x_val = data[training_samples:training_samples+validation_samples] # shape (10000,
y_val = labels[training_samples:training_samples+validation_samples] # shape (10000
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector

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

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

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_9 (Embedding)     (None, 150, 100)          1000000

 flatten_5 (Flatten)         (None, 15000)             0

 dense_6 (Dense)             (None, 32)                480032

 dense_7 (Dense)             (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
16/16 [==============================] - 3s 101ms/step - loss: 1.1760 - acc: 0.556
0 - val_loss: 0.6991 - val_acc: 0.4992
Epoch 2/10
16/16 [==============================] - 1s 93ms/step - loss: 0.9018 - acc: 0.5660
- val_loss: 1.0212 - val_acc: 0.4994
Epoch 3/10
16/16 [==============================] - 1s 92ms/step - loss: 0.6755 - acc: 0.6620
- val_loss: 0.7436 - val_acc: 0.4982
Epoch 4/10
16/16 [==============================] - 1s 94ms/step - loss: 0.5494 - acc: 0.7420
- val_loss: 1.4646 - val_acc: 0.4996
Epoch 5/10
16/16 [==============================] - 1s 95ms/step - loss: 0.5725 - acc: 0.7260
- val_loss: 0.7732 - val_acc: 0.5026
Epoch 6/10
16/16 [==============================] - 1s 93ms/step - loss: 0.3420 - acc: 0.8340
- val_loss: 0.9621 - val_acc: 0.4996
Epoch 7/10
16/16 [==============================] - 1s 83ms/step - loss: 0.2685 - acc: 0.9000
- val_loss: 1.3509 - val_acc: 0.5007
Epoch 8/10
16/16 [==============================] - 1s 94ms/step - loss: 0.0867 - acc: 0.9860
- val_loss: 0.8890 - val_acc: 0.5011
Epoch 9/10
16/16 [==============================] - 1s 75ms/step - loss: 0.1584 - acc: 0.9480
- val_loss: 0.9346 - val_acc: 0.5035
Epoch 10/10
16/16 [==============================] - 1s 95ms/step - loss: 0.0325 - acc: 1.0000
- val_loss: 1.0754 - val_acc: 0.4977
```



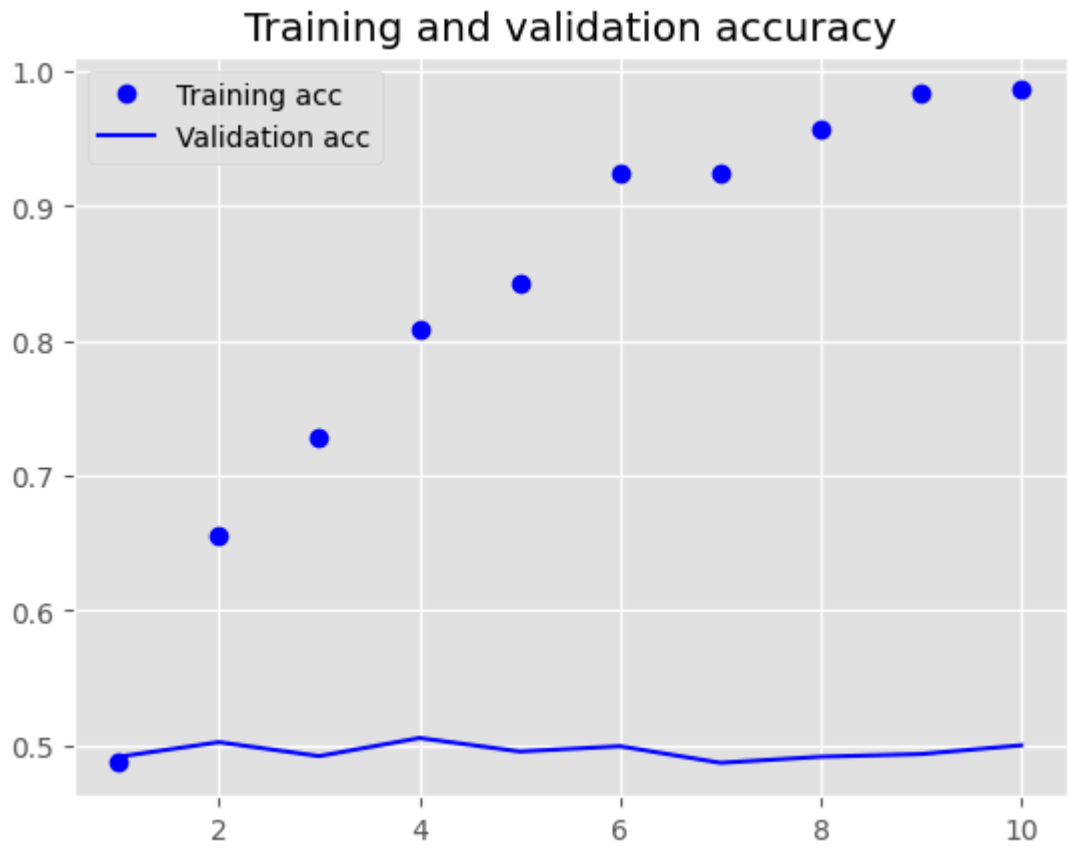


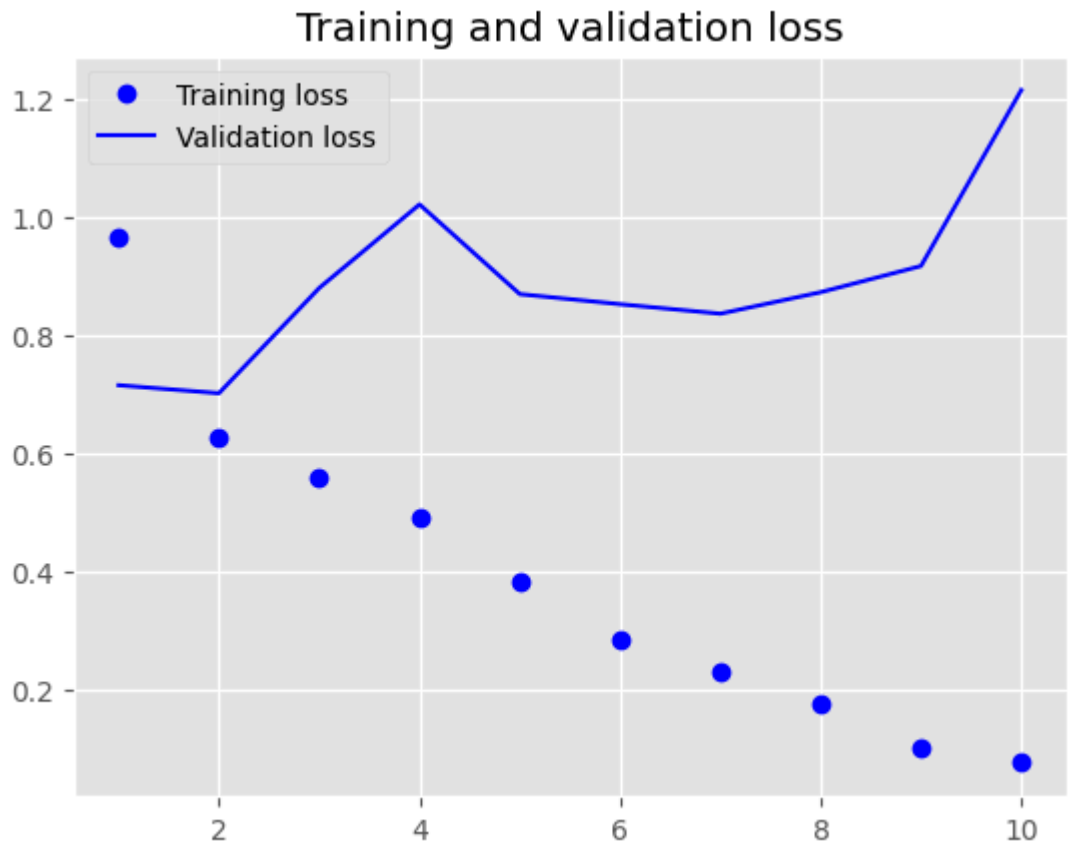
In [17]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

maxlen = 150 # cuts off review after 150 words
training_samples = 1000 #Trains on 1000 samples
validation_samples = 10000 # Validates o 10000 samples
max_words = 10000 # Considers only the top 10000 words in the dataset
```

```
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index      # Length: 88582
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print("Shape of data tensor:", data.shape)
print("Shape of label tensor:", labels.shape)

indices = np.arange(data.shape[0]) # splits data into training and validation sets,
# however since the samples are arranged, it shuffles the data: all negatives first
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples] # (200, 100)
y_train = labels[:training_samples] # shape (200,)
x_val = data[training_samples:training_samples+validation_samples] # shape (10000,
y_val = labels[training_samples:training_samples+validation_samples] # shape (10000
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector

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

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

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 150, 100)          1000000

 flatten_6 (Flatten)         (None, 15000)             0

 dense_8 (Dense)             (None, 32)                480032

 dense_9 (Dense)             (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
32/32 [==============================] - 1s 27ms/step - loss: 0.9644 - acc: 0.4880
- val_loss: 0.7155 - val_acc: 0.4916
Epoch 2/10
32/32 [==============================] - 1s 21ms/step - loss: 0.6266 - acc: 0.6560
- val_loss: 0.7019 - val_acc: 0.5024
Epoch 3/10
32/32 [==============================] - 1s 22ms/step - loss: 0.5585 - acc: 0.7290
- val_loss: 0.8798 - val_acc: 0.4921
Epoch 4/10
32/32 [==============================] - 1s 22ms/step - loss: 0.4901 - acc: 0.8090
- val_loss: 1.0210 - val_acc: 0.5055
Epoch 5/10
32/32 [==============================] - 1s 24ms/step - loss: 0.3824 - acc: 0.8420
- val_loss: 0.8694 - val_acc: 0.4955
Epoch 6/10
32/32 [==============================] - 1s 21ms/step - loss: 0.2840 - acc: 0.9240
- val_loss: 0.8524 - val_acc: 0.4994
Epoch 7/10
32/32 [==============================] - 1s 24ms/step - loss: 0.2326 - acc: 0.9250
- val_loss: 0.8363 - val_acc: 0.4871
Epoch 8/10
32/32 [==============================] - 1s 22ms/step - loss: 0.1753 - acc: 0.9570
- val_loss: 0.8726 - val_acc: 0.4916
Epoch 9/10
32/32 [==============================] - 1s 25ms/step - loss: 0.1029 - acc: 0.9830
- val_loss: 0.9169 - val_acc: 0.4936
Epoch 10/10
32/32 [==============================] - 1s 47ms/step - loss: 0.0772 - acc: 0.9860
- val_loss: 1.2143 - val_acc: 0.5001
```



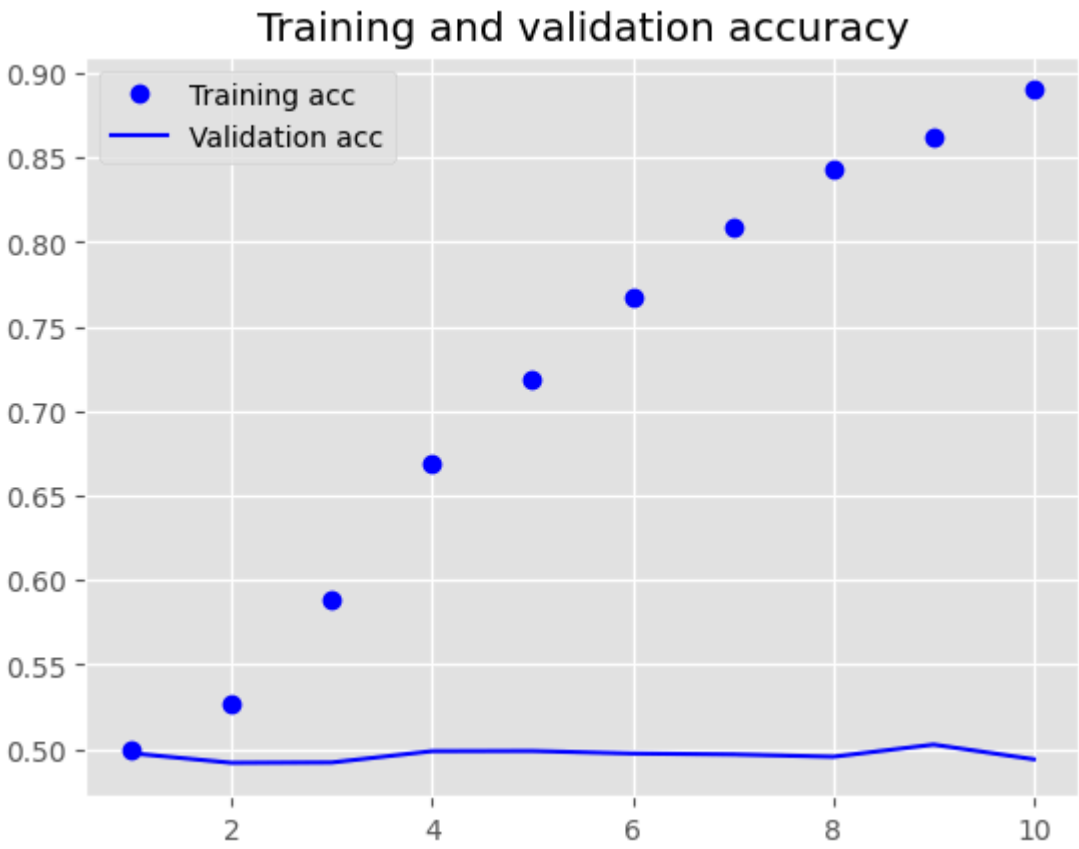


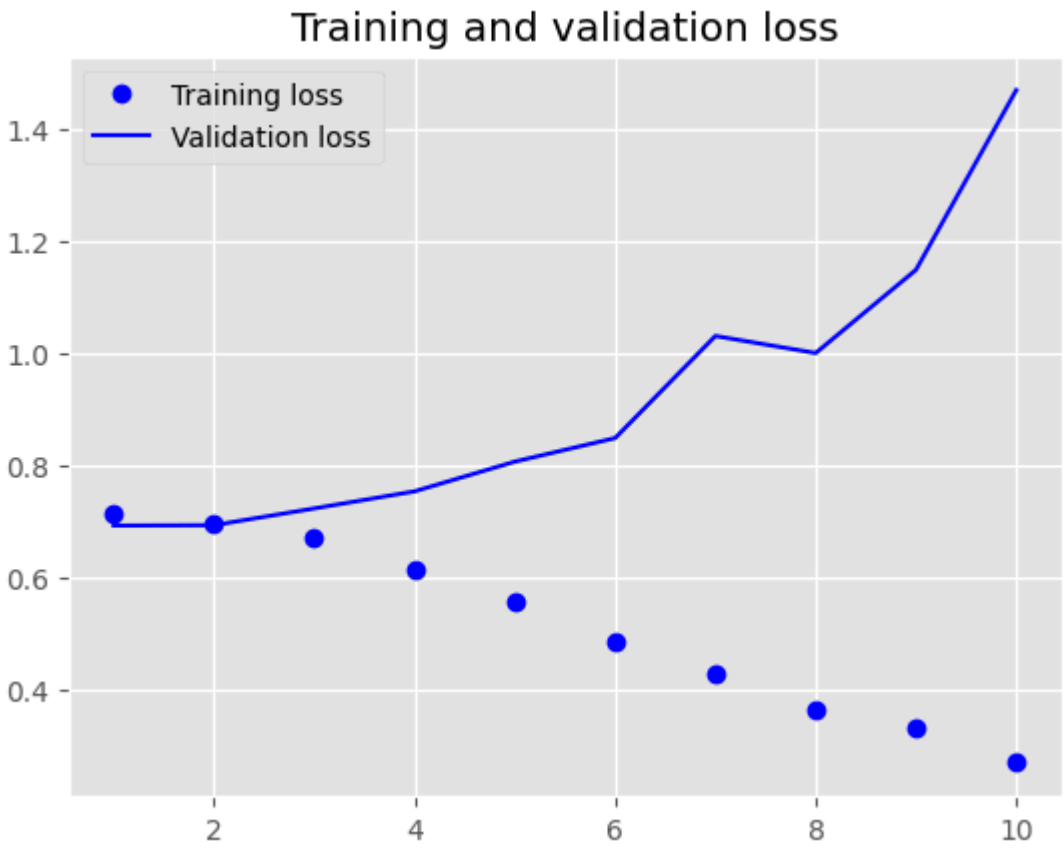
In [18]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

maxlen = 150 # cuts off review after 150 words
training_samples = 10000 # Trains on 10000 samples
validation_samples = 10000 # Validates o 10000 samples
max_words = 10000 # Considers only the top 10000 words in the dataset
```

```
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index       # Length: 88582
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print("Shape of data tensor:", data.shape)
print("Shape of label tensor:", labels.shape)

indices = np.arange(data.shape[0]) # splits data into training and validation sets,
# however since the samples are arranged, it shuffles the data: all negatives first
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples] # (200, 100)
y_train = labels[:training_samples] # shape (200,)
x_val = data[training_samples:training_samples+validation_samples] # shape (10000,
y_val = labels[training_samples:training_samples+validation_samples] # shape (10000
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector

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

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

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 150, 100)          1000000

 flatten_7 (Flatten)         (None, 15000)             0

 dense_10 (Dense)            (None, 32)                480032

 dense_11 (Dense)            (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
313/313 [==============================] - 4s 10ms/step - loss: 0.7133 - acc: 0.49
91 - val_loss: 0.6932 - val_acc: 0.4975
Epoch 2/10
313/313 [==============================] - 2s 8ms/step - loss: 0.6962 - acc: 0.526
4 - val_loss: 0.6938 - val_acc: 0.4918
Epoch 3/10
313/313 [==============================] - 3s 8ms/step - loss: 0.6721 - acc: 0.587
8 - val_loss: 0.7237 - val_acc: 0.4920
Epoch 4/10
313/313 [==============================] - 3s 10ms/step - loss: 0.6129 - acc: 0.66
91 - val_loss: 0.7542 - val_acc: 0.4987
Epoch 5/10
313/313 [==============================] - 3s 10ms/step - loss: 0.5560 - acc: 0.71
83 - val_loss: 0.8072 - val_acc: 0.4988
Epoch 6/10
313/313 [==============================] - 3s 9ms/step - loss: 0.4851 - acc: 0.767
5 - val_loss: 0.8493 - val_acc: 0.4973
Epoch 7/10
313/313 [==============================] - 3s 9ms/step - loss: 0.4279 - acc: 0.808
6 - val_loss: 1.0316 - val_acc: 0.4968
Epoch 8/10
313/313 [==============================] - 3s 9ms/step - loss: 0.3641 - acc: 0.843
3 - val_loss: 1.0011 - val_acc: 0.4954
Epoch 9/10
313/313 [==============================] - 2s 8ms/step - loss: 0.3305 - acc: 0.862
3 - val_loss: 1.1496 - val_acc: 0.5026
Epoch 10/10
313/313 [==============================] - 2s 6ms/step - loss: 0.2696 - acc: 0.890
0 - val_loss: 1.4705 - val_acc: 0.4939
```


Training and validation accuracy
Training acc
Validation acc
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
1.4
1.2
1.0
0.8
0.6
0.4
2
4
6
8
10