In [78]:
```
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 [79]:
```
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
maximum_features = 10000
# After this amount of words, cut the texts
#(among top max_features most common words)
max_len = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=maximum_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=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
from keras.models import Sequential
from keras.layers import Flatten, Dense
```

In [80]:
```
model = Sequential()

# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs

model.add(Embedding(10000, 8, input_length=max_len))

# 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'))

#compiling the model
```

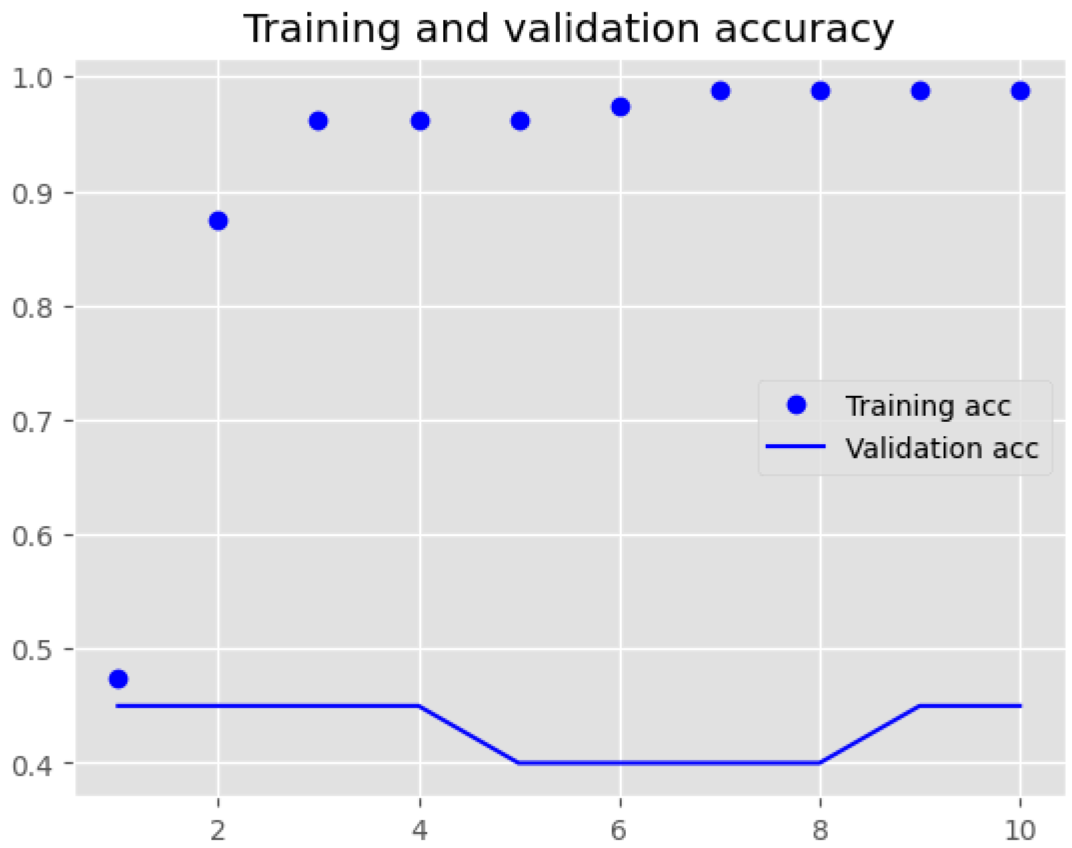
```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

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

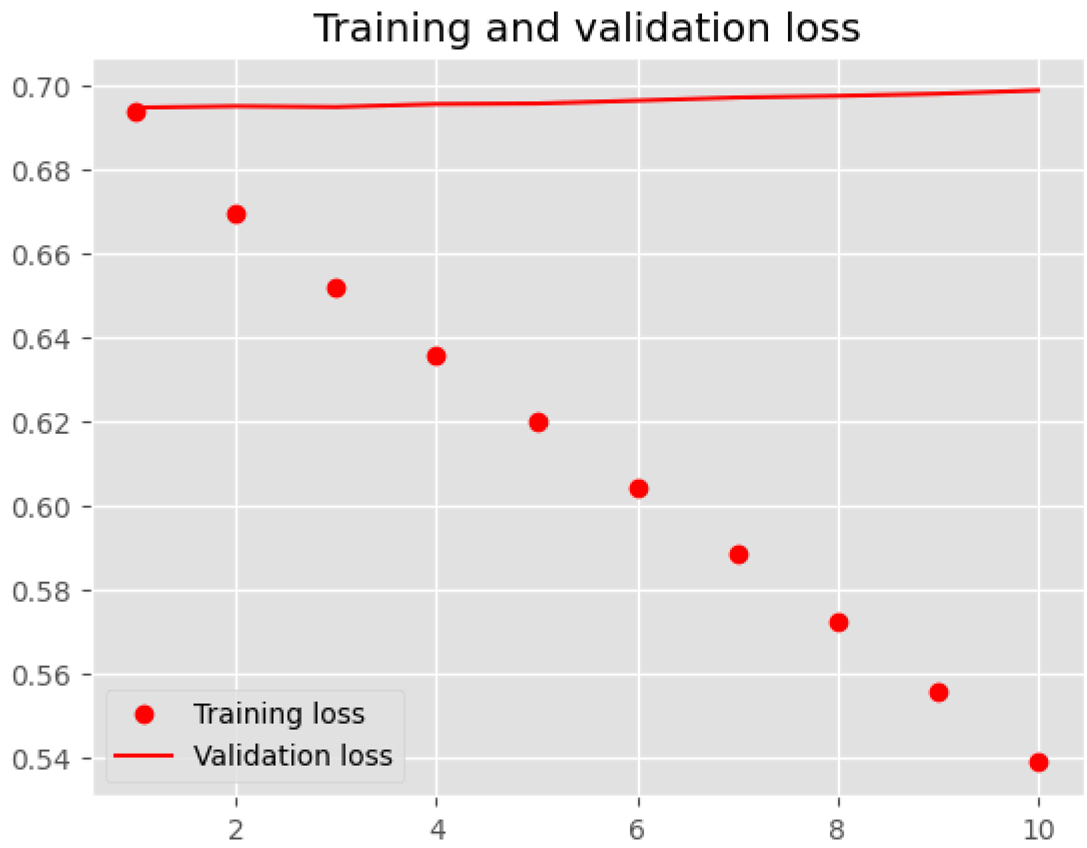
```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_42 (Embedding)    (None, 150, 8)            80000

 flatten_26 (Flatten)        (None, 1200)              0

 dense_37 (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 [==============================] - 1s 99ms/step - loss: 0.6939 - acc: 0.4750 - val_loss: 0.6947 - val_acc: 0.4500
Epoch 2/10
3/3 [==============================] - 0s 20ms/step - loss: 0.6695 - acc: 0.8750 - val_loss: 0.6950 - val_acc: 0.4500
Epoch 3/10
3/3 [==============================] - 0s 18ms/step - loss: 0.6516 - acc: 0.9625 - val_loss: 0.6948 - val_acc: 0.4500
Epoch 4/10
3/3 [==============================] - 0s 19ms/step - loss: 0.6355 - acc: 0.9625 - val_loss: 0.6955 - val_acc: 0.4500
Epoch 5/10
3/3 [==============================] - 0s 19ms/step - loss: 0.6197 - acc: 0.9625 - val_loss: 0.6956 - val_acc: 0.4000
Epoch 6/10
3/3 [==============================] - 0s 19ms/step - loss: 0.6042 - acc: 0.9750 - val_loss: 0.6963 - val_acc: 0.4000
Epoch 7/10
3/3 [==============================] - 0s 18ms/step - loss: 0.5884 - acc: 0.9875 - val_loss: 0.6971 - val_acc: 0.4000
Epoch 8/10
3/3 [==============================] - 0s 19ms/step - loss: 0.5722 - acc: 0.9875 - val_loss: 0.6974 - val_acc: 0.4000
Epoch 9/10
3/3 [==============================] - 0s 18ms/step - loss: 0.5558 - acc: 0.9875 - val_loss: 0.6980 - val_acc: 0.4500
Epoch 10/10
3/3 [==============================] - 0s 19ms/step - loss: 0.5387 - acc: 0.9875 - val_loss: 0.6988 - val_acc: 0.4500
```

In [81]:

```
import matplotlib.pyplot as plt

# Training accuracy
acc = history.history["acc"]
# Validation accuracy
valid_accuracy = history.history["val_acc"]
# Training loss
loss = history.history["loss"]
# Validation loss
valid_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, valid_accuracy, "b", label = "Validation acc") # "b" gives line pl
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, "ro", label = "Training loss")
plt.plot(epochs, valid_loss, "r", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```





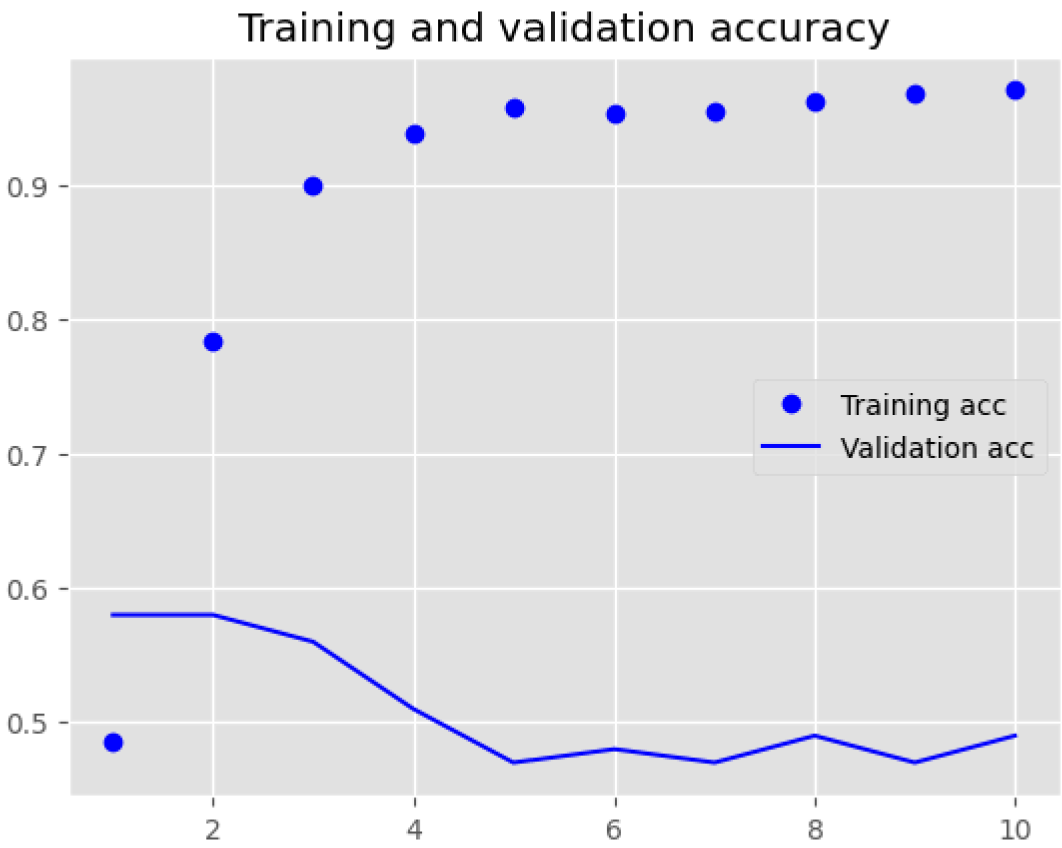
In [82]:
```
test_loss, T_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

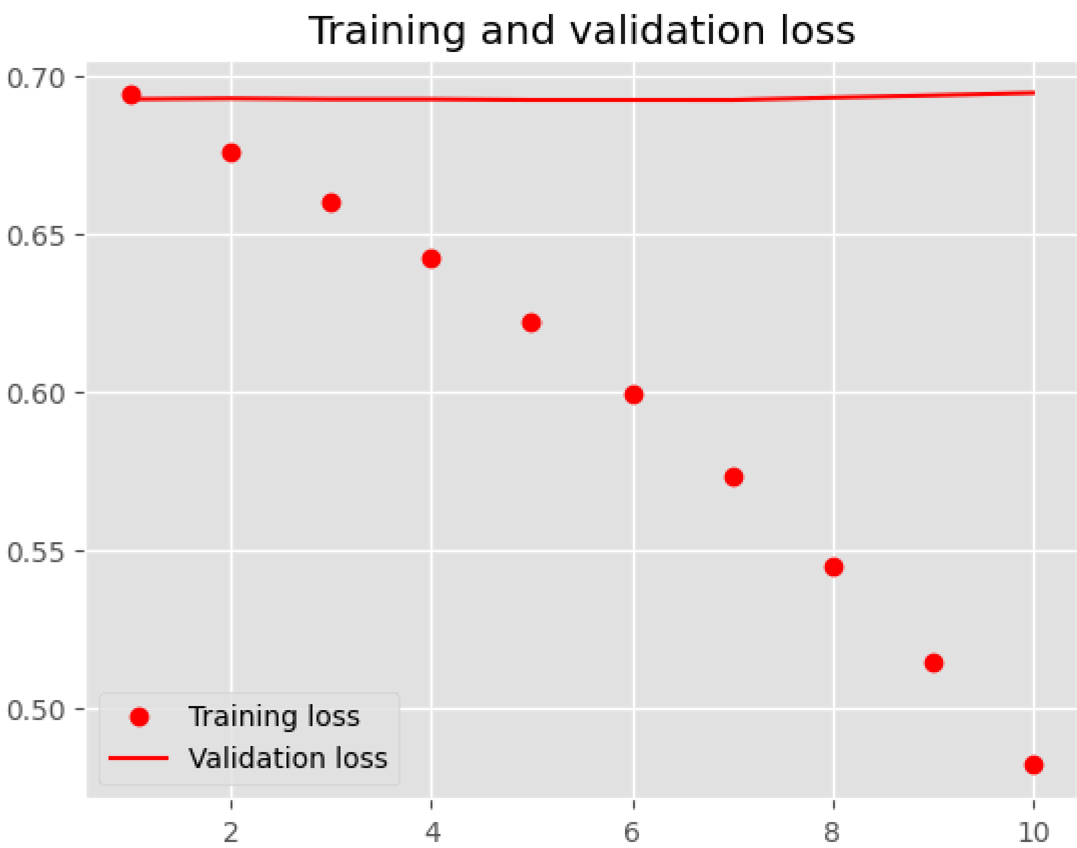
```
782/782 [==============================] - 1s 2ms/step - loss: 0.6944 - acc: 0.498
1
Test loss: 0.6943515539169312
Test accuracy: 0.49807998538017273
```

In [83]:
```
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
maximum_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
max_len = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=maximum_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=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
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=max_len))
# 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
valid_accuracy = history.history["val_acc"] # Validation accuracy
loss = history.history["loss"] # Training loss
valid_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, valid_accuracy, "b", label = "Validation acc") # "b" gives line pl
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, "ro", label = "Training loss")
plt.plot(epochs, valid_loss, "r", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```

```
Model: "sequential_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_44 (Embedding)    (None, 150, 8)            80000     
                                                                 
 flatten_27 (Flatten)        (None, 1200)              0         
                                                                 
 dense_38 (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 [==============================] - 1s 34ms/step - loss: 0.6939 - acc: 0.4850
- val_loss: 0.6925 - val_acc: 0.5800
Epoch 2/10
13/13 [==============================] - 0s 10ms/step - loss: 0.6756 - acc: 0.7825
- val_loss: 0.6927 - val_acc: 0.5800
Epoch 3/10
13/13 [==============================] - 0s 10ms/step - loss: 0.6598 - acc: 0.9000
- val_loss: 0.6925 - val_acc: 0.5600
Epoch 4/10
13/13 [==============================] - 0s 9ms/step - loss: 0.6423 - acc: 0.9375
- val_loss: 0.6925 - val_acc: 0.5100
Epoch 5/10
13/13 [==============================] - 0s 10ms/step - loss: 0.6222 - acc: 0.9575
- val_loss: 0.6923 - val_acc: 0.4700
Epoch 6/10
13/13 [==============================] - 0s 11ms/step - loss: 0.5992 - acc: 0.9525
- val_loss: 0.6922 - val_acc: 0.4800
Epoch 7/10
13/13 [==============================] - 0s 10ms/step - loss: 0.5734 - acc: 0.9550
- val_loss: 0.6922 - val_acc: 0.4700
Epoch 8/10
13/13 [==============================] - 0s 11ms/step - loss: 0.5450 - acc: 0.9625
- val_loss: 0.6930 - val_acc: 0.4900
Epoch 9/10
13/13 [==============================] - 0s 11ms/step - loss: 0.5142 - acc: 0.9675
- val_loss: 0.6937 - val_acc: 0.4700
Epoch 10/10
13/13 [==============================] - 0s 11ms/step - loss: 0.4818 - acc: 0.9700
- val_loss: 0.6945 - val_acc: 0.4900
```





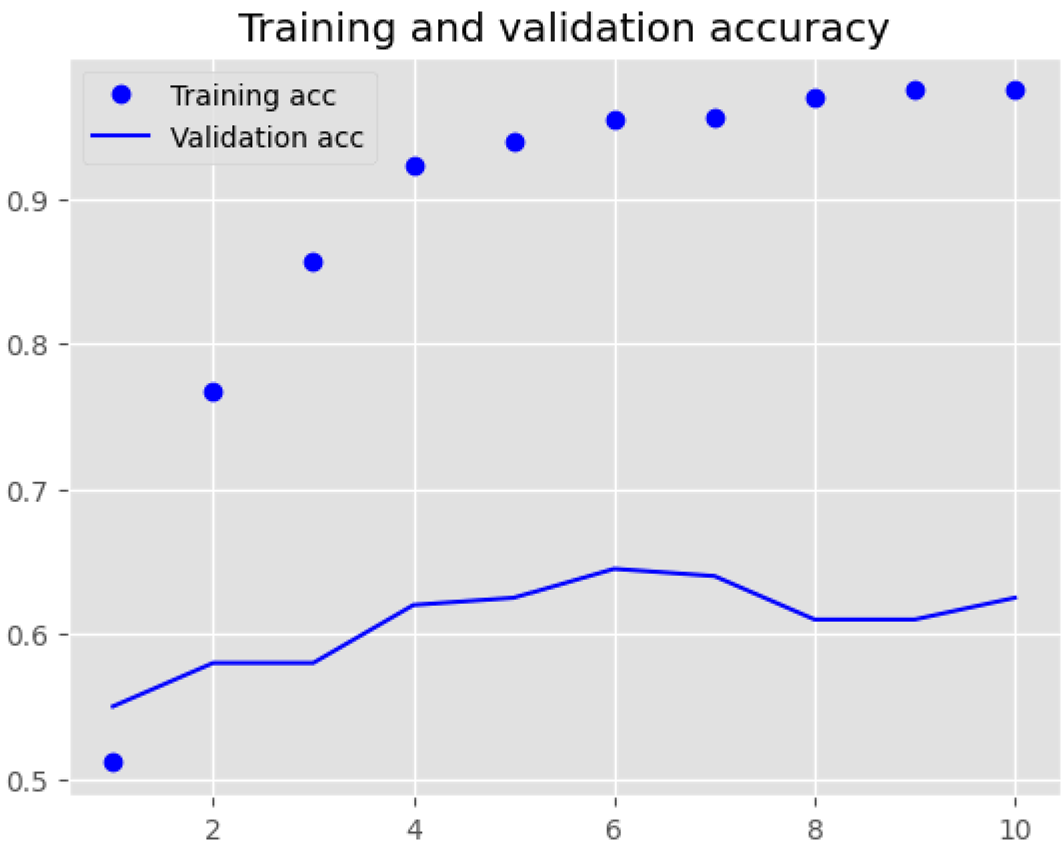
In [84]:
```
test_loss, T_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

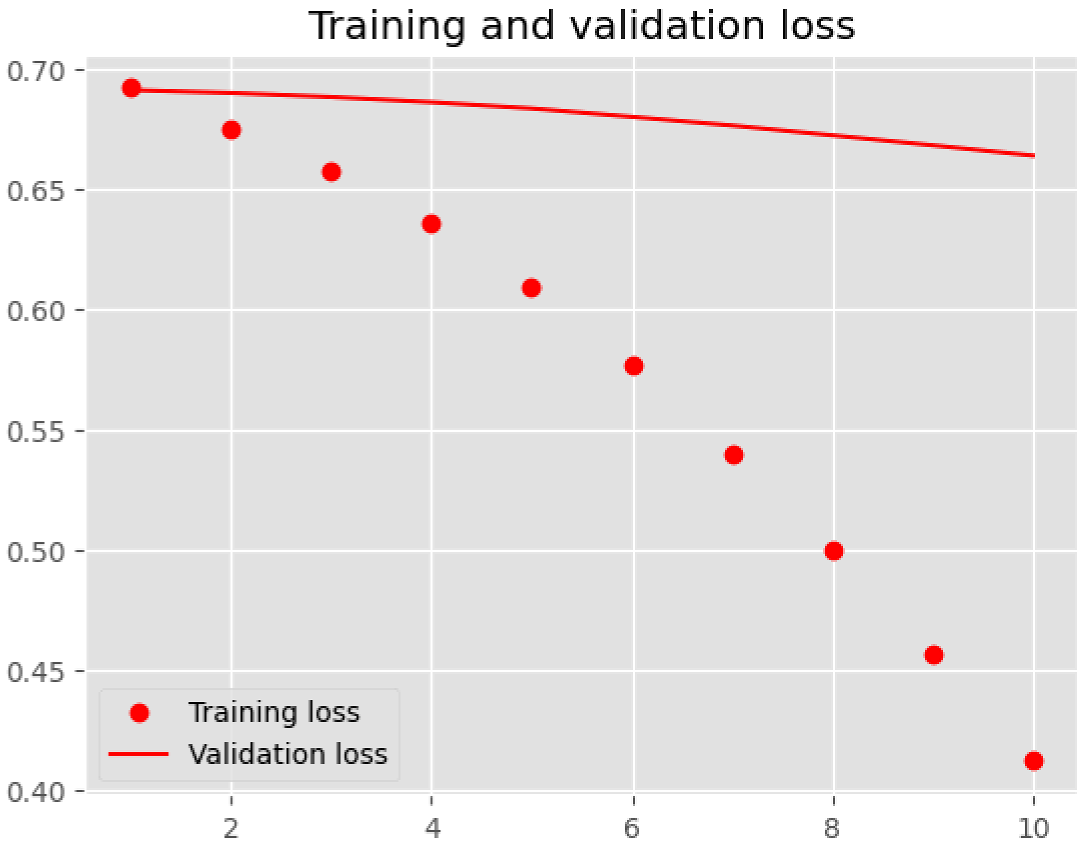
```
782/782 [==============================] - 2s 3ms/step - loss: 0.6916 - acc: 0.526
0
Test loss: 0.6915744543075562
Test accuracy: 0.5259600281715393
```

In [85]:
```
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
maximum_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
max_len = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=maximum_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=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
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=max_len))
# 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
valid_accuracy = history.history["val_acc"] # Validation accuracy
loss = history.history["loss"] # Training loss
valid_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, valid_accuracy, "b", label = "Validation acc") # "b" gives line pl
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, "ro", label = "Training loss")
plt.plot(epochs, valid_loss, "r", label = "Validation loss")
plt.title("Training and validation loss")
```

```
plt.legend()

plt.show()
```

```
Model: "sequential_28"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_46 (Embedding)    (None, 150, 8)            80000     
                                                                 
 flatten_28 (Flatten)        (None, 1200)              0         
                                                                 
 dense_39 (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 [==============================] - 1s 12ms/step - loss: 0.6920 - acc: 0.5113
- val_loss: 0.6911 - val_acc: 0.5500
Epoch 2/10
25/25 [==============================] - 0s 4ms/step - loss: 0.6748 - acc: 0.7675
- val_loss: 0.6899 - val_acc: 0.5800
Epoch 3/10
25/25 [==============================] - 0s 4ms/step - loss: 0.6573 - acc: 0.8575
- val_loss: 0.6883 - val_acc: 0.5800
Epoch 4/10
25/25 [==============================] - 0s 5ms/step - loss: 0.6355 - acc: 0.9237
- val_loss: 0.6861 - val_acc: 0.6200
Epoch 5/10
25/25 [==============================] - 0s 5ms/step - loss: 0.6088 - acc: 0.9400
- val_loss: 0.6835 - val_acc: 0.6250
Epoch 6/10
25/25 [==============================] - 0s 4ms/step - loss: 0.5769 - acc: 0.9550
- val_loss: 0.6800 - val_acc: 0.6450
Epoch 7/10
25/25 [==============================] - 0s 5ms/step - loss: 0.5403 - acc: 0.9563
- val_loss: 0.6764 - val_acc: 0.6400
Epoch 8/10
25/25 [==============================] - 0s 5ms/step - loss: 0.5000 - acc: 0.9700
- val_loss: 0.6723 - val_acc: 0.6100
Epoch 9/10
25/25 [==============================] - 0s 4ms/step - loss: 0.4567 - acc: 0.9750
- val_loss: 0.6681 - val_acc: 0.6100
Epoch 10/10
25/25 [==============================] - 0s 5ms/step - loss: 0.4123 - acc: 0.9750
- val_loss: 0.6639 - val_acc: 0.6250
```





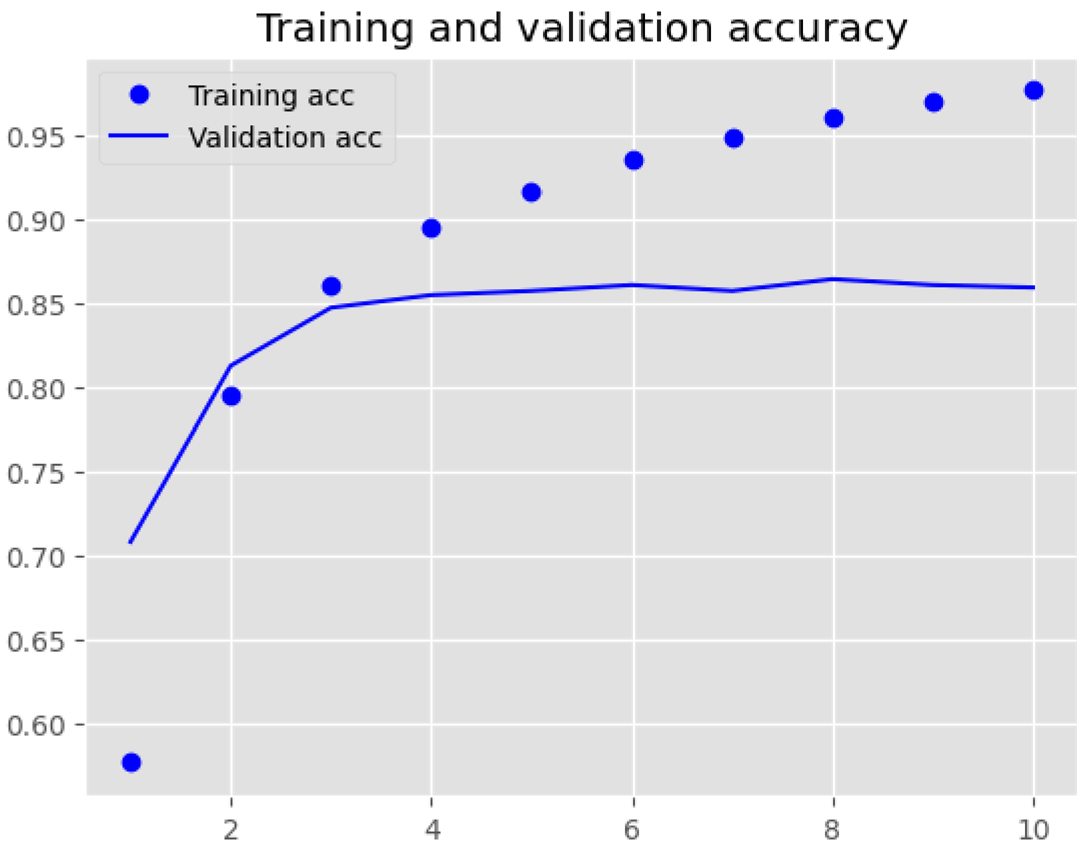
In [86]:
```
test_loss, T_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

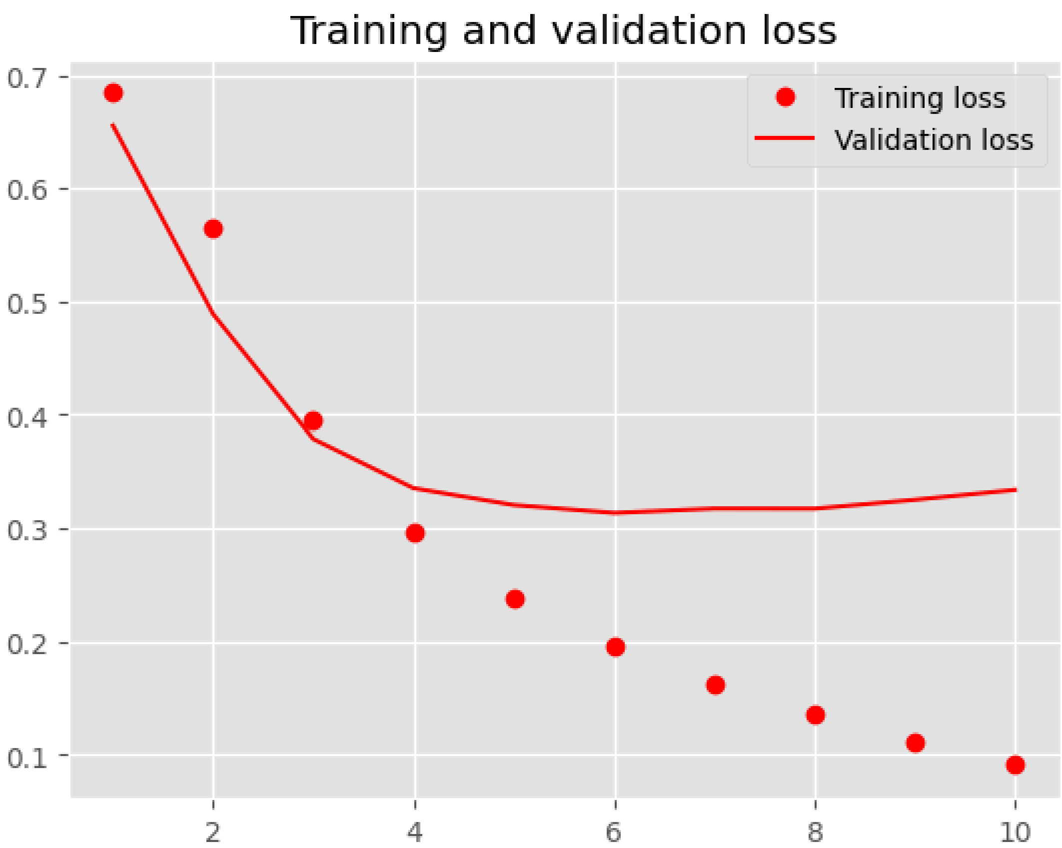
```
782/782 [==============================] - 1s 2ms/step - loss: 0.6790 - acc: 0.568
6
Test loss: 0.679002583026886
Test accuracy: 0.5685999989509583
```

In [87]:
```
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
maximum_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
max_len = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=maximum_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=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
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=max_len))
# 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
valid_accuracy = history.history["val_acc"] # Validation accuracy
loss = history.history["loss"] # Training loss
valid_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, valid_accuracy, "b", label = "Validation acc") # "b" gives line pl
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss, "ro", label = "Training loss")
plt.plot(epochs, valid_loss, "r", label = "Validation loss")
plt.title("Training and validation loss")
```

```
plt.legend()

plt.show()
```

```
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_48 (Embedding)    (None, 150, 8)            80000     
                                                                 
 flatten_29 (Flatten)        (None, 1200)              0         
                                                                 
 dense_40 (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 [==============================] - 2s 7ms/step - loss: 0.6842 - acc: 0.577
3 - val_loss: 0.6554 - val_acc: 0.7085
Epoch 2/10
250/250 [==============================] - 1s 6ms/step - loss: 0.5643 - acc: 0.795
8 - val_loss: 0.4888 - val_acc: 0.8130
Epoch 3/10
250/250 [==============================] - 1s 5ms/step - loss: 0.3952 - acc: 0.860
8 - val_loss: 0.3789 - val_acc: 0.8475
Epoch 4/10
250/250 [==============================] - 2s 6ms/step - loss: 0.2972 - acc: 0.895
7 - val_loss: 0.3356 - val_acc: 0.8550
Epoch 5/10
250/250 [==============================] - 2s 7ms/step - loss: 0.2389 - acc: 0.916
8 - val_loss: 0.3207 - val_acc: 0.8575
Epoch 6/10
250/250 [==============================] - 2s 6ms/step - loss: 0.1965 - acc: 0.935
1 - val_loss: 0.3137 - val_acc: 0.8610
Epoch 7/10
250/250 [==============================] - 1s 6ms/step - loss: 0.1631 - acc: 0.949
1 - val_loss: 0.3175 - val_acc: 0.8575
Epoch 8/10
250/250 [==============================] - 1s 6ms/step - loss: 0.1353 - acc: 0.961
3 - val_loss: 0.3174 - val_acc: 0.8645
Epoch 9/10
250/250 [==============================] - 1s 6ms/step - loss: 0.1112 - acc: 0.969
7 - val_loss: 0.3253 - val_acc: 0.8610
Epoch 10/10
250/250 [==============================] - 1s 3ms/step - loss: 0.0911 - acc: 0.976
8 - val_loss: 0.3340 - val_acc: 0.8595
```





In [88]:
```
test_loss, T_accuracy_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

```
782/782 [==============================] - 1s 2ms/step - loss: 0.3409 - acc: 0.855
2
Test loss: 0.3409438133239746
Test accuracy: 0.5685999989509583
```

In [89]:
```
!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  37.2M      0  0:00:02  0:00:02 --:--:-- 37.2M
```

In [90]:
```
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)
```

Pretrained word embeddings are an option if there is insufficient training data to effectively learn word embeddings with the problem you want to solve.

The individual training reviews are collected into a list of strings, one string per review, and also the review labels (positive/negative) are collected into a labels list.

Tokenizing the data

In [91]:
```
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 (1000€
```

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

Downloading and Preprocessing the GloVe word embedding

In [92]:
```
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 €
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.
```

Subsequently, an embedding matrix that fits inside an embedding layer is needed. The matrix must be in the form of a 10000 x 100 matrix with the dimensions (max words, embedding dim). 100 × 400000 is the GloVe.

Preparing the GloVe word embeddings matrix

In [93]:
```
embedding_dimension = 100

embedding_matrix = np.zeros((max_words, embedding_dimension))
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 [94]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embedding_dimension, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_49 (Embedding)    (None, 150, 100)          1000000   
                                                                 
 flatten_30 (Flatten)        (None, 15000)             0         
                                                                 
 dense_41 (Dense)            (None, 32)                480032    
                                                                 
 dense_42 (Dense)            (None, 1)                 33        
                                                                 
=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [95]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

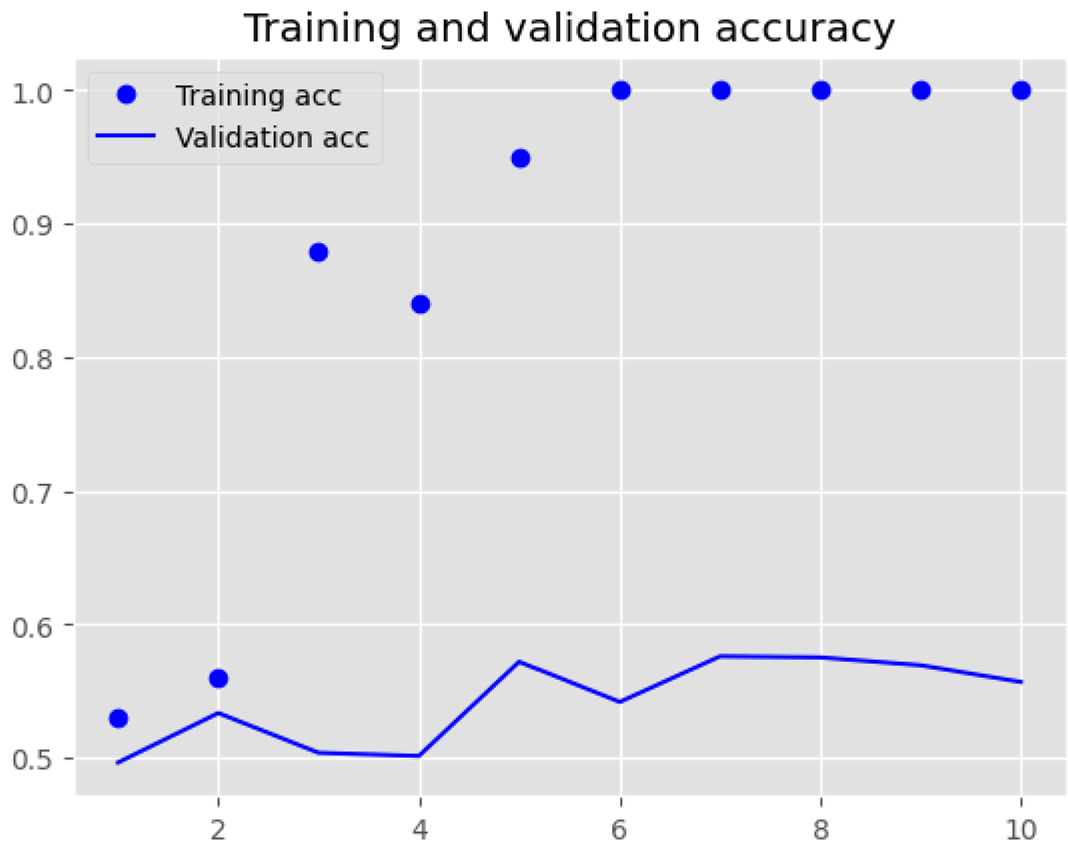
Adding pretrained word embedding to the Embeddig layer to make sure the Embedding layer is not trainable when you call it, we may set this to False. If trainable = True is selected, the optimization method will have the ability to change the word embedding settings. In order to keep pretrained sections from forgetting what they already "know," it is preferable to not update them while they are still practicing.

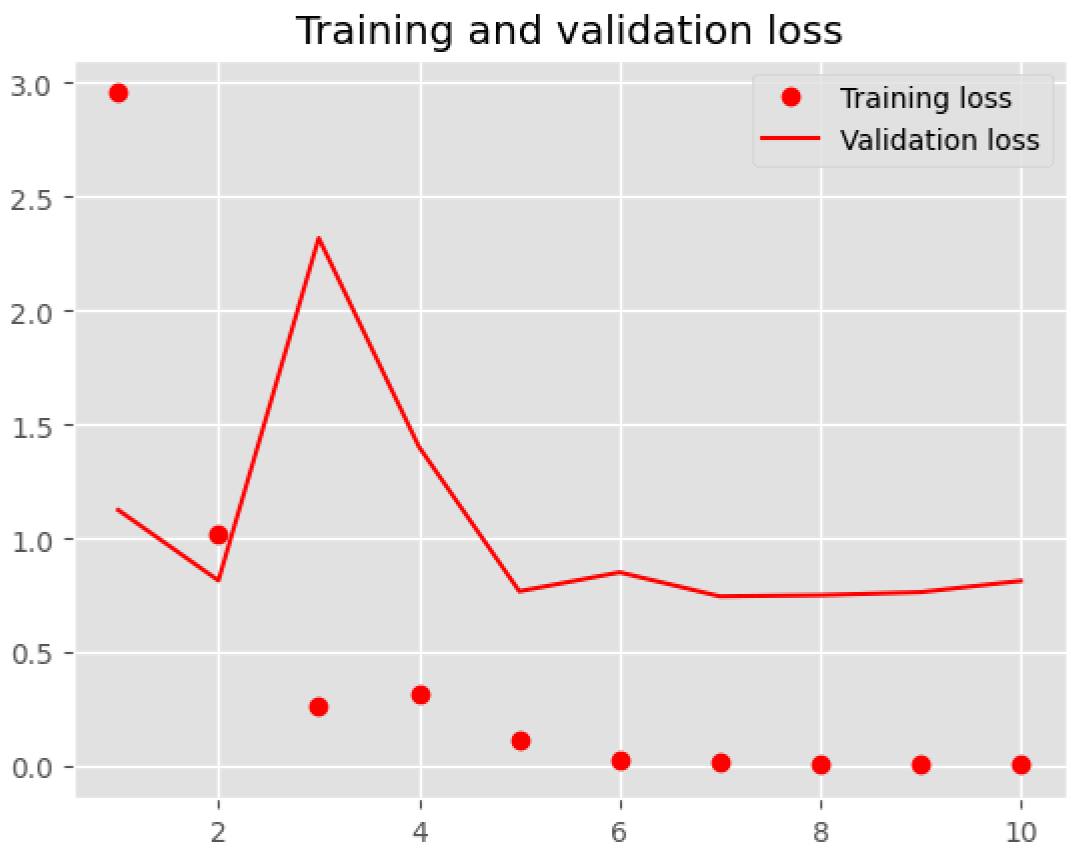
In [96]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
Epoch 1/10
4/4 [==============================] - 2s 336ms/step - loss: 2.9563 - acc: 0.5300
- val_loss: 1.1263 - val_acc: 0.4966
Epoch 2/10
4/4 [==============================] - 1s 438ms/step - loss: 1.0247 - acc: 0.5600
- val_loss: 0.8164 - val_acc: 0.5337
Epoch 3/10
4/4 [==============================] - 1s 269ms/step - loss: 0.2684 - acc: 0.8800
- val_loss: 2.3196 - val_acc: 0.5039
Epoch 4/10
4/4 [==============================] - 1s 438ms/step - loss: 0.3147 - acc: 0.8400
- val_loss: 1.4028 - val_acc: 0.5015
Epoch 5/10
4/4 [==============================] - 1s 436ms/step - loss: 0.1191 - acc: 0.9500
- val_loss: 0.7692 - val_acc: 0.5721
Epoch 6/10
4/4 [==============================] - 1s 296ms/step - loss: 0.0296 - acc: 1.0000
- val_loss: 0.8525 - val_acc: 0.5420
Epoch 7/10
4/4 [==============================] - 1s 437ms/step - loss: 0.0224 - acc: 1.0000
- val_loss: 0.7470 - val_acc: 0.5762
Epoch 8/10
4/4 [==============================] - 1s 438ms/step - loss: 0.0119 - acc: 1.0000
- val_loss: 0.7521 - val_acc: 0.5754
Epoch 9/10
4/4 [==============================] - 1s 422ms/step - loss: 0.0096 - acc: 1.0000
- val_loss: 0.7653 - val_acc: 0.5695
Epoch 10/10
4/4 [==============================] - 3s 872ms/step - loss: 0.0070 - acc: 1.0000
- val_loss: 0.8149 - val_acc: 0.5570
```

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

In [97]:
```python
import matplotlib.pyplot as plt

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

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

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

plt.figure()

plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, valid_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





In [98]:

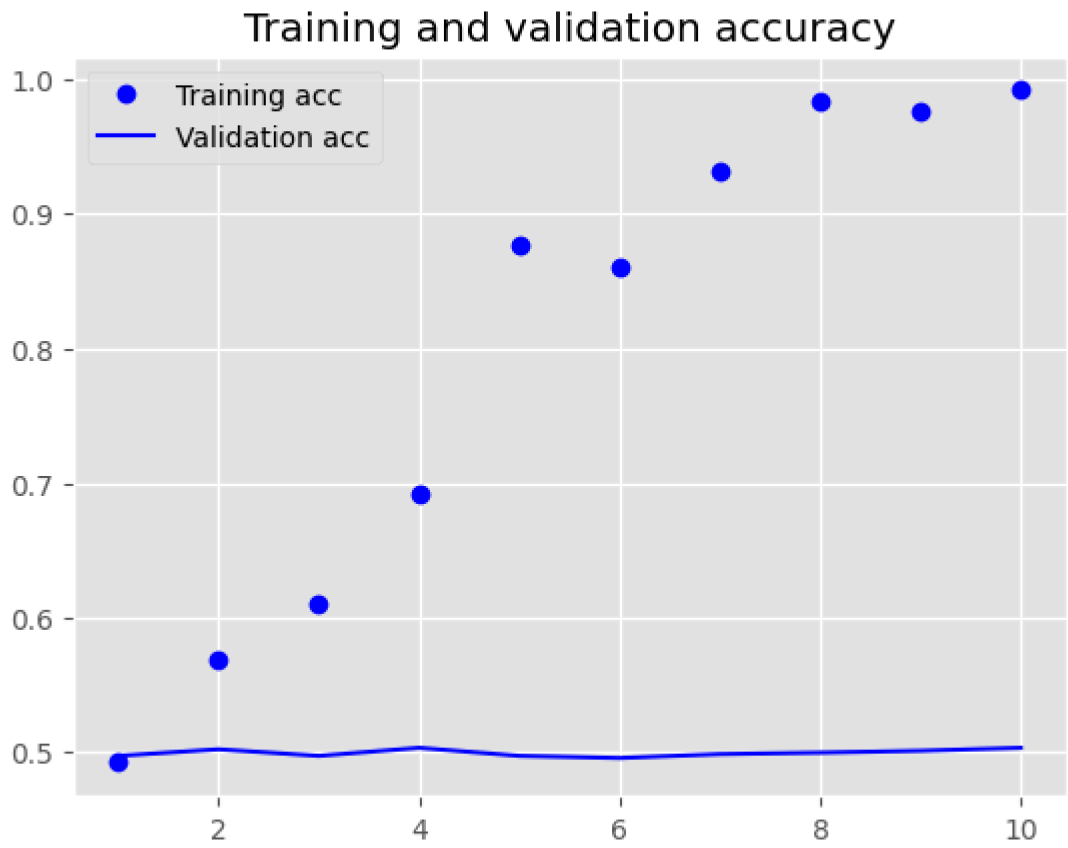
```
test_loss, T_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

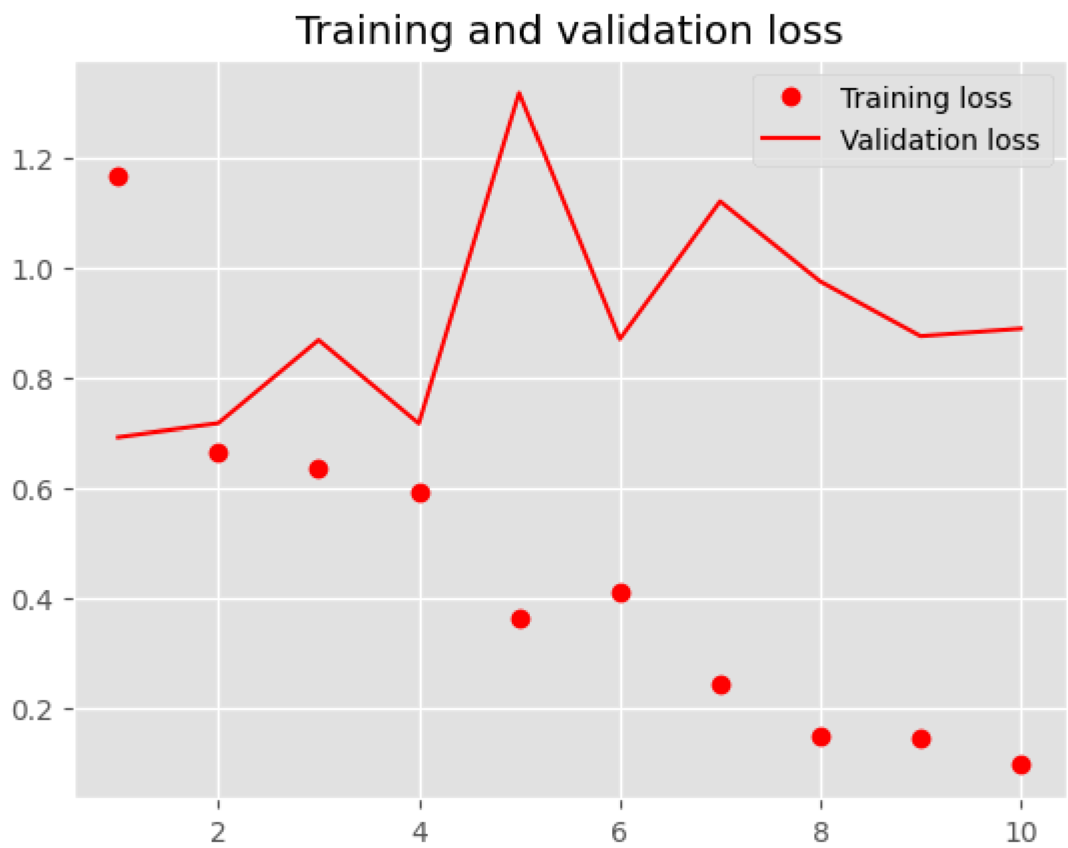
```
782/782 [==============================] - 4s 5ms/step - loss: 0.9387 - acc: 0.495
6
Test loss: 0.9387204647064209
Test accuracy: 0.4955599904060364
```

In [99]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

max_len = 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=max_len)

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 (1000
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']
```

```
valid_accuracy = history.history['val_acc']
loss = history.history['loss']
valid_loss = history.history['val_loss']

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

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

plt.figure()

plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, valid_loss, 'r', 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_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_50 (Embedding)    (None, 150, 100)          1000000   
                                                                 
 flatten_31 (Flatten)        (None, 15000)             0         
                                                                 
 dense_43 (Dense)            (None, 32)                480032    
                                                                 
 dense_44 (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 [==============================] - 4s 201ms/step - loss: 1.1693 - acc: 0.492
0 - val_loss: 0.6932 - val_acc: 0.4970
Epoch 2/10
16/16 [==============================] - 3s 184ms/step - loss: 0.6655 - acc: 0.568
0 - val_loss: 0.7181 - val_acc: 0.5019
Epoch 3/10
16/16 [==============================] - 2s 113ms/step - loss: 0.6366 - acc: 0.610
0 - val_loss: 0.8693 - val_acc: 0.4970
Epoch 4/10
16/16 [==============================] - 2s 106ms/step - loss: 0.5914 - acc: 0.692
0 - val_loss: 0.7174 - val_acc: 0.5030
Epoch 5/10
16/16 [==============================] - 2s 117ms/step - loss: 0.3656 - acc: 0.876
0 - val_loss: 1.3191 - val_acc: 0.4970
Epoch 6/10
16/16 [==============================] - 3s 185ms/step - loss: 0.4124 - acc: 0.860
0 - val_loss: 0.8710 - val_acc: 0.4954
Epoch 7/10
16/16 [==============================] - 1s 93ms/step - loss: 0.2432 - acc: 0.9320
- val_loss: 1.1217 - val_acc: 0.4983
Epoch 8/10
16/16 [==============================] - 1s 94ms/step - loss: 0.1482 - acc: 0.9840
- val_loss: 0.9758 - val_acc: 0.4994
Epoch 9/10
16/16 [==============================] - 1s 69ms/step - loss: 0.1471 - acc: 0.9760
- val_loss: 0.8764 - val_acc: 0.5008
Epoch 10/10
16/16 [==============================] - 1s 93ms/step - loss: 0.0967 - acc: 0.9920
- val_loss: 0.8901 - val_acc: 0.5030
```





In [100…

```
test_loss, T_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

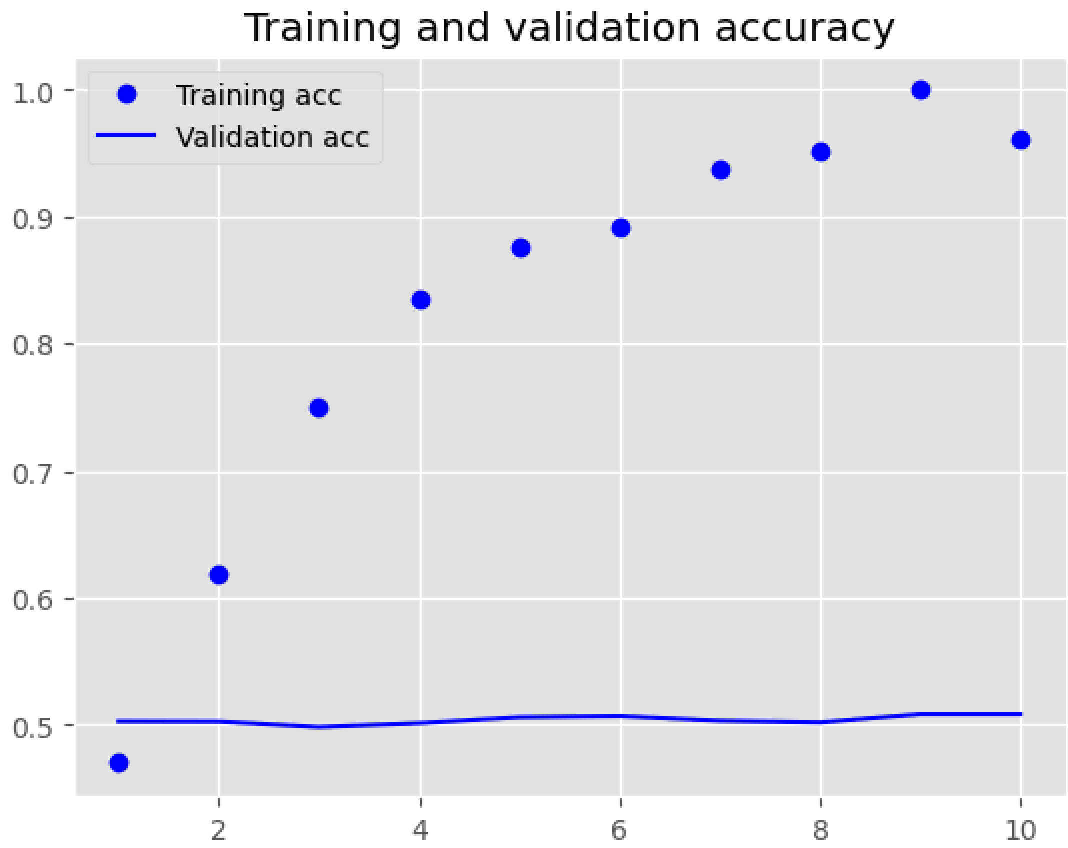
```
782/782 [==============================] - 3s 3ms/step - loss: 0.9058 - acc: 0.502
0
Test loss: 0.9057888388633728
Test accuracy: 0.5019999742507935
```

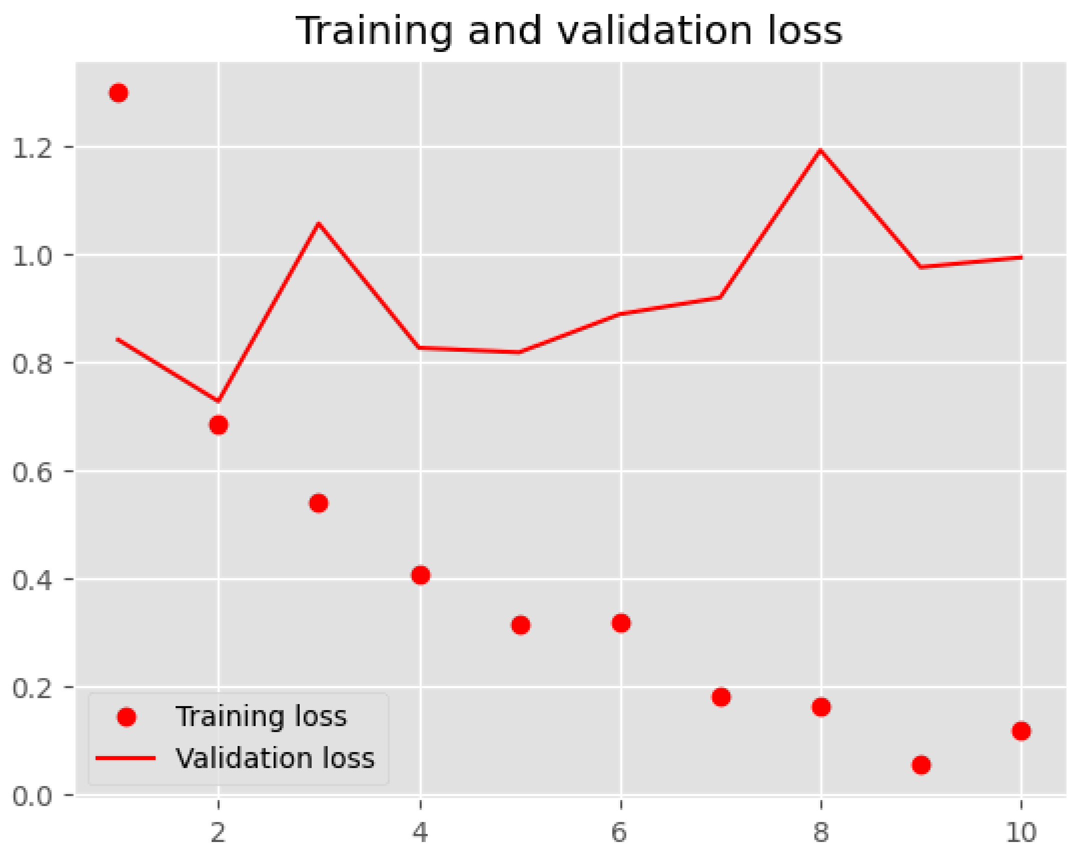
In [101…

```
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']
valid_accuracy = history.history['val_acc']
loss = history.history['loss']
valid_loss = history.history['val_loss']

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

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

plt.figure()

plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, valid_loss, 'r', 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_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_51 (Embedding)    (None, 150, 100)          1000000

 flatten_32 (Flatten)        (None, 15000)             0

 dense_45 (Dense)            (None, 32)                480032

 dense_46 (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 [==============================] - 3s 59ms/step - loss: 1.2978 - acc: 0.4700
- val_loss: 0.8410 - val_acc: 0.5029
Epoch 2/10
32/32 [==============================] - 1s 34ms/step - loss: 0.6853 - acc: 0.6190
- val_loss: 0.7270 - val_acc: 0.5027
Epoch 3/10
32/32 [==============================] - 1s 33ms/step - loss: 0.5394 - acc: 0.7500
- val_loss: 1.0562 - val_acc: 0.4984
Epoch 4/10
32/32 [==============================] - 2s 49ms/step - loss: 0.4095 - acc: 0.8350
- val_loss: 0.8259 - val_acc: 0.5014
Epoch 5/10
32/32 [==============================] - 1s 33ms/step - loss: 0.3147 - acc: 0.8760
- val_loss: 0.8175 - val_acc: 0.5061
Epoch 6/10
32/32 [==============================] - 2s 49ms/step - loss: 0.3199 - acc: 0.8920
- val_loss: 0.8882 - val_acc: 0.5070
Epoch 7/10
32/32 [==============================] - 2s 50ms/step - loss: 0.1826 - acc: 0.9370
- val_loss: 0.9188 - val_acc: 0.5033
Epoch 8/10
32/32 [==============================] - 1s 32ms/step - loss: 0.1627 - acc: 0.9520
- val_loss: 1.1918 - val_acc: 0.5020
Epoch 9/10
32/32 [==============================] - 1s 35ms/step - loss: 0.0548 - acc: 1.0000
- val_loss: 0.9752 - val_acc: 0.5085
Epoch 10/10
32/32 [==============================] - 3s 94ms/step - loss: 0.1192 - acc: 0.9620
- val_loss: 0.9932 - val_acc: 0.5085
```





In [102…

```
test_loss, T_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

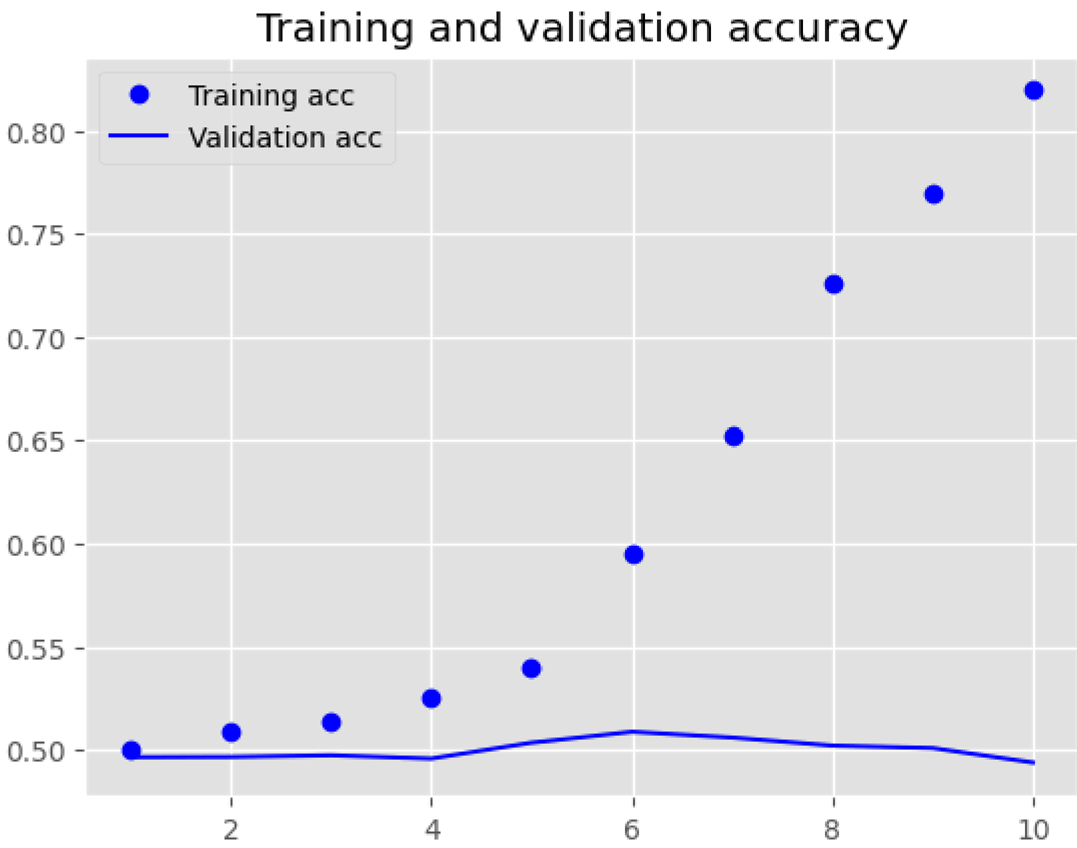
```
782/782 [==============================] - 4s 6ms/step - loss: 0.9900 - acc: 0.501
2
Test loss: 0.9899849891662598
Test accuracy: 0.5012400150299072
```

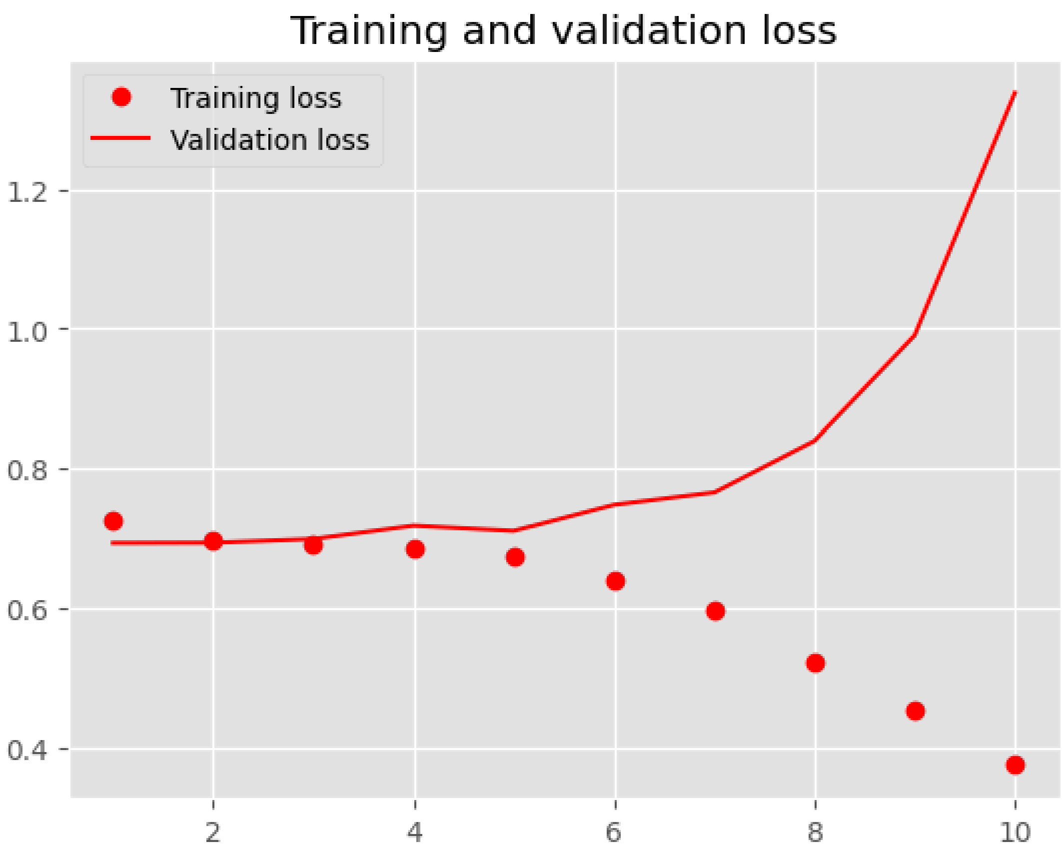
In [103…

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

max_len = 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 (1000
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']
valid_accuracy = history.history['val_acc']
```

```
loss = history.history['loss']
valid_loss = history.history['val_loss']

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

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

plt.figure()

plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, valid_loss, 'r', 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_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_52 (Embedding)    (None, 150, 100)          1000000

 flatten_33 (Flatten)        (None, 15000)             0

 dense_47 (Dense)            (None, 32)                480032

 dense_48 (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 [==============================] - 6s 16ms/step - loss: 0.7263 - acc: 0.50
01 - val_loss: 0.6932 - val_acc: 0.4964
Epoch 2/10
313/313 [==============================] - 6s 20ms/step - loss: 0.6967 - acc: 0.50
85 - val_loss: 0.6935 - val_acc: 0.4965
Epoch 3/10
313/313 [==============================] - 6s 20ms/step - loss: 0.6924 - acc: 0.51
40 - val_loss: 0.6990 - val_acc: 0.4973
Epoch 4/10
313/313 [==============================] - 4s 13ms/step - loss: 0.6858 - acc: 0.52
57 - val_loss: 0.7181 - val_acc: 0.4957
Epoch 5/10
313/313 [==============================] - 3s 10ms/step - loss: 0.6748 - acc: 0.53
95 - val_loss: 0.7106 - val_acc: 0.5034
Epoch 6/10
313/313 [==============================] - 4s 12ms/step - loss: 0.6406 - acc: 0.59
47 - val_loss: 0.7480 - val_acc: 0.5087
Epoch 7/10
313/313 [==============================] - 6s 18ms/step - loss: 0.5973 - acc: 0.65
27 - val_loss: 0.7657 - val_acc: 0.5059
Epoch 8/10
313/313 [==============================] - 6s 20ms/step - loss: 0.5214 - acc: 0.72
60 - val_loss: 0.8395 - val_acc: 0.5020
Epoch 9/10
313/313 [==============================] - 5s 17ms/step - loss: 0.4540 - acc: 0.76
95 - val_loss: 0.9904 - val_acc: 0.5008
Epoch 10/10
313/313 [==============================] - 4s 13ms/step - loss: 0.3744 - acc: 0.81
95 - val_loss: 1.3383 - val_acc: 0.4938
```





In [104…

```
test_loss, T_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', T_accuracy)
```

```
782/782 [==============================] - 2s 3ms/step - loss: 1.3057 - acc: 0.501
2
Test loss: 1.3056845664978027
Test accuracy: 0.5012000203132629
```