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
```

The IMDB dataset has labels for both positive and negative sentiments related to movie reviews.

Every review is converted into a set of word embeddings during the dataset's preparation, where each word is represented by a fixed-size vector.

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

# The Embedding layer requires a minimum of two inputs:
# The maximum word index plus one, or 1000, is the number of potential tokens.
# and the embeddings' dimensions, in this case 64.
embedd_lay = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing
from keras.utils import pad_sequences
```

A custom-embedding layer with a 100-size training sample

In [3]:
```
# The number of words that should be considered as features
num_of_fea = 10000
# Remove the text after this number of words(from the top max_features most common wor
length = 150

# Data loading to integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_of_fea)

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

# The integer lists are now transformed into a 2D integer tensor with the shape of {(s
x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)
from keras.models import Sequential
from keras.layers import Flatten, Dense

model1 = Sequential()
# In order to finally flatten the embedded inputs, the maximum length of the input to
model1.add(Embedding(10000, 8, input_length=length))
# After the Embedding layer, our activations have shape `(samples, maxlen, 8)`.
```

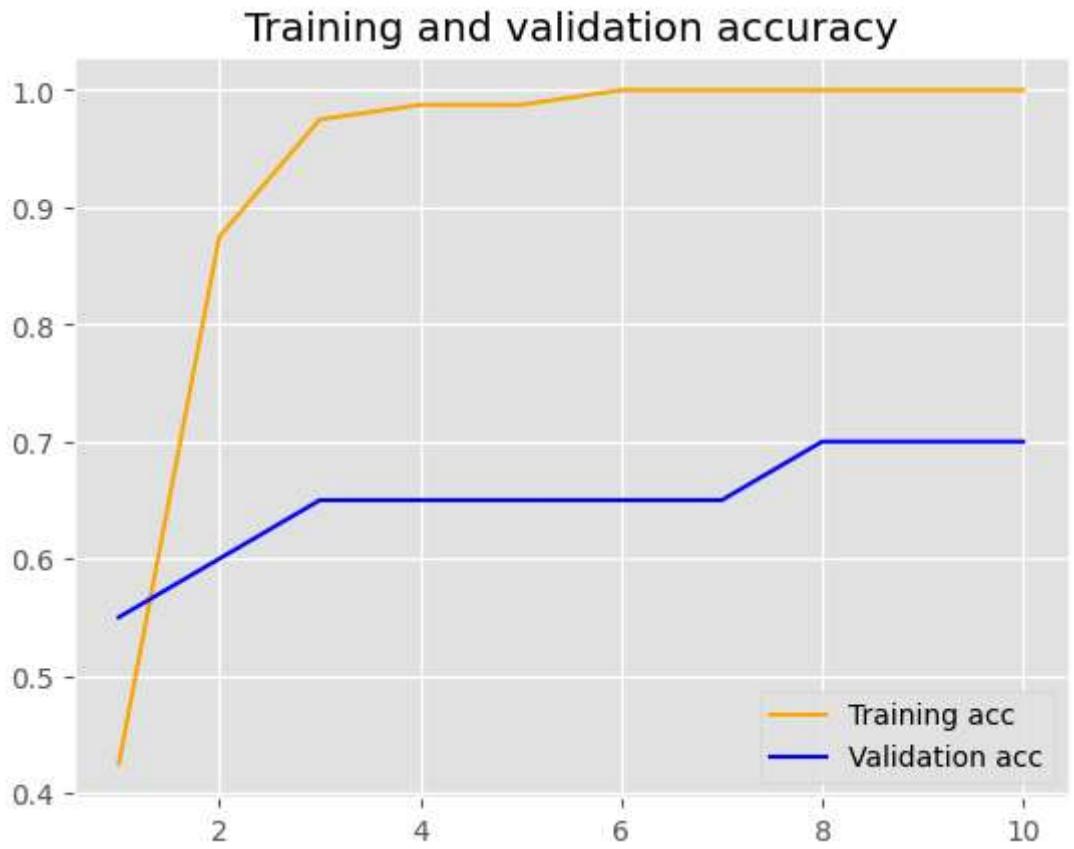
```
# We flatten the 3D tensor of embeddings into a 2D tensor of shape
#`(samples, maxlen * 8)`
model1.add(Flatten())

# We add the classifier on top
model1.add(Dense(1, activation='sigmoid'))
model1.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model1.summary()

history1 = model1.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

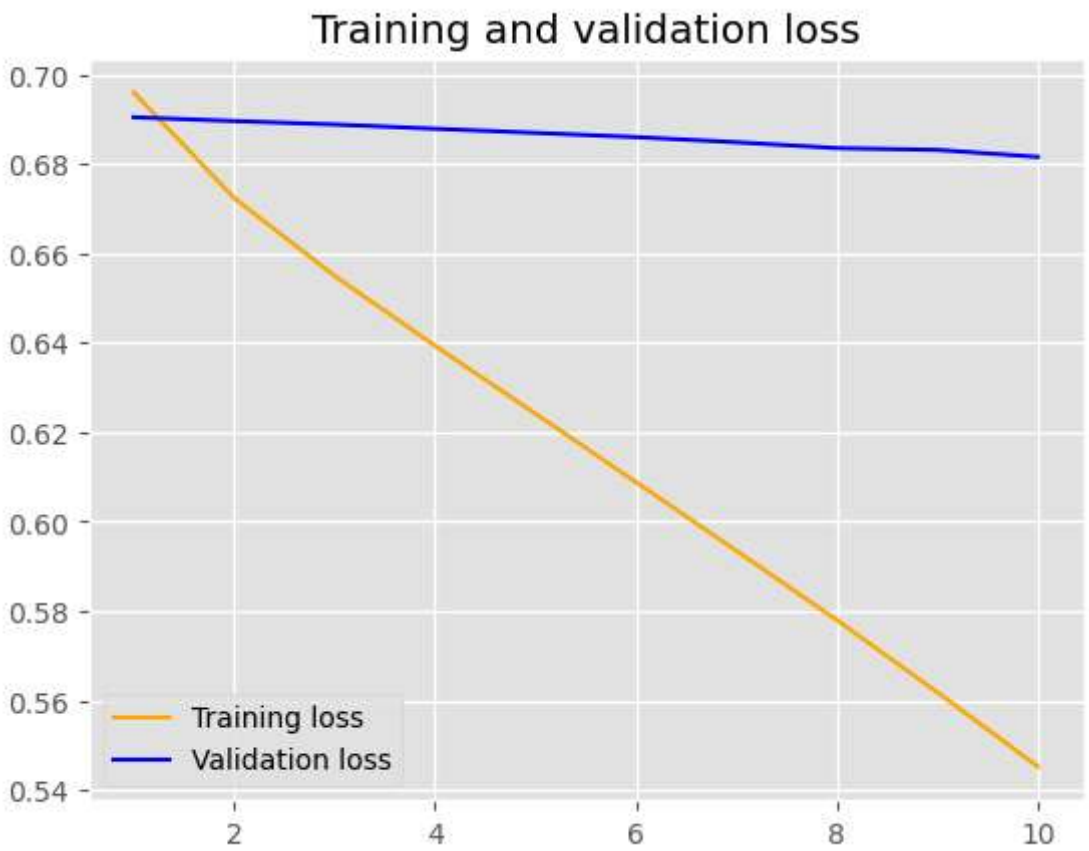
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imd
b.npz
17464789/17464789 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              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 210ms/step - loss: 0.6961 - acc: 0.4250 - v
al_loss: 0.6905 - val_acc: 0.5500
Epoch 2/10
3/3 [==============================] - 0s 132ms/step - loss: 0.6724 - acc: 0.8750 - v
al_loss: 0.6896 - val_acc: 0.6000
Epoch 3/10
3/3 [==============================] - 0s 134ms/step - loss: 0.6550 - acc: 0.9750 - v
al_loss: 0.6889 - val_acc: 0.6500
Epoch 4/10
3/3 [==============================] - 0s 132ms/step - loss: 0.6393 - acc: 0.9875 - v
al_loss: 0.6879 - val_acc: 0.6500
Epoch 5/10
3/3 [==============================] - 0s 133ms/step - loss: 0.6242 - acc: 0.9875 - v
al_loss: 0.6870 - val_acc: 0.6500
Epoch 6/10
3/3 [==============================] - 0s 132ms/step - loss: 0.6088 - acc: 1.0000 - v
al_loss: 0.6860 - val_acc: 0.6500
Epoch 7/10
3/3 [==============================] - 0s 124ms/step - loss: 0.5935 - acc: 1.0000 - v
al_loss: 0.6849 - val_acc: 0.6500
Epoch 8/10
3/3 [==============================] - 0s 126ms/step - loss: 0.5780 - acc: 1.0000 - v
al_loss: 0.6836 - val_acc: 0.7000
Epoch 9/10
3/3 [==============================] - 0s 121ms/step - loss: 0.5618 - acc: 1.0000 - v
al_loss: 0.6832 - val_acc: 0.7000
Epoch 10/10
3/3 [==============================] - 0s 123ms/step - loss: 0.5452 - acc: 1.0000 - v
al_loss: 0.6816 - val_acc: 0.7000
```

In [4]:

```python
import matplotlib.pyplot as plt

# Training accuracy
training_accu = history1.history["acc"]
# Validation accuracy
valid_accu = history1.history["val_acc"]
# Training loss
training_loss = history1.history["loss"]
# Validation loss
valid_loss = history1.history["val_loss"]
```

```
epochs = range(1, len(training_accu ) + 1)

plt.plot(epochs, training_accu , "orange", label = "Training acc")
plt.plot(epochs, valid_accu, "b", label = "Validation acc")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, training_loss, "orange", label = "Training loss")
plt.plot(epochs, valid_loss, "b", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```





In [5]:
```
test_loss, test_acc = model1.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.6938 - acc: 0.5012
Test loss: 0.6938053369522095
Test accuracy: 0.5012400150299072
```

A custom trained embedding layer with a 5000 training sample size

In [6]:
```
num_of_fea=10000
length=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_of_fea)

x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)

texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((x_train, x_test), axis=0)

x_train = x_train[:5000]
y_train = y_train[:5000]
```

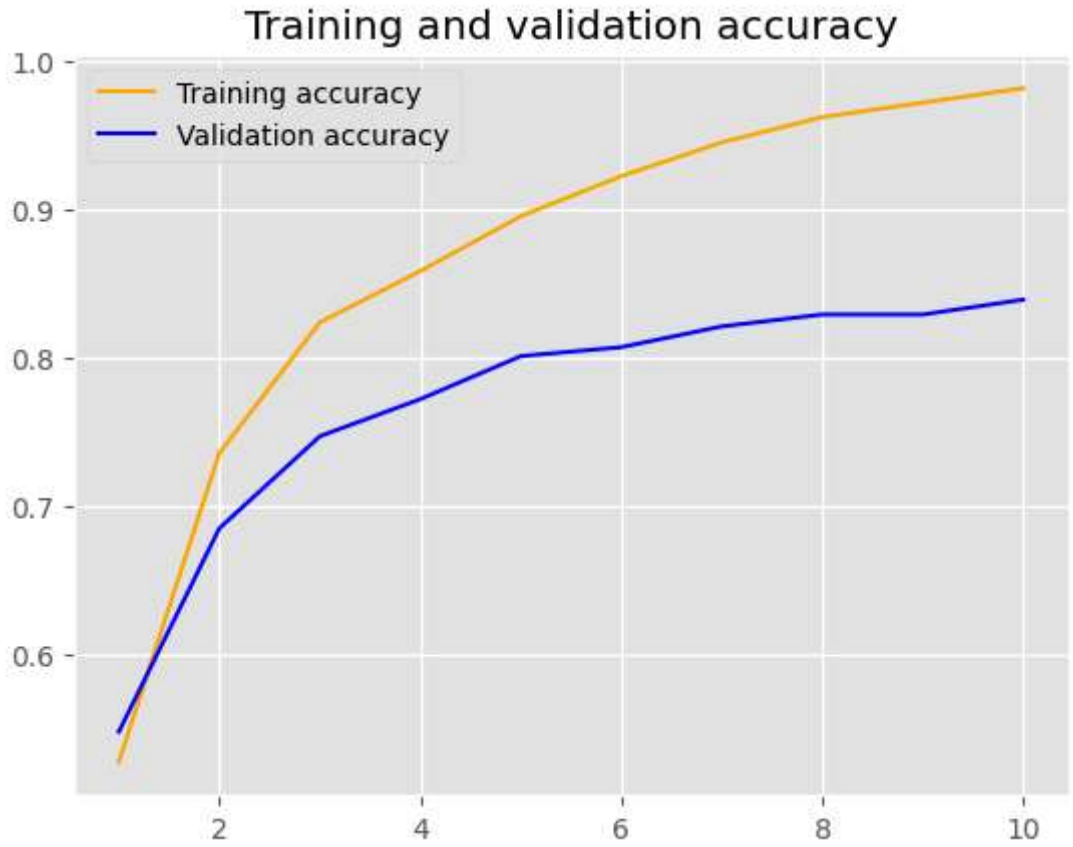
In [7]:
```
model2 = Sequential()
model2.add(Embedding(10000, 8, input_length=length))
model2.add(Flatten())
model2.add(Dense(1, activation='sigmoid'))
model2.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model2.summary()
history2 = model2.fit(x_train, y_train,
                      epochs=10,
```

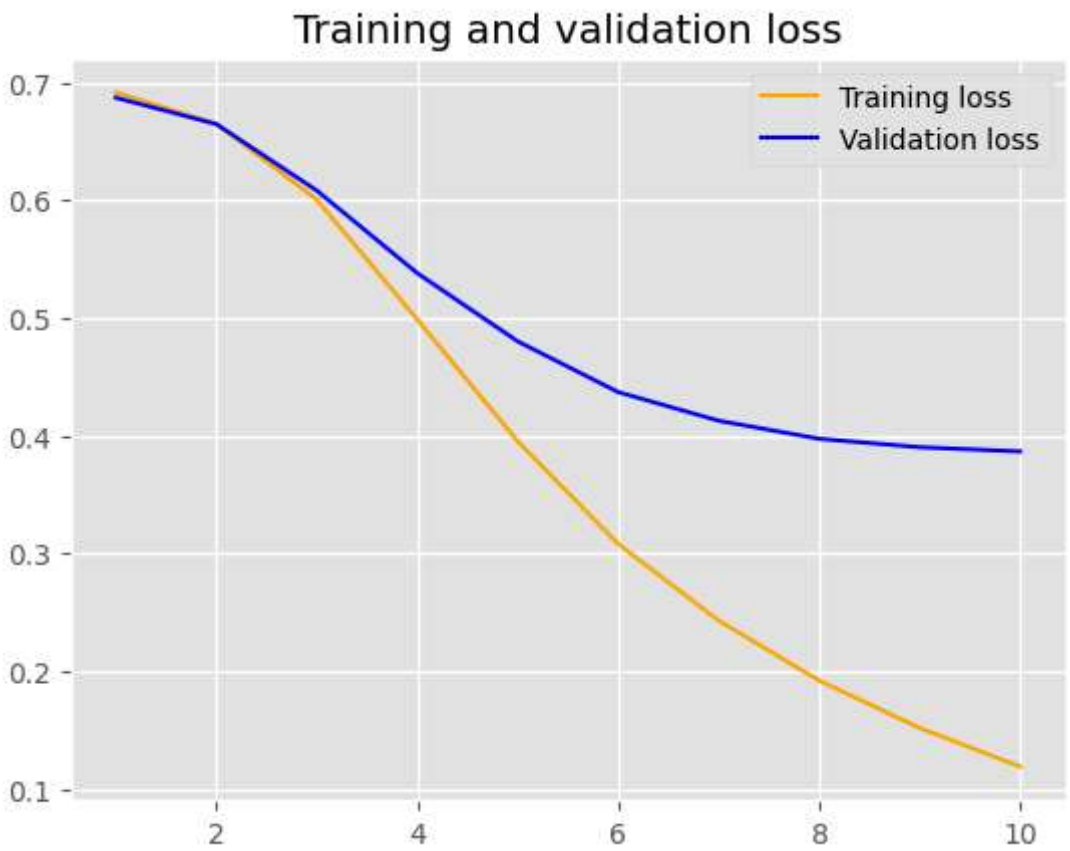
```
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (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
125/125 [==============================] - 11s 82ms/step - loss: 0.6917 - acc: 0.5278
- val_loss: 0.6871 - val_acc: 0.5480
Epoch 2/10
125/125 [==============================] - 6s 45ms/step - loss: 0.6649 - acc: 0.7358
- val_loss: 0.6647 - val_acc: 0.6850
Epoch 3/10
125/125 [==============================] - 3s 28ms/step - loss: 0.6006 - acc: 0.8238
- val_loss: 0.6082 - val_acc: 0.7470
Epoch 4/10
125/125 [==============================] - 2s 18ms/step - loss: 0.4987 - acc: 0.8583
- val_loss: 0.5381 - val_acc: 0.7720
Epoch 5/10
125/125 [==============================] - 2s 14ms/step - loss: 0.3953 - acc: 0.8953
- val_loss: 0.4802 - val_acc: 0.8010
Epoch 6/10
125/125 [==============================] - 1s 7ms/step - loss: 0.3081 - acc: 0.9222 -
val_loss: 0.4371 - val_acc: 0.8070
Epoch 7/10
125/125 [==============================] - 1s 9ms/step - loss: 0.2431 - acc: 0.9450 -
val_loss: 0.4128 - val_acc: 0.8210
Epoch 8/10
125/125 [==============================] - 1s 10ms/step - loss: 0.1922 - acc: 0.9620
- val_loss: 0.3975 - val_acc: 0.8290
Epoch 9/10
125/125 [==============================] - 1s 9ms/step - loss: 0.1519 - acc: 0.9718 -
val_loss: 0.3903 - val_acc: 0.8290
Epoch 10/10
125/125 [==============================] - 1s 8ms/step - loss: 0.1191 - acc: 0.9815 -
val_loss: 0.3867 - val_acc: 0.8390
```

In [8]:
```
training_accu2 = history2.history['acc']
valid_accur2 = history2.history['val_acc']
training_loss2 = history2.history['loss']
valid_loss2 = history2.history['val_loss']

epochs = range(1, len(training_accu2) + 1)

plt.plot(epochs, training_accu2, 'orange', label='Training accuracy')
plt.plot(epochs, valid_accur2, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
```

```
plt.figure()

plt.plot(epochs, training_loss2, 'orange', label='Training loss')
plt.plot(epochs, valid_loss2, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





In [9]:
```
test_loss2, test_accuracy2 = model2.evaluate(x_test, y_test)
print('Test loss:', test_loss2)
print('Test accuracy:', test_accuracy2)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3758 - acc: 0.8284
Test loss: 0.3757614493370056
Test accuracy: 0.8284000158309937
```

A custom trained embedding layer with a 1000 training sample size

In [10]:
```
num_of_fea=10000
length=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_of_fea)

x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)

texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((x_train, x_test), axis=0)

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

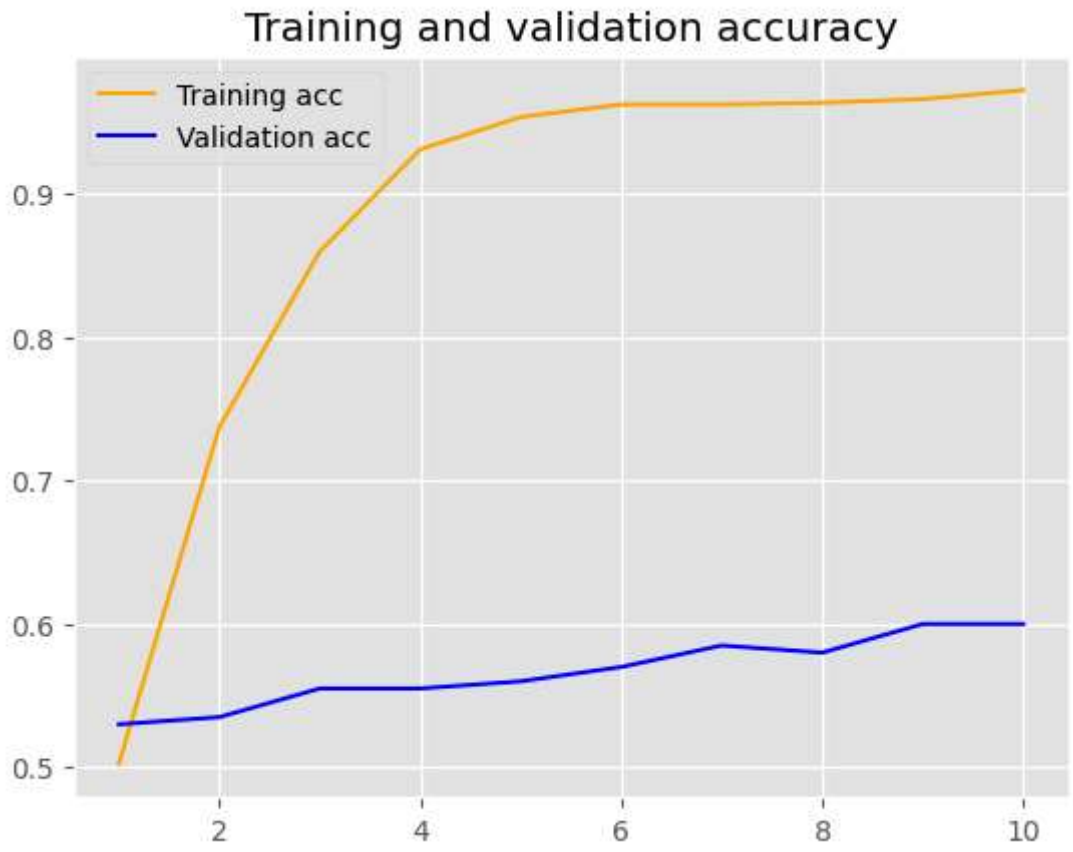
In [11]:
```
model3 = Sequential()
model3.add(Embedding(10000, 8, input_length=length))
model3.add(Flatten())
model3.add(Dense(1, activation='sigmoid'))
model3.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model3.summary()
history3 = model3.fit(x_train, y_train,
                      epochs=10,
```

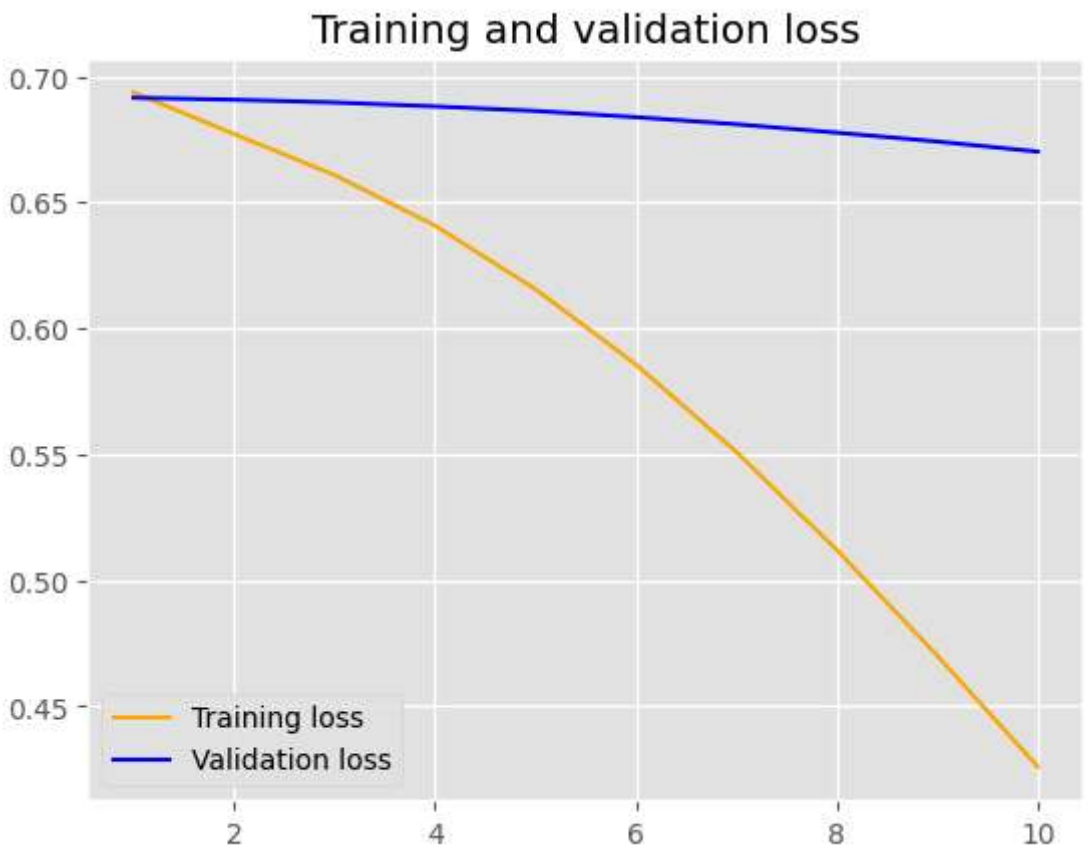
```
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (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 [==============================] - 2s 81ms/step - loss: 0.6937 - acc: 0.5025 -
val_loss: 0.6915 - val_acc: 0.5300
Epoch 2/10
25/25 [==============================] - 2s 65ms/step - loss: 0.6771 - acc: 0.7375 -
val_loss: 0.6907 - val_acc: 0.5350
Epoch 3/10
25/25 [==============================] - 1s 51ms/step - loss: 0.6609 - acc: 0.8600 -
val_loss: 0.6896 - val_acc: 0.5550
Epoch 4/10
25/25 [==============================] - 1s 46ms/step - loss: 0.6408 - acc: 0.9312 -
val_loss: 0.6881 - val_acc: 0.5550
Epoch 5/10
25/25 [==============================] - 1s 34ms/step - loss: 0.6156 - acc: 0.9538 -
val_loss: 0.6863 - val_acc: 0.5600
Epoch 6/10
25/25 [==============================] - 1s 47ms/step - loss: 0.5857 - acc: 0.9625 -
val_loss: 0.6838 - val_acc: 0.5700
Epoch 7/10
25/25 [==============================] - 1s 29ms/step - loss: 0.5510 - acc: 0.9625 -
val_loss: 0.6810 - val_acc: 0.5850
Epoch 8/10
25/25 [==============================] - 1s 36ms/step - loss: 0.5121 - acc: 0.9638 -
val_loss: 0.6777 - val_acc: 0.5800
Epoch 9/10
25/25 [==============================] - 1s 27ms/step - loss: 0.4701 - acc: 0.9663 -
val_loss: 0.6741 - val_acc: 0.6000
Epoch 10/10
25/25 [==============================] - 1s 34ms/step - loss: 0.4260 - acc: 0.9725 -
val_loss: 0.6701 - val_acc: 0.6000
```

In [12]:
```
training_accu3 = history3.history["acc"]
valid_accu3 = history3.history["val_acc"]
training_loss3 = history3.history["loss"]
valid_loss3 = history3.history["val_loss"]

epochs = range(1, len(training_accu3) + 1)

plt.plot(epochs, training_accu3, "orange", label = "Training acc")
plt.plot(epochs, valid_accu3, "b", label = "Validation acc")
plt.title("Training and validation accuracy")
plt.legend()
```

```
plt.figure()

plt.plot(epochs, training_loss3, "orange", label = "Training loss")
plt.plot(epochs, valid_loss3, "b", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```





In [13]:
```
test_loss3, test_accuracy3 = model3.evaluate(x_test, y_test)
print('Test loss:', test_loss3)
print('Test accuracy:', test_accuracy3)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.6815 - acc: 0.5641
Test loss: 0.6815415620803833
Test accuracy: 0.564079999923706
```

A custom trained embedding layer with 10,000 training samples

In [14]:
```
num_of_fea=10000
length=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_of_fea)

x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)

texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((x_train, x_test), axis=0)

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

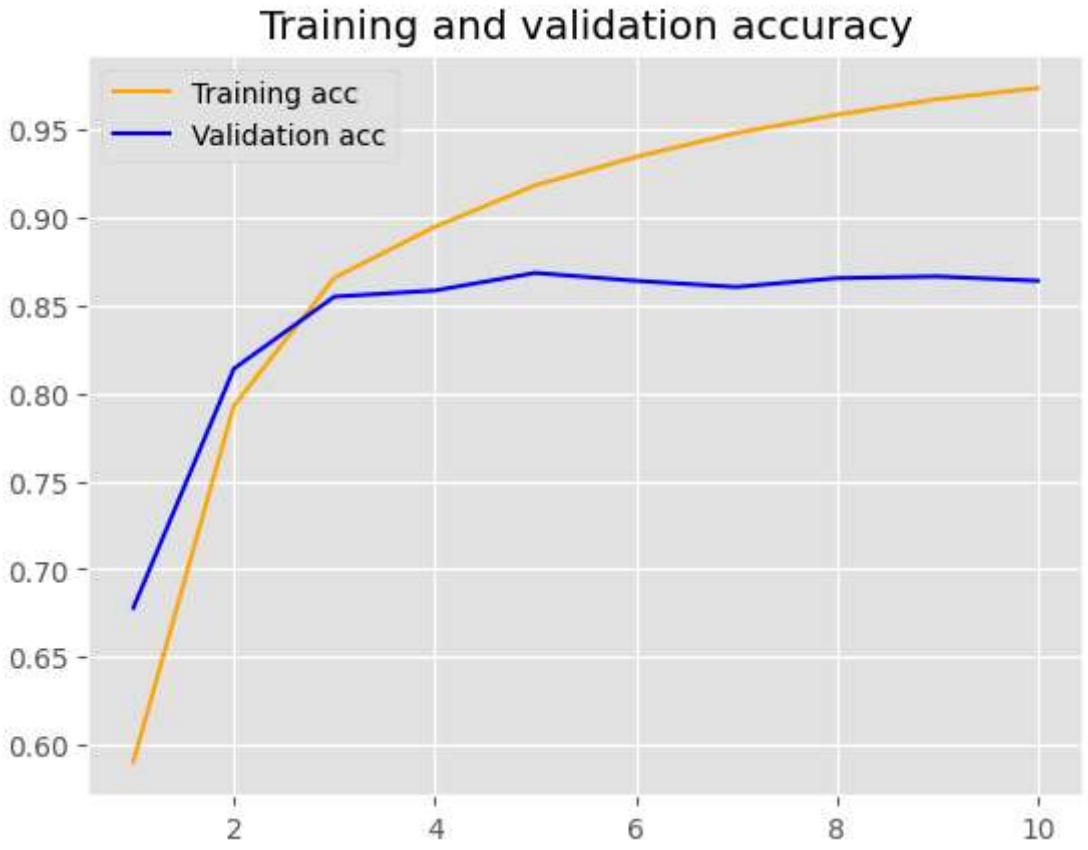
In [15]:
```
model4 = Sequential()
model4.add(Embedding(10000, 8, input_length=length))
model4.add(Flatten())
model4.add(Dense(1, activation='sigmoid'))
model4.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model4.summary()
history4 = model4.fit(x_train, y_train,
                      epochs=10,
```

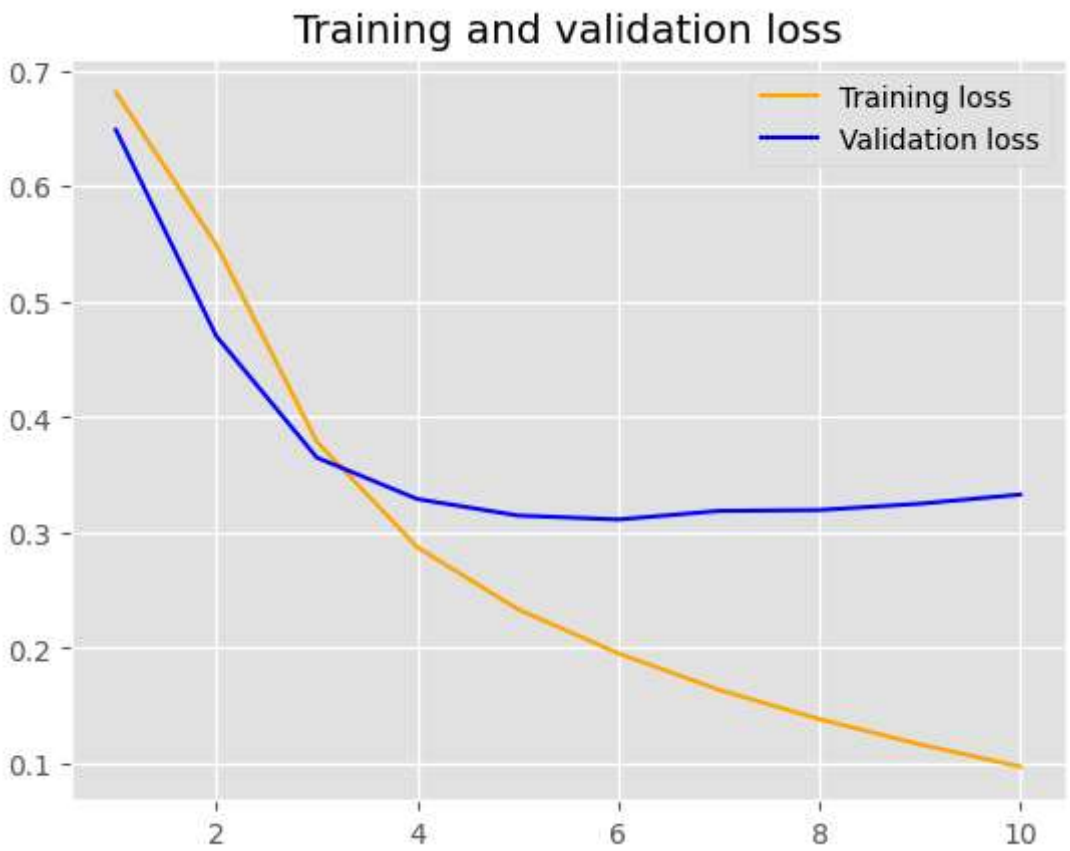
```
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (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 [==============================] - 11s 43ms/step - loss: 0.6814 - acc: 0.5904
- val_loss: 0.6489 - val_acc: 0.6780
Epoch 2/10
250/250 [==============================] - 4s 15ms/step - loss: 0.5493 - acc: 0.7930
- val_loss: 0.4702 - val_acc: 0.8140
Epoch 3/10
250/250 [==============================] - 2s 9ms/step - loss: 0.3789 - acc: 0.8658 -
val_loss: 0.3650 - val_acc: 0.8550
Epoch 4/10
250/250 [==============================] - 2s 9ms/step - loss: 0.2874 - acc: 0.8947 -
val_loss: 0.3291 - val_acc: 0.8585
Epoch 5/10
250/250 [==============================] - 1s 6ms/step - loss: 0.2338 - acc: 0.9184 -
val_loss: 0.3148 - val_acc: 0.8685
Epoch 6/10
250/250 [==============================] - 2s 6ms/step - loss: 0.1955 - acc: 0.9345 -
val_loss: 0.3114 - val_acc: 0.8640
Epoch 7/10
250/250 [==============================] - 1s 5ms/step - loss: 0.1640 - acc: 0.9481 -
val_loss: 0.3188 - val_acc: 0.8605
Epoch 8/10
250/250 [==============================] - 1s 4ms/step - loss: 0.1385 - acc: 0.9585 -
val_loss: 0.3196 - val_acc: 0.8655
Epoch 9/10
250/250 [==============================] - 1s 4ms/step - loss: 0.1167 - acc: 0.9674 -
val_loss: 0.3251 - val_acc: 0.8665
Epoch 10/10
250/250 [==============================] - 1s 5ms/step - loss: 0.0974 - acc: 0.9736 -
val_loss: 0.3330 - val_acc: 0.8640
```

In [16]:
```
training_accu4 = history4.history["acc"]
valid_accu4 = history4.history["val_acc"]
training_loss4 = history4.history["loss"]
valid_loss4 = history4.history["val_loss"]

epochs = range(1, len(training_accu4) + 1)

plt.plot(epochs, training_accu4, "orange", label = "Training acc")
plt.plot(epochs, valid_accu4, "b", label = "Validation acc")
plt.title("Training and validation accuracy")
plt.legend()
```

```
plt.figure()

plt.plot(epochs, training_loss4, "orange", label = "Training loss")
plt.plot(epochs, valid_loss4, "b", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```





In [17]:
```
test_loss4, test_accuracy4 = model4.evaluate(x_test, y_test)
print('Test loss:', test_loss4)
print('Test accuracy:', test_accuracy4)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3362 - acc: 0.8589
Test loss: 0.336191326379776
Test accuracy: 0.8588799834251404
```

In [18]:
```
!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  5237k      0  0:00:15  0:00:15 --:--:-- 8387k
```

In [19]:
```
import os
import shutil

imdb = 'aclImdb'
training = os.path.join(imdb, 'train')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(training, 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)
```

Making Use of Trained Word Embeds Use pretrained word embeddings if there is insufficient training data to obtain word embeddings linked with the desired solution.

Tokenizing the data

In [20]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

length2 = 150 # cut off review after 150 words
training_data = 100 # Training sample 100
validation_data = 10000 # Validation sample 10000
words = 10000 # Considers only the top 10000 words in the dataset

tokenizer1 = Tokenizer(num_words=words)
tokenizer1.fit_on_texts(texts)
sequences = tokenizer1.texts_to_sequences(texts)
word_index = tokenizer1.word_index
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=length2)

labels = np.asarray(labels)
print("Shape of data tensor:", data.shape)
print("Shape of label tensor:", labels.shape)
# Splits data into training and validation set, but shuffles is, since samples are ord
# all negatives first, then all positive
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_data] # (200, 100)
y_train = labels[:training_data] # shape (200,)
x_validation = data[training_data:training_data+validation_data] # shape (10000, 100)
y_validation = labels[training_data:training_data+validation_data] # shape (10000,)
```

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

Installing and setting up the GloVe word embedding

In [21]:
```
import numpy as np
import requests
from io import BytesIO
import zipfile

glove_url = 'https://nlp.stanford.edu/data/glove.6B.zip'  # URL to download GloVe embe
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.
```

Using Gigaword 5 and Wikipedia data, we trained the 6B version of the GloVe model, which comprises 400,000 words and 6 billion tokens.

Preparing the GloVe word embeddings matrix

pretrained word embedding layer with training sample size = 100

In [22]:
```
emb_dim = 100

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

In [23]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(words, emb_dim , input_length=length2))
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_5 (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 [24]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

In the Embeddig layer, pretrained word embedding is loaded. To guarantee that the Embedding layer cannot be trained, set this to False before executing it. Setting trainable = True will allow the optimization procedure to modify the word embedding settings. In order to keep students from forgetting what they already "know," it is essential to avoid updating pretrained parts while they are still receiving instruction.

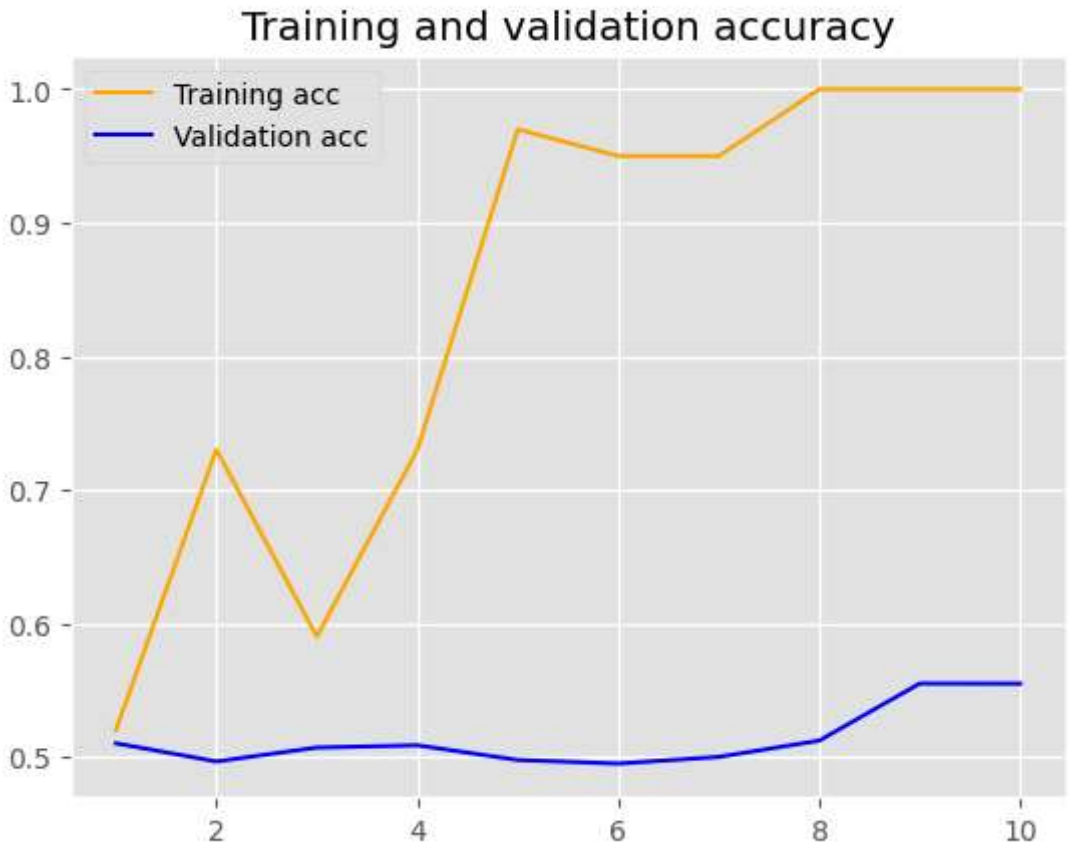
In [25]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_validation, y_validation))
model.save_weights('pre_trained_glove_model.h5')
```

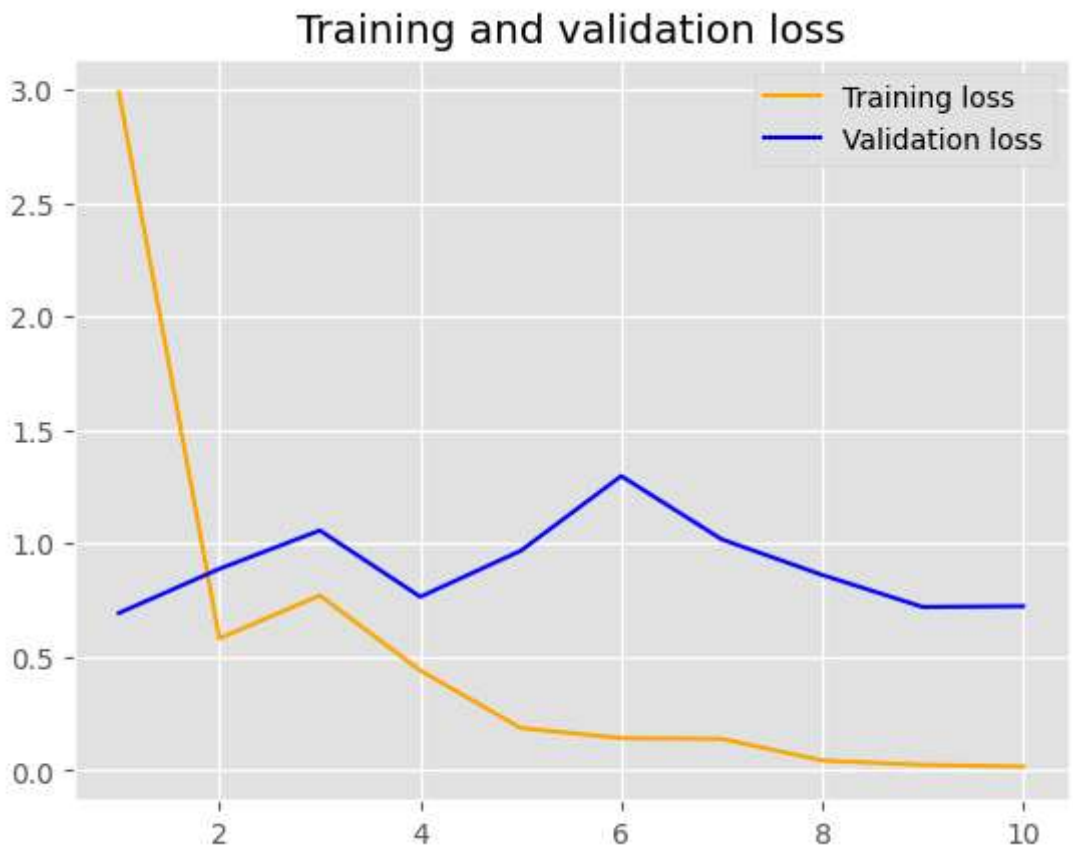
```
Epoch 1/10
4/4 [==============================] - 2s 242ms/step - loss: 2.9865 - acc: 0.5200 - v
al_loss: 0.6930 - val_acc: 0.5101
Epoch 2/10
4/4 [==============================] - 1s 193ms/step - loss: 0.5812 - acc: 0.7300 - v
al_loss: 0.8878 - val_acc: 0.4965
Epoch 3/10
4/4 [==============================] - 1s 187ms/step - loss: 0.7709 - acc: 0.5900 - v
al_loss: 1.0571 - val_acc: 0.5069
Epoch 4/10
4/4 [==============================] - 1s 191ms/step - loss: 0.4404 - acc: 0.7300 - v
al_loss: 0.7648 - val_acc: 0.5086
Epoch 5/10
4/4 [==============================] - 1s 192ms/step - loss: 0.1869 - acc: 0.9700 - v
al_loss: 0.9676 - val_acc: 0.4976
Epoch 6/10
4/4 [==============================] - 1s 191ms/step - loss: 0.1423 - acc: 0.9500 - v
al_loss: 1.2966 - val_acc: 0.4949
Epoch 7/10
4/4 [==============================] - 1s 189ms/step - loss: 0.1392 - acc: 0.9500 - v
al_loss: 1.0178 - val_acc: 0.4999
Epoch 8/10
4/4 [==============================] - 1s 191ms/step - loss: 0.0441 - acc: 1.0000 - v
al_loss: 0.8610 - val_acc: 0.5121
Epoch 9/10
4/4 [==============================] - 1s 188ms/step - loss: 0.0247 - acc: 1.0000 - v
al_loss: 0.7201 - val_acc: 0.5549
Epoch 10/10
4/4 [==============================] - 1s 194ms/step - loss: 0.0171 - acc: 1.0000 - v
al_loss: 0.7234 - val_acc: 0.5548
```

Given the limited amount of training data, this rapid overfitting of the model is to be expected. The wide variance in validation accuracy can be explained by the same mechanism.

In [26]:
```
import matplotlib.pyplot as plt

training_accu = history.history['acc']
valid_accu = history.history['val_acc']
```

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

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

plt.plot(epochs, training_accu, 'orange', label='Training acc')
plt.plot(epochs, valid_accu, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, training_loss, 'orange', label='Training loss')
plt.plot(epochs, valid_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





In [27]:
```
test_loss, test_accuracy= model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

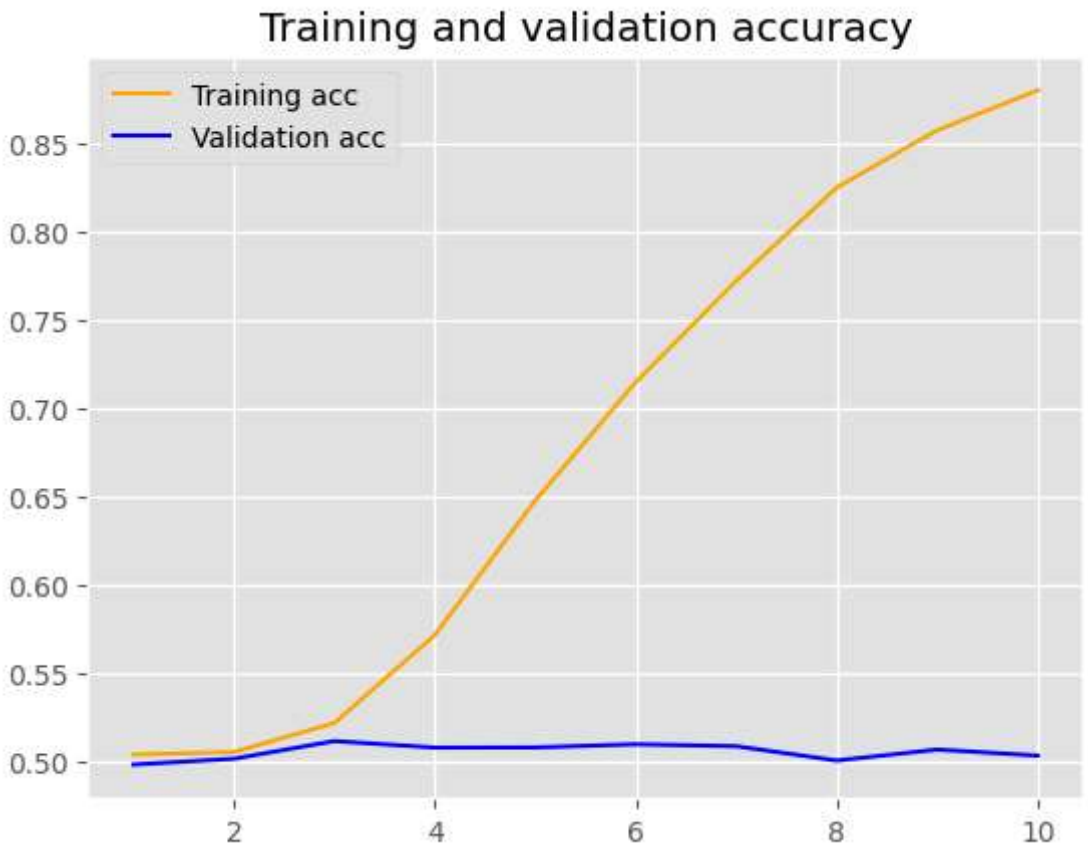
```
782/782 [==============================] - 2s 2ms/step - loss: 0.7788 - acc: 0.4980
Test loss: 0.778832733631134
Test accuracy: 0.49803999066352844
```

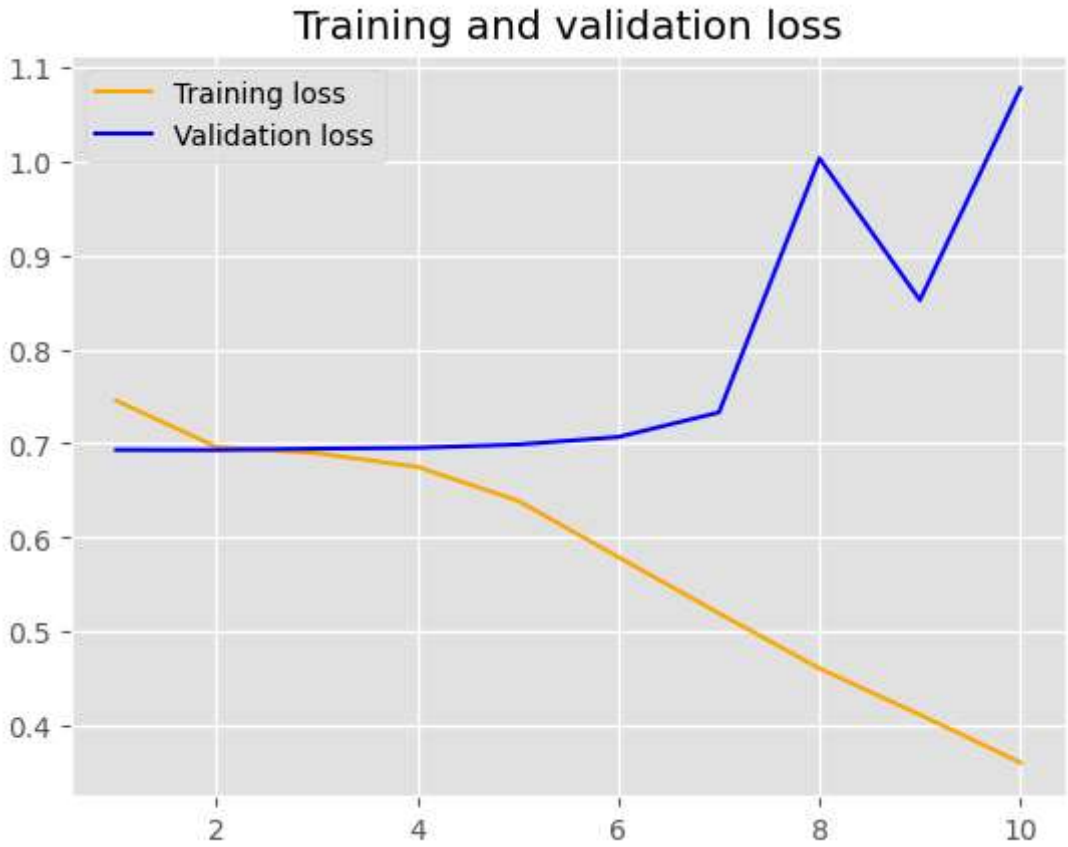
pretrained word embedding layer with training sample size = 5000

In [28]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

length2 = 150
training_data = 5000 # Training sample is 5000
validation_data = 10000
words = 10000

tokenizer2 = Tokenizer(num_words=words)
tokenizer2.fit_on_texts(texts)
sequences = tokenizer2.texts_to_sequences(texts)
word_index = tokenizer2.word_index
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=length2)

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

indices = np.arange(data.shape[0])
```

```
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_data]
y_train = labels[:training_data]
x_validation = data[training_data:training_data+validation_data]
y_validation = labels[training_data:training_data+validation_data]
embedd_di = 100

embedd_matrix = np.zeros((words, embedd_di))
for word, i in word_index.items():
    embedd_vector = embeddings_index.get(word)
    if i < words:
        if embedd_vector is not None:

            embedd_matrix[i] = embedd_vector

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

model11 = Sequential()
model11.add(Embedding(words, embedd_di, input_length=length2))
model11.add(Flatten())
model11.add(Dense(32, activation='relu'))
model11.add(Dense(1, activation='sigmoid'))
model11.summary()

model11.layers[0].set_weights([embedding_matrix])
model11.layers[0].trainable = False
model11.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history11 = model11.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_validation, y_validation))
model11.save_weights('pre_trained_glove_model.h5')
import matplotlib.pyplot as plt

accuracy11 = history11.history['acc']
valid_acc11 = history11.history['val_acc']
train_loss11 = history11.history['loss']
valid_loss11 = history11.history['val_loss']

epochs = range(1, len(accuracy11) + 1)

plt.plot(epochs, accuracy11, 'orange', label='Training acc')
plt.plot(epochs, valid_acc11, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, train_loss11, 'orange', label='Training loss')
plt.plot(epochs, valid_loss11, '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_6 (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
157/157 [==============================] - 2s 8ms/step - loss: 0.7456 - acc: 0.5038 -
val_loss: 0.6932 - val_acc: 0.4981
Epoch 2/10
157/157 [==============================] - 1s 7ms/step - loss: 0.6962 - acc: 0.5052 -
val_loss: 0.6931 - val_acc: 0.5014
Epoch 3/10
157/157 [==============================] - 1s 7ms/step - loss: 0.6897 - acc: 0.5216 -
val_loss: 0.6945 - val_acc: 0.5114
Epoch 4/10
157/157 [==============================] - 1s 7ms/step - loss: 0.6751 - acc: 0.5714 -
val_loss: 0.6954 - val_acc: 0.5077
Epoch 5/10
157/157 [==============================] - 1s 6ms/step - loss: 0.6392 - acc: 0.6476 -
val_loss: 0.6989 - val_acc: 0.5078
Epoch 6/10
157/157 [==============================] - 1s 7ms/step - loss: 0.5792 - acc: 0.7146 -
val_loss: 0.7069 - val_acc: 0.5096
Epoch 7/10
157/157 [==============================] - 1s 7ms/step - loss: 0.5196 - acc: 0.7718 -
val_loss: 0.7330 - val_acc: 0.5085
Epoch 8/10
157/157 [==============================] - 1s 7ms/step - loss: 0.4610 - acc: 0.8246 -
val_loss: 1.0030 - val_acc: 0.5004
Epoch 9/10
157/157 [==============================] - 1s 7ms/step - loss: 0.4117 - acc: 0.8570 -
val_loss: 0.8523 - val_acc: 0.5066
Epoch 10/10
157/157 [==============================] - 1s 6ms/step - loss: 0.3609 - acc: 0.8796 -
val_loss: 1.0775 - val_acc: 0.5032
```





In [29]:
```
test_loss11, test_accuracy11 = model11.evaluate(x_test, y_test)
print('Test loss:', test_loss11)
print('Test accuracy:', test_accuracy11)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 1.0668 - acc: 0.4964
Test loss: 1.0668143033981323
Test accuracy: 0.49636000394821167
```

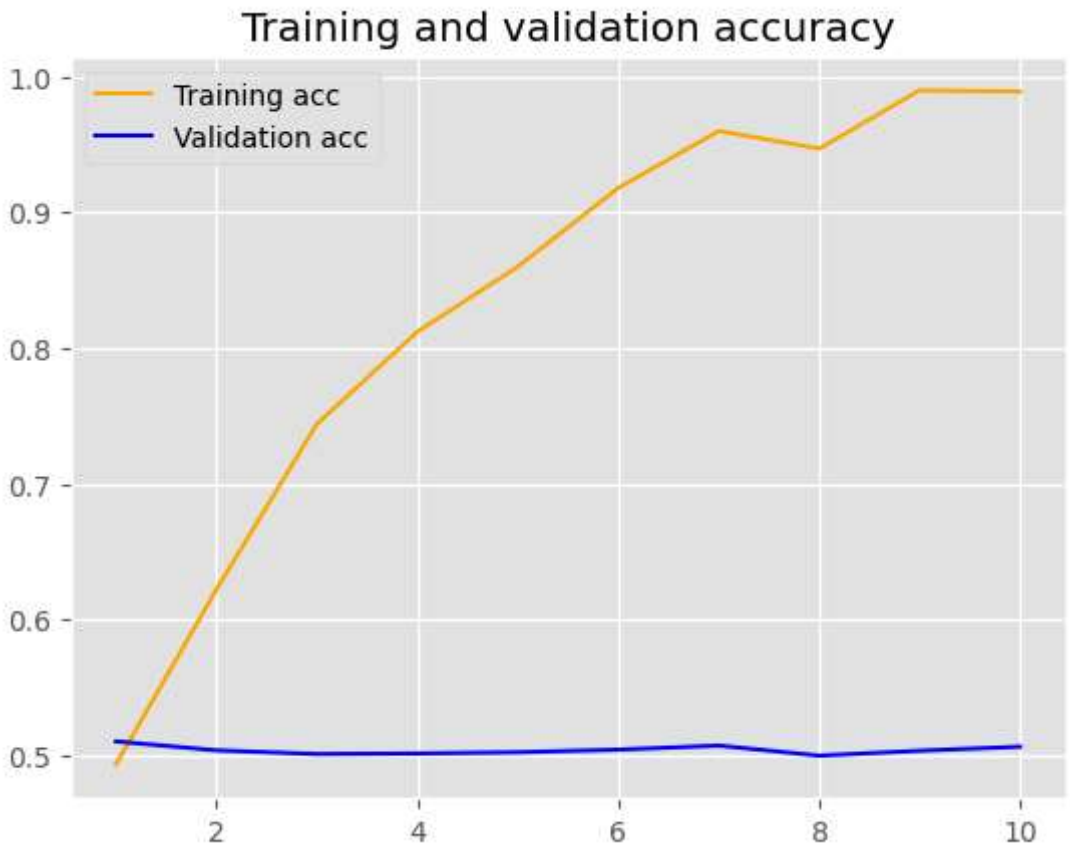
pretrained word embedding layer with training sample size = 1000

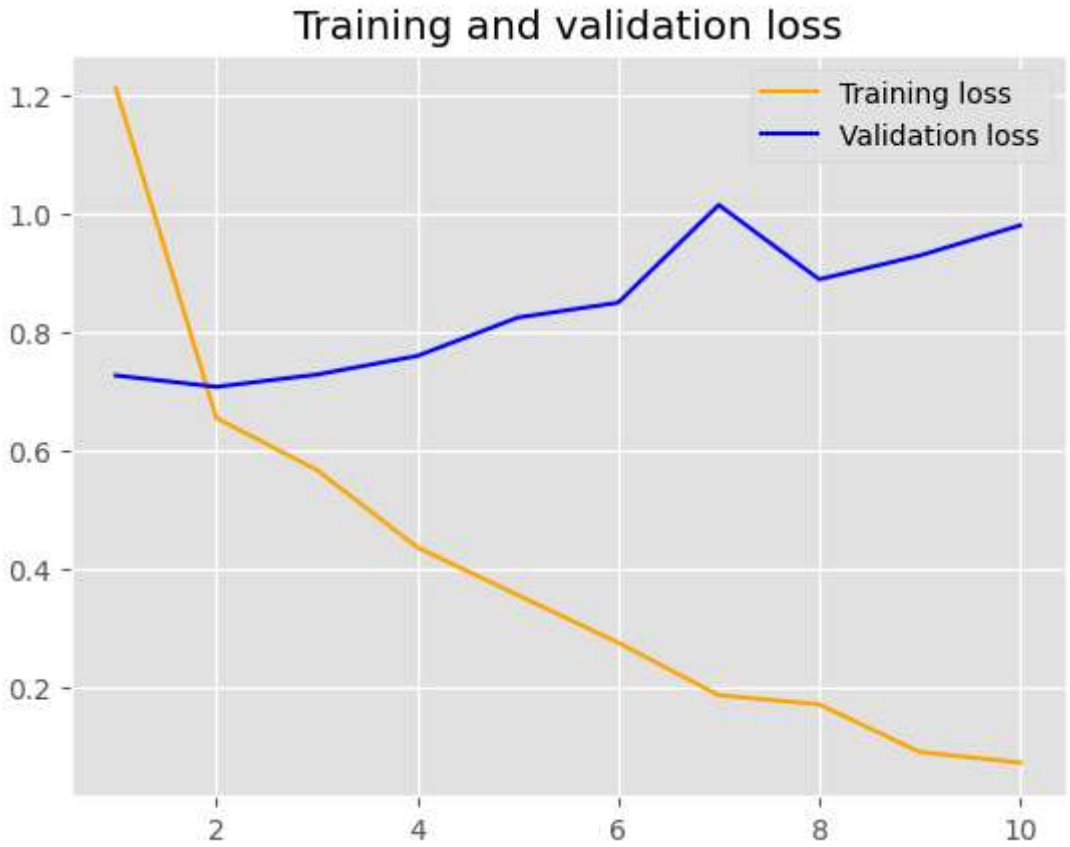
In [30]:

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

length = 150
training_data = 1000 #Trains on 1000 samples
validation_data = 10000
words = 10000

tokenizer3 = Tokenizer(num_words=words)
tokenizer3.fit_on_texts(texts)
sequences = tokenizer3.texts_to_sequences(texts)
word_index = tokenizer3.word_index
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=length)

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

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

x_train = data[:training_data]
y_train = labels[:training_data]
x_val = data[training_data:training_data+validation_data]
y_val = labels[training_data:training_data+validation_data]
embedding_dim = 100

embedd_matrix = np.zeros((words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < words:
        if embedding_vector is not None:

            embedd_matrix[i] = embedding_vector

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

model12 = Sequential()
model12.add(Embedding(words, embedding_dim, input_length=length))
model12.add(Flatten())
model12.add(Dense(32, activation='relu'))
model12.add(Dense(1, activation='sigmoid'))
model12.summary()

model12.layers[0].set_weights([embedding_matrix])
model12.layers[0].trainable = False
model12.compile(optimizer='rmsprop',
```

```
              loss='binary_crossentropy',
              metrics=['acc'])
history12 = model12.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model12.save_weights('pre_trained_glove_model.h5')
import matplotlib.pyplot as plt

acc12 = history12.history['acc']
val_acc12 = history12.history['val_acc']
loss12 = history12.history['loss']
val_loss12 = history12.history['val_loss']

epochs = range(1, len(acc12) + 1)

plt.plot(epochs, acc12, 'orange', label='Training acc')
plt.plot(epochs, val_acc12, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss12, 'orange', label='Training loss')
plt.plot(epochs, val_loss12, '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_7 (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: 1.2118 - acc: 0.4930 -
val_loss: 0.7261 - val_acc: 0.5101
Epoch 2/10
32/32 [==============================] - 1s 21ms/step - loss: 0.6540 - acc: 0.6220 -
val_loss: 0.7068 - val_acc: 0.5035
Epoch 3/10
32/32 [==============================] - 1s 22ms/step - loss: 0.5663 - acc: 0.7440 -
val_loss: 0.7276 - val_acc: 0.5008
Epoch 4/10
32/32 [==============================] - 1s 22ms/step - loss: 0.4353 - acc: 0.8120 -
val_loss: 0.7591 - val_acc: 0.5012
Epoch 5/10
32/32 [==============================] - 1s 22ms/step - loss: 0.3548 - acc: 0.8600 -
val_loss: 0.8240 - val_acc: 0.5021
Epoch 6/10
32/32 [==============================] - 1s 22ms/step - loss: 0.2741 - acc: 0.9180 -
val_loss: 0.8492 - val_acc: 0.5040
Epoch 7/10
32/32 [==============================] - 1s 21ms/step - loss: 0.1853 - acc: 0.9600 -
val_loss: 1.0146 - val_acc: 0.5070
Epoch 8/10
32/32 [==============================] - 1s 21ms/step - loss: 0.1698 - acc: 0.9470 -
val_loss: 0.8887 - val_acc: 0.4996
Epoch 9/10
32/32 [==============================] - 1s 21ms/step - loss: 0.0892 - acc: 0.9900 -
val_loss: 0.9289 - val_acc: 0.5032
Epoch 10/10
32/32 [==============================] - 1s 21ms/step - loss: 0.0713 - acc: 0.9890 -
val_loss: 0.9799 - val_acc: 0.5062
```





In [31]:
```
test_loss12, test_accuracy12 = model12.evaluate(x_test, y_test)
print('Test loss:', test_loss12)
print('Test accuracy:', test_accuracy12)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.9887 - acc: 0.4953
Test loss: 0.9887002110481262
Test accuracy: 0.495279997587204
```

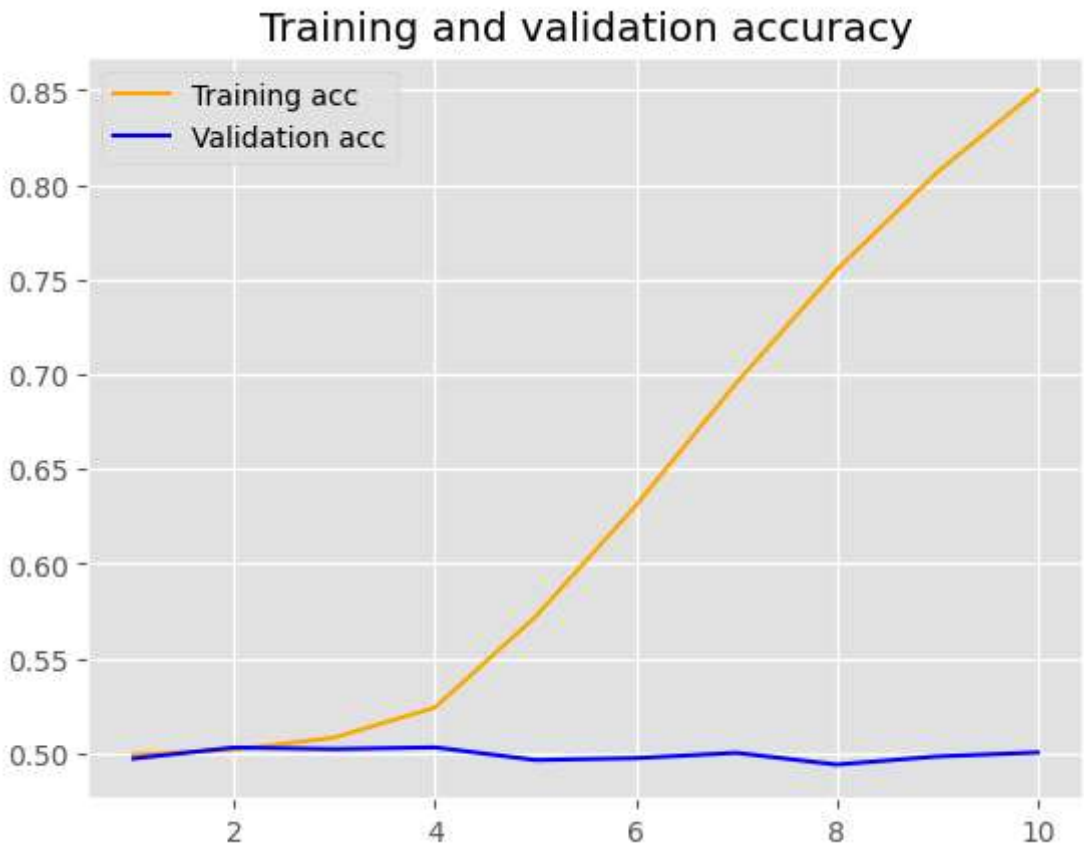
pretrained word embedding layer with training sample size = 10000

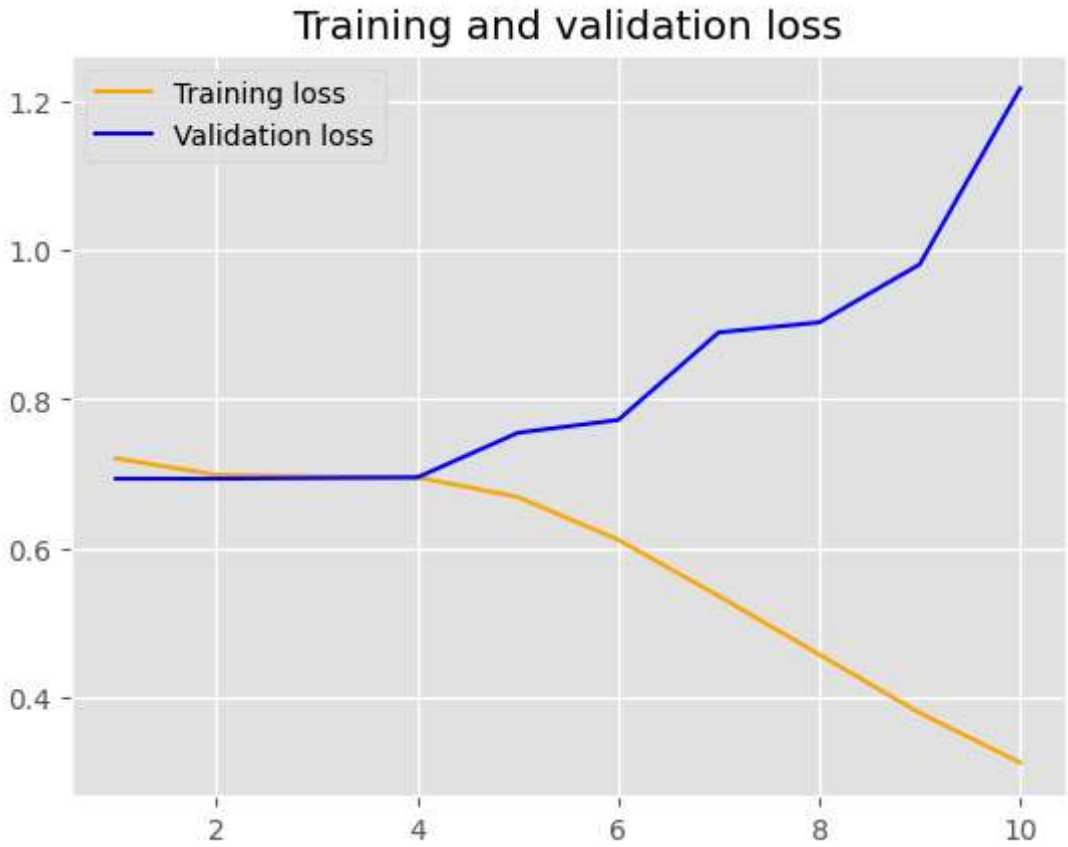
In [32]:

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

length = 150
training_ds = 10000 # Trains on 10000 samples
validation_ds = 10000
words = 10000

tokenizer4 = Tokenizer(num_words=words)
tokenizer4.fit_on_texts(texts)
sequences = tokenizer4.texts_to_sequences(texts)
word_index = tokenizer4.word_index
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=length)

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

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

x_train = data[:training_ds]
y_train = labels[:training_ds]
x_val = data[training_ds:training_ds+validation_ds]
y_val = labels[training_ds:training_ds+validation_ds]
embedd_dim = 100

embedd_matrix = np.zeros((words, embedd_dim))
for word, i in word_index.items():
    embedd_vector = embeddings_index.get(word)
    if i < words:
        if embedd_vector is not None:

            embedd_matrix[i] = embedd_vector

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

model13 = Sequential()
model13.add(Embedding(words, embedding_dim, input_length=length))
model13.add(Flatten())
model13.add(Dense(32, activation='relu'))
model13.add(Dense(1, activation='sigmoid'))
model13.summary()

model13.layers[0].set_weights([embedding_matrix])
model13.layers[0].trainable = False
model13.compile(optimizer='rmsprop',
```

```
              loss='binary_crossentropy',
              metrics=['acc'])
history13 = model13.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model13.save_weights('pre_trained_glove_model.h5')
import matplotlib.pyplot as plt

accuracy13 = history13.history['acc']
valid_acc13 = history13.history['val_acc']
loss13 = history13.history['loss']
valid_loss13 = history13.history['val_loss']

epochs = range(1, len(accuracy13) + 1)

plt.plot(epochs, accuracy13, 'orange', label='Training acc')
plt.plot(epochs, valid_acc13, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss13, 'orange', label='Training loss')
plt.plot(epochs, valid_loss13, '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_8 (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 [==============================] - 2s 5ms/step - loss: 0.7204 - acc: 0.4992 -
val_loss: 0.6932 - val_acc: 0.4972
Epoch 2/10
313/313 [==============================] - 1s 5ms/step - loss: 0.6983 - acc: 0.5019 -
val_loss: 0.6931 - val_acc: 0.5030
Epoch 3/10
313/313 [==============================] - 1s 5ms/step - loss: 0.6961 - acc: 0.5082 -
val_loss: 0.6941 - val_acc: 0.5022
Epoch 4/10
313/313 [==============================] - 1s 5ms/step - loss: 0.6951 - acc: 0.5241 -
val_loss: 0.6948 - val_acc: 0.5031
Epoch 5/10
313/313 [==============================] - 1s 5ms/step - loss: 0.6686 - acc: 0.5720 -
val_loss: 0.7548 - val_acc: 0.4964
Epoch 6/10
313/313 [==============================] - 1s 5ms/step - loss: 0.6110 - acc: 0.6307 -
val_loss: 0.7721 - val_acc: 0.4974
Epoch 7/10
313/313 [==============================] - 1s 5ms/step - loss: 0.5355 - acc: 0.6954 -
val_loss: 0.8898 - val_acc: 0.5002
Epoch 8/10
313/313 [==============================] - 1s 5ms/step - loss: 0.4569 - acc: 0.7552 -
val_loss: 0.9032 - val_acc: 0.4940
Epoch 9/10
313/313 [==============================] - 1s 5ms/step - loss: 0.3783 - acc: 0.8067 -
val_loss: 0.9812 - val_acc: 0.4983
Epoch 10/10
313/313 [==============================] - 2s 5ms/step - loss: 0.3115 - acc: 0.8500 -
val_loss: 1.2179 - val_acc: 0.5004
```





In [33]:
```
test_loss13, test_accuracy13 = model13.evaluate(x_test, y_test)
print('Test loss:', test_loss13)
print('Test accuracy:', test_accuracy13)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 1.2224 - acc: 0.4965
Test loss: 1.2223607301712036
Test accuracy: 0.4965200126171112
```