Assignment 10

DSC 650

Jake Meyer

05/20/2023

Using code examples from Chapter 6 of First Edition: deep-learning-with-python-notebooks

In [1]:

```
## Import the necessary modules for the assignment.
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import sklearn
import itertools
import string
import nltk
from numpy import array
from numpy import argmax
from sklearn.model_selection import train_test_split
from pathlib import Path
from contextlib import redirect_stdout
import time
import os

## Import the necessary keras components.
from keras import layers, models, preprocessing
from keras.datasets import imdb
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing import sequence
from keras.utils import to_categorical, np_utils
from keras.models import Sequential, load_model
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout, Activation, Flatten, Embedding
from keras.optimizers import RMSprop
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
WARNING:tensorflow:From C:\Users\jkmey\anaconda3\lib\site-packages\tensorflow\python\compa
t\v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope)
is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:

```
## Print versions of essential packages
print("keras version: {}".format(keras.__version__))
print("tensorflow version: {}".format(tf.__version__))
print("pandas version: {}".format(pd.__version__))
print("numpy version: {}".format(np.__version__))
```

```
keras version: 2.9.0
tensorflow version: 2.9.1
pandas version: 1.3.4
numpy version: 1.20.3
```

Assignment 10.1

In the first part of the assignment, you will implement basic text-preprocessing functions in Python. These functions do not need to scale to large text documents and will only need to handle small inputs.

Assignment 10.1.a

Create a tokenize function that splits a sentence into words. Ensure that your tokenizer removes basic punctuation.

In [3]:
```
'''
Purpose: As specified in the instructions, function will split a sentence into words and r
Argument: sentence
Returns: tokens
'''

def tokenize(sentence):
    ## Sentence broken down into words by spaces.
    words = sentence.split()
    ## Remove punctuation from the sentence
    punctuation_removal_table = str.maketrans('', '', string.punctuation)
    tokens = [word.translate(punctuation_removal_table) for word in words]
    return tokens
```

In [4]:
```
## Show that the tokenize() function is working as expected.
sentence = "Currently, I am working on assignment 10 and trying to see if this function wo
word_list = tokenize(sentence)
print(word_list)
```

```
['Currently', 'I', 'am', 'working', 'on', 'assignment', '10', 'and', 'trying', 'to', 'se
e', 'if', 'this', 'function', 'works']
```

Assignment 10.1.b

Implement an 'ngram' function that splits tokens into N-grams.

In [5]:
```
'''
Purpose: As specified in the instructions, function will split tokens into N-grams.
Arguments: tokens and n
Returns: n_grams
'''

def ngrams(tokens, n):
    n_grams = nltk.ngrams(tokens, n)
    return n_grams
```

In [6]:
```
## Show that ngrams() function is working as expected.
sentence = "Currently, I am working on assignment 10 and trying to see if this function wo
tokens = tokenize(sentence)
## print(tokens)
n_grams = ngrams(tokens, 3)
print(type(n_grams))
print(n_grams)
```

```
<class 'zip'>
<zip object at 0x0000027FAF2CEC00>
```

In [7]:
```
## Show the contents of n_grams to validate it is working as expected.
for gram in n_grams:
```

```
    print(gram)
```

```
('Currently', 'I', 'am')
('I', 'am', 'working')
('am', 'working', 'on')
('working', 'on', 'assignment')
('on', 'assignment', '10')
('assignment', '10', 'and')
('10', 'and', 'trying')
('and', 'trying', 'to')
('trying', 'to', 'see')
('to', 'see', 'if')
('see', 'if', 'this')
('if', 'this', 'function')
('this', 'function', 'works')
```

Assignment 10.1.c

Implement an one_hot_encode function to create a vector from a numerical vector from a list of tokens.

In [8]:

```
'''
Purpose: As specified in the instructions, function will create a vector from a numerical
Tried following example 6.1 from text for word-level one-hot encoding (toy example)
Arguments: tokens and number of words
Returns: results
'''
def one_hot_encode(tokens, num_words):
    token_index =  {}
    for sample in tokens:
        for word in sample.split():
            if word in sample.split():
                if word not in token_index:
                    token_index[word] = len(token_index) + 1
    max_length = 10
    results = np.zeros(shape = (len(tokens), max_length, max(token_index.values()) + 1))
    for i, sample in enumerate(tokens):
        for j, word in list(enumerate(sample.split()))[:max_length]:
            index = token_index.get(word)
            results[i, j, index] = 1
    return results
```

In [9]:

```
## Show that the one_hot_encode() function is working correctly.
sentence = "Currently, I am working on assignment 10 and trying to see if this function wo
tokens = tokenize(sentence)
## print(tokens)
one_hot_example = one_hot_encode(tokens, 15)
print(type(one_hot_example))
print(one_hot_example)
```

```
<class 'numpy.ndarray'>
[[[0. 1. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 1. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
```

```
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

...

[[0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

[[0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

[[0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]]
```

Assignment 10.2

Using listings 6.16, 6.17, and 6.18 in Deep Learning with Python as a guide, train a sequential model with embeddings on the IMDB data found in data/external/imdb/. Produce the model performance metrics and training and validation accuracy curves within the Jupyter notebook.

In [10]:

```
## Define directories for storing results and loading the data (train and test).
imdb_dir = Path('C:/Users/jkmey/Documents/Github/DSC650_Course_Assignments/dsc650/data/ext
results_dir = Path('C:/Users/jkmey/Documents/Github/DSC650_Course_Assignments/'
                   'dsc650/dsc650/assignments/assignment10/').joinpath('results').joinpath
results_dir.mkdir(parents = True, exist_ok = True)
test_dir = os.path.join(imdb_dir, 'test')
train_dir = os.path.join(imdb_dir, 'train')
```

In [11]:

```
## Using 6.8 as an additional reference from Deep Learning with Python code.
## Code will process the labels of the raw IMDB data.
labels = []
texts = []

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

```
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

In [12]:

```
## Using 6.9 as an additional reference from Deep Learning with Python code.
## Code will Tokenize text of the raw IMDB data.

maxlen = 100  # We will cut reviews after 100 words
training_samples = 200  # We will be training on 200 samples
validation_samples = 10000  # We will be validating on 10000 samples
max_words = 10000  # We will only consider the top 10,000 words in the dataset
embedding_dim = 100

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

word_index = tokenizer.word_index
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)

# Split the data into a training set and a validation set
# But first, shuffle the data, since we started from data
# where sample are ordered (all negative first, then all positive).
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

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

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 100)
Shape of label tensor: (25000,)
```

In [13]:

```
## Using 6.16 as reference from Deep Learning with Python code.
## Code will define the model.
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:From C:\Users\jkmey\anaconda3\lib\site-packages\keras\initializers\init
ializers_v1.py:277: calling RandomUniform.__init__ (from tensorflow.python.ops.init_ops) w
ith dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 embedding (Embedding)        (None, 100, 100)          1000000

 flatten (Flatten)            (None, 10000)             0

 dense (Dense)                (None, 32)                320032

 dense_1 (Dense)              (None, 1)                 33

=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```python
## Save the summary for this model to the results directory.

summary_file = results_dir.joinpath('assignment_10-2_Model_Summary.txt')
with open(summary_file, 'w') as f:
    with redirect_stdout(f):
        model.summary()
```

In [15]:
```python
## Train and Evaluate the model.

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))

result_model_file = results_dir.joinpath('model_10-2.h5')
model.save_weights(result_model_file)
```

```
Train on 200 samples, validate on 10000 samples
WARNING:tensorflow:OMP_NUM_THREADS is no longer used by the default Keras config. To confi
gure the number of threads, use tf.config.threading APIs.
Epoch 1/10
 32/200 [===>..........................] - ETA: 0s - loss: 0.6951 - acc: 0.4375
C:\Users\jkmey\anaconda3\lib\site-packages\keras\engine\training_v1.py:2045: UserWarning:
`Model.state_updates` will be removed in a future version. This property should not be use
d in TensorFlow 2.0, as `updates` are applied automatically.
  updates = self.state_updates
200/200 [==============================] - 0s 2ms/sample - loss: 0.6947 - acc: 0.5350 - va
l_loss: 0.6940 - val_acc: 0.4985
Epoch 2/10
200/200 [==============================] - 0s 1ms/sample - loss: 0.5189 - acc: 0.9900 - va
l_loss: 0.7003 - val_acc: 0.4977
Epoch 3/10
200/200 [==============================] - 0s 1ms/sample - loss: 0.3050 - acc: 0.9850 - va
l_loss: 0.7077 - val_acc: 0.5038
Epoch 4/10
200/200 [==============================] - 0s 2ms/sample - loss: 0.1481 - acc: 1.0000 - va
l_loss: 0.7038 - val_acc: 0.5086
Epoch 5/10
200/200 [==============================] - 0s 2ms/sample - loss: 0.0704 - acc: 1.0000 - va
l_loss: 0.7089 - val_acc: 0.5124
Epoch 6/10
200/200 [==============================] - 0s 1ms/sample - loss: 0.0364 - acc: 1.0000 - va
l_loss: 0.7173 - val_acc: 0.5107
Epoch 7/10
200/200 [==============================] - 0s 2ms/sample - loss: 0.0198 - acc: 1.0000 - va
l_loss: 0.7303 - val_acc: 0.5075
```

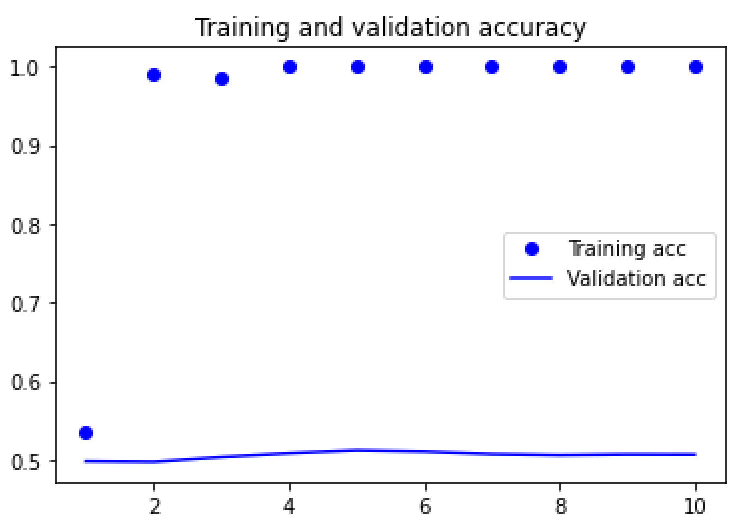
```
Epoch 8/10
200/200 [==============================] - 0s 1ms/sample - loss: 0.0113 - acc: 1.0000 - va
l_loss: 0.7382 - val_acc: 0.5062
Epoch 9/10
200/200 [==============================] - 0s 1ms/sample - loss: 0.0068 - acc: 1.0000 - va
l_loss: 0.7624 - val_acc: 0.5072
Epoch 10/10
200/200 [==============================] - 0s 1ms/sample - loss: 0.0042 - acc: 1.0000 - va
l_loss: 0.7574 - val_acc: 0.5070
```

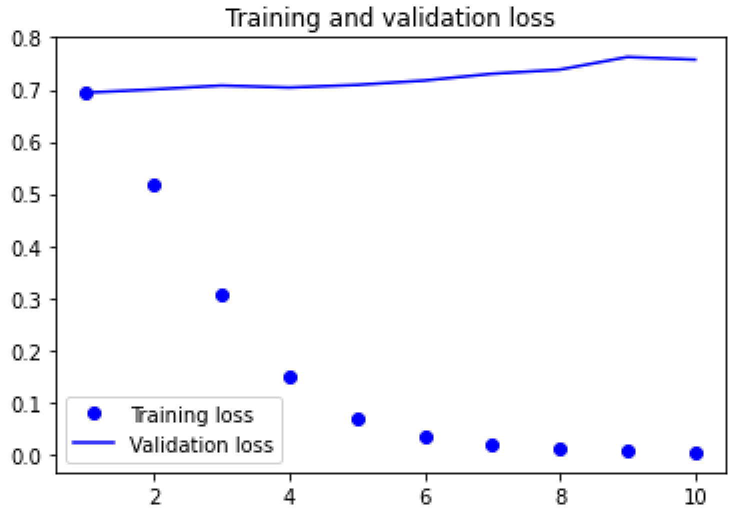
In [16]:

```
result_model_file = results_dir.joinpath('model_10-2.h5')
model.save_weights(result_model_file)
```

In [17]:

```
## Plotting the accuracy and results.

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

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

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

plt.figure()

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





In [18]:

```
## Using 6.17 as reference Tokenize the data of the test set.

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

labels = []
texts = []

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

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

In [19]:

```
## Load and evaluate the model.
model.load_weights(result_model_file)
eval = model.evaluate(x_test, y_test)
```

In [20]:

```
## Show the x_test and y_test accuracy results.
print(eval)
```

```
[0.7548423609542847, 0.51572]
```

Assignment 10.3

Using listing 6.27 in Deep Learning with Python as a guide, fit the same data with an LSTM layer. Produce the model performance metrics and training and validation accuracy curves within the Jupyter notebook.

In [21]:

```
## Define results directory for this section.
results_dir2 = Path('C:/Users/jkmey/Documents/Github/DSC650_Course_Assignments/'
                    'dsc650/dsc650/assignments/assignment10/').joinpath('results').joinpatl
results_dir2.mkdir(parents = True, exist_ok = True)
```

In [22]:

```
## Preprocess the data.
max_features = 10000  # number of words to consider as features
maxlen = 500  # cut texts after this number of words (among top max_features most common w
batch_size = 32

print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')

print('Pad sequences (samples x time)')
input_train = pad_sequences(input_train, maxlen=maxlen)
input_test = pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
input_train shape: (25000, 500)
input_test shape: (25000, 500)
```

In [34]:

```
## Use 6.27 as a refereece for code.
## Use the LSTM layer in Keras.
model_2 = Sequential()
model_2.add(Embedding(max_features, 32))
model_2.add(LSTM(32))
model_2.add(Dense(1, activation='sigmoid'))

model_2.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history_2 = model_2.fit(input_train, y_train,
                      epochs=10,
                      batch_size=128,
                      validation_split=0.2)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.4957 - acc: 0.7693
- val_loss: 0.3534 - val_acc: 0.8622
Epoch 2/10
20000/20000 [==============================] - 21s 1ms/sample - loss: 0.2846 - acc: 0.8864
- val_loss: 0.3139 - val_acc: 0.8730
Epoch 3/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.2296 - acc: 0.9130
- val_loss: 0.3548 - val_acc: 0.8440
Epoch 4/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.1947 - acc: 0.9275
- val_loss: 0.3393 - val_acc: 0.8526
Epoch 5/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.1696 - acc: 0.9395
- val_loss: 0.3416 - val_acc: 0.8680
Epoch 6/10
20000/20000 [==============================] - 21s 1ms/sample - loss: 0.1481 - acc: 0.9461
- val_loss: 0.3171 - val_acc: 0.8740
Epoch 7/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.1361 - acc: 0.9516
- val_loss: 0.4277 - val_acc: 0.8516
Epoch 8/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.1239 - acc: 0.9565
- val_loss: 0.4455 - val_acc: 0.8374
Epoch 9/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.1165 - acc: 0.9596
```

```
- val_loss: 0.3411 - val_acc: 0.8808
Epoch 10/10
20000/20000 [==============================] - 22s 1ms/sample - loss: 0.1096 - acc: 0.9612
- val_loss: 0.4113 - val_acc: 0.8816
```

In [35]:

```python
## Save the summary for this model to the results directory.

summary_file_2 = results_dir2.joinpath('assignment_10-3_Model_Summary.txt')
with open(summary_file_2, 'w') as f:
    with redirect_stdout(f):
        model_2.summary()
```

In [36]:

```python
model_2.summary()
```

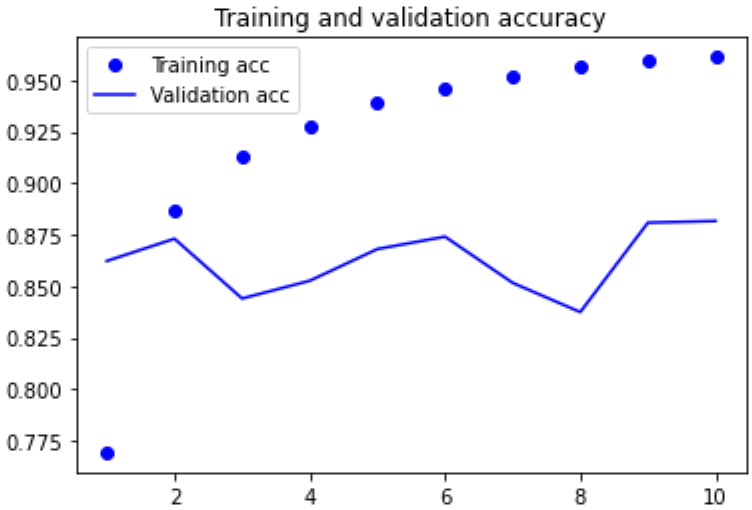
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 32)          320000

 lstm_1 (LSTM)               (None, 32)                8320

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```

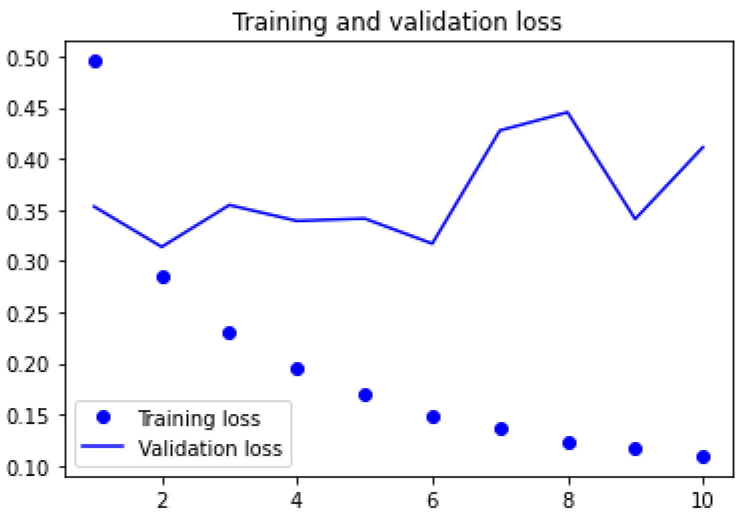
In [37]:

```python
result_model_file_2 = results_dir2.joinpath('model_10-3.h5')
model_2.save_weights(result_model_file_2)
```

In [38]:

```python
## Plotting the accuracy and results.

acc_2 = history_2.history['acc']
val_acc_2 = history_2.history['val_acc']
loss_2 = history_2.history['loss']
val_loss_2 = history_2.history['val_loss']

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

plt.plot(epochs, acc_2, 'bo', label='Training acc')
plt.plot(epochs, val_acc_2, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_2, 'bo', label='Training loss')
plt.plot(epochs, val_loss_2, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





In [40]:
```
## Load and evaluate the model_2.
model_2.load_weights(result_model_file_2)
eval_2 = model_2.evaluate(x_test, y_test)
```

In [41]:
```
## Show the x_test and y_test accuracy results.
print(eval_2)
```

```
[1.780648521823883, 0.52356]
```

Assignment 10.4

Using listing 6.46 in Deep Learning with Python as a guide, fit the same data with a simple 1D convnet. Produce the model performance metrics and training and validation accuracy curves within the Jupyter notebook.

In [42]:
```
## Define results directory for this section.
results_dir3 = Path('C:/Users/jkmey/Documents/Github/DSC650_Course_Assignments/'
                    'dsc650/dsc650/assignments/assignment10/').joinpath('results').joinpatl
results_dir3.mkdir(parents = True, exist_ok = True)
```

In [44]:
```
max_features = 10000  # number of words to consider as features
max_len = 500  # cut texts after this number of words (among top max_features most common

print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')
```

```
print('Pad sequences (samples x time)')
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 500)
x_test shape: (25000, 500)
```

In [46]:

```
## Use code from 6.46 to Train and Evaluate a Simple 1D Convnet.
model_3 = Sequential()
model_3.add(layers.Embedding(max_features, 128, input_length=max_len))
model_3.add(layers.Conv1D(32, 7, activation='relu'))
model_3.add(layers.MaxPooling1D(5))
model_3.add(layers.Conv1D(32, 7, activation='relu'))
model_3.add(layers.GlobalMaxPooling1D())
model_3.add(layers.Dense(1))

model_3.summary()

model_3.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history_3 = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_4 (Embedding)     (None, 500, 128)          1280000   
                                                                 
 conv1d_2 (Conv1D)           (None, 494, 32)           28704     
                                                                 
 max_pooling1d_1 (MaxPooling  (None, 98, 32)           0         
 1D)                                                             
                                                                 
 conv1d_3 (Conv1D)           (None, 92, 32)            7200      
                                                                 
 global_max_pooling1d_1 (Glo  (None, 32)               0         
 balMaxPooling1D)                                                
                                                                 
 dense_5 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 19s 931us/sample - loss: 0.6370 - acc: 0.75
24 - val_loss: 0.6372 - val_acc: 0.7200
Epoch 2/10
20000/20000 [==============================] - 19s 947us/sample - loss: 0.5762 - acc: 0.80
95 - val_loss: 0.5578 - val_acc: 0.7866
Epoch 3/10
20000/20000 [==============================] - 19s 957us/sample - loss: 0.4653 - acc: 0.84
```

```
04 - val_loss: 0.4420 - val_acc: 0.8288
Epoch 4/10
20000/20000 [==============================] - 19s 946us/sample - loss: 0.3680 - acc: 0.86
51 - val_loss: 0.3982 - val_acc: 0.8484
Epoch 5/10
20000/20000 [==============================] - 19s 934us/sample - loss: 0.3198 - acc: 0.88
65 - val_loss: 0.3996 - val_acc: 0.8590
Epoch 6/10
20000/20000 [==============================] - 18s 901us/sample - loss: 0.2836 - acc: 0.90
09 - val_loss: 0.4128 - val_acc: 0.8588
Epoch 7/10
20000/20000 [==============================] - 18s 877us/sample - loss: 0.2579 - acc: 0.91
29 - val_loss: 0.4422 - val_acc: 0.8612
Epoch 8/10
20000/20000 [==============================] - 18s 910us/sample - loss: 0.2327 - acc: 0.92
22 - val_loss: 0.4704 - val_acc: 0.8668
Epoch 9/10
20000/20000 [==============================] - 18s 893us/sample - loss: 0.2095 - acc: 0.93
31 - val_loss: 0.4793 - val_acc: 0.8678
Epoch 10/10
20000/20000 [==============================] - 18s 904us/sample - loss: 0.1910 - acc: 0.94
01 - val_loss: 0.5061 - val_acc: 0.8688
```

In [47]:

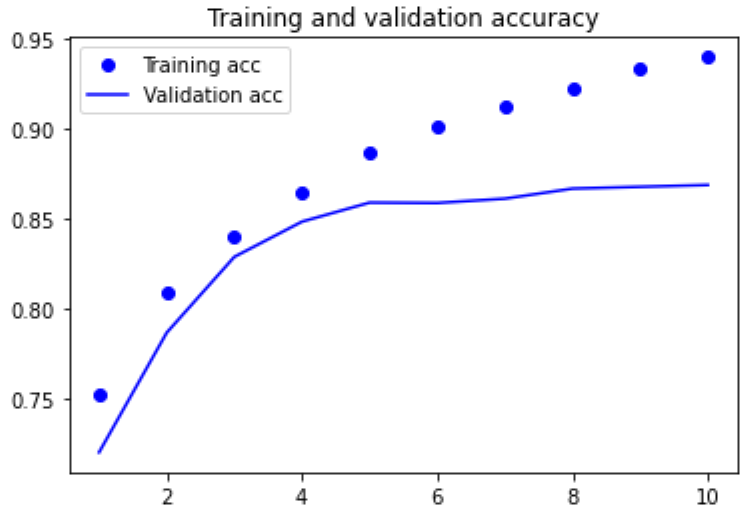
```
## Save the summary for this model to the results directory.

summary_file_3 = results_dir3.joinpath('assignment_10-4_Model_Summary.txt')
with open(summary_file_3, 'w') as f:
    with redirect_stdout(f):
        model_3.summary()
```

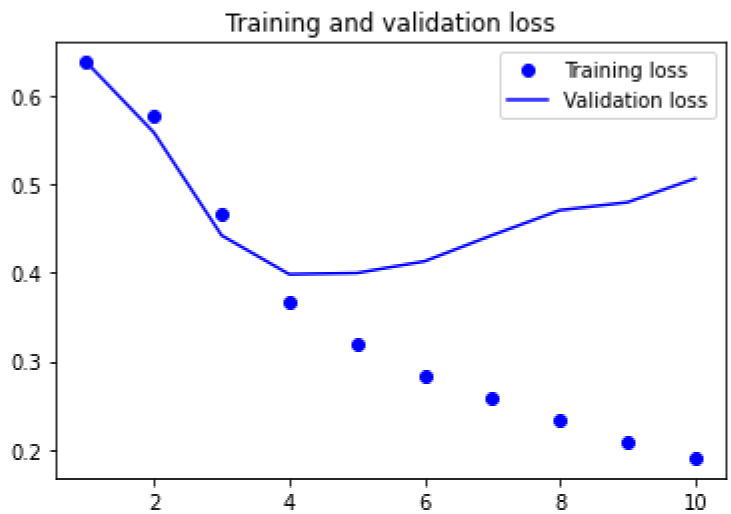
In [48]:

```
result_model_file_3 = results_dir3.joinpath('model_10-4.h5')
model_3.save_weights(result_model_file_3)
```

In [50]:

```
## Plotting the accuracy and results.

acc_3 = history_3.history['acc']
val_acc_3 = history_3.history['val_acc']
loss_3 = history_3.history['loss']
val_loss_3 = history_3.history['val_loss']

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

plt.plot(epochs, acc_3, 'bo', label='Training acc')
plt.plot(epochs, val_acc_3, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_3, 'bo', label='Training loss')
plt.plot(epochs, val_loss_3, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





In [51]:

```
## Load and evaluate the model_3.
model_3.load_weights(result_model_file_3)
eval_3 = model_3.evaluate(x_test, y_test)
```

In [52]:

```
## Show the x_test and y_test accuracy results.
print(eval_3)
```

```
[1.3791219604492186, 0.5]
```