assignment10

January 31, 2022

```
[1]: import os
     import string
     import re
     from pathlib import Path
```

```
[3]: current_dir = Path(os.getcwd()).absolute()
     results_dir = current_dir.joinpath('results')
     results_dir.mkdir(parents=True, exist_ok=True)
     data_dir = current_dir.joinpath('data')
     data_dir.mkdir(parents=True, exist_ok=True)
     external_data_dir = current_dir.parent.parent.parent.joinpath('data')
     imdb_dir = external_data_dir.joinpath(r'external/imdb/aclImdb')




     print(current_dir)
     print(results_dir)
     print(data_dir)
     print(imdb_dir)
```

```
/home/jovyan/dsc650/dsc650/assignments/assignment10
/home/jovyan/dsc650/dsc650/assignments/assignment10/results
/home/jovyan/dsc650/dsc650/assignments/assignment10/data
/home/jovyan/dsc650/data/external/imdb/aclImdb
```

Assignment 10.1.a Create a tokenize function that splits a sentence into words. Ensure that your tokenizer removes basic punctuation.

```
[5]: file_name = "sample.txt"
     file_path = f"{data_dir}/{file_name}"
     file_path
```

```
[5]: '/home/jovyan/dsc650/dsc650/assignments/assignment10/data/sample.txt'
```

```
[6]: def tokenize(sentence):
         tokens = []
         words = sentence.split()
```

```
    # tokenize the sentence
    for word in words:
        # allowing only alphabets
        word = re.sub("[^a-zA-Z]", "", word)
        tokens.append(word)
    return tokens
```

[7]:
```
def call_tokenize():
    with open(f'{file_path}','r') as f:
        for line in f:
            line = line.lower()

            return tokenize(line)

call_tokenize()
```

[7]:
```
['it',
 'was',
 '',
 'minutes',
 'after',
 'midnight',
 'the',
 'dog',
 'was',
 'lying',
 'on',
 'the',
 'grass',
 'in',
 'the',
 'middle',
 'of',
 'the',
 'lawn',
 'in',
 'front',
 'of',
 'mrs',
 'shears',
 'house',
 'its',
 'eyes',
 'were',
 'closed',
 'it',
 'looked',
 'as',
 'if',
 'it',
 'was',
 'running',
 'on',
 'its',
 'side',
 'the',
 'way',
 'dogs',
 'run',
 'when',
 'they',
 'think',
 'they',
 'are',
 'chasing',
 'a',
 'cat',
 'in',
 'a',
 'dream',
 'but',
 'the',
 'dog',
 'was',
 'not',
 'running',
 'or',
 'asleep',
 'the',
 'dog',
 'was',
 'dead',
 'there',
 'was',
 'a',
 'garden',
 'fork',
 'sticking',
 'out',
 'of',
 'the',
 'dog',
 'the',
 'points',
 'of',
 'the',
 'fork',
 'must',
 'have',
 'gone',
 'all',
 'the',
 'way',
 'through',
 'the',
 'dog',
 'and',
 'into',
 'the',
 'ground',
 'because',
 'the',
 'fork',
 'had',
 'not',
 'fallen',
 'over',
 'i',
 'decided',
 'that',
 'the',
 'dog',
 'was',
 'probably',
 'killed',
 'with',
 'the',
 'fork',
 'because',
 'i',
 'could',
 'not',
 'see',
 'any',
 'other',
 'wounds',
 'in',
 'the',
 'dog',
 'and',
 'i',
 'do',
 'not',
 'think',
 'you',
 'would',
 'stick',
 'a',
 'garden',
 'fork',
 'into',
 'a',
 'dog',
 'after',
 'it',
 'had',
 'died',
 'for',
 'some',
 'other',
 'reason',
 'like',
 'cancer',
 'for',
 'example',
 'or',
 'a',
 'road',
 'accident',
 'but',
 'i',
 'could',
 'not',
 'be',
 'certain',
 'about',
 'this']
```

Assignment 10.1.b Implement an `ngram` function that splits tokens into N-grams.

[8]:
```python
def ngram(tokens, n):
    # ngrams = []
    # Create ngrams
    # return ngrams
    return print(list(zip(*[tokens[i:] for i in range(n)])))
```

```
[9]: def call_ngram():
         with open(f'{file_path}','r') as f:
             for line in f:
                 line = line.lower()
                 # print(line, end = 'XX')
                 return ngram(tokenize(line),3)

     call_ngram()
```

```
[('it', 'was', ''), ('was', '', 'minutes'), ('', 'minutes', 'after'),
('minutes', 'after', 'midnight'), ('after', 'midnight', 'the'), ('midnight',
'the', 'dog'), ('the', 'dog', 'was'), ('dog', 'was', 'lying'), ('was', 'lying',
'on'), ('lying', 'on', 'the'), ('on', 'the', 'grass'), ('the', 'grass', 'in'),
('grass', 'in', 'the'), ('in', 'the', 'middle'), ('the', 'middle', 'of'),
('middle', 'of', 'the'), ('of', 'the', 'lawn'), ('the', 'lawn', 'in'), ('lawn',
'in', 'front'), ('in', 'front', 'of'), ('front', 'of', 'mrs'), ('of', 'mrs',
'shears'), ('mrs', 'shears', 'house'), ('shears', 'house', 'its'), ('house',
'its', 'eyes'), ('its', 'eyes', 'were'), ('eyes', 'were', 'closed'), ('were',
'closed', 'it'), ('closed', 'it', 'looked'), ('it', 'looked', 'as'), ('looked',
'as', 'if'), ('as', 'if', 'it'), ('if', 'it', 'was'), ('it', 'was', 'running'),
('was', 'running', 'on'), ('running', 'on', 'its'), ('on', 'its', 'side'),
('its', 'side', 'the'), ('side', 'the', 'way'), ('the', 'way', 'dogs'), ('way',
'dogs', 'run'), ('dogs', 'run', 'when'), ('run', 'when', 'they'), ('when',
'they', 'think'), ('they', 'think', 'they'), ('think', 'they', 'are'), ('they',
'are', 'chasing'), ('are', 'chasing', 'a'), ('chasing', 'a', 'cat'), ('a',
'cat', 'in'), ('cat', 'in', 'a'), ('in', 'a', 'dream'), ('a', 'dream', 'but'),
('dream', 'but', 'the'), ('but', 'the', 'dog'), ('the', 'dog', 'was'), ('dog',
'was', 'not'), ('was', 'not', 'running'), ('not', 'running', 'or'), ('running',
'or', 'asleep'), ('or', 'asleep', 'the'), ('asleep', 'the', 'dog'), ('the',
'dog', 'was'), ('dog', 'was', 'dead'), ('was', 'dead', 'there'), ('dead',
'there', 'was'), ('there', 'was', 'a'), ('was', 'a', 'garden'), ('a', 'garden',
'fork'), ('garden', 'fork', 'sticking'), ('fork', 'sticking', 'out'),
('sticking', 'out', 'of'), ('out', 'of', 'the'), ('of', 'the', 'dog'), ('the',
'dog', 'the'), ('dog', 'the', 'points'), ('the', 'points', 'of'), ('points',
'of', 'the'), ('of', 'the', 'fork'), ('the', 'fork', 'must'), ('fork', 'must',
'have'), ('must', 'have', 'gone'), ('have', 'gone', 'all'), ('gone', 'all',
'the'), ('all', 'the', 'way'), ('the', 'way', 'through'), ('way', 'through',
'the'), ('through', 'the', 'dog'), ('the', 'dog', 'and'), ('dog', 'and',
'into'), ('and', 'into', 'the'), ('into', 'the', 'ground'), ('the', 'ground',
'because'), ('ground', 'because', 'the'), ('because', 'the', 'fork'), ('the',
'fork', 'had'), ('fork', 'had', 'not'), ('had', 'not', 'fallen'), ('not',
'fallen', 'over'), ('fallen', 'over', 'i'), ('over', 'i', 'decided'), ('i',
'decided', 'that'), ('decided', 'that', 'the'), ('that', 'the', 'dog'), ('the',
'dog', 'was'), ('dog', 'was', 'probably'), ('was', 'probably', 'killed'),
('probably', 'killed', 'with'), ('killed', 'with', 'the'), ('with', 'the',
'fork'), ('the', 'fork', 'because'), ('fork', 'because', 'i'), ('because', 'i',
'could'), ('i', 'could', 'not'), ('could', 'not', 'see'), ('not', 'see', 'any'),
('see', 'any', 'other'), ('any', 'other', 'wounds'), ('other', 'wounds', 'in'),
('wounds', 'in', 'the'), ('in', 'the', 'dog'), ('the', 'dog', 'and'), ('dog',
'and', 'i'), ('and', 'i', 'do'), ('i', 'do', 'not'), ('do', 'not', 'think'),
('not', 'think', 'you'), ('think', 'you', 'would'), ('you', 'would', 'stick'),
('would', 'stick', 'a'), ('stick', 'a', 'garden'), ('a', 'garden', 'fork'),
('garden', 'fork', 'into'), ('fork', 'into', 'a'), ('into', 'a', 'dog'), ('a',
'dog', 'after'), ('dog', 'after', 'it'), ('after', 'it', 'had'), ('it', 'had',
'died'), ('had', 'died', 'for'), ('died', 'for', 'some'), ('for', 'some',
'other'), ('some', 'other', 'reason'), ('other', 'reason', 'like'), ('reason',
'like', 'cancer'), ('like', 'cancer', 'for'), ('cancer', 'for', 'example'),
('for', 'example', 'or'), ('example', 'or', 'a'), ('or', 'a', 'road'), ('a',
'road', 'accident'), ('road', 'accident', 'but'), ('accident', 'but', 'i'),
('but', 'i', 'could'), ('i', 'could', 'not'), ('could', 'not', 'be'), ('not',
'be', 'certain'), ('be', 'certain', 'about'), ('certain', 'about', 'this')]
```

Assignment 10.1.c Implement an one_hot_encode function to create a vector from a numerical vector from a list of tokens.

[10]:
```python
def one_hot_encode(tokens, num_words):
    token_index = {}
    results = ''
    return results
```

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.

[11]:
```python
# Processing the labels of the raw IMDB data

def load_raw_imdb(source_dir):

    labels = []
    texts = []

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

    labels = np.asarray(labels)
    return texts,labels
```

[12]:
```
# Plotting the results from the training and validation set

import matplotlib.pyplot as plt

def plot_train_val(acc,val_acc,loss, val_loss):

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

    plt.plot(epochs, acc, 'bo', label='Training acc')
    if len(val_acc) == 0:
        plt.title('Training Accuracy')
    else:
        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')
    if len(val_loss) == 0:
        plt.title('Training loss')
    else:
        plt.plot(epochs, val_loss, 'b', label='Validation loss')
        plt.title('Training and validation loss')

    plt.legend()
    plt.show()
```

[13]:
```
# Tokenizing the text of the raw IMDB data

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


train_dir = imdb_dir.joinpath('train')
(texts,labels) = load_raw_imdb(train_dir)

# cuts off reviews after 100 words
maxlen = 100
# trains 200 samples
training_samples = 200
#validates on 10,000 samples
validation_samples = 10000
# considers only the top 10,000 words in the dataset
```

```
max_words = 10000

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

# Splits the data into a training set and a validation set, but first shuffles␣
 ↪the data,
# because you're starting with data in which samples 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,)
```

[14]:
```
# Parsing the GloVe word-embeddings file
glove_dir = '/home/jovyan/glove.6B'

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

print('Found %s word vectors.' % len(embeddings_index))

# Preparing the GloVe word-embeddings matrix
```

```
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

# Training the same model without pretrained word embeddings
# Model Definition

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()
```

```
Found 400000 word vectors.
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
_________________________________________________________________
```

[15]:
```
# Training and Evaluating 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))
```

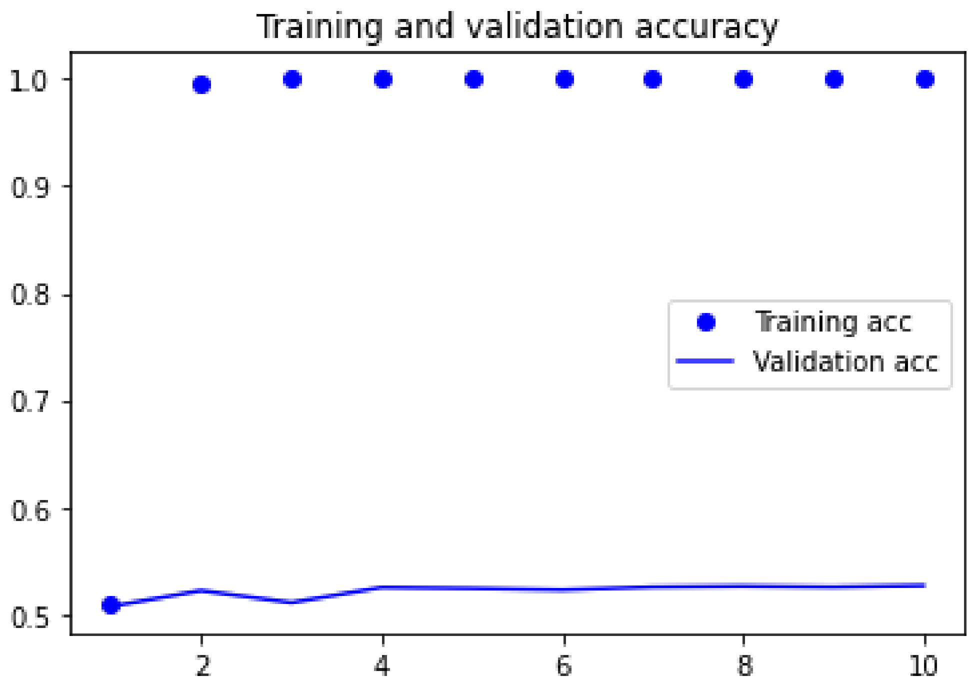
```
Epoch 1/10
7/7 [==============================] - 1s 124ms/step - loss: 0.6924 - acc:
0.5100 - val_loss: 0.6936 - val_acc: 0.5075
Epoch 2/10
7/7 [==============================] - 1s 107ms/step - loss: 0.4739 - acc:
0.9950 - val_loss: 0.6922 - val_acc: 0.5223
Epoch 3/10
7/7 [==============================] - 1s 106ms/step - loss: 0.2491 - acc:
1.0000 - val_loss: 0.7039 - val_acc: 0.5111
Epoch 4/10
7/7 [==============================] - 1s 102ms/step - loss: 0.1051 - acc:
1.0000 - val_loss: 0.7021 - val_acc: 0.5253
Epoch 5/10
7/7 [==============================] - 1s 101ms/step - loss: 0.0485 - acc:
1.0000 - val_loss: 0.7085 - val_acc: 0.5244
Epoch 6/10
7/7 [==============================] - 1s 107ms/step - loss: 0.0250 - acc:
1.0000 - val_loss: 0.7173 - val_acc: 0.5230
Epoch 7/10
7/7 [==============================] - 1s 102ms/step - loss: 0.0138 - acc:
1.0000 - val_loss: 0.7189 - val_acc: 0.5257
Epoch 8/10
7/7 [==============================] - 1s 103ms/step - loss: 0.0081 - acc:
1.0000 - val_loss: 0.7262 - val_acc: 0.5264
Epoch 9/10
7/7 [==============================] - 1s 99ms/step - loss: 0.0048 - acc: 1.0000
- val_loss: 0.7332 - val_acc: 0.5257
Epoch 10/10
7/7 [==============================] - 1s 103ms/step - loss: 0.0030 - acc:
1.0000 - val_loss: 0.7359 - val_acc: 0.5270
```

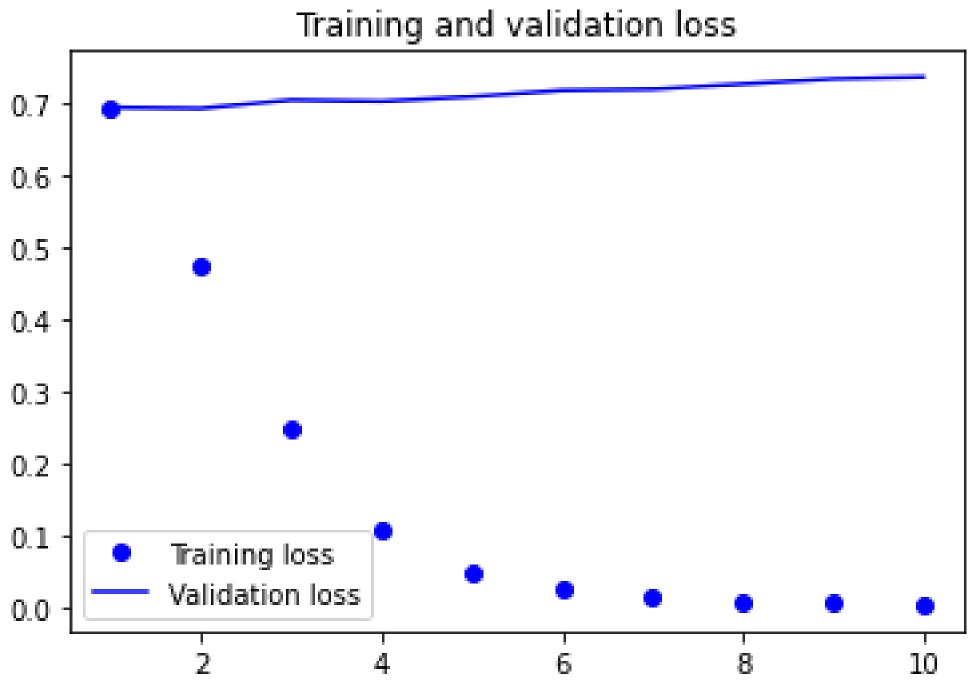
The accuracy = 1.0

[16]:
```
#saving the model
model.save_weights('glove_model.h5')
```

[17]:
```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

plot_train_val(acc,val_acc,loss,val_loss)
```





```
[18]: # Tokenizing the data for the test set
      test_dir = imdb_dir.joinpath('test')
```

```
(texts,labels) = load_raw_imdb(test_dir)

sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
print('Shape of data tensor:', x_test.shape)
print('Shape of label tensor:', y_test.shape)
```

```
Shape of data tensor: (25000, 100)
Shape of label tensor: (25000,)
```

[19]:
```
# Load and evaluate the model on the test set
model.load_weights('glove_model.h5')
model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.7387 - acc:
0.5227
```

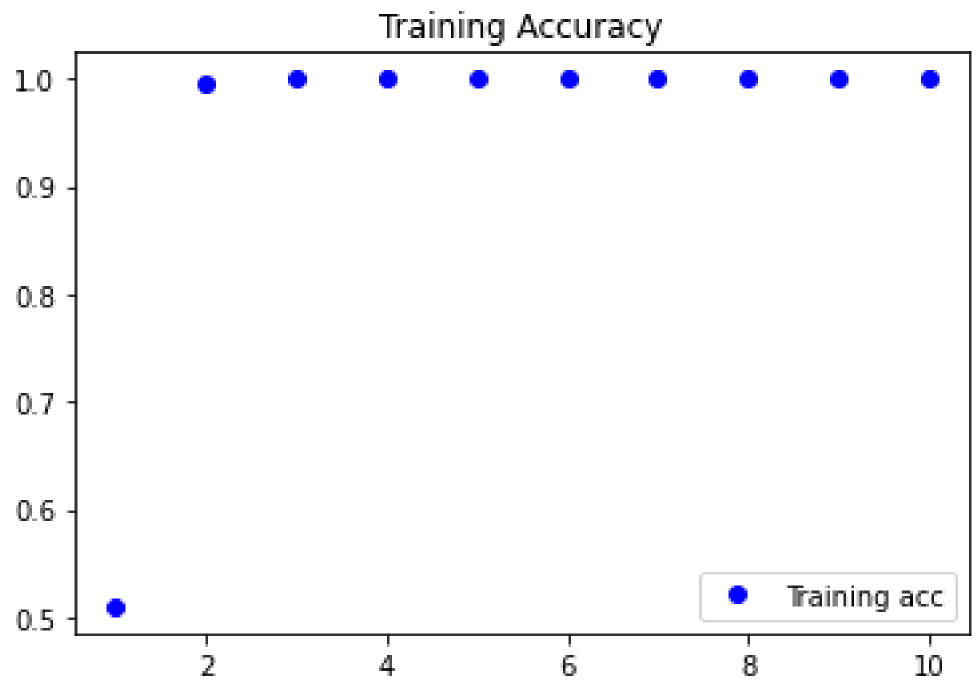
[19]:
```
[0.7387346029281616, 0.5227199792861938]
```

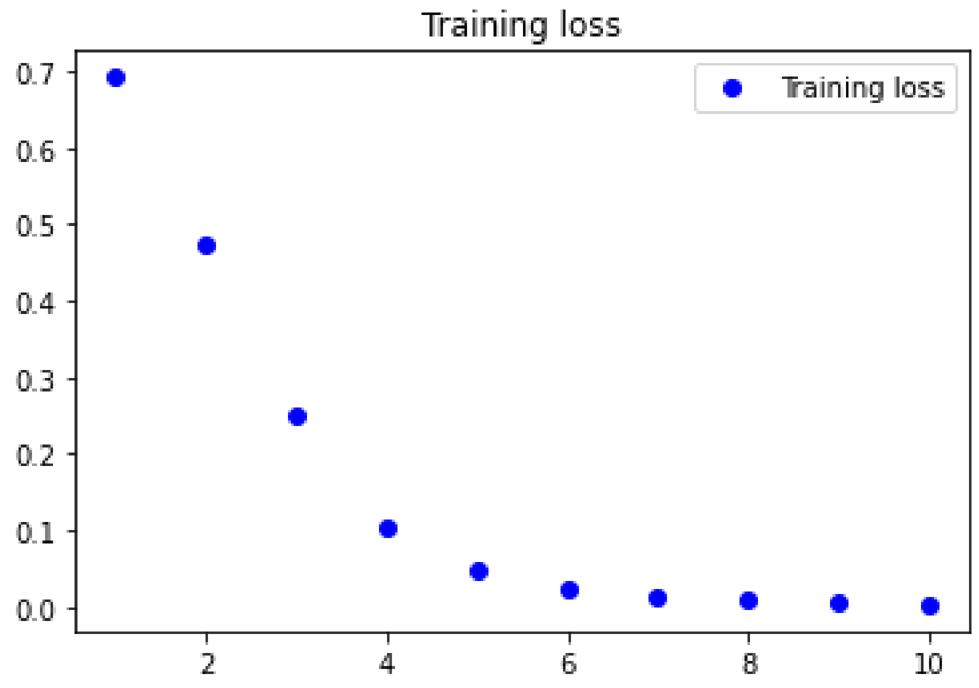
[20]:
```
#### The accuracy is 53.68 / 52.34 / 55.11
```

[21]:
```
# Plotting the results from the test set

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

plot_train_val(acc,val_acc,loss,val_loss)
```





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.

[22]:
```
# Processing the labels of the raw IMDB data
train_dir = imdb_dir.joinpath('train')
print('Loading Training data...')

(input_train, y_train) = load_raw_imdb(train_dir)
print(len(input_train), 'train sequences')

test_dir = imdb_dir.joinpath('test')
print('Loading Test data...')

(input_test, y_test) = load_raw_imdb(test_dir)
print(len(input_test), 'test sequences')
```

```
Loading Training data…
25000 train sequences
Loading Test data…
25000 test sequences
```

[23]:
```
# Using the same train and test data set from the above dataset
# Preparing the dataset differently
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence

# Number of words to consider in the features
max_features = 10000
# Cuts off texts after this many words
# (among the max_features most common words)
maxlen = 500
batch_size = 32

train_dir = imdb_dir.joinpath('train')
test_dir = imdb_dir.joinpath('test')
print('Loading data...')
(input_train, y_train) =  load_raw_imdb(train_dir)
(input_test, y_test) = load_raw_imdb(test_dir)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')

tokenizer = Tokenizer(num_words=max_features)

print('Pad sequences (samples x time)')
tokenizer.fit_on_texts(input_train)
sequences = tokenizer.texts_to_sequences(input_train)
input_train = sequence.pad_sequences(sequences, maxlen=maxlen)

tokenizer.fit_on_texts(input_test)
```

```
sequences = tokenizer.texts_to_sequences(input_test)
input_test = sequence.pad_sequences(sequences, 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)
```

[24]:
```
# Using LSTM layers in keras
# Model summary

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

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
lstm (LSTM)                  (None, 32)                8320
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```

[25]:
```
# Training and evaluating the model

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

```
Epoch 1/10
157/157 [==============================] - 68s 434ms/step - loss: 0.5133 - acc:
0.7466 - val_loss: 0.4263 - val_acc: 0.9150
Epoch 2/10
157/157 [==============================] - 67s 426ms/step - loss: 0.2955 - acc:
0.8846 - val_loss: 0.5512 - val_acc: 0.7664
Epoch 3/10
157/157 [==============================] - 66s 424ms/step - loss: 0.2331 - acc:
0.9118 - val_loss: 0.2378 - val_acc: 0.9088
Epoch 4/10
157/157 [==============================] - 67s 426ms/step - loss: 0.1982 - acc:
0.9261 - val_loss: 0.4095 - val_acc: 0.8612
Epoch 5/10
157/157 [==============================] - 68s 433ms/step - loss: 0.1721 - acc:
0.9380 - val_loss: 0.7598 - val_acc: 0.7462
Epoch 6/10
157/157 [==============================] - 67s 424ms/step - loss: 0.1592 - acc:
0.9423 - val_loss: 1.0047 - val_acc: 0.6344
Epoch 7/10
157/157 [==============================] - 68s 430ms/step - loss: 0.1386 - acc:
0.9508 - val_loss: 0.3880 - val_acc: 0.8498
Epoch 8/10
157/157 [==============================] - 74s 472ms/step - loss: 0.1242 - acc:
0.9576 - val_loss: 0.6583 - val_acc: 0.7432
Epoch 9/10
157/157 [==============================] - 74s 473ms/step - loss: 0.1172 - acc:
0.9605 - val_loss: 0.4909 - val_acc: 0.8358
Epoch 10/10
157/157 [==============================] - 74s 474ms/step - loss: 0.1061 - acc:
0.9645 - val_loss: 0.6454 - val_acc: 0.7912
```

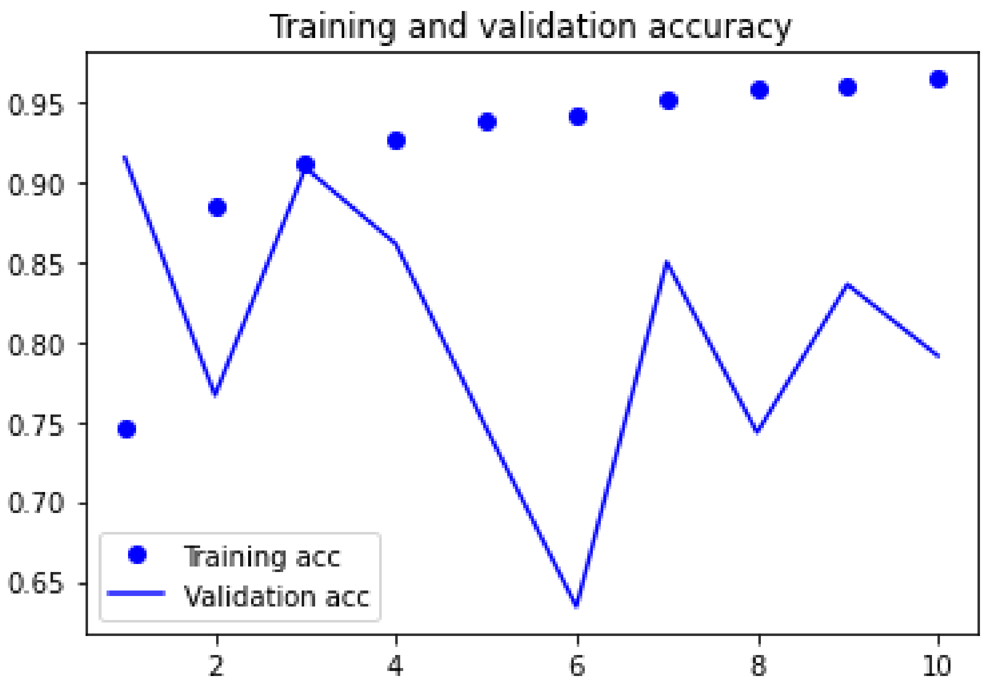
Accuracy = 96.45

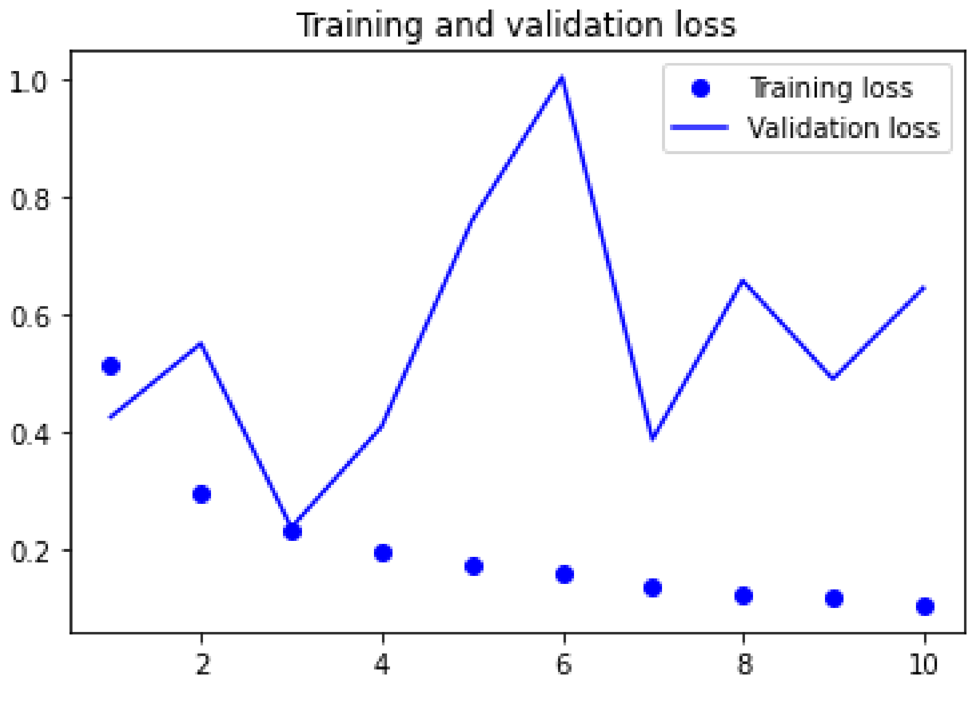
[26]:
```
#saving the model
model.save_weights('LSTM_model.h5')
```

[27]:
```
# Plotting the results from Training and validation accurary

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

plot_train_val(acc,val_acc,loss,val_loss)
```





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.

[28]:
```
# Defining a model with simple 1D convnet on the same IMDB raw data

from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=maxlen))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 128)          1280000

_________________________________________________________________
conv1d (Conv1D)              (None, 494, 32)           28704

_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 98, 32)            0

_________________________________________________________________
conv1d_1 (Conv1D)            (None, 92, 32)            7200

_________________________________________________________________
global_max_pooling1d (Global (None, 32)                0

_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0

_________________________________________________________________
```

[29]:
```
# Training and evaluating the model

model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

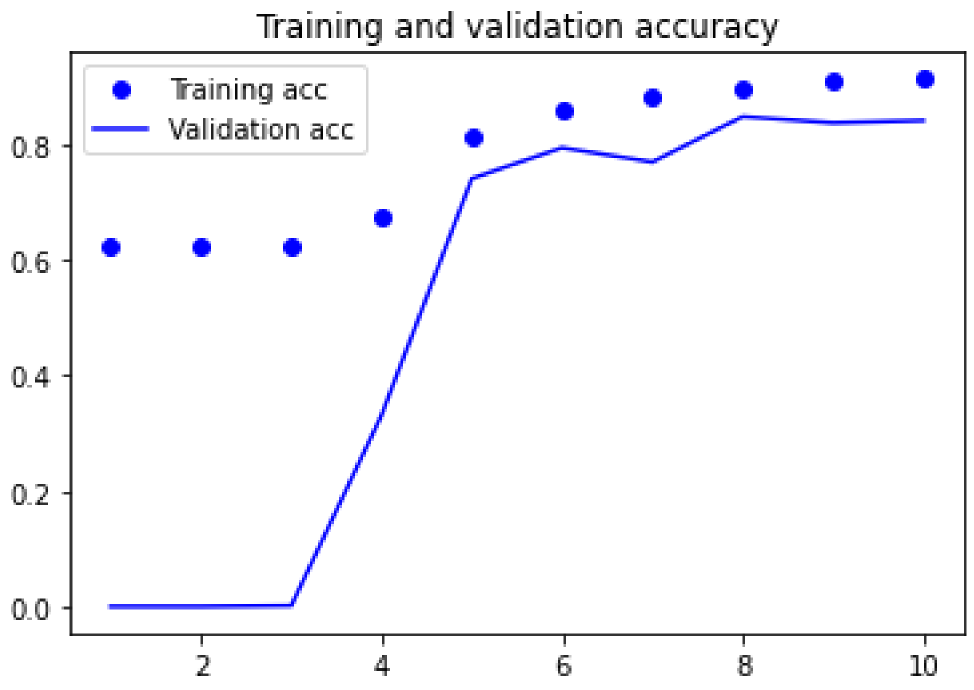
```
Epoch 1/10
157/157 [==============================] - 15s 98ms/step - loss: 0.6636 - acc:
0.6239 - val_loss: 0.9732 - val_acc: 0.0000e+00
Epoch 2/10
157/157 [==============================] - 16s 101ms/step - loss: 0.6522 - acc:
0.6250 - val_loss: 0.9588 - val_acc: 0.0000e+00
Epoch 3/10
157/157 [==============================] - 16s 101ms/step - loss: 0.6306 - acc:
0.6251 - val_loss: 0.9168 - val_acc: 0.0014
Epoch 4/10
157/157 [==============================] - 16s 105ms/step - loss: 0.5666 - acc:
0.6758 - val_loss: 0.8132 - val_acc: 0.3312
Epoch 5/10
157/157 [==============================] - 16s 104ms/step - loss: 0.4559 - acc:
0.8142 - val_loss: 0.5637 - val_acc: 0.7414
Epoch 6/10
157/157 [==============================] - 16s 103ms/step - loss: 0.3622 - acc:
0.8589 - val_loss: 0.4786 - val_acc: 0.7946
Epoch 7/10
157/157 [==============================] - 16s 100ms/step - loss: 0.3033 - acc:
0.8810 - val_loss: 0.5067 - val_acc: 0.7698
Epoch 8/10
157/157 [==============================] - 15s 96ms/step - loss: 0.2674 - acc:
0.8954 - val_loss: 0.3735 - val_acc: 0.8486
Epoch 9/10
157/157 [==============================] - 13s 82ms/step - loss: 0.2416 - acc:
0.9087 - val_loss: 0.3900 - val_acc: 0.8378
Epoch 10/10
157/157 [==============================] - 11s 72ms/step - loss: 0.2213 - acc:
0.9144 - val_loss: 0.3765 - val_acc: 0.8418
```

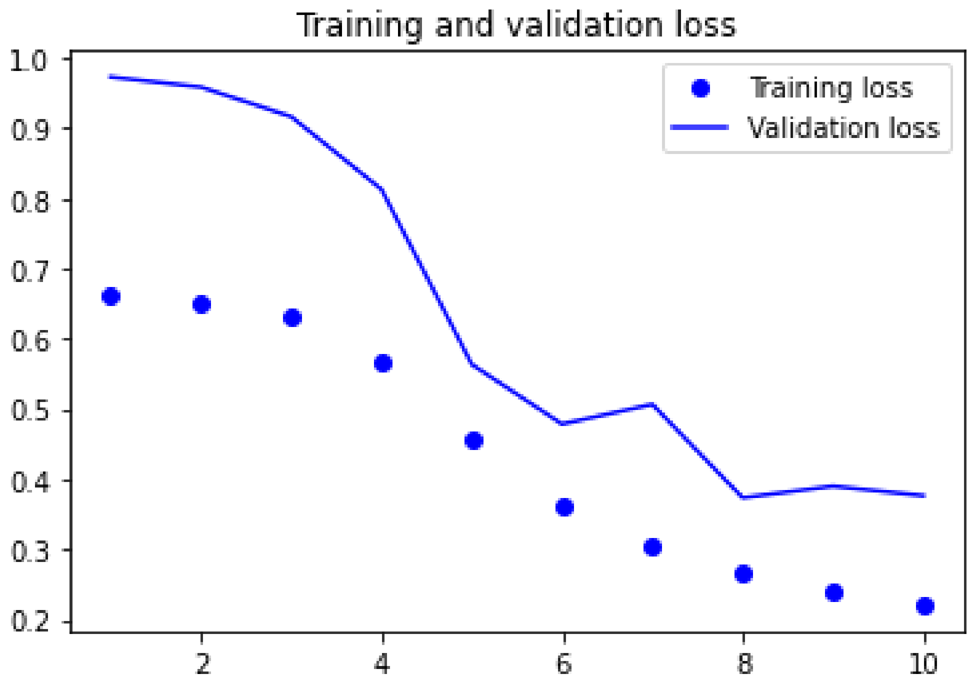
[30]:
```
#saving the model
model.save_weights('Conv1D_model.h5')
```

[31]:
```
# Plotting the results from Training and validation accurary

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

plot_train_val(acc,val_acc,loss,val_loss)
```


Training and validation accuracy
Training acc
Validation acc
0.0
0.2
0.4
0.6
0.8
2
4
6
8
10


Training and validation loss
Training loss
Validation loss
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
2
4
6
8
10

```
[ ]:
```