# Assignment10KudaimiBilal

January 28, 2022

[1]:
```python
#Importing the necessary libraries
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras import layers
from keras.layers import Embedding, Flatten, Dense
from keras.preprocessing.sequence import pad_sequences
from keras.layers import LSTM
from keras.optimizers import RMSprop
import matplotlib.pyplot as plt
import numpy as np
import os

#10.1a

def tokenize(sentence):
    #Splitting each word and removing all punctuation
    tokens = []
    word_list = sentence.split(' ')
    punctuation = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
    for word in word_list:
        for char in word:
            if char in punctuation:
                word = word.replace(char, "")
        word = word.lower()
        tokens.append(word)
    # tokenize the sentence
    return tokens
```

[2]:
```python
#10.1b

#Splitting a list of tokens into n-grams
def ngram(tokens, n):
    ngrams = zip(*[tokens[i:] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]
    return ngrams
```

[3]:
```python
#10.1c

#One-hot encoding a list of tokens
def one_hot_encode(tokens, num_words):
    labelencoder = LabelEncoder()
    int_encode = labelencoder.fit_transform(tokens)
    int_encode = int_encode.reshape(len(int_encode), num_words)

    OHencode = OneHotEncoder(sparse = False)
    encoded = OHencode.fit_transform(int_encode)
    return encoded
```

[4]:
```python
#Testing the above three functions
gettysburg = '''
Four score and seven years ago our fathers brought forth on this continent, a␣
 ↪new nation, conceived in Liberty, and dedicated to the proposition that all␣
 ↪men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any␣
 ↪nation so conceived and so dedicated, can long endure. We are met on a great␣
 ↪battle-field of that war. We have come to dedicate a portion of that field,␣
 ↪as a final resting place for those who here gave their lives that that␣
 ↪nation might live. It is altogether fitting and proper that we should do␣
 ↪this.

But, in a larger sense, we can not dedicate-we can not consecrate-we can not␣
 ↪hallow-this ground. The brave men, living and dead, who struggled here, have␣
 ↪consecrated it, far above our poor power to add or detract. The world will␣
 ↪little note, nor long remember what we say here, but it can never forget␣
 ↪what they did here. It is for us the living, rather, to be dedicated here to␣
 ↪the unfinished work which they who fought here have thus far so nobly␣
 ↪advanced. It is rather for us to be here dedicated to the great task␣
 ↪remaining before us-that from these honored dead we take increased devotion␣
 ↪to that cause for which they gave the last full measure of devotion-that we␣
 ↪here highly resolve that these dead shall not have died in vain-that this␣
 ↪nation, under God, shall have a new birth of freedom-and that government of␣
 ↪the people, by the people, for the people, shall not perish from the earth.
'''

tokens = tokenize(gettysburg)
ng = ngram(tokens, 1)
OHencode = one_hot_encode(ng, 1)
print('One-hot encoded string: {}'.format(OHencode))
```

```
One-hot encoded string: [[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.]]
```

[5]:
```
#10.2
#Training a sequential model with embeddings on the IMDB data using section 6.
↪16-6.18 in the book

#Loading the directories
imdb_dir = '/home/jovyan/dsc650/data/external/imdb/aclImdb/'
test_dir = os.path.join(imdb_dir, 'test')
train_dir = os.path.join(imdb_dir, 'train')
```

[6]:
```
#Getting the IMDb data
labels = []
texts = []
for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname))
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

[7]:
```
#Tokenizing the IMDb data text
maxlen = 100
training_samples = 200
validation_samples = 10000
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 {} unique tokens.'.format(len(word_index)))

data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

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

```
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,)
```

[8]:
```
#Building the neural network
embedding_dim = 100
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length = maxlen))
model.add(Flatten())
model.add(Dense(32, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
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

_________________________________________________________________
```

[9]:
```
#Training the model
model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics =␣
 ↪['acc'])
history = model.fit(x_train, y_train, epochs = 10, batch_size = 5,␣
 ↪validation_data = (x_val, y_val))
```

```
Epoch 1/10
40/40 [==============================] - 4s 98ms/step - loss: 0.6915 - acc:
0.4900 - val_loss: 0.6919 - val_acc: 0.5195
Epoch 2/10
```

```
40/40 [==============================] - 4s 92ms/step - loss: 0.3120 - acc:
0.9850 - val_loss: 0.7328 - val_acc: 0.5167
Epoch 3/10
40/40 [==============================] - 4s 94ms/step - loss: 0.0467 - acc:
0.9950 - val_loss: 0.7420 - val_acc: 0.5165
Epoch 4/10
40/40 [==============================] - 4s 93ms/step - loss: 0.0053 - acc:
1.0000 - val_loss: 0.8568 - val_acc: 0.5187
Epoch 5/10
40/40 [==============================] - 4s 95ms/step - loss: 9.2602e-04 - acc:
1.0000 - val_loss: 0.8215 - val_acc: 0.5188
Epoch 6/10
40/40 [==============================] - 4s 94ms/step - loss: 1.1087e-04 - acc:
1.0000 - val_loss: 0.9328 - val_acc: 0.5171
Epoch 7/10
40/40 [==============================] - 4s 95ms/step - loss: 1.2459e-05 - acc:
1.0000 - val_loss: 0.9209 - val_acc: 0.5212
Epoch 8/10
40/40 [==============================] - 4s 99ms/step - loss: 2.1283e-06 - acc:
1.0000 - val_loss: 0.9752 - val_acc: 0.5184
Epoch 9/10
40/40 [==============================] - 4s 99ms/step - loss: 4.5864e-07 - acc:
1.0000 - val_loss: 1.0300 - val_acc: 0.5183
Epoch 10/10
40/40 [==============================] - 4s 95ms/step - loss: 1.3590e-07 - acc:
1.0000 - val_loss: 1.0732 - val_acc: 0.5183
```

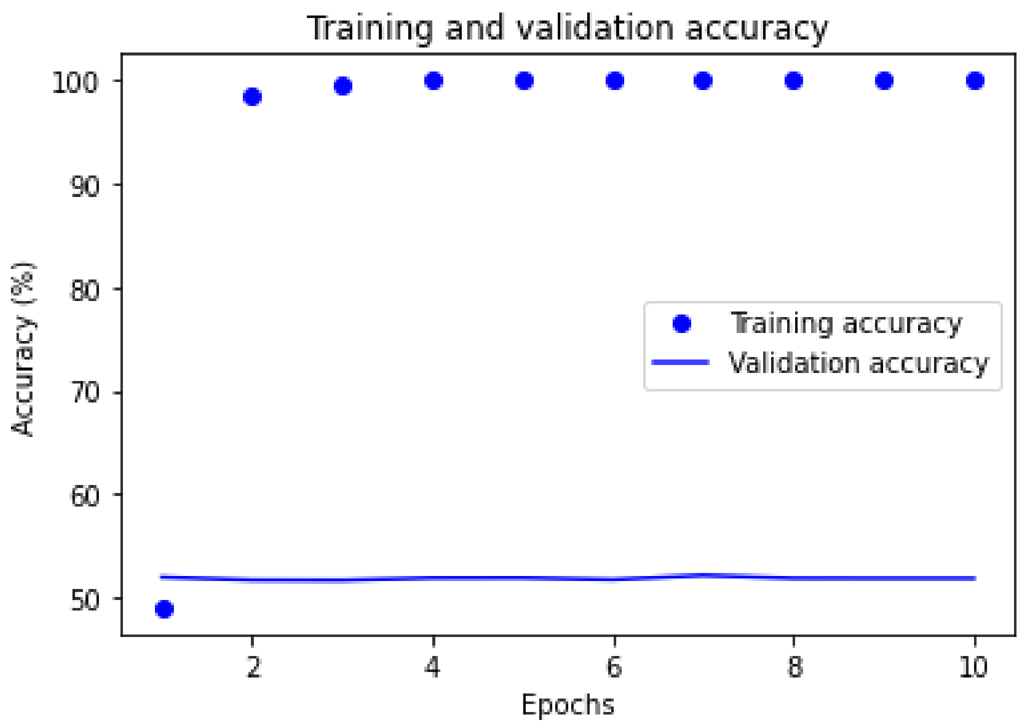
[10]:

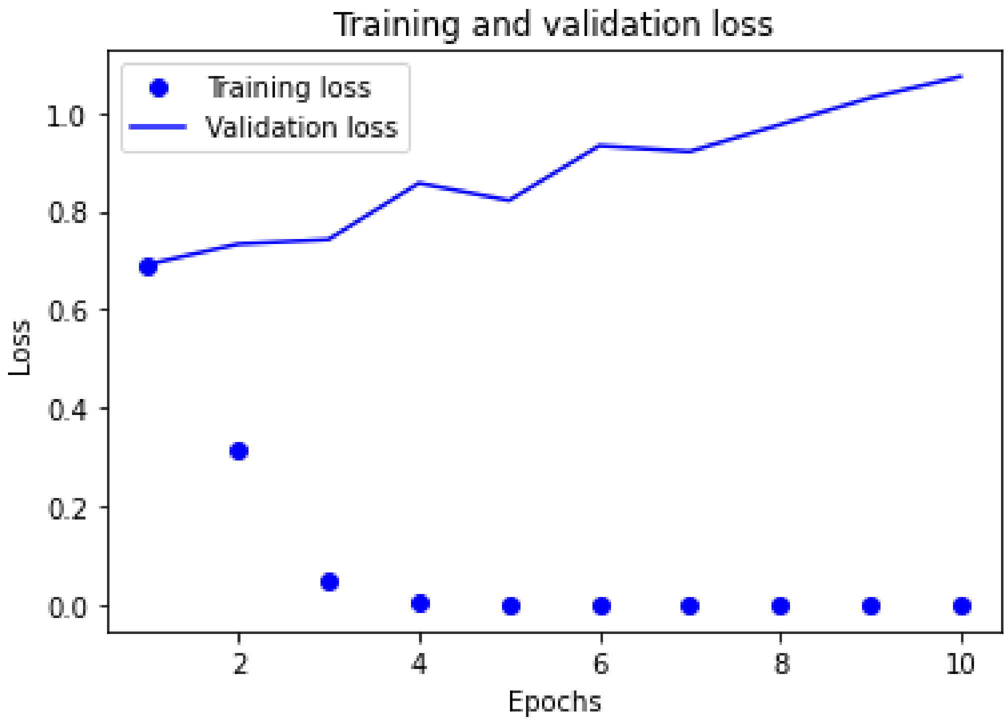
```python
#Plottig the training and validation accuracy and loss
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, [a*100 for a in accuracy], 'bo', label = 'Training accuracy')
plt.plot(epochs, [b*100 for b in val_accuracy], 'b', label = 'Validation␣
 ↪accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
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.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.title('Training and validation loss')
plt.legend()
plt.show()
```






```
[11]: #Getting and tokenizing the testing dataset for evaluation
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))
        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)
```

```
[12]: #Getting the testing dataset lengths
print('Length of testing data: {}'.format(len(x_test)))
print('Length of testing labels: {}'.format(len(y_test)))
```

```
Length of testing data: 12500
```

```
Length of testing labels: 12500
```

[13]:
```
#Evaluating the model on the testing dataset
result = model.evaluate(x_test, y_test)
```

```
391/391 [==============================] - 1s 3ms/step - loss: 0.7781 - acc:
0.6365
```

[14]:
```
#Printing the evaluation accuracy
print('Evaluation accuracy: {} percent'.format(round(result[1], 3)*100))
```

```
Evaluation accuracy: 63.6 percent
```

[15]:
```
#10.3

#Defining a model with an LSTM layer using 6.27 in the book.
max_features = 10000
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation = 'sigmoid'))
```

[16]:
```
#Training the model
model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics =␣
 ↪['acc'])
history = model.fit(x_train, y_train, epochs = 10, batch_size = 128,␣
 ↪validation_split = 0.2)
```

```
Epoch 1/10
2/2 [==============================] - 0s 231ms/step - loss: 0.6933 - acc:
0.5188 - val_loss: 0.6946 - val_acc: 0.3750
Epoch 2/10
2/2 [==============================] - 0s 75ms/step - loss: 0.6882 - acc: 0.7312
- val_loss: 0.6956 - val_acc: 0.4000
Epoch 3/10
2/2 [==============================] - 0s 75ms/step - loss: 0.6814 - acc: 0.8250
- val_loss: 0.6933 - val_acc: 0.4250
Epoch 4/10
2/2 [==============================] - 0s 78ms/step - loss: 0.6718 - acc: 0.9500
- val_loss: 0.6996 - val_acc: 0.4000
Epoch 5/10
2/2 [==============================] - 0s 80ms/step - loss: 0.6546 - acc: 0.9000
- val_loss: 0.6909 - val_acc: 0.5250
Epoch 6/10
2/2 [==============================] - 0s 76ms/step - loss: 0.6162 - acc: 0.9625
- val_loss: 0.7576 - val_acc: 0.4000
Epoch 7/10
2/2 [==============================] - 0s 78ms/step - loss: 0.5086 - acc: 0.8125
```

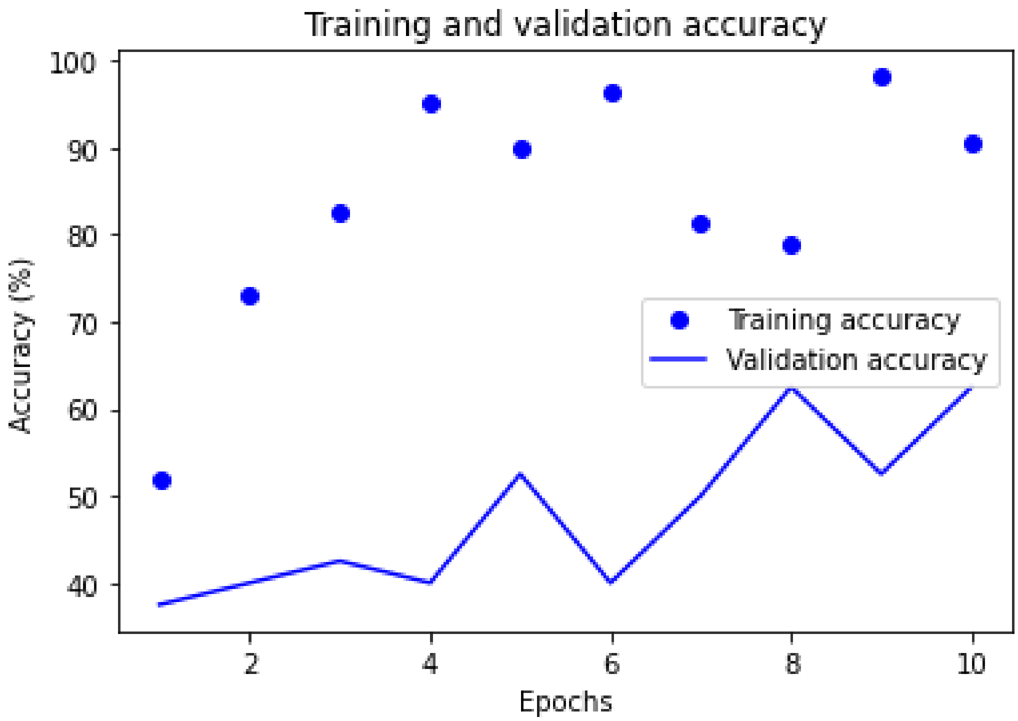
```
- val_loss: 1.0097 - val_acc: 0.5000
Epoch 8/10
2/2 [==============================] - 0s 85ms/step - loss: 0.5154 - acc: 0.7875
- val_loss: 0.6376 - val_acc: 0.6250
Epoch 9/10
2/2 [==============================] - 0s 78ms/step - loss: 0.3007 - acc: 0.9812
- val_loss: 0.9582 - val_acc: 0.5250
Epoch 10/10
2/2 [==============================] - 0s 74ms/step - loss: 0.3267 - acc: 0.9062
- val_loss: 0.6981 - val_acc: 0.6250
```

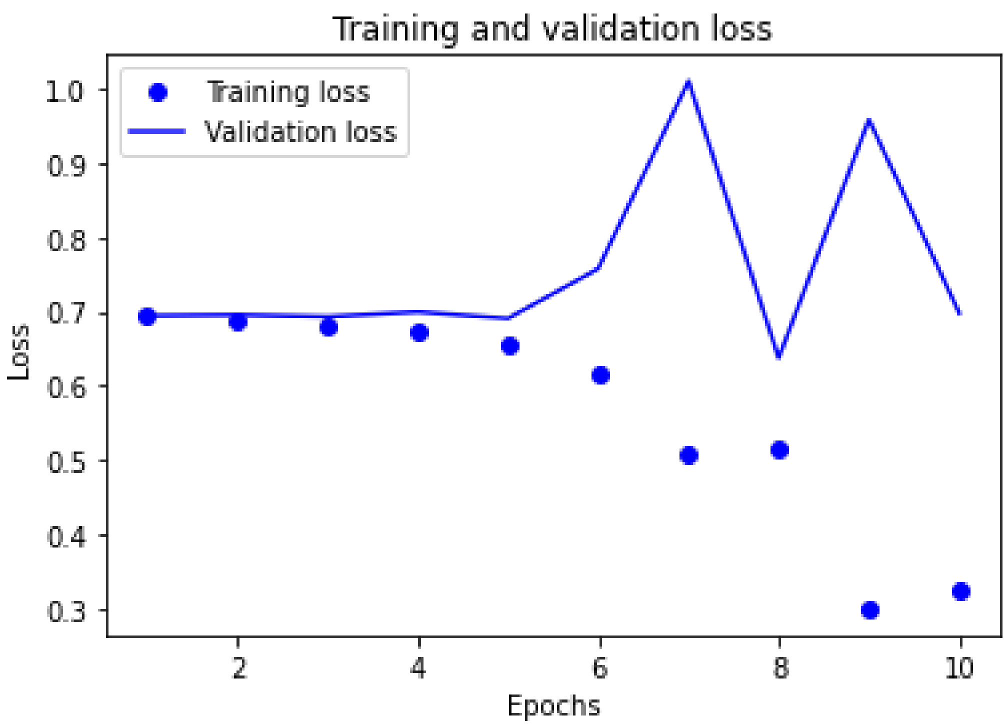
[17]:

```
#Plotting the training and validation accuracy and loss
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, [a*100 for a in accuracy], 'bo', label = 'Training accuracy')
plt.plot(epochs, [b*100 for b in val_accuracy], 'b', label = 'Validation␣
↪accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
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.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training accuracy
Validation accuracy
Accuracy (%)
Epochs
100
90
80
70
60
50
40
2
4
6
8
10


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

[18]:
```
#Getting and tokenizing the testing dataset for evaluation
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))
        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)
```

[19]:
```
#Evaluating the model on the testing dataset
result = model.evaluate(x_test, y_test)
```

```
391/391 [==============================] - 7s 19ms/step - loss: 0.5625 - acc:
0.7198
```

[20]:
```
#Printing the evaluation accuracy
print('Evaluation accuracy: {} percent'.format(round(result[1], 3)*100))
```

```
Evaluation accuracy: 72.0 percent
```

[21]:
```
#10.4

#Building a 1D convnet to fit the data using 6.46 in the book.
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))
```

[22]:
```
#Summarizing the model
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
embedding_2 (Embedding)      (None, 100, 128)         1280000
_________________________________________________________________
conv1d (Conv1D)              (None, 94, 32)           28704
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 18, 32)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 12, 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
_________________________________________________________________
```

[23]:

```
#Training the model
model.compile(optimizer = RMSprop(lr = 1e-4), loss = 'binary_crossentropy',␣
↪metrics = ['acc'])
history = model.fit(x_train, y_train, epochs = 10, batch_size = 128,␣
↪validation_split = 0.2)
```

```
Epoch 1/10
2/2 [==============================] - 0s 73ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 2/10
2/2 [==============================] - 0s 26ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 3/10
2/2 [==============================] - 0s 26ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 4/10
2/2 [==============================] - 0s 25ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 5/10
2/2 [==============================] - 0s 28ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 6/10
2/2 [==============================] - 0s 28ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 7/10
2/2 [==============================] - 0s 28ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 8/10
2/2 [==============================] - 0s 28ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 9/10
```

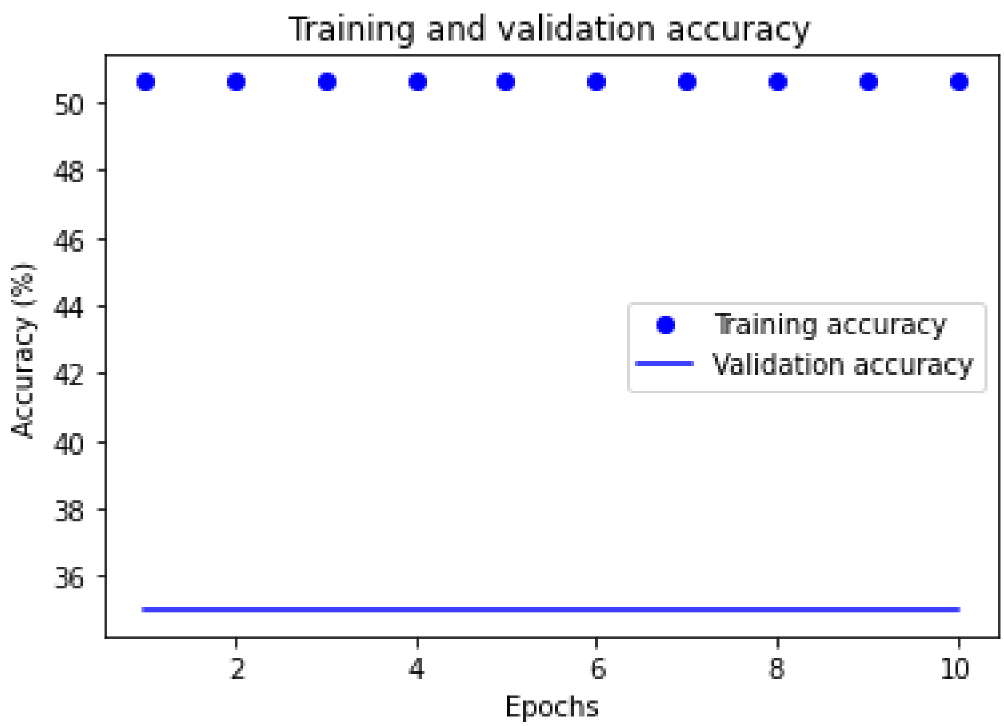
```
2/2 [==============================] - 0s 25ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
Epoch 10/10
2/2 [==============================] - 0s 31ms/step - loss: 7.6161 - acc: 0.5063
- val_loss: 10.0262 - val_acc: 0.3500
```

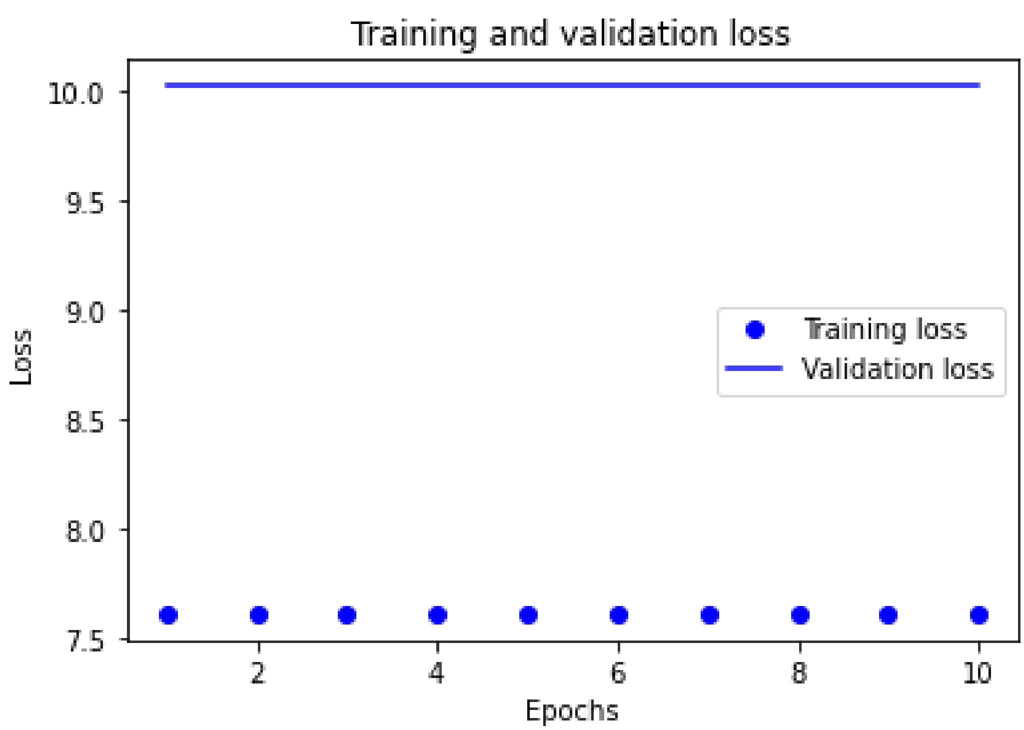
[24]:

```
#Plotting the training and validation accuracy and loss
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, [a*100 for a in accuracy], 'bo', label = 'Training accuracy')
plt.plot(epochs, [b*100 for b in val_accuracy], 'b', label = 'Validation␣
↪accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
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.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Accuracy (%)
50
48
46
44
42
40
38
36
2
4
6
8
10
Epochs
Training accuracy
Validation accuracy


Training and validation loss
Loss
10.0
9.5
9.0
8.5
8.0
7.5
2
4
6
8
10
Epochs
Training loss
Validation loss

[25]:
```
#Getting and tokenizing the testing dataset for evaluation
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))
        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)
```

[26]:
```
#Evaluating the model on the testing dataset
result = model.evaluate(x_test, y_test)
```

```
391/391 [==============================] - 1s 4ms/step - loss: 15.4249 - acc:
0.0000e+00
```

[27]:
```
#Printing the evaluation accuracy
print('Evaluation accuracy: {} percent'.format(round(result[1], 3)*100))
```

```
Evaluation accuracy: 0.0 percent
```