Assignment 10

November 9, 2020

1 Assignment 10

Brandon Sams

07Nov2020

1.1 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.

1.1.1 Assignment 10.1.a

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

```
def tokenize(sentence):
    tokens = []
    # tokenize the sentence
    return tokens
```

[1]:
```
import keras
```

[2]:
```
def tokenize(sentence):
    tokens = keras.preprocessing.text.text_to_word_sequence(sentence)
    return(tokens)
```

[3]:
```
tokens = tokenize("The quick brown fox jumped over the lazy dog.")
tokens
```

[3]:
```
['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog']
```

1.1.2 Assignment 10.1.b

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

```
def ngram(tokens, n):
    ngrams = []
    # Create ngrams
    return ngrams
```

[4]:
```
def ngram(tokens, n):
    ngrams = []
    for i in range(len(tokens)-n+1):
        ngram = ' '.join(word_list for word_list in tokens[i:i+n])
        ngrams.append(ngram)
    return(ngrams)
```

[5]:
```
ngram = ngram(tokens,4)
ngram
```

[5]:
```
['the quick brown fox',
 'quick brown fox jumped',
 'brown fox jumped over',
 'fox jumped over the',
 'jumped over the lazy',
 'over the lazy dog']
```

1.1.3 Assignment 10.1.c

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

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

[6]:
```
def one_hot_encode(tokens, num_words = len(set(tokens))):
    num_words += 1 # Add an extra column, as this method always produces an␣
↪empty first column
    tokenizer = keras.preprocessing.text.Tokenizer(num_words = num_words)
    tokenizer.fit_on_texts(tokens)
    sequences = tokenizer.texts_to_sequences(tokens)
    results = tokenizer.texts_to_matrix(tokens, mode='binary')
    results = results[:, 1:] # Remove first column, as it is always zeros
    token_index = tokenizer.word_index
    return(results)
```

[7]:
```
one_hot_encode(tokens,15)
```

[7]:
```
array([[1., 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., 1., 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., 1., 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.],
       [1., 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., 1., 0., 0., 0., 0., 0., 0., 0.]])
```

[8]:
```
one_hot_encode(tokens)
```

[8]:
```
array([[1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.]])
```

1.2 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.

[9]:
```
import os
import numpy as np

imdb_dir = '/home/jovyan/dsc650/data/external/imdb/aclImdb'
train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

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

[10]:
```
max_words = 10000
embedding_dim = 100
maxlen = 100
training_samples = 200
validation_samples = 10000
```

[11]:
```
tokenizer = keras.preprocessing.text.Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

data = keras.preprocessing.sequence.pad_sequences(sequences,maxlen=maxlen)
labels = np.asarray(labels)
```

[12]:
```
indices = np.arange(data.shape[0])
np.random.shuffle(indices)

data = data[indices]
labels = labels[indices]
```

[13]:
```
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]
```

[14]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
#model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

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

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          1000000

_________________________________________________________________
dense (Dense)                (None, 100, 32)           3232

_________________________________________________________________
dense_1 (Dense)              (None, 100, 1)            33
=================================================================
Total params: 1,003,265
```

```
Trainable params: 1,003,265
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
7/7 [==============================] - 1s 194ms/step - loss: 0.6922 - acc:
0.5195 - val_loss: 0.6924 - val_acc: 0.5181
Epoch 2/10
7/7 [==============================] - 1s 136ms/step - loss: 0.6854 - acc:
0.6123 - val_loss: 0.6920 - val_acc: 0.5207
Epoch 3/10
7/7 [==============================] - 1s 128ms/step - loss: 0.6802 - acc:
0.6308 - val_loss: 0.6917 - val_acc: 0.5228
Epoch 4/10
7/7 [==============================] - 1s 129ms/step - loss: 0.6745 - acc:
0.6417 - val_loss: 0.6914 - val_acc: 0.5248
Epoch 5/10
7/7 [==============================] - 1s 136ms/step - loss: 0.6680 - acc:
0.6486 - val_loss: 0.6921 - val_acc: 0.5255
Epoch 6/10
7/7 [==============================] - 1s 133ms/step - loss: 0.6600 - acc:
0.6496 - val_loss: 0.6930 - val_acc: 0.5241
Epoch 7/10
7/7 [==============================] - 1s 133ms/step - loss: 0.6520 - acc:
0.6495 - val_loss: 0.6950 - val_acc: 0.5243
Epoch 8/10
7/7 [==============================] - 1s 133ms/step - loss: 0.6429 - acc:
0.6531 - val_loss: 0.6991 - val_acc: 0.5240
Epoch 9/10
7/7 [==============================] - 1s 132ms/step - loss: 0.6347 - acc:
0.6537 - val_loss: 0.7017 - val_acc: 0.5240
Epoch 10/10
7/7 [==============================] - 1s 138ms/step - loss: 0.6263 - acc:
0.6561 - val_loss: 0.7047 - val_acc: 0.5237
```

[15]:

```
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)
```

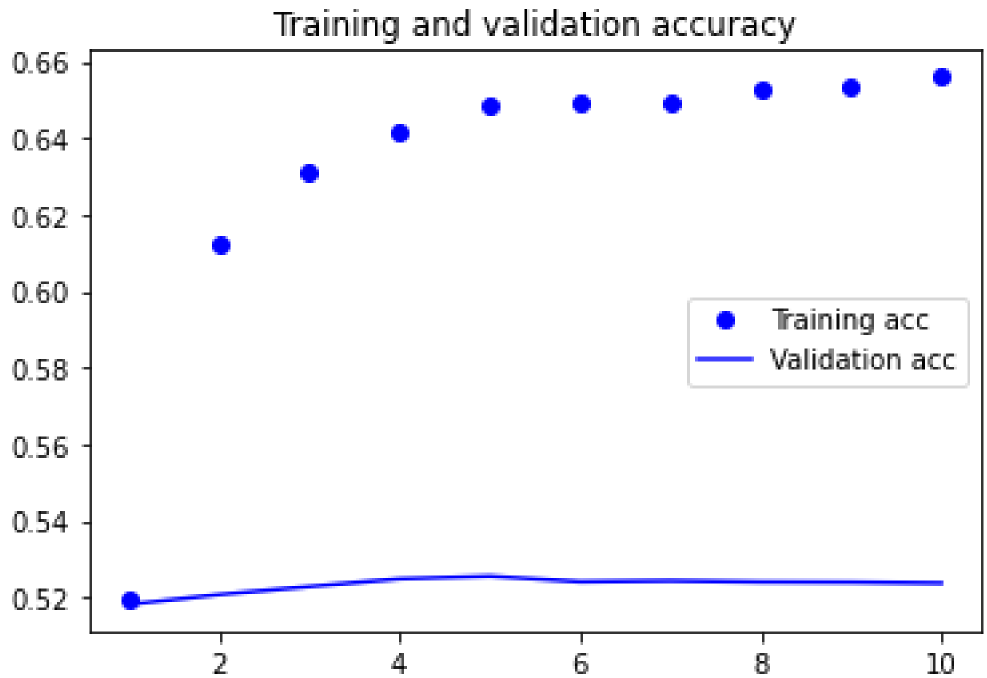
[16]:
```
sequences = tokenizer.texts_to_sequences(texts)
x_test = keras.preprocessing.sequence.pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

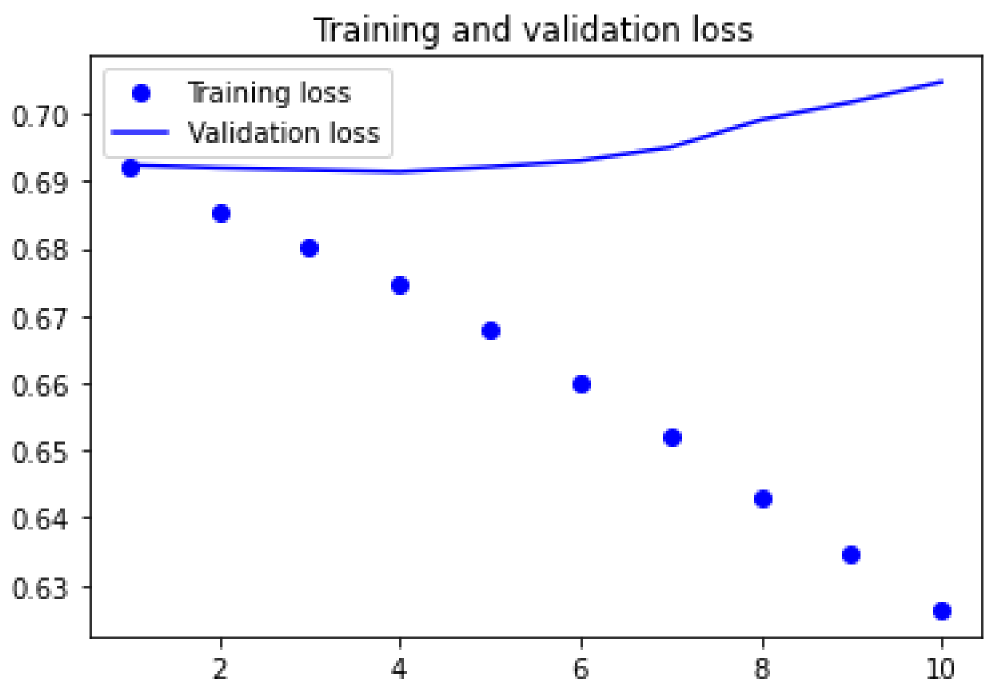
[17]:
```
model.evaluate(x_test,y_test)
```

```
782/782 [==============================] - 2s 3ms/step - loss: 0.7044 - acc:
0.5246
```

[17]: [0.7043618559837341, 0.52463299036026]

[18]:
```
import matplotlib.pyplot as plt

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()
```


Training and validation accuracy
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0.52
2
4
6
8
10
Training acc
Validation acc


Training and validation loss
0.70
0.69
0.68
0.67
0.66
0.65
0.64
0.63
2
4
6
8
10
Training loss
Validation loss

1.3 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.

[19]:
```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

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 [==============================] - 1s 256ms/step - loss: 0.6936 - acc:
0.4875 - val_loss: 0.6892 - val_acc: 0.6500
Epoch 2/10
2/2 [==============================] - 0s 83ms/step - loss: 0.6797 - acc: 0.7250
- val_loss: 0.6854 - val_acc: 0.6500
Epoch 3/10
2/2 [==============================] - 0s 76ms/step - loss: 0.6617 - acc: 0.8313
- val_loss: 0.6742 - val_acc: 0.6000
Epoch 4/10
2/2 [==============================] - 0s 79ms/step - loss: 0.6301 - acc: 0.7563
- val_loss: 0.6489 - val_acc: 0.6250
Epoch 5/10
2/2 [==============================] - 0s 77ms/step - loss: 0.5583 - acc: 0.6125
- val_loss: 0.8138 - val_acc: 0.6250
Epoch 6/10
2/2 [==============================] - 0s 74ms/step - loss: 0.6859 - acc: 0.6562
- val_loss: 0.6261 - val_acc: 0.6500
Epoch 7/10
2/2 [==============================] - 0s 78ms/step - loss: 0.4426 - acc: 0.9875
- val_loss: 0.5958 - val_acc: 0.6750
Epoch 8/10
2/2 [==============================] - 0s 75ms/step - loss: 0.3699 - acc: 0.9875
- val_loss: 0.5915 - val_acc: 0.7000
Epoch 9/10
2/2 [==============================] - 0s 79ms/step - loss: 0.3211 - acc: 0.9875
- val_loss: 0.5916 - val_acc: 0.7000
```

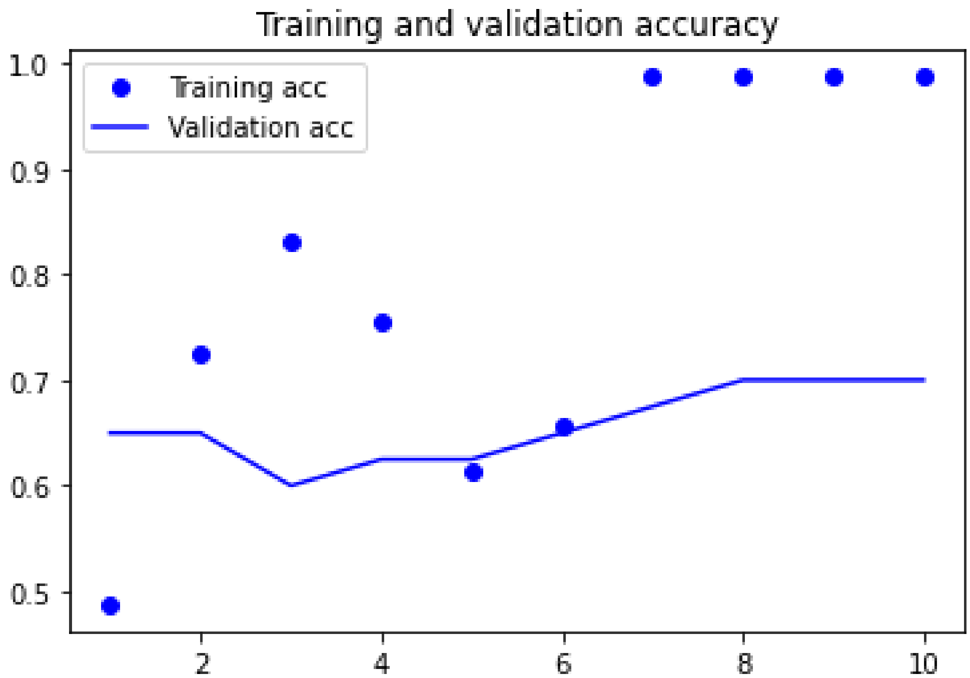
```
Epoch 10/10
2/2 [==============================] - 0s 80ms/step - loss: 0.2798 - acc: 0.9875
- val_loss: 0.5789 - val_acc: 0.7000
```

[20]:
```
model.evaluate(x_test,y_test)
```

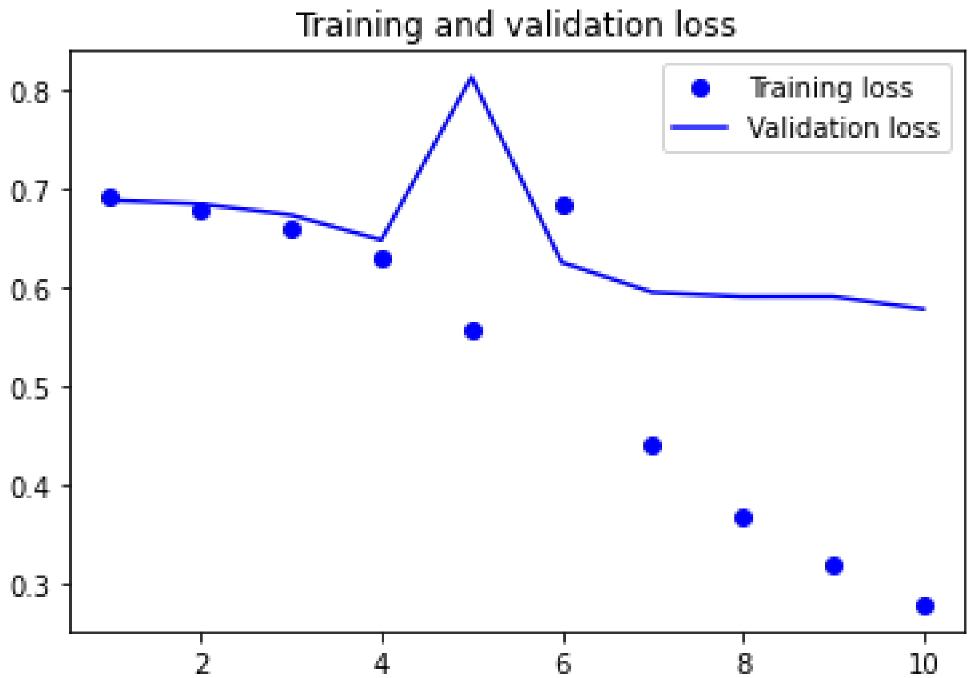
```
782/782 [==============================] - 15s 19ms/step - loss: 0.7062 - acc:
0.6226
```

[20]:
```
[0.7062131762504578, 0.6226000189781189]
```

[21]:
```
import matplotlib.pyplot as plt

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()
```



Training and validation accuracy
Training acc
Validation acc
1.0
0.9
0.8
0.7
0.6
0.5
2
4
6
8
10



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

1.4 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.

[22]:
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

max_features = 10000
max_len = 100

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(1))

model.summary()

model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=100,
                    batch_size=128,
                    validation_split=0.2)
```

```
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
_________________________________________________________________
flatten (Flatten)            (None, 384)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 385
=================================================================
Total params: 1,316,289
```

```
Trainable params: 1,316,289
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
2/2 [==============================] - 0s 69ms/step - loss: 6.0584 - acc: 0.4750
- val_loss: 5.1235 - val_acc: 0.3750
Epoch 2/100
2/2 [==============================] - 0s 28ms/step - loss: 3.8327 - acc: 0.4750
- val_loss: 2.9919 - val_acc: 0.3750
Epoch 3/100
2/2 [==============================] - 0s 28ms/step - loss: 2.3551 - acc: 0.4750
- val_loss: 2.2869 - val_acc: 0.3750
Epoch 4/100
2/2 [==============================] - 0s 26ms/step - loss: 1.9890 - acc: 0.4750
- val_loss: 2.0544 - val_acc: 0.3750
Epoch 5/100
2/2 [==============================] - 0s 25ms/step - loss: 1.6679 - acc: 0.4750
- val_loss: 1.8484 - val_acc: 0.3750
Epoch 6/100
2/2 [==============================] - 0s 25ms/step - loss: 1.3634 - acc: 0.4750
- val_loss: 1.8139 - val_acc: 0.3750
Epoch 7/100
2/2 [==============================] - 0s 26ms/step - loss: 1.3225 - acc: 0.4750
- val_loss: 1.7766 - val_acc: 0.3750
Epoch 8/100
2/2 [==============================] - 0s 28ms/step - loss: 1.2877 - acc: 0.4750
- val_loss: 1.7473 - val_acc: 0.3750
Epoch 9/100
2/2 [==============================] - 0s 25ms/step - loss: 1.2599 - acc: 0.4750
- val_loss: 1.7167 - val_acc: 0.3750
Epoch 10/100
2/2 [==============================] - 0s 24ms/step - loss: 1.2314 - acc: 0.4750
- val_loss: 1.6871 - val_acc: 0.3750
Epoch 11/100
2/2 [==============================] - 0s 32ms/step - loss: 1.2045 - acc: 0.4750
- val_loss: 1.6537 - val_acc: 0.3750
Epoch 12/100
2/2 [==============================] - 0s 28ms/step - loss: 1.1758 - acc: 0.4750
- val_loss: 1.6250 - val_acc: 0.3750
Epoch 13/100
2/2 [==============================] - 0s 27ms/step - loss: 1.1499 - acc: 0.4750
- val_loss: 1.5939 - val_acc: 0.3750
Epoch 14/100
2/2 [==============================] - 0s 24ms/step - loss: 1.1241 - acc: 0.4750
- val_loss: 1.5680 - val_acc: 0.3750
Epoch 15/100
2/2 [==============================] - 0s 29ms/step - loss: 1.1013 - acc: 0.4750
- val_loss: 1.5397 - val_acc: 0.3750
```

```
Epoch 16/100
2/2 [==============================] - 0s 29ms/step - loss: 1.0771 - acc: 0.4750
- val_loss: 1.5084 - val_acc: 0.3750
Epoch 17/100
2/2 [==============================] - 0s 25ms/step - loss: 1.0515 - acc: 0.4750
- val_loss: 1.4786 - val_acc: 0.3750
Epoch 18/100
2/2 [==============================] - 0s 21ms/step - loss: 1.0267 - acc: 0.4750
- val_loss: 1.4439 - val_acc: 0.3750
Epoch 19/100
2/2 [==============================] - 0s 27ms/step - loss: 0.9998 - acc: 0.4750
- val_loss: 1.4161 - val_acc: 0.3750
Epoch 20/100
2/2 [==============================] - 0s 29ms/step - loss: 0.9768 - acc: 0.4750
- val_loss: 1.3884 - val_acc: 0.3750
Epoch 21/100
2/2 [==============================] - 0s 26ms/step - loss: 0.9544 - acc: 0.4750
- val_loss: 1.3643 - val_acc: 0.3750
Epoch 22/100
2/2 [==============================] - 0s 29ms/step - loss: 0.9342 - acc: 0.4750
- val_loss: 1.3398 - val_acc: 0.3750
Epoch 23/100
2/2 [==============================] - 0s 27ms/step - loss: 0.9134 - acc: 0.4750
- val_loss: 1.3100 - val_acc: 0.3750
Epoch 24/100
2/2 [==============================] - 0s 26ms/step - loss: 0.8899 - acc: 0.4750
- val_loss: 1.2834 - val_acc: 0.3750
Epoch 25/100
2/2 [==============================] - 0s 50ms/step - loss: 0.8681 - acc: 0.4750
- val_loss: 1.2580 - val_acc: 0.3750
Epoch 26/100
2/2 [==============================] - 0s 26ms/step - loss: 0.8469 - acc: 0.4750
- val_loss: 1.2318 - val_acc: 0.3750
Epoch 27/100
2/2 [==============================] - 0s 27ms/step - loss: 0.8255 - acc: 0.4750
- val_loss: 1.2070 - val_acc: 0.3750
Epoch 28/100
2/2 [==============================] - 0s 28ms/step - loss: 0.8048 - acc: 0.4750
- val_loss: 1.1810 - val_acc: 0.3750
Epoch 29/100
2/2 [==============================] - 0s 33ms/step - loss: 0.7840 - acc: 0.4750
- val_loss: 1.1585 - val_acc: 0.3750
Epoch 30/100
2/2 [==============================] - 0s 23ms/step - loss: 0.7650 - acc: 0.4750
- val_loss: 1.1371 - val_acc: 0.3750
Epoch 31/100
2/2 [==============================] - 0s 27ms/step - loss: 0.7460 - acc: 0.4750
- val_loss: 1.1129 - val_acc: 0.3750
```

```
Epoch 32/100
2/2 [==============================] - 0s 27ms/step - loss: 0.7258 - acc: 0.4750
- val_loss: 1.0904 - val_acc: 0.3750
Epoch 33/100
2/2 [==============================] - 0s 28ms/step - loss: 0.7064 - acc: 0.4750
- val_loss: 1.0654 - val_acc: 0.3750
Epoch 34/100
2/2 [==============================] - 0s 24ms/step - loss: 0.6863 - acc: 0.4750
- val_loss: 1.0451 - val_acc: 0.3750
Epoch 35/100
2/2 [==============================] - 0s 29ms/step - loss: 0.6681 - acc: 0.4750
- val_loss: 1.0234 - val_acc: 0.3750
Epoch 36/100
2/2 [==============================] - 0s 27ms/step - loss: 0.6504 - acc: 0.4750
- val_loss: 1.0096 - val_acc: 0.3750
Epoch 37/100
2/2 [==============================] - 0s 32ms/step - loss: 0.6350 - acc: 0.4750
- val_loss: 0.9885 - val_acc: 0.3750
Epoch 38/100
2/2 [==============================] - 0s 25ms/step - loss: 0.6167 - acc: 0.4750
- val_loss: 0.9645 - val_acc: 0.3750
Epoch 39/100
2/2 [==============================] - 0s 27ms/step - loss: 0.5980 - acc: 0.4750
- val_loss: 0.9478 - val_acc: 0.3750
Epoch 40/100
2/2 [==============================] - 0s 25ms/step - loss: 0.5820 - acc: 0.4750
- val_loss: 0.9305 - val_acc: 0.3750
Epoch 41/100
2/2 [==============================] - 0s 30ms/step - loss: 0.5660 - acc: 0.4750
- val_loss: 0.9142 - val_acc: 0.3750
Epoch 42/100
2/2 [==============================] - 0s 29ms/step - loss: 0.5510 - acc: 0.4812
- val_loss: 0.9025 - val_acc: 0.3750
Epoch 43/100
2/2 [==============================] - 0s 25ms/step - loss: 0.5367 - acc: 0.5000
- val_loss: 0.8788 - val_acc: 0.3750
Epoch 44/100
2/2 [==============================] - 0s 26ms/step - loss: 0.5188 - acc: 0.5437
- val_loss: 0.8615 - val_acc: 0.3750
Epoch 45/100
2/2 [==============================] - 0s 28ms/step - loss: 0.5037 - acc: 0.5938
- val_loss: 0.8490 - val_acc: 0.3750
Epoch 46/100
2/2 [==============================] - 0s 27ms/step - loss: 0.4899 - acc: 0.6250
- val_loss: 0.8302 - val_acc: 0.3750
Epoch 47/100
2/2 [==============================] - 0s 27ms/step - loss: 0.4747 - acc: 0.7188
- val_loss: 0.8188 - val_acc: 0.3750
```

```
Epoch 48/100
2/2 [==============================] - 0s 25ms/step - loss: 0.4614 - acc: 0.7812
- val_loss: 0.8073 - val_acc: 0.3750
Epoch 49/100
2/2 [==============================] - 0s 24ms/step - loss: 0.4490 - acc: 0.8750
- val_loss: 0.7995 - val_acc: 0.3750
Epoch 50/100
2/2 [==============================] - 0s 25ms/step - loss: 0.4368 - acc: 0.8938
- val_loss: 0.7833 - val_acc: 0.3750
Epoch 51/100
2/2 [==============================] - 0s 27ms/step - loss: 0.4225 - acc: 0.9500
- val_loss: 0.7739 - val_acc: 0.3750
Epoch 52/100
2/2 [==============================] - 0s 30ms/step - loss: 0.4100 - acc: 0.9563
- val_loss: 0.7660 - val_acc: 0.3750
Epoch 53/100
2/2 [==============================] - 0s 26ms/step - loss: 0.3978 - acc: 0.9750
- val_loss: 0.7493 - val_acc: 0.3750
Epoch 54/100
2/2 [==============================] - 0s 25ms/step - loss: 0.3837 - acc: 0.9875
- val_loss: 0.7388 - val_acc: 0.3750
Epoch 55/100
2/2 [==============================] - 0s 24ms/step - loss: 0.3714 - acc: 0.9937
- val_loss: 0.7292 - val_acc: 0.4000
Epoch 56/100
2/2 [==============================] - 0s 25ms/step - loss: 0.3595 - acc: 1.0000
- val_loss: 0.7254 - val_acc: 0.4000
Epoch 57/100
2/2 [==============================] - 0s 25ms/step - loss: 0.3484 - acc: 1.0000
- val_loss: 0.7200 - val_acc: 0.3500
Epoch 58/100
2/2 [==============================] - 0s 23ms/step - loss: 0.3376 - acc: 1.0000
- val_loss: 0.7212 - val_acc: 0.3500
Epoch 59/100
2/2 [==============================] - 0s 25ms/step - loss: 0.3273 - acc: 1.0000
- val_loss: 0.7156 - val_acc: 0.3750
Epoch 60/100
2/2 [==============================] - 0s 29ms/step - loss: 0.3157 - acc: 1.0000
- val_loss: 0.7096 - val_acc: 0.4000
Epoch 61/100
2/2 [==============================] - 0s 26ms/step - loss: 0.3041 - acc: 1.0000
- val_loss: 0.7051 - val_acc: 0.4250
Epoch 62/100
2/2 [==============================] - 0s 34ms/step - loss: 0.2926 - acc: 1.0000
- val_loss: 0.6962 - val_acc: 0.4500
Epoch 63/100
2/2 [==============================] - 0s 29ms/step - loss: 0.2805 - acc: 1.0000
- val_loss: 0.6870 - val_acc: 0.5250
```

```
Epoch 64/100
2/2 [==============================] - 0s 21ms/step - loss: 0.2684 - acc: 0.9937
- val_loss: 0.6815 - val_acc: 0.5250
Epoch 65/100
2/2 [==============================] - 0s 26ms/step - loss: 0.2572 - acc: 0.9937
- val_loss: 0.6783 - val_acc: 0.5750
Epoch 66/100
2/2 [==============================] - 0s 26ms/step - loss: 0.2466 - acc: 0.9937
- val_loss: 0.6817 - val_acc: 0.5250
Epoch 67/100
2/2 [==============================] - 0s 21ms/step - loss: 0.2360 - acc: 1.0000
- val_loss: 0.6780 - val_acc: 0.5750
Epoch 68/100
2/2 [==============================] - 0s 24ms/step - loss: 0.2249 - acc: 0.9937
- val_loss: 0.6729 - val_acc: 0.6000
Epoch 69/100
2/2 [==============================] - 0s 28ms/step - loss: 0.2136 - acc: 0.9875
- val_loss: 0.6694 - val_acc: 0.6000
Epoch 70/100
2/2 [==============================] - 0s 24ms/step - loss: 0.2024 - acc: 0.9875
- val_loss: 0.6632 - val_acc: 0.6250
Epoch 71/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1912 - acc: 0.9875
- val_loss: 0.6616 - val_acc: 0.6500
Epoch 72/100
2/2 [==============================] - 0s 28ms/step - loss: 0.1805 - acc: 0.9875
- val_loss: 0.6634 - val_acc: 0.6250
Epoch 73/100
2/2 [==============================] - 0s 27ms/step - loss: 0.1699 - acc: 0.9875
- val_loss: 0.6625 - val_acc: 0.6250
Epoch 74/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1592 - acc: 0.9875
- val_loss: 0.6661 - val_acc: 0.6250
Epoch 75/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1485 - acc: 0.9875
- val_loss: 0.6645 - val_acc: 0.6250
Epoch 76/100
2/2 [==============================] - 0s 24ms/step - loss: 0.1382 - acc: 0.9875
- val_loss: 0.6674 - val_acc: 0.5750
Epoch 77/100
2/2 [==============================] - 0s 27ms/step - loss: 0.1277 - acc: 0.9875
- val_loss: 0.6684 - val_acc: 0.5750
Epoch 78/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1178 - acc: 0.9875
- val_loss: 0.6710 - val_acc: 0.5750
Epoch 79/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1086 - acc: 0.9875
- val_loss: 0.6779 - val_acc: 0.5500
```

```
Epoch 80/100
2/2 [==============================] - 0s 26ms/step - loss: 0.1000 - acc: 0.9937
- val_loss: 0.6713 - val_acc: 0.6000
Epoch 81/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0902 - acc: 0.9937
- val_loss: 0.6708 - val_acc: 0.5750
Epoch 82/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0820 - acc: 0.9937
- val_loss: 0.6763 - val_acc: 0.5500
Epoch 83/100
2/2 [==============================] - 0s 24ms/step - loss: 0.0744 - acc: 0.9937
- val_loss: 0.6778 - val_acc: 0.5500
Epoch 84/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0663 - acc: 0.9937
- val_loss: 0.6738 - val_acc: 0.6000
Epoch 85/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0587 - acc: 0.9937
- val_loss: 0.6736 - val_acc: 0.6000
Epoch 86/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0518 - acc: 0.9937
- val_loss: 0.6728 - val_acc: 0.6250
Epoch 87/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0453 - acc: 0.9937
- val_loss: 0.6808 - val_acc: 0.5500
Epoch 88/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0394 - acc: 1.0000
- val_loss: 0.6703 - val_acc: 0.6000
Epoch 89/100
2/2 [==============================] - 0s 24ms/step - loss: 0.0343 - acc: 0.9937
- val_loss: 0.6709 - val_acc: 0.6250
Epoch 90/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0298 - acc: 0.9937
- val_loss: 0.6749 - val_acc: 0.5750
Epoch 91/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0257 - acc: 0.9937
- val_loss: 0.6721 - val_acc: 0.6000
Epoch 92/100
2/2 [==============================] - 0s 166ms/step - loss: 0.0227 - acc:
0.9937 - val_loss: 0.6785 - val_acc: 0.5500
Epoch 93/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0198 - acc: 0.9937
- val_loss: 0.6732 - val_acc: 0.6000
Epoch 94/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0177 - acc: 0.9937
- val_loss: 0.6737 - val_acc: 0.6000
Epoch 95/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0152 - acc: 0.9937
- val_loss: 0.6732 - val_acc: 0.6000
```

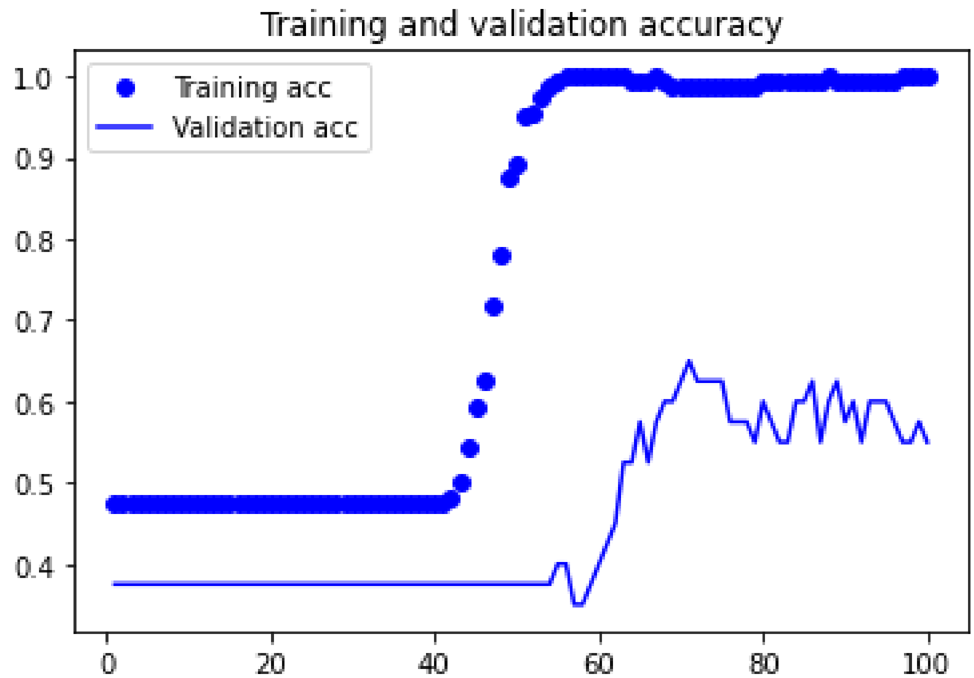
```
Epoch 96/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0138 - acc: 0.9937
- val_loss: 0.6757 - val_acc: 0.5750
Epoch 97/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0126 - acc: 1.0000
- val_loss: 0.6778 - val_acc: 0.5500
Epoch 98/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0113 - acc: 1.0000
- val_loss: 0.6775 - val_acc: 0.5500
Epoch 99/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0104 - acc: 1.0000
- val_loss: 0.6760 - val_acc: 0.5750
Epoch 100/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0100 - acc: 1.0000
- val_loss: 0.6782 - val_acc: 0.5500
```

[23]:
```
model.evaluate(x_test,y_test)
```

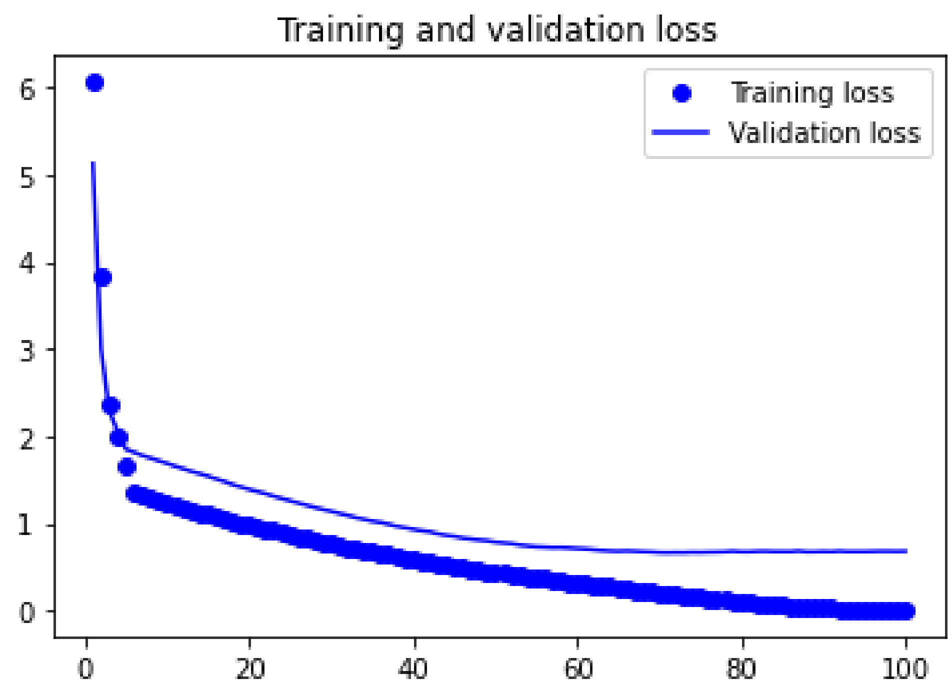
```
782/782 [==============================] - 2s 3ms/step - loss: 0.6900 - acc:
0.5376
```

[23]:
```
[0.690026581287384, 0.5375999808311462]
```

[24]:
```
import matplotlib.pyplot as plt

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()
```



Training and validation accuracy
Training acc
Validation acc
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100



Training and validation loss
Training loss
Validation loss
6
5
4
3
2
1
0
0
20
40
60
80
100

[ ]:

```
[ ]:
```