aml-4

July 27, 2024

To Load the libraries and IMDB Data

[1]:
```
import os
from operator import itemgetter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
get_ipython().magic(u'matplotlib inline')
plt.style.use('ggplot')

import tensorflow as tf

from keras import models, regularizers, layers, optimizers, losses, metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

[16]:
```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# The number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing
from keras.utils import pad_sequences

# Basic Model that shows how embedding and cutoff works:
# Number of words to consider as features
max_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
max_len = 150

# Data should be loaded as lists of integers
```

```
(train_data, train_labels), (test_data, test_labels) = imdb.
 ↪load_data(num_words=max_features)

train_data = train_data[:100]
train_labels = train_labels[:100]

# This turns our lists of integers into a 2D integer tensor of shape
# `(samples, maxlen)`
train_data = pad_sequences(train_data, maxlen=max_len)
test_data = pad_sequences(test_data, maxlen=max_len)

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

model = Sequential()
# We provide our Embedding layer a maximum input length specification
# in order to flatten the embedded inputs later
model.add(Embedding(10000, 8, input_length=max_len))
# After the Embedding layer, our activations have shape `(samples, maxlen, 8)`.

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

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

history = model.fit(train_data, train_labels,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

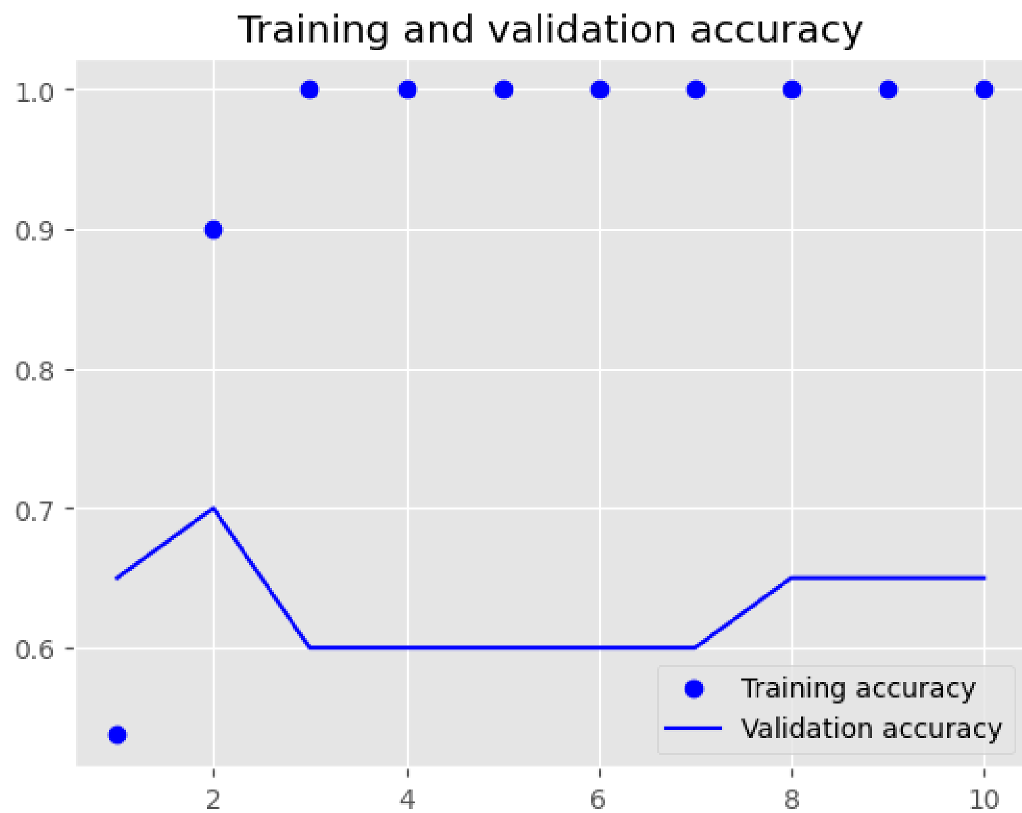
```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_9 (Embedding)     (None, 150, 8)            80000

 flatten_4 (Flatten)         (None, 1200)              0

 dense_4 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
```

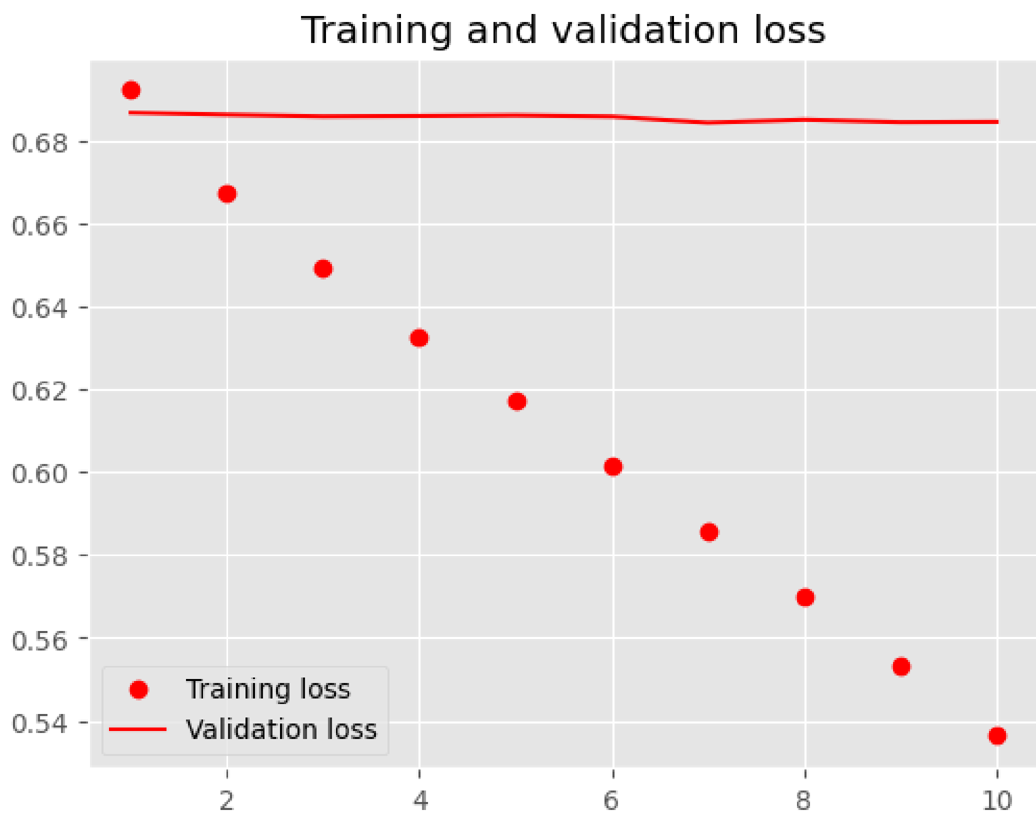
```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
3/3 [==============================] - 1s 184ms/step - loss: 0.6921 - acc:
0.5375 - val_loss: 0.6866 - val_acc: 0.6500
Epoch 2/10
3/3 [==============================] - 0s 30ms/step - loss: 0.6672 - acc: 0.9000
- val_loss: 0.6861 - val_acc: 0.7000
Epoch 3/10
3/3 [==============================] - 0s 35ms/step - loss: 0.6493 - acc: 1.0000
- val_loss: 0.6857 - val_acc: 0.6000
Epoch 4/10
3/3 [==============================] - 0s 35ms/step - loss: 0.6327 - acc: 1.0000
- val_loss: 0.6858 - val_acc: 0.6000
Epoch 5/10
3/3 [==============================] - 0s 33ms/step - loss: 0.6171 - acc: 1.0000
- val_loss: 0.6860 - val_acc: 0.6000
Epoch 6/10
3/3 [==============================] - 0s 31ms/step - loss: 0.6014 - acc: 1.0000
- val_loss: 0.6857 - val_acc: 0.6000
Epoch 7/10
3/3 [==============================] - 0s 34ms/step - loss: 0.5858 - acc: 1.0000
- val_loss: 0.6843 - val_acc: 0.6000
Epoch 8/10
3/3 [==============================] - 0s 39ms/step - loss: 0.5697 - acc: 1.0000
- val_loss: 0.6850 - val_acc: 0.6500
Epoch 9/10
3/3 [==============================] - 0s 47ms/step - loss: 0.5531 - acc: 1.0000
- val_loss: 0.6844 - val_acc: 0.6500
Epoch 10/10
3/3 [==============================] - 0s 42ms/step - loss: 0.5363 - acc: 1.0000
- val_loss: 0.6845 - val_acc: 0.6500
```

[17]:

```python
import matplotlib.pyplot as plt

# Training accuracy
training_acc = history.history["acc"]
# Validation accuracy
validation_acc = history.history["val_acc"]
# Training loss
training_loss = history.history["loss"]
# Validation loss
validation_loss = history.history["val_loss"]

# Plots for each epoch, here 10
epochs = range(1, len(training_acc) + 1)
```

```
plt.plot(epochs, training_acc, "bo", label="Training accuracy")  # "bo" gives␣
  ↪dot plot
plt.plot(epochs, validation_acc, "b", label="Validation accuracy")  # "b" gives␣
  ↪line plot
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, training_loss, "ro", label="Training loss")
plt.plot(epochs, validation_loss, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```





[21]:
```
test_loss, test_accuracy = model.evaluate(test_data, test_labels) # Use the␣
 ↪correct variable names: 'model', 'test_data', and 'test_labels'
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
782/782 [==============================] - 5s 7ms/step - loss: 0.6942 - acc:
0.4980
Test loss: 0.6941725611686707
Test accuracy: 0.4979999594688416
```

Without using an embedding layer to restrict the training, validation, and test samples, the model's accuracy was 0.4979.

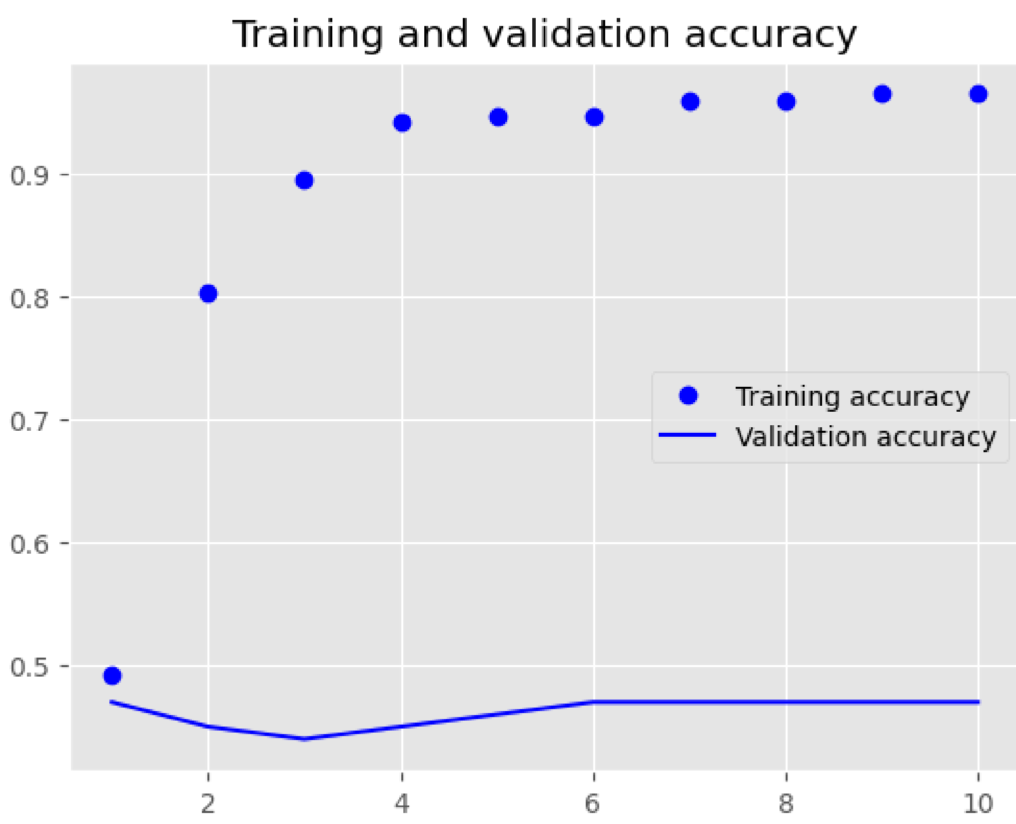
[22]:
```
from keras.layers import Embedding
from keras.datasets import imdb
from keras.utils import pad_sequences
from keras.models import Sequential
from keras.layers import Flatten, Dense
import matplotlib.pyplot as plt

# Number of words to consider as features
```

```python
max_features = 10000
# After this number of words, cut the text (only consider the top max_features most common words)
max_len = 150

# Load the data as lists of integers
(training_data, training_labels), (testing_data, testing_labels) = imdb.load_data(num_words=max_features)

# Limit training data to the first 500 samples
training_data = training_data[:500]
training_labels = training_labels[:500]

# Convert the lists of integers into a 2D integer tensor of shape `(samples, max_len)`
training_data = pad_sequences(training_data, maxlen=max_len)
testing_data = pad_sequences(testing_data, maxlen=max_len)

# Define the model
classification_model = Sequential()
classification_model.add(Embedding(max_features, 8, input_length=max_len))
classification_model.add(Flatten())
classification_model.add(Dense(1, activation='sigmoid'))

classification_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
classification_model.summary()

# Train the model
history = classification_model.fit(training_data, training_labels,
                                   epochs=10,
                                   batch_size=32,
                                   validation_split=0.2)

# Training accuracy
training_accuracy = history.history["acc"]
# Validation accuracy
validation_accuracy = history.history["val_acc"]
# Training loss
training_loss = history.history["loss"]
# Validation loss
validation_loss = history.history["val_loss"]

# Plot the results
epochs = range(1, len(training_accuracy) + 1)

plt.plot(epochs, training_accuracy, "bo", label="Training accuracy")
```

```python
plt.plot(epochs, validation_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, training_loss, "ro", label="Training loss")
plt.plot(epochs, validation_loss, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()

# Evaluate the model on the test data
test_loss, test_accuracy = classification_model.evaluate(testing_data,␣
 ↪testing_labels)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 150, 8)            80000

 flatten_5 (Flatten)         (None, 1200)              0

 dense_5 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
13/13 [==============================] - 1s 31ms/step - loss: 0.6940 - acc:
0.4925 - val_loss: 0.6958 - val_acc: 0.4700
Epoch 2/10
13/13 [==============================] - 0s 10ms/step - loss: 0.6759 - acc:
0.8025 - val_loss: 0.6955 - val_acc: 0.4500
Epoch 3/10
13/13 [==============================] - 0s 11ms/step - loss: 0.6603 - acc:
0.8950 - val_loss: 0.6950 - val_acc: 0.4400
Epoch 4/10
13/13 [==============================] - 0s 11ms/step - loss: 0.6430 - acc:
0.9425 - val_loss: 0.6947 - val_acc: 0.4500
Epoch 5/10
13/13 [==============================] - 0s 10ms/step - loss: 0.6232 - acc:
```

```
0.9475 - val_loss: 0.6945 - val_acc: 0.4600
Epoch 6/10
13/13 [==============================] - 0s 19ms/step - loss: 0.6004 - acc:
0.9475 - val_loss: 0.6942 - val_acc: 0.4700
Epoch 7/10
13/13 [==============================] - 0s 12ms/step - loss: 0.5747 - acc:
0.9600 - val_loss: 0.6941 - val_acc: 0.4700
Epoch 8/10
13/13 [==============================] - 0s 12ms/step - loss: 0.5467 - acc:
0.9600 - val_loss: 0.6941 - val_acc: 0.4700
Epoch 9/10
13/13 [==============================] - 0s 11ms/step - loss: 0.5160 - acc:
0.9650 - val_loss: 0.6946 - val_acc: 0.4700
Epoch 10/10
13/13 [==============================] - 0s 10ms/step - loss: 0.4836 - acc:
0.9650 - val_loss: 0.6950 - val_acc: 0.4700
```




```
782/782 [==============================] - 3s 3ms/step - loss: 0.6918 - acc:
0.5211
Test loss: 0.6917838454246521
Test accuracy: 0.5210800170898438
```

[23]:

```python
# Evaluate the model on the test data
test_loss, test_accuracy = classification_model.evaluate(testing_data,␣
 ↪testing_labels)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.6918 - acc:
0.5211
Test loss: 0.6917838454246521
Test accuracy: 0.5210800170898438
```

[24]:

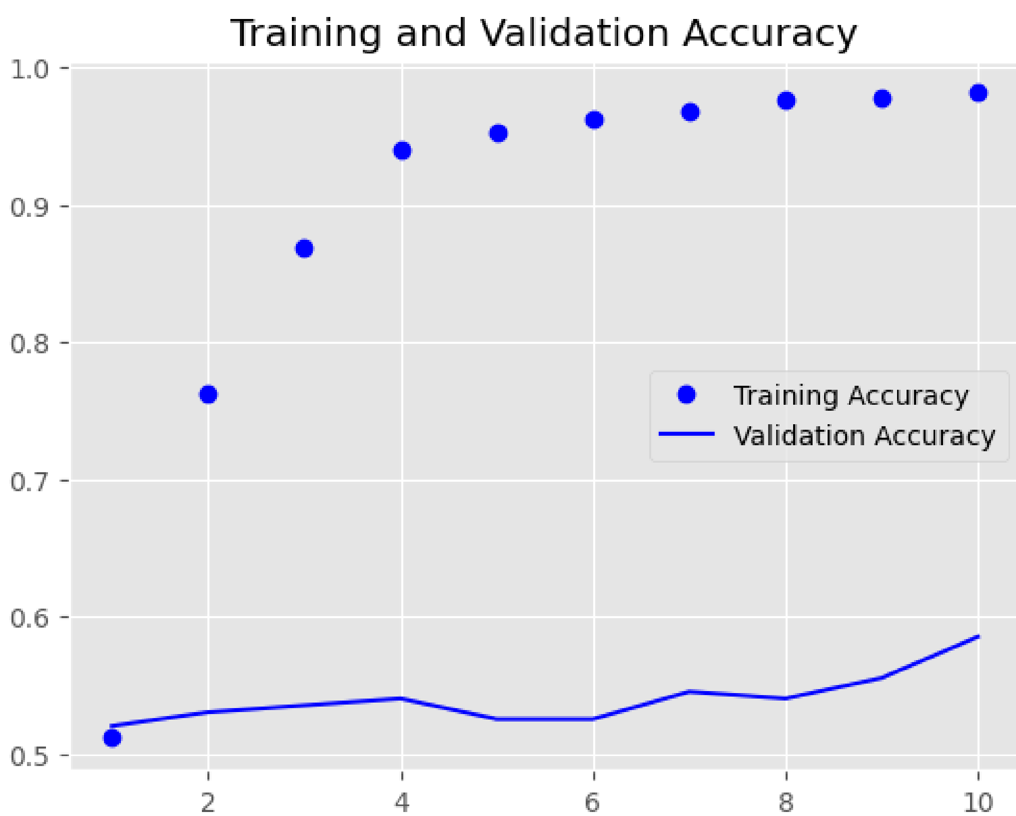
```python
from keras.layers import Embedding
from keras.datasets import imdb
from keras import preprocessing
from keras.models import Sequential
from keras.layers import Flatten, Dense
```

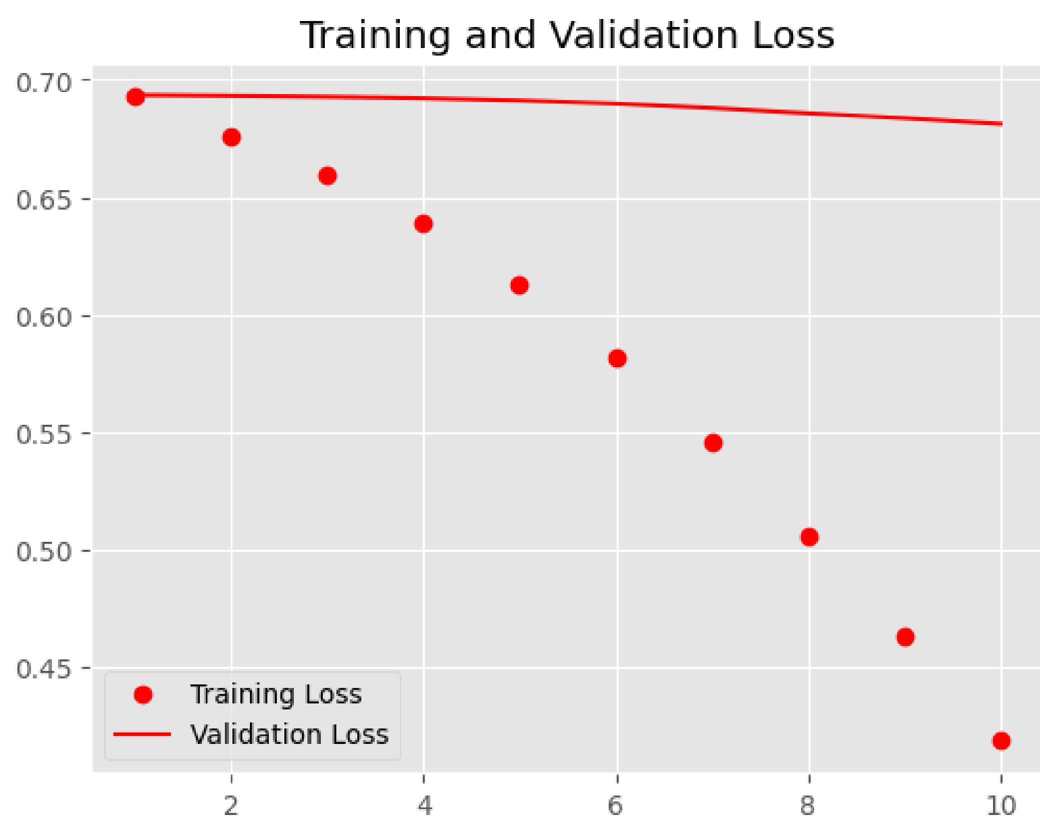
```python
import matplotlib.pyplot as plt
from keras.utils import pad_sequences

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)

# Number of words to consider as features
max_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
max_len = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

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

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, max_len)`
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

model = Sequential()
# We provide our Embedding layer a maximum input length specification
# in order to flatten the embedded inputs later
model.add(Embedding(max_features, 8, input_length=max_len))
# After the Embedding layer,
# our activations have shape `(samples, max_len, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, max_len * 8)`
model.add(Flatten())
# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)

train_acc = history.history["acc"]  # Training accuracy
val_acc = history.history["val_acc"]  # Validation accuracy
```

```
train_loss = history.history["loss"]  # Training loss
val_loss = history.history["val_loss"]  # Validation loss

epochs_range = range(1, len(train_acc) + 1)  # plots every epoch, here 10

plt.plot(epochs_range, train_acc, "bo", label="Training Accuracy")  # "bo"␣
 ↪gives dot plot
plt.plot(epochs_range, val_acc, "b", label="Validation Accuracy")  # "b" gives␣
 ↪line plot
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs_range, train_loss, "ro", label="Training Loss")
plt.plot(epochs_range, val_loss, "r", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.legend()

plt.show()

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_12 (Embedding)    (None, 150, 8)            80000

 flatten_6 (Flatten)         (None, 1200)              0

 dense_6 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
25/25 [==============================] - 1s 12ms/step - loss: 0.6927 - acc:
0.5113 - val_loss: 0.6935 - val_acc: 0.5200
Epoch 2/10
25/25 [==============================] - 0s 7ms/step - loss: 0.6762 - acc:
0.7625 - val_loss: 0.6931 - val_acc: 0.5300
Epoch 3/10
```

```
25/25 [==============================] - 0s 6ms/step - loss: 0.6596 - acc:
0.8687 - val_loss: 0.6927 - val_acc: 0.5350
Epoch 4/10
25/25 [==============================] - 0s 6ms/step - loss: 0.6388 - acc:
0.9400 - val_loss: 0.6921 - val_acc: 0.5400
Epoch 5/10
25/25 [==============================] - 0s 6ms/step - loss: 0.6131 - acc:
0.9525 - val_loss: 0.6912 - val_acc: 0.5250
Epoch 6/10
25/25 [==============================] - 0s 6ms/step - loss: 0.5820 - acc:
0.9625 - val_loss: 0.6898 - val_acc: 0.5250
Epoch 7/10
25/25 [==============================] - 0s 6ms/step - loss: 0.5462 - acc:
0.9675 - val_loss: 0.6880 - val_acc: 0.5450
Epoch 8/10
25/25 [==============================] - 0s 6ms/step - loss: 0.5061 - acc:
0.9762 - val_loss: 0.6857 - val_acc: 0.5400
Epoch 9/10
25/25 [==============================] - 0s 5ms/step - loss: 0.4632 - acc:
0.9775 - val_loss: 0.6837 - val_acc: 0.5550
Epoch 10/10
25/25 [==============================] - 0s 6ms/step - loss: 0.4187 - acc:
0.9812 - val_loss: 0.6813 - val_acc: 0.5850
```






```
782/782 [==============================] - 2s 3ms/step - loss: 0.6810 - acc:
0.5644
Test loss: 0.6810483336448669
Test accuracy: 0.5644400119781494
```

[25]:

```python
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
782/782 [==============================] - 5s 7ms/step - loss: 0.6810 - acc:
0.5644
Test loss: 0.6810483336448669
Test accuracy: 0.5644400119781494
```

[26]:

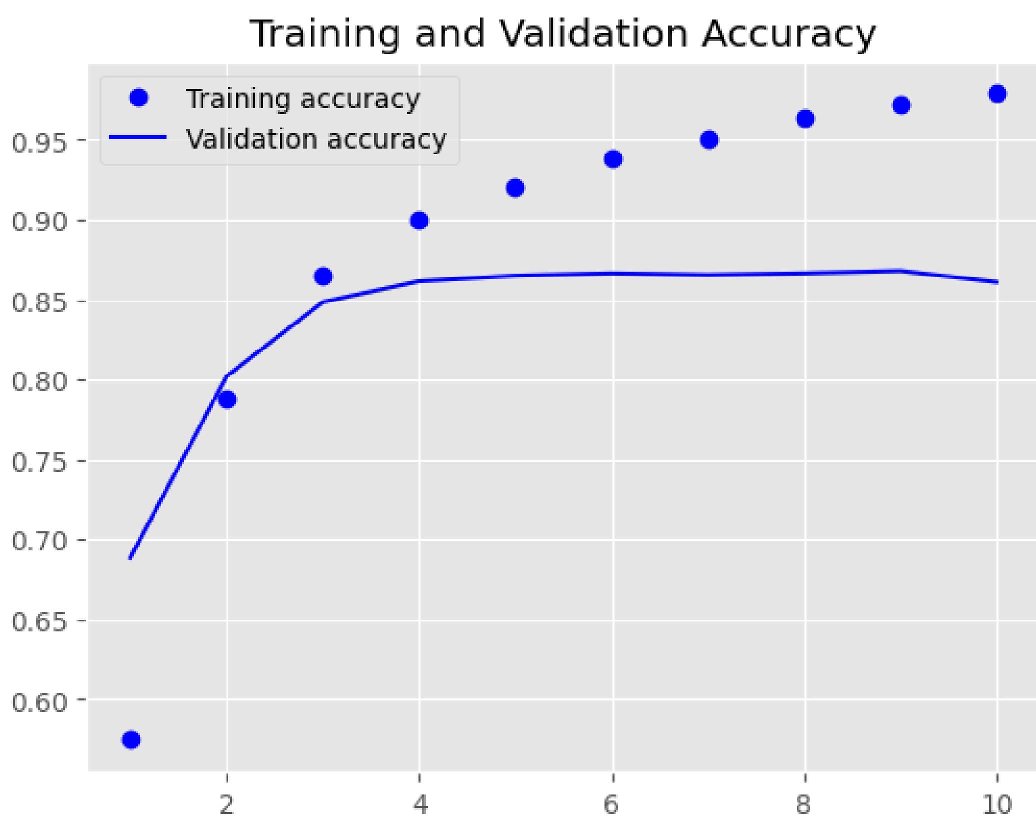
```python
from keras.layers import Embedding
from keras.datasets import imdb
from keras import preprocessing
from keras.models import Sequential
```

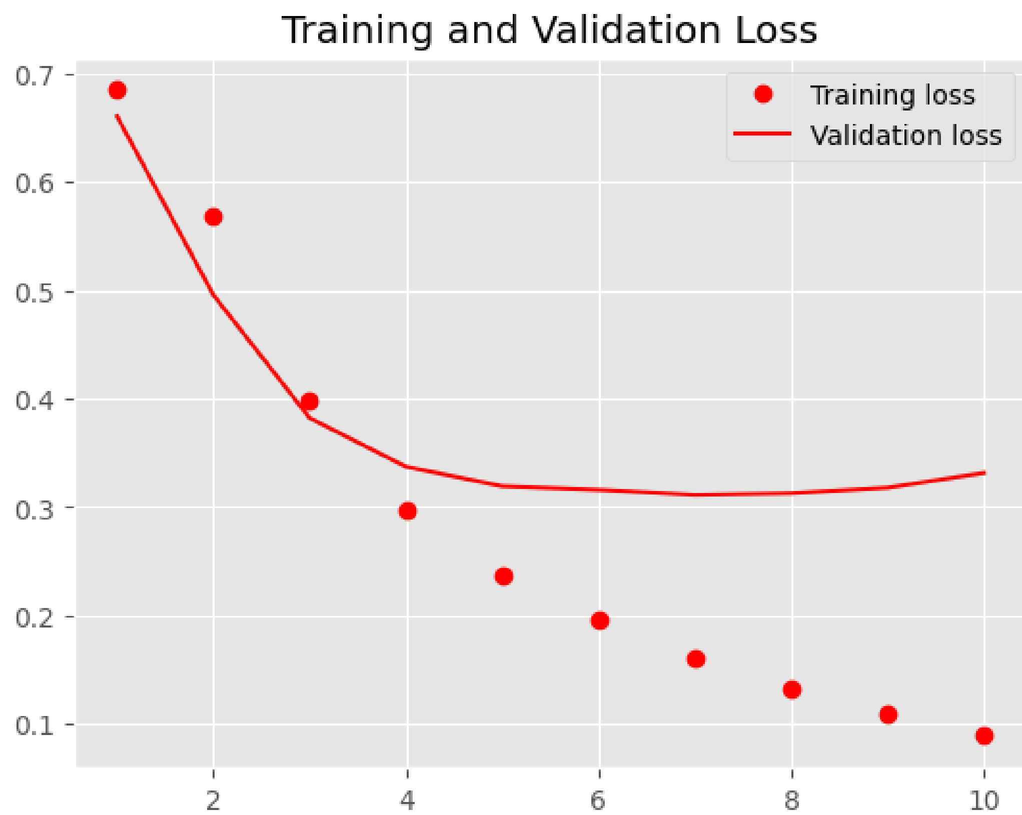
```python
from keras.layers import Flatten, Dense
from keras.utils import pad_sequences
import matplotlib.pyplot as plt

# Number of words to consider as features
max_features = 10000
# After this amount of words, cut the texts
# (among top max_features most common words)
max_len = 150

# Data should be loaded as lists of integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

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

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

model = Sequential()
# We provide our Embedding layer a maximum input length specification
# in order to flatten the embedded inputs later
model.add(Embedding(max_features, 8, input_length=max_len))
# After the Embedding layer,
# our activations have shape `(samples, max_len, 8)`.

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

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

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
train_acc = history.history["acc"] # Training accuracy
val_acc = history.history["val_acc"] # Validation accuracy
train_loss = history.history["loss"] # Training loss
val_loss = history.history["val_loss"] # Validation loss

epochs = range(1, len(train_acc) + 1) #plots every epoch, here 10
```

```python
plt.plot(epochs, train_acc, "bo", label = "Training accuracy") # "bo" gives dot
 ↪plot
plt.plot(epochs, val_acc, "b", label = "Validation accuracy") # "b" gives line
 ↪plot
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, train_loss, "ro", label = "Training loss")
plt.plot(epochs, val_loss, "r", label = "Validation loss")
plt.title("Training and Validation Loss")
plt.legend()

plt.show()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_13 (Embedding)    (None, 150, 8)            80000

 flatten_7 (Flatten)         (None, 1200)              0

 dense_7 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
250/250 [==============================] - 3s 8ms/step - loss: 0.6848 - acc:
0.5746 - val_loss: 0.6605 - val_acc: 0.6885
Epoch 2/10
250/250 [==============================] - 2s 6ms/step - loss: 0.5686 - acc:
0.7884 - val_loss: 0.4957 - val_acc: 0.8020
Epoch 3/10
250/250 [==============================] - 2s 6ms/step - loss: 0.3978 - acc:
0.8650 - val_loss: 0.3822 - val_acc: 0.8485
Epoch 4/10
250/250 [==============================] - 2s 7ms/step - loss: 0.2973 - acc:
0.8996 - val_loss: 0.3373 - val_acc: 0.8615
Epoch 5/10
250/250 [==============================] - 2s 7ms/step - loss: 0.2377 - acc:
0.9209 - val_loss: 0.3193 - val_acc: 0.8650
Epoch 6/10
```

```
250/250 [==============================] - 2s 7ms/step - loss: 0.1953 - acc:
0.9391 - val_loss: 0.3159 - val_acc: 0.8665
Epoch 7/10
250/250 [==============================] - 2s 7ms/step - loss: 0.1615 - acc:
0.9509 - val_loss: 0.3113 - val_acc: 0.8655
Epoch 8/10
250/250 [==============================] - 2s 7ms/step - loss: 0.1331 - acc:
0.9634 - val_loss: 0.3128 - val_acc: 0.8665
Epoch 9/10
250/250 [==============================] - 2s 7ms/step - loss: 0.1095 - acc:
0.9722 - val_loss: 0.3179 - val_acc: 0.8680
Epoch 10/10
250/250 [==============================] - 1s 3ms/step - loss: 0.0889 - acc:
0.9789 - val_loss: 0.3314 - val_acc: 0.8610
```





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

```
782/782 [==============================] - 1s 2ms/step - loss: 0.3423 - acc:
0.8536
Test loss: 0.34232082962989807
Test accuracy: 0.8536400198936462
```

[28]:
```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
782/782 [==============================] - 3s 4ms/step - loss: 0.3423 - acc:
0.8536
Test loss: 0.34232082962989807
Test accuracy: 0.8536400198936462
```

[29]:
```
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  21.4M      0  0:00:03  0:00:03 --:--:-- 21.4M
```

[30]:
```python
imdb_directory = 'aclImdb'
training_directory = os.path.join(imdb_directory, 'train')

text_labels = []
text_data = []

for sentiment_type in ['neg', 'pos']:
    directory_name = os.path.join(training_directory, sentiment_type)
    for file_name in os.listdir(directory_name):
        if file_name.endswith('.txt'):
            with open(os.path.join(directory_name, file_name),␣
 ↪encoding='utf-8') as file:
                text_data.append(file.read())
            if sentiment_type == 'neg':
                text_labels.append(0)
            else:
                text_labels.append(1)
```

You can use pre-existing word embeddings if there isn't enough training data to learn word embeddings alongside the particular problem you're addressing.

Each review is compiled into a list of strings, one string for each review, and the labels (positive/negative) that correspond to each string are compiled into another list.

Tokenizing the data(it involves splitting the data into smaller units called tokens)

[31]:
```python
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

max_len = 150  # Cuts off review after 150 words
num_train_samples = 1000  # Trains on 1000 samples
num_val_samples = 10000  # Validates 10000 samples
num_words = 10000  # Considers only the top 10000 words in the dataset

text_tokenizer = Tokenizer(num_words=num_words)
text_tokenizer.fit_on_texts(texts)
text_sequences = text_tokenizer.texts_to_sequences(texts)
word_index = text_tokenizer.word_index  # Length: 88582
print("Found %s unique tokens." % len(word_index))

padded_data = pad_sequences(text_sequences, maxlen=max_len)

labels_array = np.asarray(labels)
```

```
print("Shape of data tensor:", padded_data.shape)
print("Shape of label tensor:", labels_array.shape)

shuffled_indices = np.arange(padded_data.shape[0])  # Splits data into training␣
 ↪and validation set, but shuffles it since samples are ordered:
# all negatives first, then all positives
np.random.shuffle(shuffled_indices)
shuffled_data = padded_data[shuffled_indices]
shuffled_labels = labels_array[shuffled_indices]

x_train = shuffled_data[:num_train_samples]  # (1000, 150)
y_train = shuffled_labels[:num_train_samples]  # shape (1000,)
x_val = shuffled_data[num_train_samples:num_train_samples + num_val_samples]  #␣
 ↪shape (10000, 150)
y_val = shuffled_labels[num_train_samples:num_train_samples + num_val_samples] ␣
 ↪# shape (10000,)
```

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

Downloading and Preprocessing the GloVe word embedding

[32]:
```
import numpy as np
import requests
from io import BytesIO
import zipfile  # importing zipfile module

glove_url = 'https://nlp.stanford.edu/data/glove.6B.zip'  # URL to download␣
 ↪GloVe embeddings
glove_response = requests.get(glove_url)

# Unzip the contents
with zipfile.ZipFile(BytesIO(glove_response.content)) as zip_file:
    zip_file.extractall('/content/glove')

# Loading GloVe embeddings into memory
embedding_index = {}
with open('/content/glove/glove.6B.100d.txt', encoding='utf-8') as file:
    for line in file:
        values = line.split()
        word = values[0]
        coefficients = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefficients

print("Found %s word vectors." % len(embedding_index))
```

```
Found 400000 word vectors.
```

Making an embedding matrix appropriate for an embedding layer is the next step. It should have the following measurements: 10000 x 100 (max words, embedding dimension). The original size of the GloVe embedding was 100 x 400000.

Preparing the GloVe word embeddings matrix

[34]:
```
embedding_dim = 100

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

[35]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(Max_words, embedding_dim, input_length=Max_len))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_14 (Embedding)    (None, 150, 100)          1000000

 flatten_8 (Flatten)         (None, 15000)             0

 dense_8 (Dense)             (None, 32)                480032

 dense_9 (Dense)             (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[36]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

The Embedding layer won't be trainable if this is set to False, which prevents the optimization algorithm from changing the word embedding values. On the other hand, when it is set to True, the algorithm is able to update the pretrained embeddings. In order to prevent pretrained embeddings from forgetting what they have already learned, it is generally advised against updating them during training.

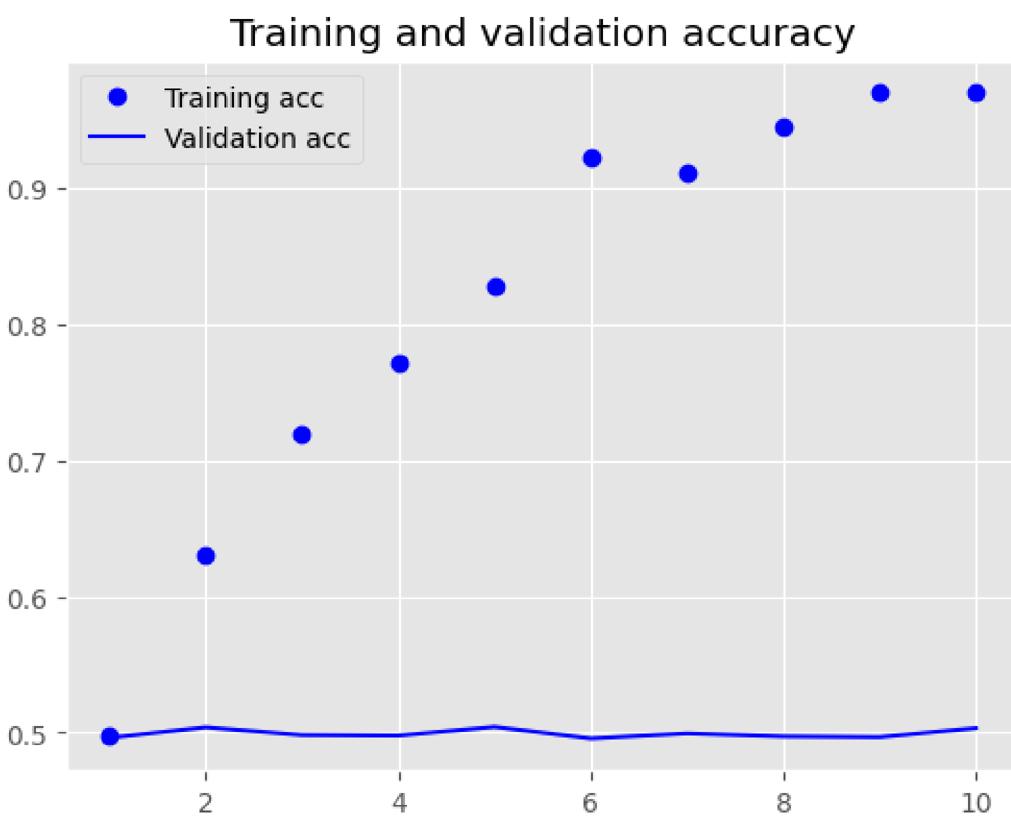
[37]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

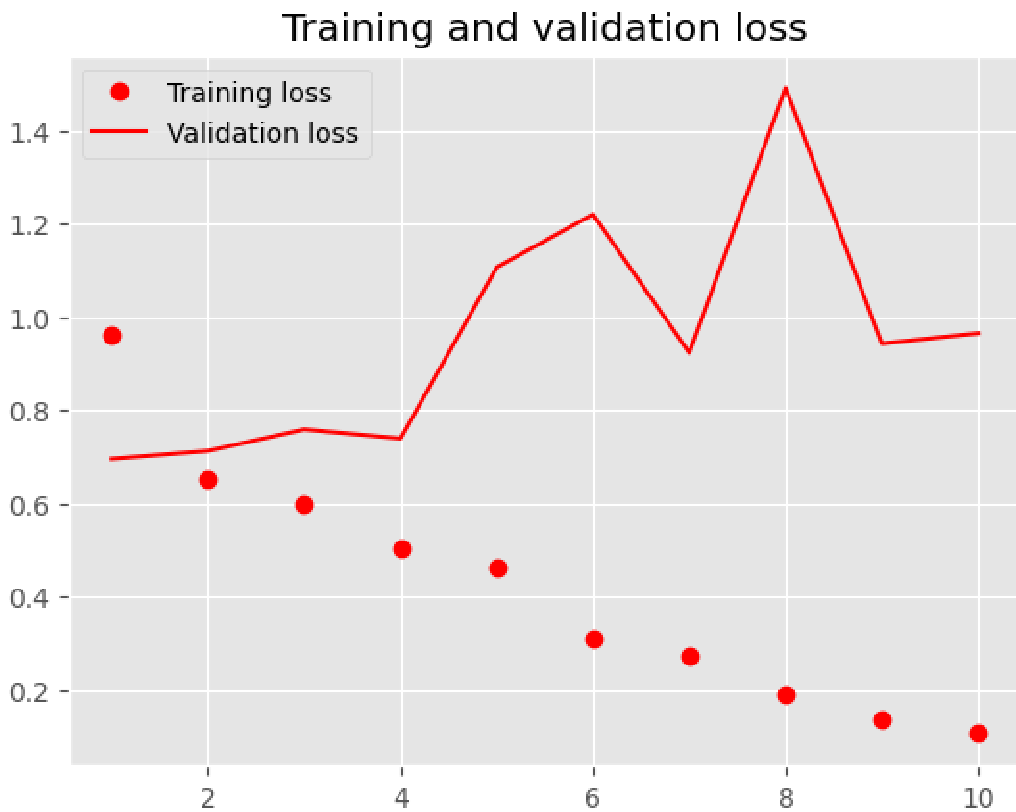
```
Epoch 1/10
32/32 [==============================] - 5s 111ms/step - loss: 0.9626 - acc:
0.4980 - val_loss: 0.6958 - val_acc: 0.4961
Epoch 2/10
32/32 [==============================] - 3s 101ms/step - loss: 0.6533 - acc:
0.6300 - val_loss: 0.7127 - val_acc: 0.5038
Epoch 3/10
32/32 [==============================] - 2s 78ms/step - loss: 0.5967 - acc:
0.7200 - val_loss: 0.7586 - val_acc: 0.4981
Epoch 4/10
32/32 [==============================] - 3s 100ms/step - loss: 0.5037 - acc:
0.7710 - val_loss: 0.7392 - val_acc: 0.4977
Epoch 5/10
32/32 [==============================] - 3s 97ms/step - loss: 0.4606 - acc:
0.8280 - val_loss: 1.1067 - val_acc: 0.5041
Epoch 6/10
32/32 [==============================] - 1s 46ms/step - loss: 0.3077 - acc:
0.9220 - val_loss: 1.2206 - val_acc: 0.4956
Epoch 7/10
32/32 [==============================] - 2s 49ms/step - loss: 0.2731 - acc:
0.9110 - val_loss: 0.9231 - val_acc: 0.4993
Epoch 8/10
32/32 [==============================] - 2s 50ms/step - loss: 0.1899 - acc:
0.9450 - val_loss: 1.4924 - val_acc: 0.4971
Epoch 9/10
32/32 [==============================] - 2s 49ms/step - loss: 0.1372 - acc:
0.9700 - val_loss: 0.9430 - val_acc: 0.4966
Epoch 10/10
32/32 [==============================] - 1s 41ms/step - loss: 0.1053 - acc:
0.9700 - val_loss: 0.9650 - val_acc: 0.5032
```

As expected with a small training dataset, the model starts overfitting quickly. The wide range of validation accuracy results is also due to this limited amount of data.

[38]:

```
import matplotlib.pyplot as plt

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

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

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

plt.figure()

plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, validation_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





[39]:
```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
782/782 [==============================] - 3s 3ms/step - loss: 0.9603 - acc:
0.5003
Test loss: 0.9603413343429565
Test accuracy: 0.5002800226211548
```

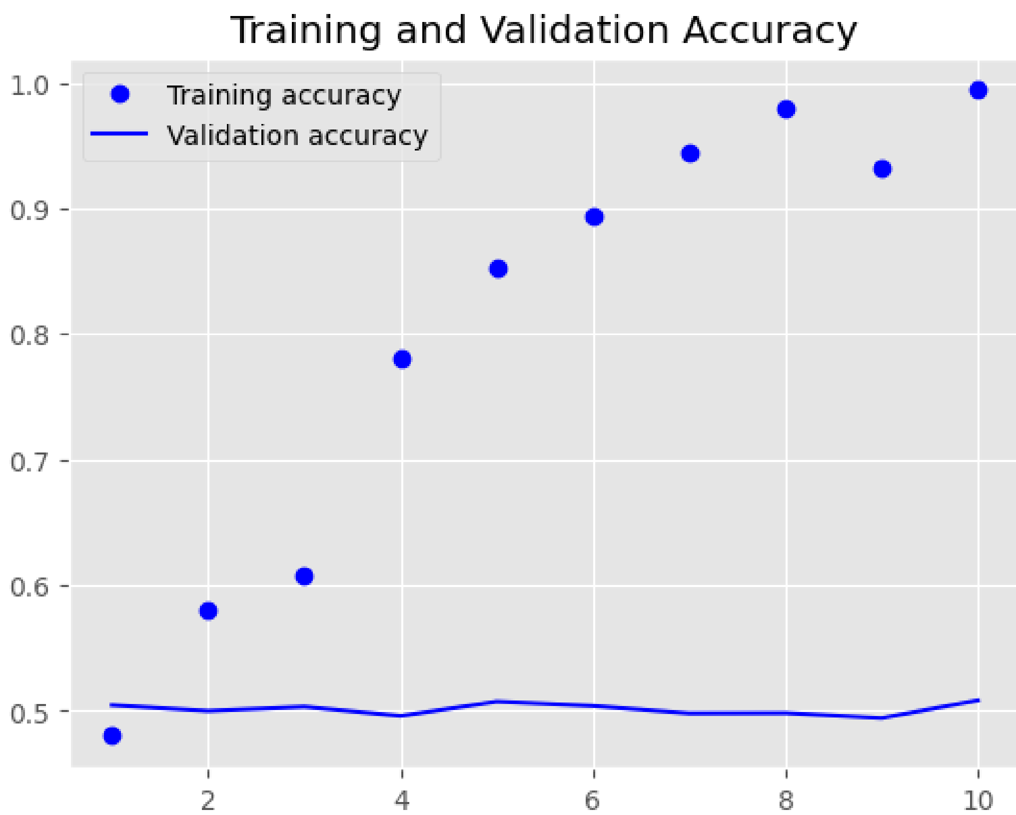
[41]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

max_len = 150  # cuts off review after 150 words
training_samples = 500   # Trains on 500 samples
validation_samples = 10000   # Validates 10000 samples
max_words = 10000  # Considers only the top 10000 words in the dataset

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
```

```python
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index    # Length: 88582
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=max_len)

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

indices = np.arange(data.shape[0])  # Splits data into training and validation sets, shuffles it
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]  # (500, 150)
y_train = labels[:training_samples]  # shape (500,)
x_val = data[training_samples:training_samples+validation_samples]  # shape (10000, 150)
y_val = labels[training_samples:training_samples+validation_samples]  # shape (10000,)

embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, index in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if index < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[index] = embedding_vector

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

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

model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
```

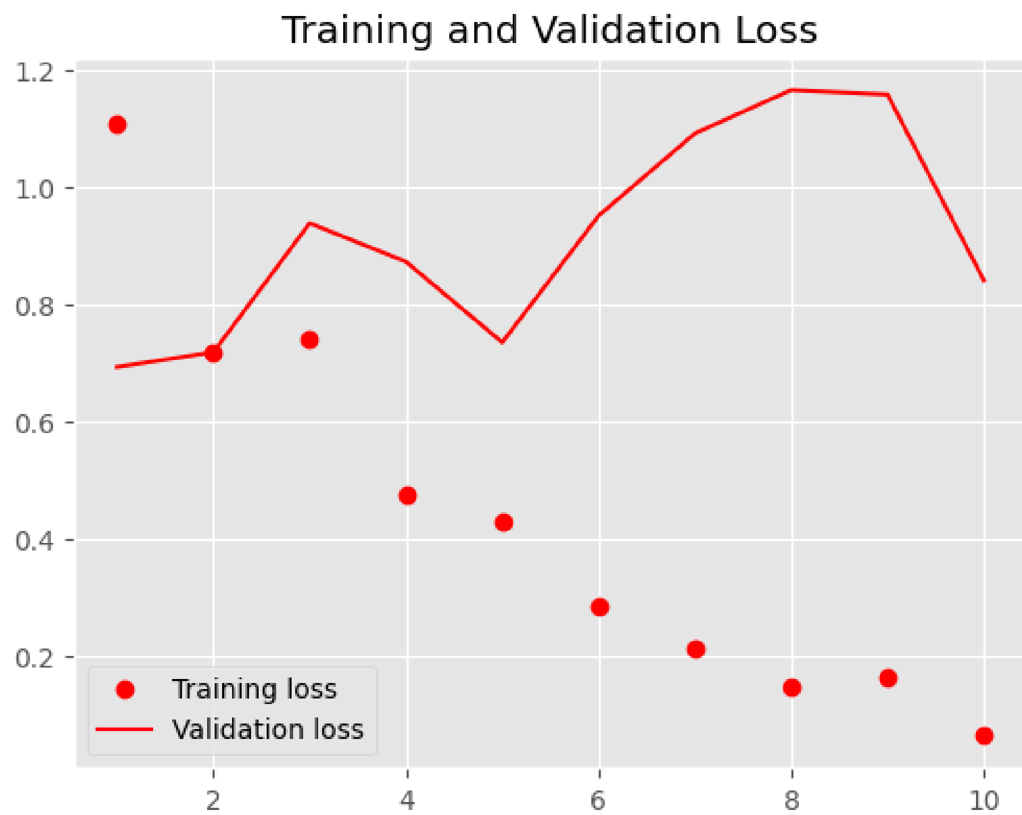
```python
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')

import matplotlib.pyplot as plt
training_acc = history.history['acc']
validation_acc = history.history['val_acc']
training_loss = history.history['loss']
validation_loss = history.history['val_loss']

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

plt.plot(epochs, training_acc, 'bo', label='Training accuracy')
plt.plot(epochs, validation_acc, 'b', label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, training_loss, 'ro', label='Training loss')
plt.plot(epochs, validation_loss, 'r', label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.show()
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_15 (Embedding)    (None, 150, 100)          1000000

 flatten_9 (Flatten)         (None, 15000)             0

 dense_10 (Dense)            (None, 32)                480032

 dense_11 (Dense)            (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
-----------------------------------------------------------------
Epoch 1/10
16/16 [==============================] - 3s 143ms/step - loss: 1.1083 - acc:
0.4800 - val_loss: 0.6931 - val_acc: 0.5045
Epoch 2/10
16/16 [==============================] - 2s 140ms/step - loss: 0.7171 - acc:
0.5800 - val_loss: 0.7180 - val_acc: 0.4999
Epoch 3/10
16/16 [==============================] - 3s 186ms/step - loss: 0.7396 - acc:
0.6080 - val_loss: 0.9381 - val_acc: 0.5032
Epoch 4/10
16/16 [==============================] - 1s 94ms/step - loss: 0.4752 - acc:
0.7800 - val_loss: 0.8724 - val_acc: 0.4959
Epoch 5/10
16/16 [==============================] - 1s 93ms/step - loss: 0.4279 - acc:
0.8520 - val_loss: 0.7349 - val_acc: 0.5073
Epoch 6/10
16/16 [==============================] - 1s 95ms/step - loss: 0.2864 - acc:
0.8940 - val_loss: 0.9508 - val_acc: 0.5039
Epoch 7/10
16/16 [==============================] - 1s 65ms/step - loss: 0.2120 - acc:
0.9440 - val_loss: 1.0912 - val_acc: 0.4980
Epoch 8/10
16/16 [==============================] - 1s 93ms/step - loss: 0.1479 - acc:
0.9800 - val_loss: 1.1650 - val_acc: 0.4982
Epoch 9/10
16/16 [==============================] - 1s 96ms/step - loss: 0.1636 - acc:
0.9320 - val_loss: 1.1574 - val_acc: 0.4942
Epoch 10/10
16/16 [==============================] - 3s 179ms/step - loss: 0.0639 - acc:
0.9940 - val_loss: 0.8407 - val_acc: 0.5083
```



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



[42]:
```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
782/782 [==============================] - 10s 13ms/step - loss: 0.8476 - acc:
0.4946
Test loss: 0.8475740551948547
Test accuracy: 0.49459999799728394
```

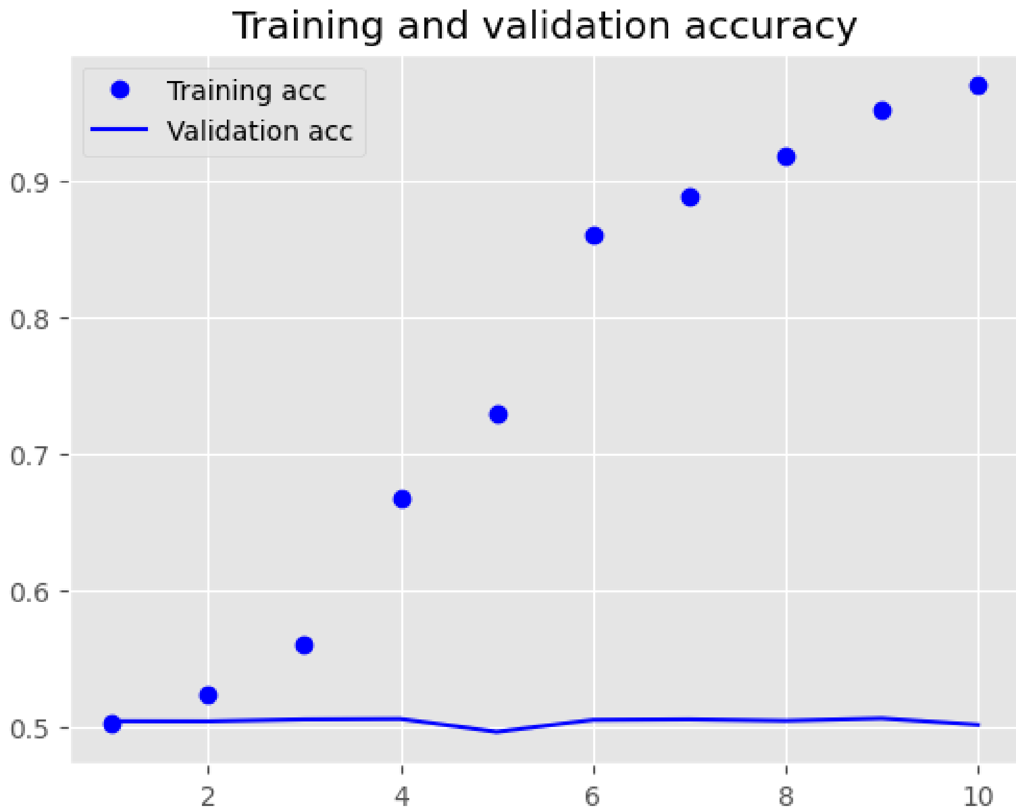
[43]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

max_len = 150 # cuts off review after 150 words
training_samples = 1000 # Trains on 1000 samples
validation_samples = 10000 # Validates on 10000 samples
max_words = 10000 # Considers only the top 10000 words in the dataset

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

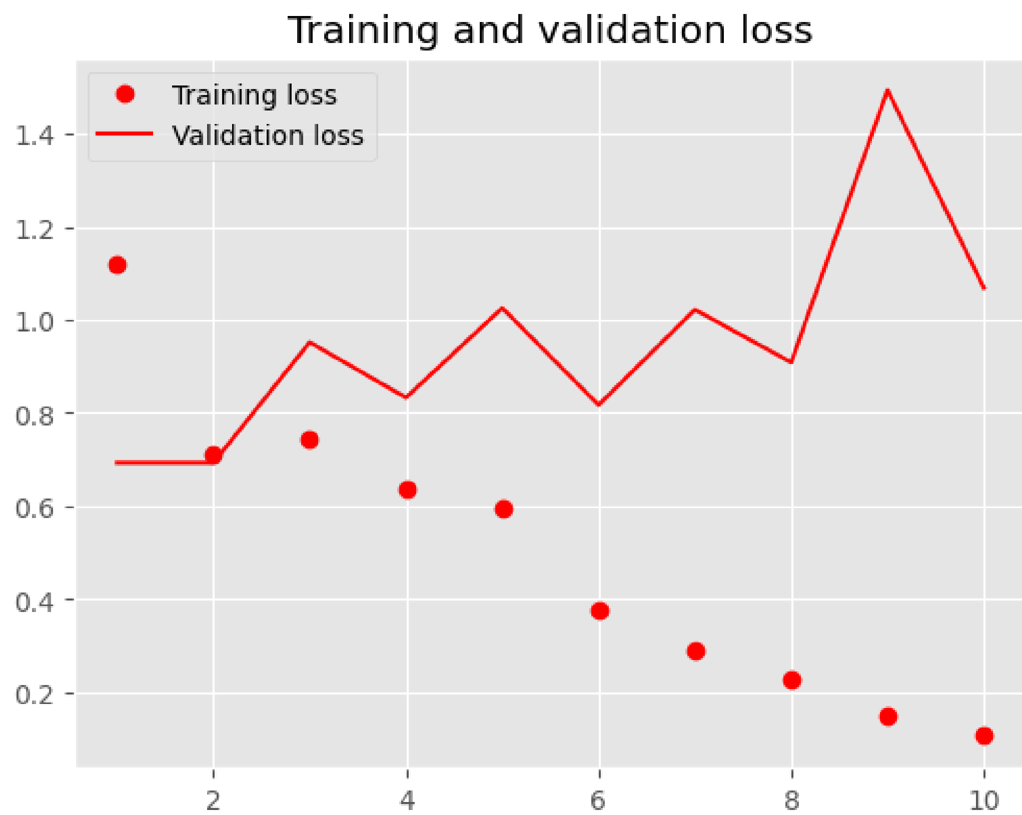
```
word_index = tokenizer.word_index      # Length: 88582
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=max_len)

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

indices = np.arange(data.shape[0]) # splits data into training and validation␣
 ↪sets,
# however since the samples are arranged, it shuffles the data: all negatives␣
 ↪first, then all positive
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[:training_samples] # (1000, 150)
y_train = labels[:training_samples] # shape (1000,)
x_val = data[training_samples:training_samples+validation_samples] # shape␣
 ↪(10000, 150)
y_val = labels[training_samples:training_samples+validation_samples] # shape␣
 ↪(10000,)

embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector

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

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

model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
```

```python
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')

import matplotlib.pyplot as plt

training_acc = history.history['acc']
validation_acc = history.history['val_acc']
training_loss = history.history['loss']
validation_loss = history.history['val_loss']

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

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

plt.figure()

plt.plot(epochs, training_loss, 'ro', label='Training loss')
plt.plot(epochs, validation_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_16 (Embedding)    (None, 150, 100)          1000000

 flatten_10 (Flatten)        (None, 15000)             0

 dense_12 (Dense)            (None, 32)                480032

 dense_13 (Dense)            (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
32/32 [==============================] - 2s 56ms/step - loss: 1.1206 - acc:
0.5020 - val_loss: 0.6931 - val_acc: 0.5041
Epoch 2/10
32/32 [==============================] - 2s 49ms/step - loss: 0.7129 - acc:
0.5240 - val_loss: 0.6931 - val_acc: 0.5040
Epoch 3/10
32/32 [==============================] - 3s 95ms/step - loss: 0.7447 - acc:
0.5600 - val_loss: 0.9519 - val_acc: 0.5054
Epoch 4/10
32/32 [==============================] - 3s 97ms/step - loss: 0.6371 - acc:
0.6670 - val_loss: 0.8330 - val_acc: 0.5057
Epoch 5/10
32/32 [==============================] - 3s 98ms/step - loss: 0.5943 - acc:
0.7290 - val_loss: 1.0255 - val_acc: 0.4962
Epoch 6/10
32/32 [==============================] - 2s 76ms/step - loss: 0.3761 - acc:
0.8600 - val_loss: 0.8172 - val_acc: 0.5050
Epoch 7/10
32/32 [==============================] - 3s 96ms/step - loss: 0.2904 - acc:
0.8890 - val_loss: 1.0220 - val_acc: 0.5054
Epoch 8/10
32/32 [==============================] - 2s 76ms/step - loss: 0.2287 - acc:
0.9190 - val_loss: 0.9087 - val_acc: 0.5044
Epoch 9/10
32/32 [==============================] - 3s 98ms/step - loss: 0.1515 - acc:
0.9520 - val_loss: 1.4936 - val_acc: 0.5061
Epoch 10/10
32/32 [==============================] - 1s 40ms/step - loss: 0.1065 - acc:
0.9700 - val_loss: 1.0688 - val_acc: 0.5014
```


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



[44]:
```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
782/782 [==============================] - 5s 7ms/step - loss: 1.0852 - acc:
0.5026
Test loss: 1.0852259397506714
Test accuracy: 0.5026400089263916
```

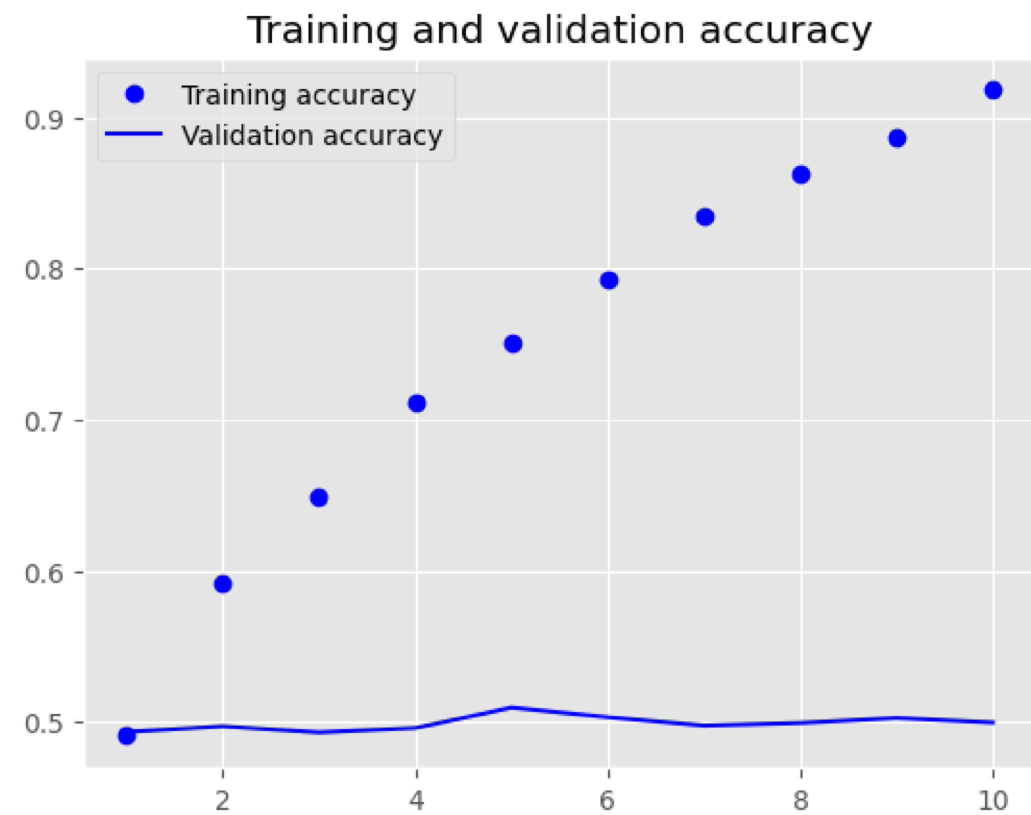
[45]:
```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

max_len = 150 # cuts off review after 150 words
training_samples = 10000 # Trains on 10000 samples
validation_samples = 10000 # Validates on 10000 samples
max_words = 10000 # Considers only the top 10000 words in the dataset

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

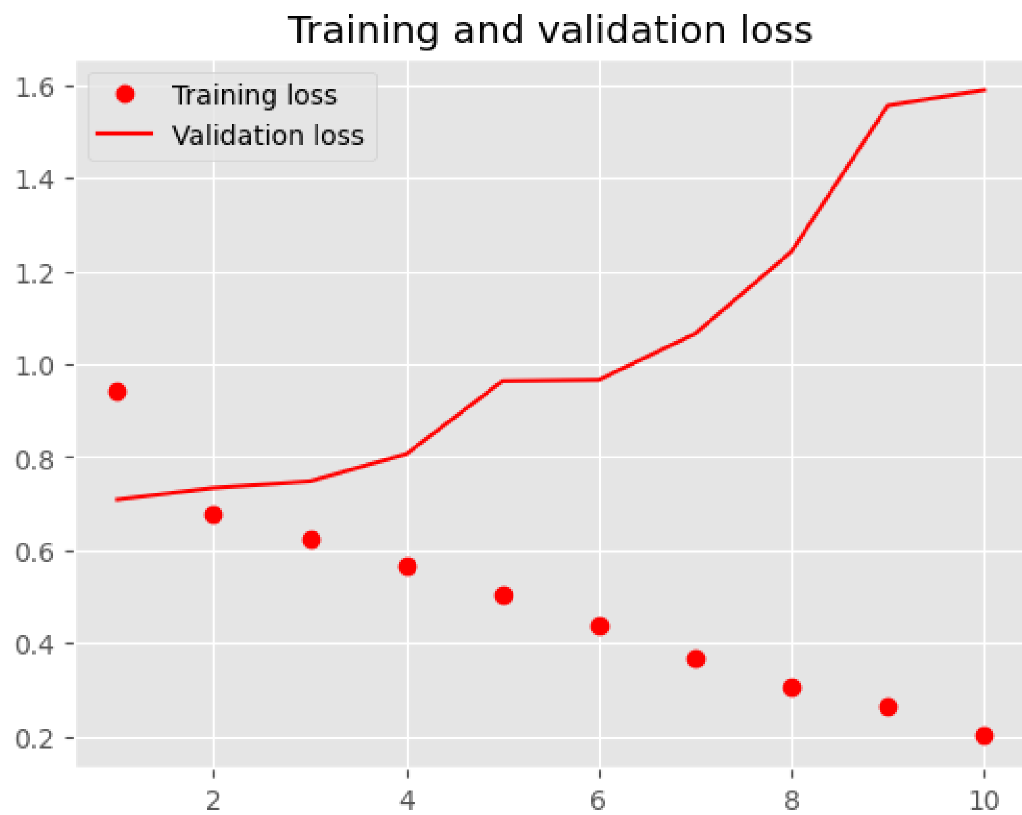
```python
word_index = tokenizer.word_index       # Length: 88582
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=max_len)

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

indices = np.arange(data.shape[0]) # splits data into training and validation sets,
# however since the samples are arranged, it shuffles the data: all negatives first, then all positive
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[:training_samples] # (10000, 150)
y_train = labels[:training_samples] # shape (10000,)
x_val = data[training_samples:training_samples+validation_samples] # shape (10000, 150)
y_val = labels[training_samples:training_samples+validation_samples] # shape (10000,)

embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector

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

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_len))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train,
```

```python
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')

import matplotlib.pyplot as plt

training_acc = history.history['accuracy']
validation_acc = history.history['val_accuracy']
training_loss = history.history['loss']
validation_loss = history.history['val_loss']

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

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

plt.figure()

plt.plot(epochs, training_loss, 'ro', label='Training loss')
plt.plot(epochs, validation_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_17 (Embedding)    (None, 150, 100)          1000000

 flatten_11 (Flatten)        (None, 15000)             0

 dense_14 (Dense)            (None, 32)                480032

 dense_15 (Dense)            (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
-----------------------------------------------------------------
Epoch 1/10
313/313 [==============================] - 8s 22ms/step - loss: 0.9403 -
accuracy: 0.4907 - val_loss: 0.7088 - val_accuracy: 0.4933
Epoch 2/10
313/313 [==============================] - 7s 22ms/step - loss: 0.6788 -
accuracy: 0.5921 - val_loss: 0.7337 - val_accuracy: 0.4970
Epoch 3/10
313/313 [==============================] - 5s 17ms/step - loss: 0.6253 -
accuracy: 0.6487 - val_loss: 0.7481 - val_accuracy: 0.4930
Epoch 4/10
313/313 [==============================] - 8s 27ms/step - loss: 0.5650 -
accuracy: 0.7108 - val_loss: 0.8064 - val_accuracy: 0.4959
Epoch 5/10
313/313 [==============================] - 7s 21ms/step - loss: 0.5027 -
accuracy: 0.7504 - val_loss: 0.9634 - val_accuracy: 0.5094
Epoch 6/10
313/313 [==============================] - 7s 22ms/step - loss: 0.4366 -
accuracy: 0.7926 - val_loss: 0.9660 - val_accuracy: 0.5031
Epoch 7/10
313/313 [==============================] - 5s 17ms/step - loss: 0.3677 -
accuracy: 0.8342 - val_loss: 1.0651 - val_accuracy: 0.4976
Epoch 8/10
313/313 [==============================] - 3s 11ms/step - loss: 0.3065 -
accuracy: 0.8626 - val_loss: 1.2413 - val_accuracy: 0.4993
Epoch 9/10
313/313 [==============================] - 4s 12ms/step - loss: 0.2639 -
accuracy: 0.8869 - val_loss: 1.5554 - val_accuracy: 0.5026
Epoch 10/10
313/313 [==============================] - 4s 13ms/step - loss: 0.2010 -
accuracy: 0.9179 - val_loss: 1.5885 - val_accuracy: 0.4996
```


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



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

```
782/782 [==============================] - 3s 4ms/step - loss: 1.3040 - acc:
0.4990
Test loss: 1.3039577007293701
Test accuracy: 0.49904000759124756
```