import keras keras.\_\_version\_\_

'2.4.3' Using Pretrained word

embedding

from google.colab import drive drive.mount('/content/drive')

Mounted at /content/drive

import os, shutil

imdb\_dir = '/content/drive/MyDrive/aclImdb'

!ls '/content/drive/MyDrive/aclImdb' imdbEr.txt imdb.vocab 'New Text Document.txt' README test train Import data

import os 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) Tokenize the data

from keras.preprocessing.text import Tokenizer from

keras.preprocessing.sequence import pad\_sequences import numpy as np maxlen = 150 training\_samples = 100 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 %s unique tokens.' % len(word\_index)) data = pad\_sequences(sequences, maxlen=maxlen)

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

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, 150) Shape of label tensor: (25000,)

import glove 6b

glove\_dir = '/content/drive/MyDrive/glove.6B'

!ls '/content/drive/MyDrive/glove.6B' glove.6B.100d.txt glove.6B.200d.txt glove.6B.300d.txt glove.6B.50d.txt Preprocess embeddings

from keras.preprocessing.text import Tokenizer from

keras.preprocessing.sequence import pad\_sequences import numpy as np embeddings\_index = {} f = open(os.path.join(glove\_dir, 'glove.6B.100d.txt')) for line in f: for line in f:

values = line.split() word = values[0]

coefs = np.asarray(values[1:], dtype='float32') embeddings\_index[word] = coefs f.close()

print('Found %s word vectors.' % len(embeddings\_index))

Found 400000 word vectors.

embedding\_dim = 100 embedding\_matrix = np.zeros((max\_words, embedding\_dim)) for word, i in word\_index.items(): if i < max\_words: embedding\_vector = embeddings\_index.get(word) if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector Build the model

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: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 150, 100) 1000000

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ flatten (Flatten) (None, 15000) 0

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense (Dense) (None, 32) 480032

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_1

(Dense) (None, 1) 33

=================================================================

Total params: 1,480,065

Trainable params: 1,480,065 Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Load the

glove embeddings model.layers[0].set\_weights([embedding\_matrix]) model.layers[0].trainable = False Train and evaluate

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

4/4 [==============================] - 2s 296ms/step - loss: 2.2054 - acc: 0.5862 - val\_loss: 0.7813 - val\_acc: 0.5094 Epoch 2/10

4/4 [==============================] - 1s 223ms/step - loss: 0.6062 - acc: 0.6652 - val\_loss: 0.6983 - val\_acc: 0.5461 Epoch 3/10

4/4 [==============================] - 1s 222ms/step - loss: 0.3793 - acc: 0.8010 - val\_loss: 0.9853 - val\_acc: 0.5008 Epoch 4/10

4/4 [==============================] - 1s 237ms/step - loss: 0.1717 - acc: 0.9536 - val\_loss: 1.4688 - val\_acc: 0.5004 Epoch 5/10

4/4 [==============================] - 1s 248ms/step - loss: 0.3316 - acc: 0.7941 - val\_loss: 0.7181 - val\_acc: 0.5312 Epoch 6/10

4/4 [==============================] - 1s 240ms/step - loss: 0.0996 - acc: 0.9845 - val\_loss: 0.9061 - val\_acc: 0.5059 Epoch 7/10

4/4 [==============================] - 1s 247ms/step - loss: 0.0637 - acc: 1.0000 - val\_loss: 0.7631 - val\_acc: 0.5275 Epoch 8/10

4/4 [==============================] - 1s 231ms/step - loss: 0.0462 - acc: 1.0000 - val\_loss: 0.8088 - val\_acc: 0.5412 Epoch 9/10

4/4 [==============================] - 1s 217ms/step - loss: 0.0248 - acc: 1.0000 - val\_loss: 0.7189 - val\_acc: 0.5588 Epoch 10/10

4/4 [==============================] - 1s 239ms/step - loss: 0.0260 - acc: 1.0000 - val\_loss: 0.7576 - val\_acc: 0.5598 Plotting

import matplotlib.pyplot as plt acc = history.history['acc'] val\_acc =

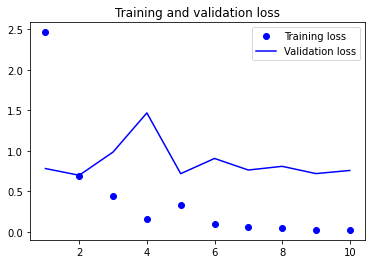
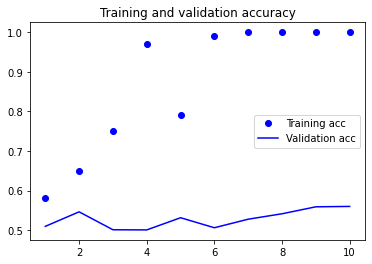
history.history['val\_acc'] loss = history.history['loss'] val\_loss = history.history['val\_loss']

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.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()



highest validation accuracy is 56% while training accuracy is 100%. model is over tting; validation loss was 0.7 @ 2epochs. Training the model

with an embedding layer

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\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_1 (Embedding) (None, 150, 100) 1000000

[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ flatten(Flatten) (None, 15000) 0 1](#_Toc7208)

[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense (Dense) (None, 32) 480032 2](#_Toc7209)

[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense 3](#_Toc7210)

(Dense) (None, 1) 33

=================================================================

Total params: 1,480,065

Trainable params: 1,480,065

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Epoch 1/10

4/4 [==============================] - 2s 293ms/step - loss: 0.6990 - acc: 0.4752 - val\_loss: 0.6924 - val\_acc: 0.5119 Epoch 2/10

4/4 [==============================] - 1s 240ms/step - loss: 0.5086 - acc: 0.9661 - val\_loss: 0.6925 - val\_acc: 0.5160 Epoch 3/10

4/4 [==============================] - 1s 235ms/step - loss: 0.3354 - acc: 0.9693 - val\_loss: 0.7030 - val\_acc: 0.5106 Epoch 4/10

4/4 [==============================] - 1s 247ms/step - loss: 0.1941 - acc: 1.0000 - val\_loss: 0.7128 - val\_acc: 0.5052 Epoch 5/10

4/4 [==============================] - 1s 228ms/step - loss: 0.1177 - acc: 1.0000 - val\_loss: 0.7054 - val\_acc: 0.5092 Epoch 6/10

4/4 [==============================] - 1s 230ms/step - loss: 0.0648 - acc: 1.0000 - val\_loss: 0.7018 - val\_acc: 0.5237 Epoch 7/10

4/4 [==============================] - 1s 244ms/step - loss: 0.0453 - acc: 1.0000 - val\_loss: 0.7043 - val\_acc: 0.5179 Epoch 8/10

4/4 [==============================] - 1s 227ms/step - loss: 0.0255 - acc: 1.0000 - val\_loss: 0.7107 - val\_acc: 0.5238 Epoch 9/10

4/4 [==============================] - 1s 223ms/step - loss: 0.0186 - acc: 1.0000 - val\_loss: 0.7180 - val\_acc: 0.5217 Epoch 10/10 4/4 [==============================] - 1s 233ms/step - loss: 0.0139 - acc: 1.0000 - val\_loss: 0.7202 - val\_acc: 0.5252

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)

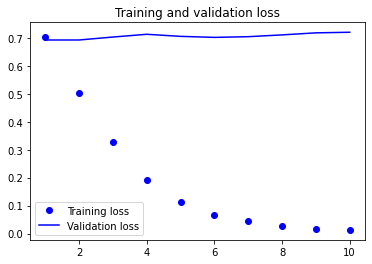
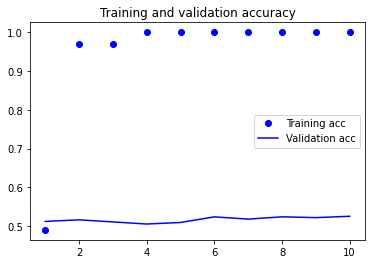
epoc s a ge( , e (acc) )

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()

Evaluate the model on the test set Tokenize the data

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) Evaluate the model

model.load\_weights('pre\_trained\_glove\_model.h5') model.evaluate(x\_test,

y\_test)

782/782 [==============================] - 2s 3ms/step - loss: 0.7657 - acc: 0.5491 [0.7656670808792114, 0.5491200089454651]

#validation accuracy is 54% The accuracy of the pretrained word embedding during validation was 56%, whereas the accuracy of the embedding layer during validation was 54%. The pretrained word embedding yielded slightly better accuracy compared to the embedding layer.