IMDB – RNN

Reginald Mordi

4/29/2021

# Using pretrained word embeddings

import keras

keras.\_\_version\_\_

from google.colab import drive

drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

import os, shutil

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

!ls '/content/drive/MyDrive/aclImdb'

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)

Tokenizing 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, 100)

Shape of label tensor: (25000,)

# Importing golve data

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

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:

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.

# Preparing the GloVe word-embeddings matrix

embedding\_dim = 100

embedding\_matrix = np.zeros((max\_words, embedding\_dim))

for word, i in word\_index.items():

if i < max\_words:

embedding\_vector = embeddings\_index.get(word)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

# Building 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, 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

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

Loading glove embedding layer

model.layers[0].set\_weights([embedding\_matrix])

model.layers[0].trainable = False

Training and Evaluation

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

7/7 [==============================] - 2s 174ms/step - loss: 1.4395 - acc: 0.5290 - val\_loss: 0.7502 - val\_acc: 0.5037

Epoch 2/10

7/7 [==============================] - 1s 98ms/step - loss: 0.5373 - acc: 0.7202 - val\_loss: 0.7355 - val\_acc: 0.5176

Epoch 3/10

7/7 [==============================] - 1s 96ms/step - loss: 0.4822 - acc: 0.7571 - val\_loss: 0.8077 - val\_acc: 0.5057

Epoch 4/10

7/7 [==============================] - 1s 100ms/step - loss: 0.4089 - acc: 0.8215 - val\_loss: 0.7428 - val\_acc: 0.5384

Epoch 5/10

7/7 [==============================] - 1s 97ms/step - loss: 0.2944 - acc: 0.8990 - val\_loss: 0.8546 - val\_acc: 0.5063

Epoch 6/10

7/7 [==============================] - 1s 98ms/step - loss: 0.2188 - acc: 0.9436 - val\_loss: 0.7052 - val\_acc: 0.5459

Epoch 7/10

7/7 [==============================] - 1s 94ms/step - loss: 0.1407 - acc: 1.0000 - val\_loss: 0.7336 - val\_acc: 0.5540

Epoch 8/10

7/7 [==============================] - 1s 100ms/step - loss: 0.1014 - acc: 0.9809 - val\_loss: 0.8381 - val\_acc: 0.5261

Epoch 9/10

7/7 [==============================] - 1s 96ms/step - loss: 0.0632 - acc: 1.0000 - val\_loss: 0.7660 - val\_acc: 0.5600

Epoch 10/10

7/7 [==============================] - 1s 96ms/step - loss: 0.0299 - acc: 1.0000 - val\_loss: 0.7808 - val\_acc: 0.5603

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.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()

Chart, scatter chart

Description automatically generated

Chart, line chart

Description automatically generated

# Training the same model without pretrained word embeddings

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, 100, 100) 1000000

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

flatten\_1 (Flatten) (None, 10000) 0

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

dense\_2 (Dense) (None, 32) 320032

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

dense\_3 (Dense) (None, 1) 33

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

Total params: 1,320,065

Trainable params: 1,320,065

Non-trainable params: 0

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

Epoch 1/10

7/7 [==============================] - 1s 126ms/step - loss: 0.6959 - acc: 0.5028 - val\_loss: 0.6958 - val\_acc: 0.5041

Epoch 2/10

7/7 [==============================] - 1s 103ms/step - loss: 0.5351 - acc: 0.9812 - val\_loss: 0.6960 - val\_acc: 0.5130

Epoch 3/10

7/7 [==============================] - 1s 103ms/step - loss: 0.3230 - acc: 0.9937 - val\_loss: 0.6968 - val\_acc: 0.5151

Epoch 4/10

7/7 [==============================] - 1s 103ms/step - loss: 0.1447 - acc: 1.0000 - val\_loss: 0.7166 - val\_acc: 0.5102

Epoch 5/10

7/7 [==============================] - 1s 101ms/step - loss: 0.0706 - acc: 1.0000 - val\_loss: 0.7185 - val\_acc: 0.5201

Epoch 6/10

7/7 [==============================] - 1s 101ms/step - loss: 0.0340 - acc: 1.0000 - val\_loss: 0.7163 - val\_acc: 0.5134

Epoch 7/10

7/7 [==============================] - 1s 100ms/step - loss: 0.0191 - acc: 1.0000 - val\_loss: 0.7280 - val\_acc: 0.5164

Epoch 8/10

7/7 [==============================] - 1s 100ms/step - loss: 0.0109 - acc: 1.0000 - val\_loss: 0.7341 - val\_acc: 0.5151

Epoch 9/10

7/7 [==============================] - 1s 101ms/step - loss: 0.0064 - acc: 1.0000 - val\_loss: 0.7376 - val\_acc: 0.5170

Epoch 10/10

7/7 [==============================] - 1s 103ms/step - loss: 0.0040 - acc: 1.0000 - val\_loss: 0.7442 - val\_acc: 0.5217

Tokenizing the data of the test set

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)

Evaluating the model on the test set

model.load\_weights('pre\_trained\_glove\_model.h5')

model.evaluate(x\_test, y\_test)

We get an appalling test accuracy of 56%. Working with just a handful of training samples is hard!