babi\_memnn

# 在 bAbI 数据集上训练一个记忆网络。

参考文献：

- Jason Weston, Antoine Bordes, Sumit Chopra, Tomas Mikolov, Alexander M. Rush,

["Towards AI-Complete Question Answering:

A Set of Prerequisite Toy Tasks"](http://arxiv.org/abs/1502.05698)

- Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus,

["End-To-End Memory Networks"](http://arxiv.org/abs/1503.08895)

120 轮迭代后，在 'single\_supporting\_fact\_10k' 任务上达到了 98.6% 的准确率。

每轮迭代时间: 3s on CPU (core i7).

```python

from \_\_future\_\_ import print\_function

from keras.models import Sequential, Model

from keras.layers.embeddings import Embedding

from keras.layers import Input, Activation, Dense, Permute, Dropout

from keras.layers import add, dot, concatenate

from keras.layers import LSTM

from keras.utils.data\_utils import get\_file

from keras.preprocessing.sequence import pad\_sequences

from functools import reduce

import tarfile

import numpy as np

import re

def tokenize(sent):

'''返回包含标点符号的句子的标记。

>>> tokenize('Bob dropped the apple. Where is the apple?')

['Bob', 'dropped', 'the', 'apple', '.', 'Where', 'is', 'the', 'apple', '?']

'''

return [x.strip() for x in re.split(r'(\W+)?', sent) if x.strip()]

def parse\_stories(lines, only\_supporting=False):

'''解析 bAbi 任务格式中提供的故事

如果 only\_supporting 为 true，

则只保留支持答案的句子。

'''

data = []

story = []

for line in lines:

line = line.decode('utf-8').strip()

nid, line = line.split(' ', 1)

nid = int(nid)

if nid == 1:

story = []

if '\t' in line:

q, a, supporting = line.split('\t')

q = tokenize(q)

if only\_supporting:

# 只选择相关的子故事

supporting = map(int, supporting.split())

substory = [story[i - 1] for i in supporting]

else:

# 提供所有子故事

substory = [x for x in story if x]

data.append((substory, q, a))

story.append('')

else:

sent = tokenize(line)

story.append(sent)

return data

def get\_stories(f, only\_supporting=False, max\_length=None):

'''给定文件名，读取文件，检索故事，

然后将句子转换为一个独立故事。

如果提供了 max\_length,

任何长于 max\_length 的故事都将被丢弃。

'''

data = parse\_stories(f.readlines(), only\_supporting=only\_supporting)

flatten = lambda data: reduce(lambda x, y: x + y, data)

data = [(flatten(story), q, answer) for story, q, answer in data

if not max\_length or len(flatten(story)) < max\_length]

return data

def vectorize\_stories(data):

inputs, queries, answers = [], [], []

for story, query, answer in data:

inputs.append([word\_idx[w] for w in story])

queries.append([word\_idx[w] for w in query])

answers.append(word\_idx[answer])

return (pad\_sequences(inputs, maxlen=story\_maxlen),

pad\_sequences(queries, maxlen=query\_maxlen),

np.array(answers))

try:

path = get\_file('babi-tasks-v1-2.tar.gz',

origin='https://s3.amazonaws.com/text-datasets/'

'babi\_tasks\_1-20\_v1-2.tar.gz')

except:

print('Error downloading dataset, please download it manually:\n'

'$ wget http://www.thespermwhale.com/jaseweston/babi/tasks\_1-20\_v1-2'

'.tar.gz\n'

'$ mv tasks\_1-20\_v1-2.tar.gz ~/.keras/datasets/babi-tasks-v1-2.tar.gz')

raise

challenges = {

# QA1 任务，10,000 样本

'single\_supporting\_fact\_10k': 'tasks\_1-20\_v1-2/en-10k/qa1\_'

'single-supporting-fact\_{}.txt',

# QA2 任务，1000 样本

'two\_supporting\_facts\_10k': 'tasks\_1-20\_v1-2/en-10k/qa2\_'

'two-supporting-facts\_{}.txt',

}

challenge\_type = 'single\_supporting\_fact\_10k'

challenge = challenges[challenge\_type]

print('Extracting stories for the challenge:', challenge\_type)

with tarfile.open(path) as tar:

train\_stories = get\_stories(tar.extractfile(challenge.format('train')))

test\_stories = get\_stories(tar.extractfile(challenge.format('test')))

vocab = set()

for story, q, answer in train\_stories + test\_stories:

vocab |= set(story + q + [answer])

vocab = sorted(vocab)

# 保留 0 以留作 pad\_sequences 进行 masking

vocab\_size = len(vocab) + 1

story\_maxlen = max(map(len, (x for x, \_, \_ in train\_stories + test\_stories)))

query\_maxlen = max(map(len, (x for \_, x, \_ in train\_stories + test\_stories)))

print('-')

print('Vocab size:', vocab\_size, 'unique words')

print('Story max length:', story\_maxlen, 'words')

print('Query max length:', query\_maxlen, 'words')

print('Number of training stories:', len(train\_stories))

print('Number of test stories:', len(test\_stories))

print('-')

print('Here\'s what a "story" tuple looks like (input, query, answer):')

print(train\_stories[0])

print('-')

print('Vectorizing the word sequences...')

word\_idx = dict((c, i + 1) for i, c in enumerate(vocab))

inputs\_train, queries\_train, answers\_train = vectorize\_stories(train\_stories)

inputs\_test, queries\_test, answers\_test = vectorize\_stories(test\_stories)

print('-')

print('inputs: integer tensor of shape (samples, max\_length)')

print('inputs\_train shape:', inputs\_train.shape)

print('inputs\_test shape:', inputs\_test.shape)

print('-')

print('queries: integer tensor of shape (samples, max\_length)')

print('queries\_train shape:', queries\_train.shape)

print('queries\_test shape:', queries\_test.shape)

print('-')

print('answers: binary (1 or 0) tensor of shape (samples, vocab\_size)')

print('answers\_train shape:', answers\_train.shape)

print('answers\_test shape:', answers\_test.shape)

print('-')

print('Compiling...')

# 占位符

input\_sequence = Input((story\_maxlen,))

question = Input((query\_maxlen,))

# 编码器

# 将输入序列编码为向量的序列

input\_encoder\_m = Sequential()

input\_encoder\_m.add(Embedding(input\_dim=vocab\_size,

output\_dim=64))

input\_encoder\_m.add(Dropout(0.3))

# 输出: (samples, story\_maxlen, embedding\_dim)

# 将输入编码为的向量的序列（向量尺寸为 query\_maxlen）

input\_encoder\_c = Sequential()

input\_encoder\_c.add(Embedding(input\_dim=vocab\_size,

output\_dim=query\_maxlen))

input\_encoder\_c.add(Dropout(0.3))

# 输出: (samples, story\_maxlen, query\_maxlen)

# 将问题编码为向量的序列

question\_encoder = Sequential()

question\_encoder.add(Embedding(input\_dim=vocab\_size,

output\_dim=64,

input\_length=query\_maxlen))

question\_encoder.add(Dropout(0.3))

# 输出: (samples, query\_maxlen, embedding\_dim)

# 编码输入序列和问题（均已索引化）为密集向量的序列

input\_encoded\_m = input\_encoder\_m(input\_sequence)

input\_encoded\_c = input\_encoder\_c(input\_sequence)

question\_encoded = question\_encoder(question)

# 计算第一个输入向量和问题向量序列的『匹配』（'match'）

# 尺寸: `(samples, story\_maxlen, query\_maxlen)`

match = dot([input\_encoded\_m, question\_encoded], axes=(2, 2))

match = Activation('softmax')(match)

# 将匹配矩阵与第二个输入向量序列相加

response = add([match, input\_encoded\_c]) # (samples, story\_maxlen, query\_maxlen)

response = Permute((2, 1))(response) # (samples, query\_maxlen, story\_maxlen)

# 拼接匹配矩阵和问题向量序列

answer = concatenate([response, question\_encoded])

# 原始论文使用一个矩阵乘法来进行归约操作。

# 我们在此选择使用 RNN。

answer = LSTM(32)(answer) # (samples, 32)

# 一个正则化层 - 可能还需要更多层

answer = Dropout(0.3)(answer)

answer = Dense(vocab\_size)(answer) # (samples, vocab\_size)

# 输出词汇表的一个概率分布

answer = Activation('softmax')(answer)

# 构建最终模型

model = Model([input\_sequence, question], answer)

model.compile(optimizer='rmsprop', loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# 训练

model.fit([inputs\_train, queries\_train], answers\_train,

batch\_size=32,

epochs=120,

validation\_data=([inputs\_test, queries\_test], answers\_test))

```

imdb\_cnn\_lstm

# 在 IMDB 情绪分类任务上训练循环卷积网络。

2 个轮次后达到 0.8498 的测试精度。K520 GPU 上为 41 秒/轮次。

```python

from \_\_future\_\_ import print\_function

from keras.preprocessing import sequence

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation

from keras.layers import Embedding

from keras.layers import LSTM

from keras.layers import Conv1D, MaxPooling1D

from keras.datasets import imdb

# Embedding

max\_features = 20000

maxlen = 100

embedding\_size = 128

# Convolution

kernel\_size = 5

filters = 64

pool\_size = 4

# LSTM

lstm\_output\_size = 70

# Training

batch\_size = 30

epochs = 2

'''

注意:

batch\_size 是高度敏感的

由于数据集非常小，因此仅需要 2 个轮次。

'''

print('Loading data...')

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

print(len(x\_train), 'train sequences')

print(len(x\_test), 'test sequences')

print('Pad sequences (samples x time)')

x\_train = sequence.pad\_sequences(x\_train, maxlen=maxlen)

x\_test = sequence.pad\_sequences(x\_test, maxlen=maxlen)

print('x\_train shape:', x\_train.shape)

print('x\_test shape:', x\_test.shape)

print('Build model...')

model = Sequential()

model.add(Embedding(max\_features, embedding\_size, input\_length=maxlen))

model.add(Dropout(0.25))

model.add(Conv1D(filters,

kernel\_size,

padding='valid',

activation='relu',

strides=1))

model.add(MaxPooling1D(pool\_size=pool\_size))

model.add(LSTM(lstm\_output\_size))

model.add(Dense(1))

model.add(Activation('sigmoid'))

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

print('Train...')

model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(x\_test, y\_test))

score, acc = model.evaluate(x\_test, y\_test, batch\_size=batch\_size)

print('Test score:', score)

print('Test accuracy:', acc)

```