keras_babi_rnn

December 1, 2016

In [1]:
```
from __future__ import print_function
from functools import reduce
import re
import tarfile

import numpy as np
np.random.seed(1337)  # for reproducibility

from keras.utils.data_utils import get_file
from keras.layers.embeddings import Embedding
from keras.layers import Dense, Merge, Dropout, RepeatVector
from keras.layers import recurrent
from keras.models import Sequential
from keras.preprocessing.sequence import pad_sequences


def tokenize(sent):
    '''Return the tokens of a sentence including punctuation.

    >>> 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('(\W+)?', sent) if x.strip()]


def parse_stories(lines, only_supporting=False):
    '''Parse stories provided in the bAbi tasks format

    If only_supporting is true, only the sentences that support the answer
    '''
    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)
                substory = None
                if only_supporting:
                    # Only select the related substory
                    supporting = map(int, supporting.split())
                    substory = [story[i - 1] for i in supporting]
                else:
                    # Provide all the substories
                    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):
        '''Given a file name, read the file, retrieve the stories, and then co

        If max_length is supplied, any stories longer than max_length tokens w
        '''
        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
        return data


    def vectorize_stories(data, word_idx, story_maxlen, query_maxlen):
        X = []
        Xq = []
        Y = []
        for story, query, answer in data:
            x = [word_idx[w] for w in story]
            xq = [word_idx[w] for w in query]
            y = np.zeros(len(word_idx) + 1)  # let's not forget that index 0 is
            y[word_idx[answer]] = 1
            X.append(x)
            Xq.append(xq)
            Y.append(y)
        return pad_sequences(X, maxlen=story_maxlen), pad_sequences(Xq, maxlen=

    RNN = recurrent.LSTM
    EMBED_HIDDEN_SIZE = 50
    SENT_HIDDEN_SIZE = 100
    QUERY_HIDDEN_SIZE = 100
```

```
BATCH_SIZE = 32
EPOCHS = 40
print('RNN / Embed / Sent / Query = {}, {}, {}, {}'.format(RNN, EMBED_HIDDE

try:
    path = get_file('babi-tasks-v1-2.tar.gz', origin='https://s3.amazonaws.
except:
    print('Error downloading dataset, please download it manually:\n'
          '$ wget http://www.thespermwhale.com/jaseweston/babi/tasks_1-20_v
          '$ mv tasks_1-20_v1-2.tar.gz ~/.keras/datasets/babi-tasks-v1-2.ta
    raise
tar = tarfile.open(path)
```

```
Using TensorFlow backend.
```

```
RNN / Embed / Sent / Query = <class 'keras.layers.recurrent.LSTM'>, 50, 100, 100
```

In [2]:
```
# Default QA1 with 1000 samples
# challenge = 'tasks_1-20_v1-2/en/qa1_single-supporting-fact_{}.txt'
# QA1 with 10,000 samples
# challenge = 'tasks_1-20_v1-2/en-10k/qa1_single-supporting-fact_{}.txt'
# QA2 with 1000 samples
# challenge = 'tasks_1-20_v1-2/en/qa2_two-supporting-facts_{}.txt'
# QA2 with 10,000 samples
challenge = 'tasks_1-20_v1-2/en-10k/qa2_two-supporting-facts_{}.txt'
train = get_stories(tar.extractfile(challenge.format('train')))
test = get_stories(tar.extractfile(challenge.format('test')))

vocab = sorted(reduce(lambda x, y: x | y, (set(story + q + [answer]) for st
# Reserve 0 for masking via pad_sequences
vocab_size = len(vocab) + 1
word_idx = dict((c, i + 1) for i, c in enumerate(vocab))
story_maxlen = max(map(len, (x for x, _, _ in train + test)))
query_maxlen = max(map(len, (x for _, x, _ in train + test)))

X, Xq, Y = vectorize_stories(train, word_idx, story_maxlen, query_maxlen)
tX, tXq, tY = vectorize_stories(test, word_idx, story_maxlen, query_maxlen)

print('vocab = {}'.format(vocab))
print('X.shape = {}'.format(X.shape))
print('Xq.shape = {}'.format(Xq.shape))
print('Y.shape = {}'.format(Y.shape))
print('story_maxlen, query_maxlen = {}, {}'.format(story_maxlen, query_maxl

print('Build model...')
```

```
        sentrnn = Sequential()
        sentrnn.add(Embedding(vocab_size, EMBED_HIDDEN_SIZE,
                              input_length=story_maxlen))
        sentrnn.add(Dropout(0.3))

        qrnn = Sequential()
        qrnn.add(Embedding(vocab_size, EMBED_HIDDEN_SIZE,
                           input_length=query_maxlen))
        qrnn.add(Dropout(0.3))
        qrnn.add(RNN(EMBED_HIDDEN_SIZE, return_sequences=False))
        qrnn.add(RepeatVector(story_maxlen))

        model = Sequential()
        model.add(Merge([sentrnn, qrnn], mode='sum'))
        model.add(RNN(EMBED_HIDDEN_SIZE, return_sequences=False))
        model.add(Dropout(0.3))
        model.add(Dense(vocab_size, activation='softmax'))
```

```
/home/ravirajukrishna/anaconda3/lib/python3.5/re.py:203: FutureWarning: split() rec
  return _compile(pattern, flags).split(string, maxsplit)


vocab = ['.', '?', 'Daniel', 'John', 'Mary', 'Sandra', 'Where', 'apple', 'back', 'b
X.shape = (10000, 552)
Xq.shape = (10000, 5)
Y.shape = (10000, 36)
story_maxlen, query_maxlen = 552, 5
Build model...
```

```
In [3]: model.summary()
```

```
____________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================
embedding_1 (Embedding)          (None, 552, 50)       1800        embedding_input_
____________________________________________________________________________________
dropout_1 (Dropout)              (None, 552, 50)       0           embedding_1[0][0
____________________________________________________________________________________
embedding_2 (Embedding)          (None, 5, 50)         1800        embedding_input_
____________________________________________________________________________________
dropout_2 (Dropout)              (None, 5, 50)         0           embedding_2[0][0
____________________________________________________________________________________
lstm_1 (LSTM)                    (None, 50)            20200       dropout_2[0][0]
____________________________________________________________________________________
repeatvector_1 (RepeatVector)    (None, 552, 50)       0           lstm_1[0][0]
____________________________________________________________________________________
lstm_2 (LSTM)                    (None, 50)            20200       merge_1[0][0]
____________________________________________________________________________________
```

```
dropout_3 (Dropout)              (None, 50)            0           lstm_2[0][0]
_________________________________________________________________________________
dense_1 (Dense)                  (None, 36)            1836        dropout_3[0][0]
=================================================================================
Total params: 45836
_________________________________________________________________________________
```

```
In [4]: model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

        print('Training')
        model.fit([X, Xq], Y, batch_size=BATCH_SIZE, nb_epoch=EPOCHS, validation_sp
        loss, acc = model.evaluate([tX, tXq], tY, batch_size=BATCH_SIZE)
        print('Test loss / test accuracy = {:.4f} / {:.4f}'.format(loss, acc))
```

```
Training
Train on 9500 samples, validate on 500 samples
Epoch 1/40
9500/9500 [==============================] - 173s - loss: 2.0364 - acc: 0.1686 - va
Epoch 2/40
9500/9500 [==============================] - 183s - loss: 1.8406 - acc: 0.1734 - va
Epoch 3/40
9500/9500 [==============================] - 179s - loss: 1.8069 - acc: 0.1963 - va
Epoch 4/40
9500/9500 [==============================] - 179s - loss: 1.7598 - acc: 0.2561 - va
Epoch 5/40
9500/9500 [==============================] - 185s - loss: 1.6991 - acc: 0.3159 - va
Epoch 6/40
9500/9500 [==============================] - 191s - loss: 1.6330 - acc: 0.3493 - va
Epoch 7/40
9500/9500 [==============================] - 184s - loss: 1.5769 - acc: 0.3713 - va
Epoch 8/40
9500/9500 [==============================] - 180s - loss: 1.5221 - acc: 0.3837 - va
Epoch 9/40
9500/9500 [==============================] - 188s - loss: 1.5118 - acc: 0.3861 - va
Epoch 10/40
9500/9500 [==============================] - 188s - loss: 1.4857 - acc: 0.3888 - va
Epoch 11/40
9500/9500 [==============================] - 198s - loss: 1.4603 - acc: 0.3971 - va
Epoch 12/40
9500/9500 [==============================] - 184s - loss: 1.4467 - acc: 0.3962 - va
Epoch 13/40
9500/9500 [==============================] - 174s - loss: 1.4353 - acc: 0.4002 - va
Epoch 14/40
9500/9500 [==============================] - 175s - loss: 1.4210 - acc: 0.4013 - va
Epoch 15/40
```

```
9500/9500 [==============================] - 174s - loss: 1.4154 - acc: 0.4031 - va
Epoch 16/40
9500/9500 [==============================] - 169s - loss: 1.4094 - acc: 0.4077 - va
Epoch 17/40
9500/9500 [==============================] - 177s - loss: 1.3876 - acc: 0.4118 - va
Epoch 18/40
9500/9500 [==============================] - 176s - loss: 1.3802 - acc: 0.4183 - va
Epoch 19/40
9500/9500 [==============================] - 180s - loss: 1.3710 - acc: 0.4214 - va
Epoch 20/40
9500/9500 [==============================] - 190s - loss: 1.3657 - acc: 0.4213 - va
Epoch 21/40
9500/9500 [==============================] - 176s - loss: 1.3549 - acc: 0.4318 - va
Epoch 22/40
9500/9500 [==============================] - 176s - loss: 1.3501 - acc: 0.4361 - va
Epoch 23/40
9500/9500 [==============================] - 178s - loss: 1.3339 - acc: 0.4407 - va
Epoch 24/40
9500/9500 [==============================] - 179s - loss: 1.3341 - acc: 0.4386 - va
Epoch 25/40
9500/9500 [==============================] - 182s - loss: 1.3132 - acc: 0.4522 - va
Epoch 26/40
9500/9500 [==============================] - 181s - loss: 1.3114 - acc: 0.4455 - va
Epoch 27/40
9500/9500 [==============================] - 189s - loss: 1.2999 - acc: 0.4547 - va
Epoch 28/40
9500/9500 [==============================] - 183s - loss: 1.2943 - acc: 0.4532 - va
Epoch 29/40
9500/9500 [==============================] - 179s - loss: 1.2865 - acc: 0.4591 - va
Epoch 30/40
9500/9500 [==============================] - 182s - loss: 1.2823 - acc: 0.4587 - va
Epoch 31/40
9500/9500 [==============================] - 186s - loss: 1.2665 - acc: 0.4638 - va
Epoch 32/40
9500/9500 [==============================] - 175s - loss: 1.2503 - acc: 0.4682 - va
Epoch 33/40
9500/9500 [==============================] - 182s - loss: 1.2478 - acc: 0.4717 - va
Epoch 34/40
9500/9500 [==============================] - 182s - loss: 1.2460 - acc: 0.4733 - va
Epoch 35/40
9500/9500 [==============================] - 184s - loss: 1.2408 - acc: 0.4803 - va
Epoch 36/40
9500/9500 [==============================] - 185s - loss: 1.2278 - acc: 0.4848 - va
Epoch 37/40
9500/9500 [==============================] - 173s - loss: 1.2280 - acc: 0.4782 - va
Epoch 38/40
9500/9500 [==============================] - 184s - loss: 1.2170 - acc: 0.4854 - va
Epoch 39/40
```

```
9500/9500 [==============================] - 188s - loss: 1.2086 - acc: 0.4939 - va
Epoch 40/40
9500/9500 [==============================] - 183s - loss: 1.2061 - acc: 0.4946 - va
1000/1000 [==============================] - 5s
Test loss / test accuracy = 1.3419 / 0.4390
```