谷歌深度学习公开课任务 6: LSTMs (http://www.hankcs.com/ml/task-lstms-6.html)

最后一次任务，至此速战速决解决了这门快餐课程。心得是作为一个“懒惰的工程师”，对常见的深度学习模型、技巧、应用有了浅显的了解。但对“好奇的求知者”来讲，则只能说看了一张模糊不清的缩略图，许多理论和细节得通过正式一些的课程去补充。

任务 6: LSTMs

训练一个长短期记忆网络预测字符串，我们已经给出了一个基本的LSTM模型，请通过解决给出的问题来优化它。

上次训练了一个skip-gram模型，这次在相同的text8语料上训练LSTM字符模型，该模型输入一个字符，预测接下来可能出现的那个字符，所以才叫字符模型。

谷歌一如既往贴心地给出了基础代码，简单地过一遍吧。代码首先将字符串语料库拆分为

训练集和验证集:

```
valid_size = 1000
valid_text = text[:valid_size]
train_text = text[valid_size:]
train_size = len(train_text)
print(train_size, train_text[:64])
print(valid_size, valid_text[:64])
```

输出

```
99999000 ons anarchists advocate social relations based upon voluntary as
1000  anarchism originated as a term of abuse first used against earl
```

第1、2行分别是训练集和验证集的前64个字符。但字符串只是训练数据的原始状态，可被神经网络接受的数据应该是一组one-hot的向量。所以需要一段转换代码:

```
batch_size=64
num_unrollings=10

class BatchGenerator(object):
  def __init__(self, text, batch_size, num_unrollings):
    self._text = text
    self._text_size = len(text)
    self._batch_size = batch_size
    self._num_unrollings = num_unrollings
    segment = self._text_size // batch_size
    self._cursor = [ offset * segment for offset in range(batch_size)]
    self._last_batch = self._next_batch()

  def _next_batch(self):
    """Generate a single batch from the current cursor position in the data
    batch = np.zeros(shape=(self._batch_size, vocabulary_size), dtype=np.flo
    for b in range(self._batch_size):
      batch[b, char2id(self._text[self._cursor[b]])] = 1.0
      self._cursor[b] = (self._cursor[b] + 1) % self._text_size
    return batch

  def next(self):
    """Generate the next array of batches from the data. The array consists
    the last batch of the previous array, followed by num_unrollings new one
    """
    batches = [self._last_batch]
    for step in range(self._num_unrollings):
      batches.append(self._next_batch())
    self._last_batch = batches[-1]
    return batches

def characters(probabilities):
  """Turn a 1-hot encoding or a probability distribution over the possible
  characters back into its (most likely) character representation."""
  return [id2char(c) for c in np.argmax(probabilities, 1)]

def batches2string(batches):
  """Convert a sequence of batches back into their (most likely) string
  representation."""
  s = [''] * batches[0].shape[0]
  for b in batches:
    s = [''.join(x) for x in zip(s, characters(b))]
  return s

train_batches = BatchGenerator(train_text, batch_size, num_unrollings)
valid_batches = BatchGenerator(valid_text, 1, 1)
```

```
print(batches2string(train_batches.next()))
print(batches2string(train_batches.next()))
print(batches2string(valid_batches.next()))
print(batches2string(valid_batches.next()))
```

next方法返回一个batches，由num_unrollings+1个batch构成，每个batch都是batch_size*27的矩阵，矩阵中每个行向量都是字母表上的one-hot向量。

理解了batches内部结构后，就可以通过batches2string把batches还原为batch_size个长num_unrollings+1的字符串。

输出

```
['ons anarchi', 'when milita', 'lleria arch', ' abbeys and', 'married urr',
['ists advoca', 'ary governm', 'hes nationa', 'd monasteri', 'raca prince',
[' a']
['an']
```

细心的话可以注意到相邻两个batches中的对应字符串都是首尾相接的:

> 'ons anarchi', 'when milita', 'lleria arch',
>
> 'ists advoca', 'ary governm', 'hes nationa',

首尾相接指的是，当前11个batch中的第一个与上11个batch中的最后一个相同。为什么要这么干呢？因为只有这么做，LSTM模型看到的才是连续的文本啊。其实谷歌的这种打印方式有点误导人啊，我还以为不同的batches相互构成训练实例呢。其实同一个batch内的每个序列构成一个训练实例，应该换行打印:

```
ists advoca
ary governm
hes nationa
d monasteri
raca prince
chard baer
rgical lang
for passeng
the nationa
took place
...
```

一共64行，也就是64个训练实例。拿“the nationa”举个例子吧：

```
num_unrollings = 10
train_data = 'the nationa'
train_inputs = train_data[:num_unrollings]
train_labels = train_data[1:]  # labels are inputs shifted by one time step

print('train_inputs=', train_inputs)
print('train_labels=', train_labels)
```

得到

```
train_inputs= the nation
train_labels= he nationa
```

这下子应该明白输入输出分别是什么吧，输出就是输入往后移动一个字符（对应的序列的one-hot向量），也就是（x,y）=（t,h）、(h,e)、(e,)……。当然它们必须构成一个连续序列，不然LSTM试图记忆的根本就不是有意义的句子。如果LSTM模型能够根据一个字符s预测该字符s的下一个字符c，我们不断地执行s+=c的话，就可以生成一整段有意义的文本了。

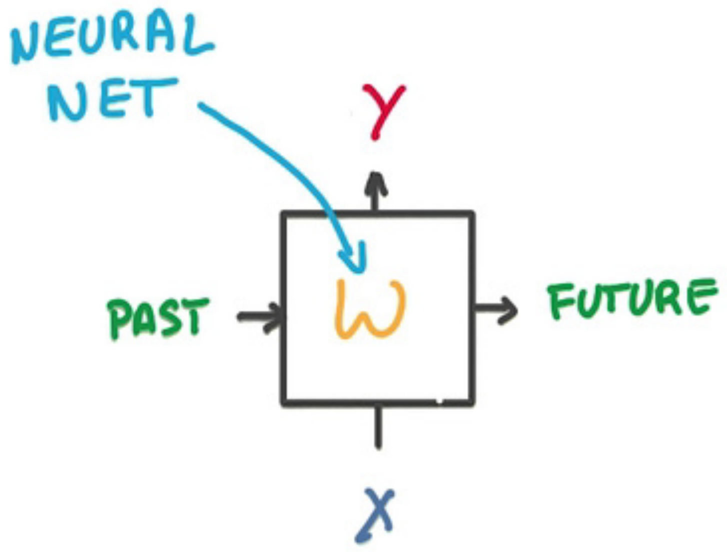
词与序列

如果说词是x的话，那么序列就是多个x组成的动态数组。

RNN

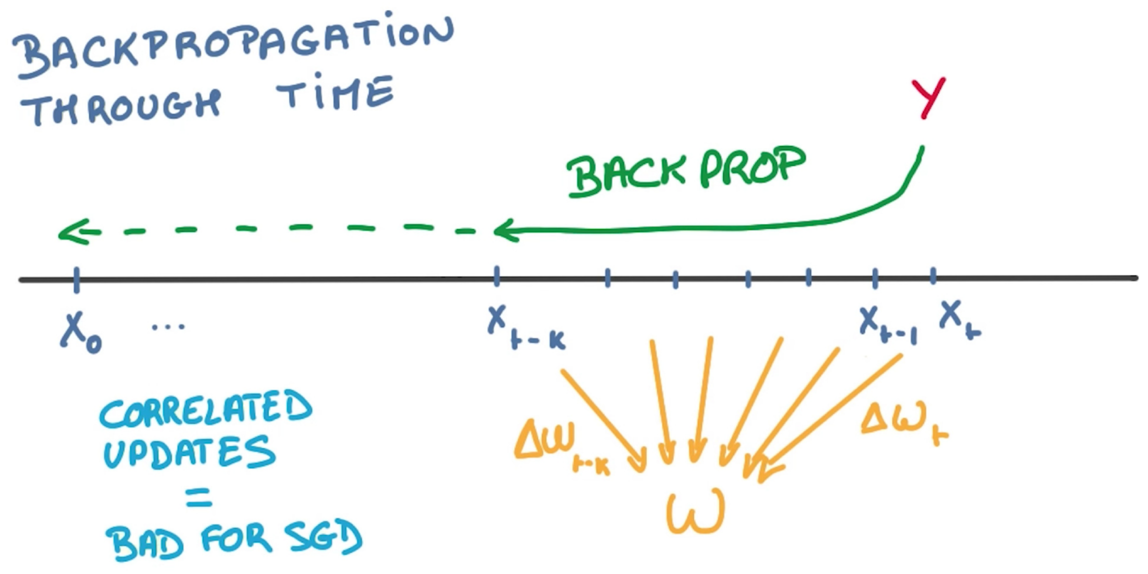
词可以方便地转为向量，但序列是变长的，不方便转为向量，也就不方便使用神经网络语

言模型来处理序列。RNN应运而生，其输入是单个向量x，但前一个输入x影响其内部状态，该状态和当前输入x同时影响分类结果。由于始终只有一个输入、一个分类器，但又确实能够处理不定长的输入，所以RNN可视作时间上的“参数共享”，与CNN在空间上的“参数共享”形成对照。



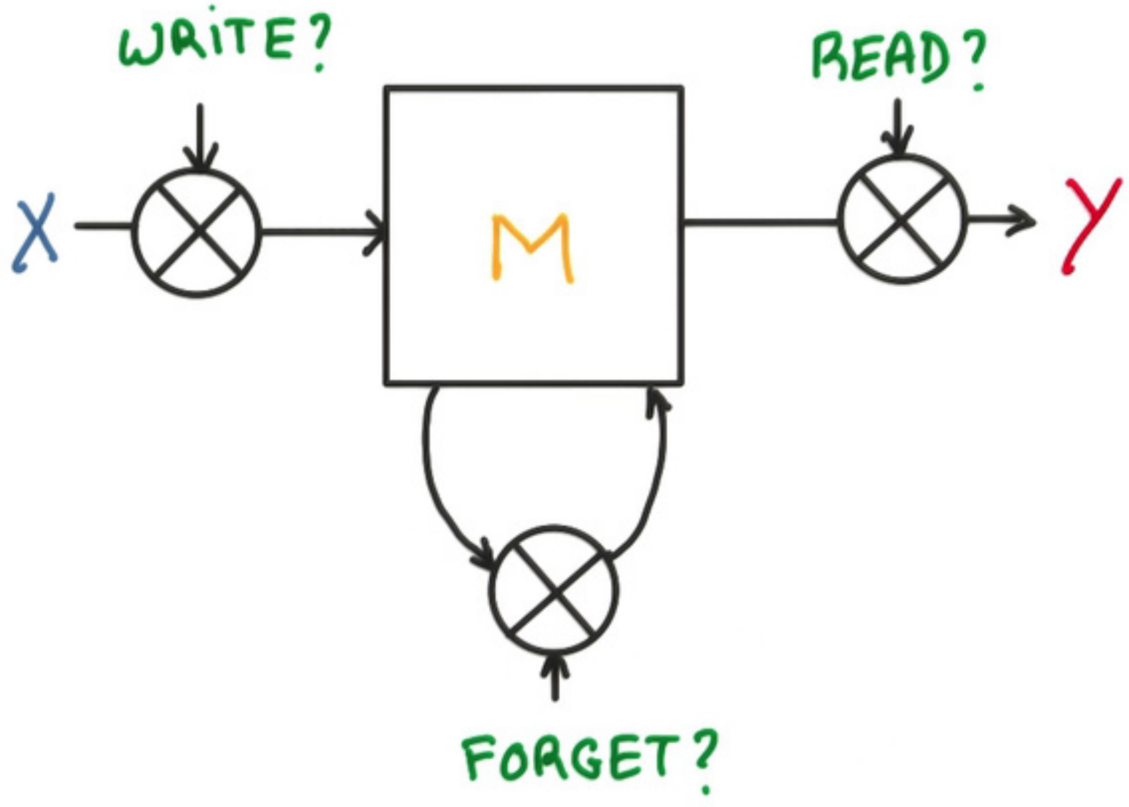
梯度消失

但过长的时间线不利于求导，对同一个参数求导次数多了之后梯度就会为0（课程没有详细深入），这会导致RNN记不住多远之前的事情（梯度为0，参数不发生变化，无法继续训练）：

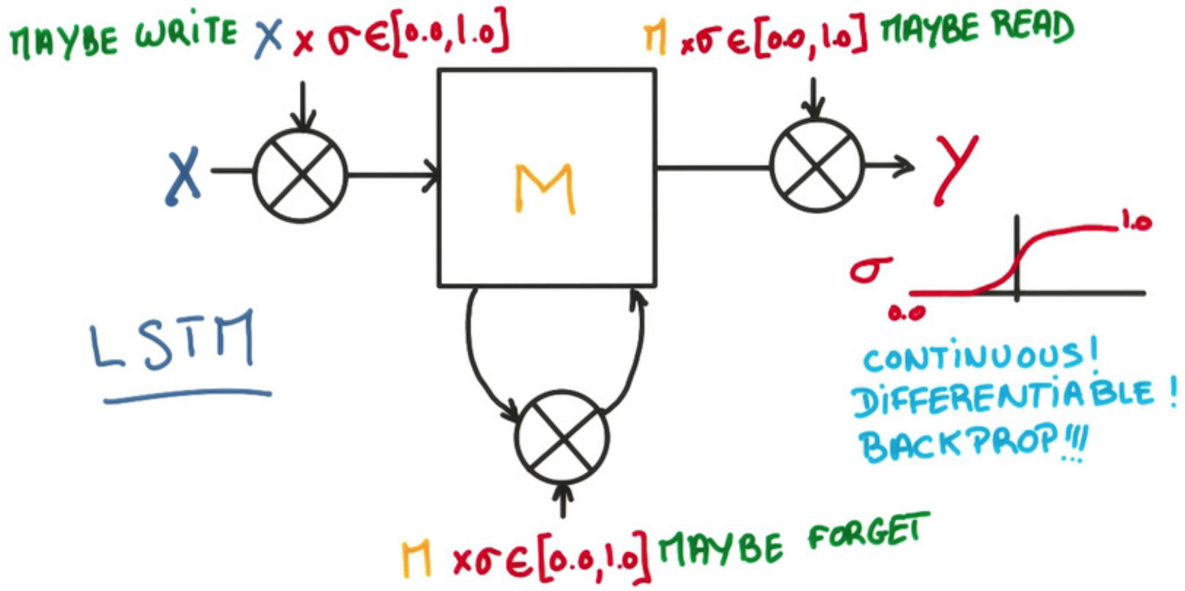


LSTM

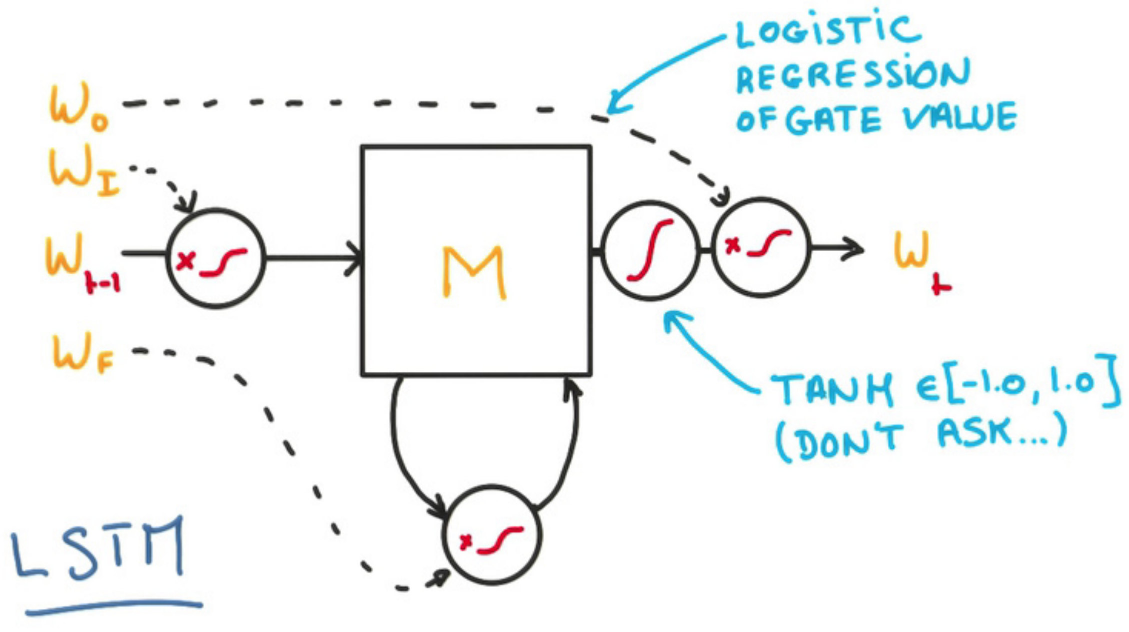
LSTM是RNN的增强版，允许遗忘部分信息，而保留重要的部分，其将RNN中的神经网络替换为LSTM-cell：



叉叉分别代表input_gate、forget_gate和output_gate，分别控制输入、遗忘和输出的程度。既然是程度而非布尔值，那就可以训练并优化了。


MAYBE WRITE X x σ ∈ [0.0, 1.0]
M x σ ∈ [0.0, 1.0] MAYBE READ
X
M
Y
σ
1.0
0.0
LSTM
CONTINUOUS!
DIFFERENTIABLE!
BACKPROP!!!
M x σ ∈ [0.0, 1.0] MAYBE FORGET

把时间线上的多个LSTM单元（实际上任何时刻都只有一个单元）串联在一起，得到如下网络：


LOGISTIC REGRESSION OF GATE VALUE
W_O
W_I
W_{t-1}
W_F
M
W_t
TANH ∈ [-1.0, 1.0] (DON'T ASK...)
LSTM

回到练习中来，仔细看看谷歌的示范代码：

```
num_nodes = 64

graph = tf.Graph()
with graph.as_default():
    # Parameters:
    # Input gate: input, previous output, and bias.
    ix = tf.Variable(tf.truncated_normal([vocabulary_size, num_nodes], -0.1,
    im = tf.Variable(tf.truncated_normal([num_nodes, num_nodes], -0.1, 0.1)
    ib = tf.Variable(tf.zeros([1, num_nodes]))
    # Forget gate: input, previous output, and bias.
    fx = tf.Variable(tf.truncated_normal([vocabulary_size, num_nodes], -0.1,
    fm = tf.Variable(tf.truncated_normal([num_nodes, num_nodes], -0.1, 0.1)
    fb = tf.Variable(tf.zeros([1, num_nodes]))
    # Memory cell: input, state and bias.
    cx = tf.Variable(tf.truncated_normal([vocabulary_size, num_nodes], -0.1,
    cm = tf.Variable(tf.truncated_normal([num_nodes, num_nodes], -0.1, 0.1)
    cb = tf.Variable(tf.zeros([1, num_nodes]))
    # Output gate: input, previous output, and bias.
    ox = tf.Variable(tf.truncated_normal([vocabulary_size, num_nodes], -0.1,
    om = tf.Variable(tf.truncated_normal([num_nodes, num_nodes], -0.1, 0.1)
    ob = tf.Variable(tf.zeros([1, num_nodes]))
    # Variables saving state across unrollings.
    saved_output = tf.Variable(tf.zeros([batch_size, num_nodes]), trainable=
    saved_state = tf.Variable(tf.zeros([batch_size, num_nodes]), trainable=F
    # Classifier weights and biases.
    w = tf.Variable(tf.truncated_normal([num_nodes, vocabulary_size], -0.1,
    b = tf.Variable(tf.zeros([vocabulary_size]))


    # Definition of the cell computation.
    def lstm_cell(i, o, state):
        """Create a LSTM cell. See e.g.: http://arxiv.org/pdf/1402.1128v1.pc
        Note that in this formulation, we omit the various connections betwe
        previous state and the gates."""
        input_gate = tf.sigmoid(tf.matmul(i, ix) + tf.matmul(o, im) + ib)
        forget_gate = tf.sigmoid(tf.matmul(i, fx) + tf.matmul(o, fm) + fb)
        update = tf.matmul(i, cx) + tf.matmul(o, cm) + cb
        state = forget_gate * state + input_gate * tf.tanh(update)
        output_gate = tf.sigmoid(tf.matmul(i, ox) + tf.matmul(o, om) + ob)
        return output_gate * tf.tanh(state), state


    # Input data.
    train_data = list()
    for _ in range(num_unrollings + 1):
        train_data.append(
            tf.placeholder(tf.float32, shape=[batch_size, vocabulary_size])
    train_inputs = train_data[:num_unrollings]
    train_labels = train_data[1:]  # labels are inputs shifted by one time 

    # Unrolled LSTM loop.
    outputs = list()
    output = saved_output
    state = saved_state
    for i in train_inputs:
        output, state = lstm_cell(i, output, state)
        outputs.append(output)

    # State saving across unrollings.
    with tf.control_dependencies([saved_output.assign(output),
                                  saved_state.assign(state)]):
        # Classifier.
        logits = tf.nn.xw_plus_b(tf.concat(0, outputs), w, b)
        loss = tf.reduce_mean(
            tf.nn.softmax_cross_entropy_with_logits(
                logits, tf.concat(0, train_labels)))

    # Optimizer.
    global_step = tf.Variable(0)
    learning_rate = tf.train.exponential_decay(
        10.0, global_step, 5000, 0.1, staircase=True)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    gradients, v = zip(*optimizer.compute_gradients(loss))
    gradients, _ = tf.clip_by_global_norm(gradients, 1.25)
    optimizer = optimizer.apply_gradients(
        zip(gradients, v), global_step=global_step)

    # Predictions.
    train_prediction = tf.nn.softmax(logits)

    # Sampling and validation eval: batch 1, no unrolling.
    sample_input = tf.placeholder(tf.float32, shape=[1, vocabulary_size])
    saved_sample_output = tf.Variable(tf.zeros([1, num_nodes]))
    saved_sample_state = tf.Variable(tf.zeros([1, num_nodes]))
    reset_sample_state = tf.group(
        saved_sample_output.assign(tf.zeros([1, num_nodes])),
        saved_sample_state.assign(tf.zeros([1, num_nodes])))
    sample_output, sample_state = lstm_cell(
        sample_input, saved_sample_output, saved_sample_state)
    with tf.control_dependencies([saved_sample_output.assign(sample_output)
                                  saved_sample_state.assign(sample_state)])
        sample_prediction = tf.nn.softmax(tf.nn.xw_plus_b(sample_output, w,
```

其中，lstm单元定义如下

```
def lstm_cell(i, o, state):
    """Create a LSTM cell. See e.g.: http://arxiv.org/pdf/1402.1128v1.pc
    Note that in this formulation, we omit the various connections betwe
    previous state and the gates."""
    input_gate = tf.sigmoid(tf.matmul(i, ix) + tf.matmul(o, im) + ib)
    forget_gate = tf.sigmoid(tf.matmul(i, fx) + tf.matmul(o, fm) + fb)
    update = tf.matmul(i, cx) + tf.matmul(o, cm) + cb
    state = forget_gate * state + input_gate * tf.tanh(update)
    output_gate = tf.sigmoid(tf.matmul(i, ox) + tf.matmul(o, om) + ob)
    return output_gate * tf.tanh(state), state
```

i是输入，o是上次输出，state是上次状态。虽然我们只有一个LSTM单元，但我们可以把时间线上连续多个时刻的LSTM串联（unroll）起来：

```
# Unrolled LSTM loop.
outputs = list()
output = saved_output
state = saved_state
for i in train_inputs:
    output, state = lstm_cell(i, output, state)
...
    input_gate = tf.sigmoid(tf.matmul(i, ix) + tf.matmul(o, im) + ib)
    forget_gate = tf.sigmoid(tf.matmul(i, fx) + tf.matmul(o, fm) + fb)
    update = tf.matmul(i, cx) + tf.matmul(o, cm) + cb
    state = forget_gate * state + input_gate * tf.tanh(update)
    output_gate = tf.sigmoid(tf.matmul(i, ox) + tf.matmul(o, om) + ob)
    return output_gate * tf.tanh(state), state
```

对应着论文 (https://arxiv.org/pdf/1402.1128v1.pdf)里面：

$$\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i)\\
f_t &= \sigma(W_{fx}x_t + W_{mf}m_{t-1} + W_{cf}c_{t-1} + b_f)\\
c_t &= f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c)\\
o_t &= \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o)\\
m_t &= o_t \odot h(c_t)\\
y_t &= W_{ym}m_t + b_y
\end{aligned}$$

ifco分别表示输入门、遗忘门、记忆单元、输出。

至于为什么要这样更新权值，那就不在这门课的讨论范围内了，毕竟听众都是“懒惰的工程师”。

运行后得到

```
================================================================================
jam also with poles instendent duss where the came incomplemented lead appea
abra been knows hoine from one nine nine nine four one one blany common majo
les manubely strugule lices for pandels clha prinal temporaty the portand or
ing doog indersevents procuctions to energos never big or copparter now to b
odners and s atmency of the english towiall were the prograge is annotles ma
================================================================================
```

题目 1

注意到上面对4个元件分别有4个矩阵，请用1个矩阵替代它。

那就把这4个矩阵拼起来喽，参数定义：

```
# Concatenate parameters
sx = tf.concat(1, [ix, fx, cx, ox])
sm = tf.concat(1, [im, fm, cm, om])
sb = tf.concat(1, [ib, fb, cb, ob])
```

lstm定义

```
# Definition of the cell computation.
def lstm_cell(i, o, state):
    """Create a LSTM cell. See e.g.: http://arxiv.org/pdf/1402.1128v1.pdf
    Note that in this formulation, we omit the various connections between t
    previous state and the gates."""
    y = tf.matmul(i, sx) + tf.matmul(o, sm) + sb
    y_input, y_forget, update, y_output = tf.split(1, 4, y)
    input_gate = tf.sigmoid(y_input)
    forget_gate = tf.sigmoid(y_forget)
    output_gate = tf.sigmoid(y_output)
    state = forget_gate * state + input_gate * tf.tanh(update)
    return output_gate * tf.tanh(state), state
```

得到的结果是类似的：

```
================================================================================
hiam result of and colle orgation this socroishn shore the world clasic to g
on boons uss involund wectre scure o canforles was the on artic world far do
ation notyarih the conjund jathers penime of the temperstory relege to not t
chardre in the dural forwershadiath didyation d isa astrace by one eight two
phility fich and dr and humentingmentiindly versmos event wave basether that
================================================================================
Validation set perplexity: 4.28
```

问题**2**

请将字符LSTM改造为Bigram-LSTM，注意Bigram数量很大，one-hot编码会极其稀疏，请：

a. 引入embedding

b. 引入Dropout

Bigram

Bigram的改造并不困难（参考 (https://github.com/ahangchen/GDLnotes/blob/master/src/rnn/embed_bigram_lstm.py)），以前词表大小为27，bigram的话为27*27。

```
bigram_vocabulary_size = vocabulary_size * vocabulary_size


class BigramBatchGenerator(object):
    def __init__(self, text, batch_size, num_unrollings):
        self._text = text
        self._text_size_in_chars = len(text)
        self._text_size = self._text_size_in_chars // 2
        self._batch_size = batch_size
        self._num_unrollings = num_unrollings
        segment = self._text_size // batch_size
        self._cursor = [offset * segment for offset in range(batch_size)]
        self._last_batch = self._next_batch()

    def _next_batch(self):
        batch = np.zeros(shape=self._batch_size, dtype=np.int)
        for b in range(self._batch_size):
            char_idx = self._cursor[b] * 2
            ch1 = char2id(self._text[char_idx])
            if self._text_size_in_chars - 1 == char_idx:
                ch2 = 0
            else:
                ch2 = char2id(self._text[char_idx + 1])
            batch[b] = ch1 * vocabulary_size + ch2
            self._cursor[b] = (self._cursor[b] + 1) % self._text_size
        return batch

    def next(self):
        batches = [self._last_batch]
        for step in range(self._num_unrollings):
            batches.append(self._next_batch())
        self._last_batch = batches[-1]
        return batches


def bi2str(encoding):
    return id2char(encoding // vocabulary_size) + id2char(encoding % vocabu


def bigrams(encodings):
    return [bi2str(e) for e in encodings]


def bibatches2string(batches):
    s = [''] * batches[0].shape[0]
    for b in batches:
        s = [''.join(x) for x in zip(s, bigrams(b))]
    return s


bi_onehot = np.zeros((bigram_vocabulary_size, bigram_vocabulary_size))
np.fill_diagonal(bi_onehot, 1)


def bigramonehot(encodings):
    return [bi_onehot[e] for e in encodings]


train_batches = BigramBatchGenerator(train_text, 8, 8)
valid_batches = BigramBatchGenerator(valid_text, 1, 1)

print (bibatches2string(train_batches.next()))
print (bibatches2string(train_batches.next()))
print (bibatches2string(valid_batches.next()))
print (bibatches2string(valid_batches.next()))
```

输出

```
['ons anarchists adv', 'on from the nation', 'significant than i', 'ain dru
['dvocate social rel', 'onal media and fro', ' in jersey and gue', 'ion ina
[' ana']
['narc']
```

注意这里的next方法不再生成one-hot向量构成的矩阵，因为该向量太稀疏，马上要用embedding替代它。

embedding

目的是用embedding_size = 128的向量来替代27*27的one-hot向量，理论上只要将这27*27个向量映射为互不相同的向量，都可以完成训练，所以这份代码直接用了随机初始化的embedding

```
# embeddings for all possible bigrams
embeddings = tf.Variable(tf.random_uniform([bigram_vocabulary_size, embeddi
```

不过我觉得这个embedding应该事先通过CBOW之类的语言模型训练出来，如此效果才

会好。

之后的输入i（id形式，因为要调用embedding_lookup所以不能转为one-hot）就替换为i的embedding了：

```
# embed input bigrams -> [batch_size, embedding_size]
output, state = lstm_cell(tf.nn.embedding_lookup(embeddings, i), output, sta
```

输出倒是和原来类似，是27*27的one-hot构成的矩阵。

```
for l in train_data[1:]:
    train_labels.append(tf.gather(bigram_one_hot, l))
```

Dropout

Dropout加在输入和输出上：

```
def lstm_cell(i, o, state):
    i = tf.nn.dropout(i, keep_prob)
    mult = tf.matmul(i, x) + tf.matmul(o, m) + biases
    input_gate = tf.sigmoid(mult[:, :num_nodes])
    forget_gate = tf.sigmoid(mult[:, num_nodes:num_nodes * 2])
    update = mult[:, num_nodes * 3:num_nodes * 4]
    state = forget_gate * state + input_gate * tf.tanh(update)
    output_gate = tf.sigmoid(mult[:, num_nodes * 3:])
    output = tf.nn.dropout(output_gate * tf.tanh(state), keep_prob)
    return output, state
```

其他与字符LSTM一模一样。

最终结果

```
================================================================================
geller depaqand as characturn essaot emonist the signatually only eight eigl
ebra colling with of specified for the however their of boun collecument wi
kxeone file following from especists tv elements in the souther song perndar
ykm u kespecame the used from english city of the sullent aiklintial pddedil
by of is a spate areal depindition religions with has are sproducules well 1
================================================================================
Validation set perplexity: 17.15
```

perplexity变差了许多，我猜就是随机embedding的原因。

题目 3

实现sequence-to-sequence LSTM，让其学会如下序列转换规律：

```
For example, if your input is:
the quick brown fox
the model should attempt to output:
eht kciuq nworb xof
```

也就是逆转序列中的每个词。这其实是个无厘头的题目，毫无实际意义。seq2seq的一个典型应用是机器翻译，比如官方的英法翻译，或者“将单词的读音翻译为单词的拼写(https://github.com/mikesj-public/rnn_spelling_bee/blob/master/spelling_bee_RNN.ipynb)”这个例子，都有趣多了。

另外，其实tf中就有一整个开箱即用的seq2seq模块，只需填充参数即可：

```
model = seq2seq_model.Seq2SeqModel(source_vocab_size=vocabulary_size,
                                   target_vocab_size=vocabulary_size,
                                   buckets=[(20, 20)],
                                   size=256,
                                   num_layers=4,
                                   max_gradient_norm=5.0,
                                   batch_size=batch_size,
                                   learning_rate=1.0,
                                   learning_rate_decay_factor=0.9,
                                   use_lstm=True,
                                   forward_only=forward_only)
```

buckets表示输入输出都是len=20的字符串（其实RNN、LSTM的特长是应该处理不定长

的序列，这里让输入输出等长没体现出其优势），至于如何将 seq->vector->lstm->vector->seq，那就是这个模块的内部细节了。作为快餐课程的听众，暂时只需要弄明白训练数据是什么样，然后会调用该模块即可，深入的学习估计会在cs224d上。

训练数据就是一个句子以及其每个词的逆转：

```
def rev_id(forward):
    temp = forward.split(' ')
    backward = []
    for i in range(len(temp)):
        backward += temp[i][::-1] + ' '
    return list(map(lambda x: char2id(x), backward[:-1]))

batches = train_batches.next()
train_sets = []
batch_encs = list(map(lambda x: list(map(lambda y: char2id(y), list(x))), ba
batch_decs = list(map(lambda x: rev_id(x), batches))
print('x=', ''.join([id2char(x) for x in batch_encs[0]]))
print('y=', ''.join([id2char(x) for x in batch_decs[0]]))
```

输出

```
x= he diggers of the e
y= eh sreggid fo eht e
```

训练耗时较长，完整代码在https://github.com/hankcs/udacity-deep-learning (https://github.com/hankcs/udacity-deep-learning) 。

分享到：更多 ()

继续浏览有关 机器学习 (http://www.hankcs.com/ml/) LSTM (http://www.hankcs.com/tag/lstm/) RNN (http://www.hankcs.com/tag/rnn/) 深度学习 (http://www.hankcs.com/tag/%e6%b7%b1%e5%ba%a6%e5%ad%a6%e4%b9%a0/) 的文章

上一篇
谷歌深度学习公开课任务 5: Word2Vec&CBOW (http://www.hankcs.com/ml/cbow-word2vec.html)

Hinton神经网络公开课2 The Perceptron learning procedure (http://www.hankcs.com/ml/the-perceptron-learning-procedure.html)
下一篇

评论 1

此处不受理任何开源项目问题，请在GitHub上发issue ，大家一起讨论，谢谢。

提交评论

昵称	昵称 (必填)
邮箱	邮箱 (必填)
网址	网址

大厉害了，，之前一直以为这个网站是一个大型网站。。。。没想到是这么文艺，又厉害的人。。 #1

xqnq2007 (http://weibo.com/2435336380) 10个月前 (12-04)

我的开源项目

HanLP自然语言处理包 (https://github.com/hankcs/…

基于DoubleArrayTrie的Aho Corasick自动机 (https:…