Lecture 8:RNN and Language model

2018 年 7 月 15 日

1 Language Model

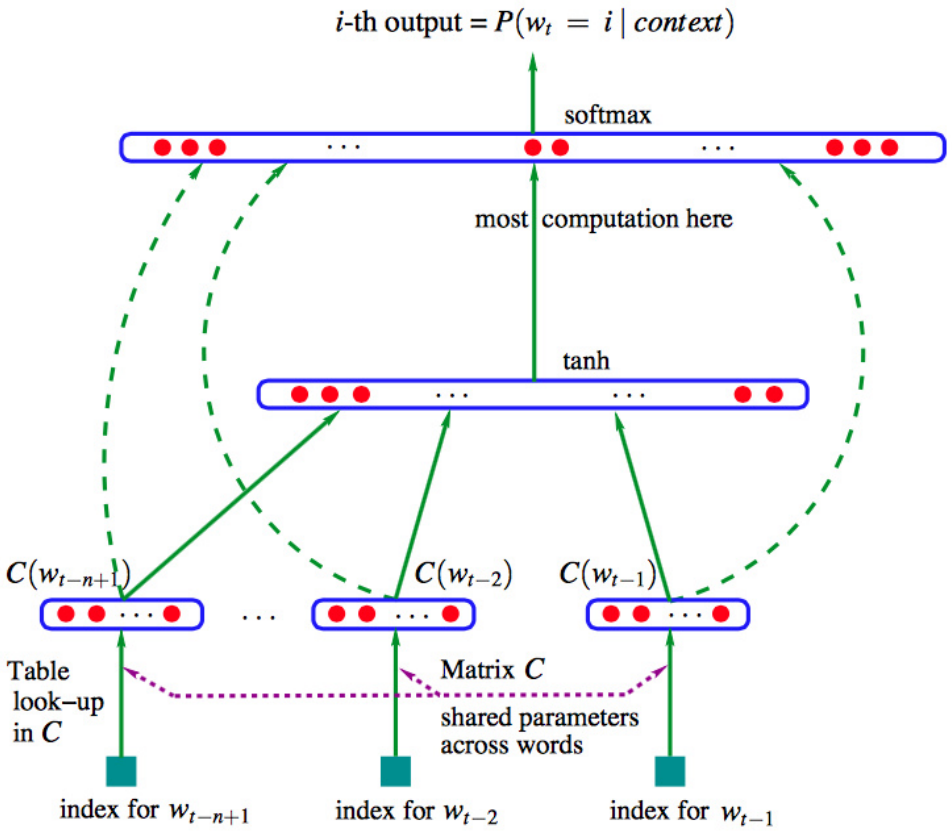
1.1 目标

引入马尔科夫假设后计算一个单词序列出现的概率

$$P(w_1, w_2, \cdots, w_m) \approx \prod_{i=1}^{m} P(w_i | w_{i-n}, \cdots, w_{i-1})$$

2 NNLM (Bengio 2003)

运用了 Markov 假设的暴力建模，给定前 n 个词预测第 $n+1$ 个词, 最小化交叉熵。



x 是长为 n 的窗口的词向量的拼接, softmax 内部要运算一个长为 $|V|$ 的巨大向量。

预测公式 ：

$$\hat{y} = softmax(W^{(2)} tanh(W^{(1)} x + b^{(1)}) + W^{(3)} x + b^{(3)})$$

模型缺陷

- 若使用 n-grams 预测，最终模型大小为 $O(|V|^n)$
- 稀疏性：若某个询问在训练语料中未出现，则条件概率为 0.

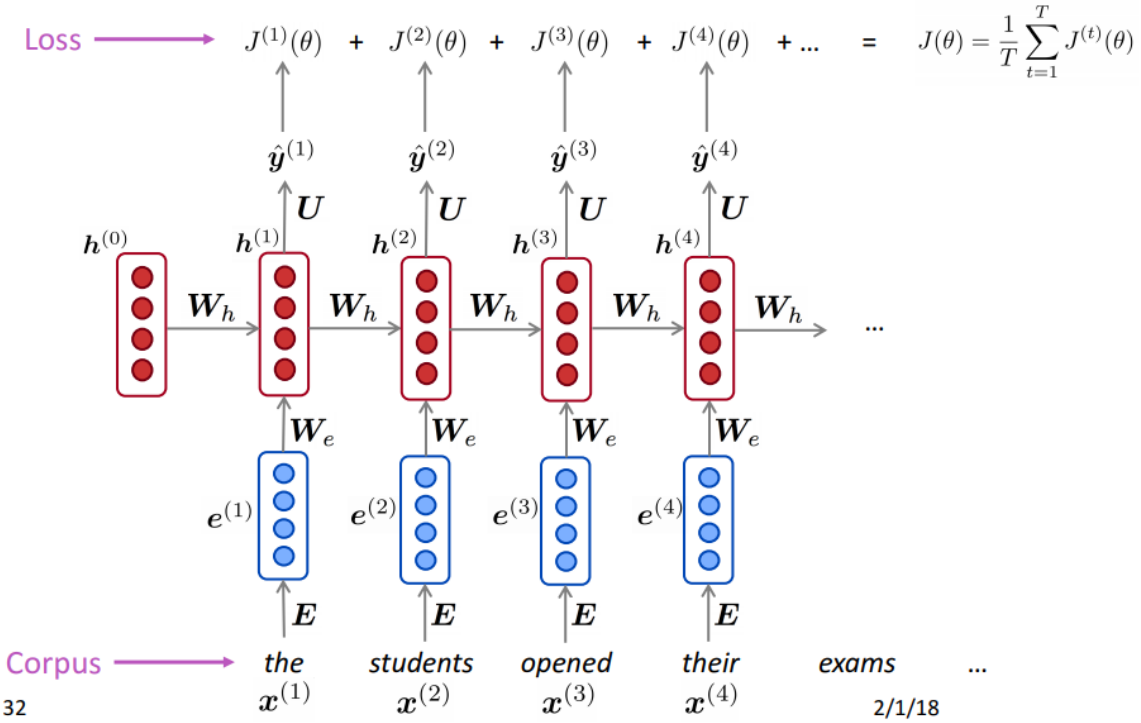
疑问：

1. 将 $b^{(2)}$ 换成 $W^{(3)}x + b^{(3)}$ 意义何在?
2. ~~训练该网络时，应该如何喂数据？将一个窗口优化到什么程度再换另一个窗口呢?~~

输入数据时，由于窗口大小固定，存在跨语句情况，如何解决？

3 Recurrent Neural Network

RNN 复用不同时刻的权重矩阵 W 以及隐层向量 h，理论上所有出现过的单词都会影响到当前预测单词。



3.1 模型参数

- $x^{(t)} \in R^d$: t 时刻输入单词的词向量
- $W_e \in R^{D_h \times d}$
- $h^{(t-1)} \in R^{D_h}$: t-1 时刻前积累的隐层向量
- $W_h \in R^{D_h \times D_h}$
- $W_S \in R^{|V| \times D_h}$ 喂给 softmax 的激活向量的权重矩阵

3.2 损失函数

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_j^{(t)} log(\hat{y}_j^{(t)})$$

3.3 训练

3.3.1 data

实际上，计算整个语料 (corpus) 的开销太大了，一般都是每隔一个/几个句子更新一次。

假设文本长度为 T, 分成 k 个长度为 $\frac{T}{k}$batch。窗口大小是 n 则将每个 batch 分成 $\frac{T}{kn}$ 个长为 n 的片段，每次训练放入 $k \times n$ 的数据, 共 $\frac{T}{kn}$ 次所有数据可全部放入，这称为一个 epoch.

由于一个 epoch 不足以让模型达到最优，所以重复以上步骤进行多个 epoch 以最小化 $J(\theta)$

3.3.2 Feed Forward

给定前 t-1 时刻的隐层信息 + t 时刻 input，更新隐层向量

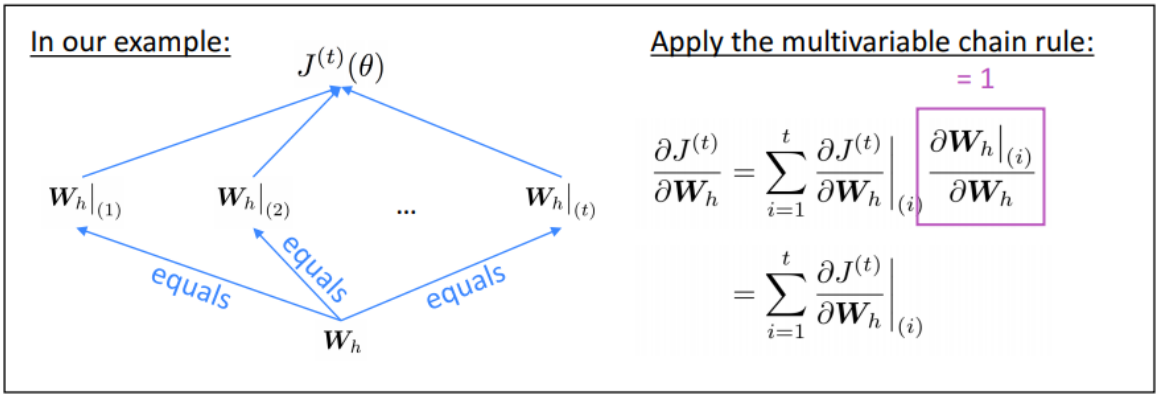
$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e x^{(t)} + b_1) = \sigma([W_h, W_e] \cdot \begin{bmatrix} h^{(t-1)} \\ x^{(t)} \end{bmatrix} + b_1)$$

t 时刻做出对 $t+1$ 时刻的预测 $\hat{y}^{(t)}$

$$\hat{y}^{(t)} = softmax(W_S h^{(t)} + b_2)$$

3.3.3 Back Propagation

有意思的是 T 长度时间序列上的求导:



$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{T} \frac{\partial J^{(t)}}{\partial W_h}|_{(i)}$$

$$\frac{\partial J^{(t)}}{\partial W_e} = \sum_{i=1}^{T} \frac{\partial J^{(t)}}{\partial W_e}|_{(i)}$$

4 梯度爆炸 and 梯度消失

Simplified annotation:

$$h_t = W f(h_{t-1}) + W_e x^{(t)}$$

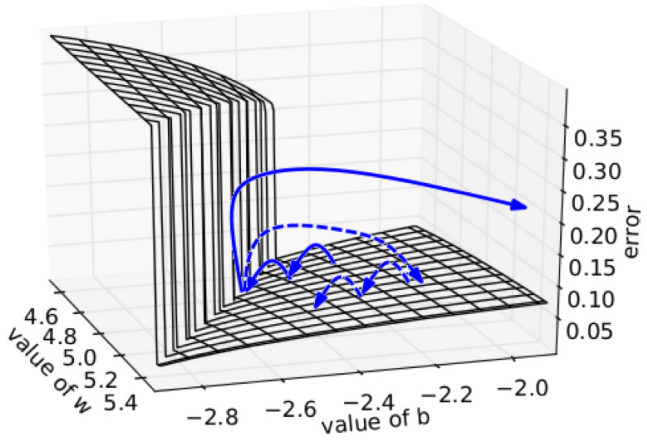
参数 W 关于损失函数 E_t 的梯度:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \tag{1}$$

对链式法则第三项:

$$\begin{aligned} \|\frac{\partial h_j}{\partial h_{j-1}}\| &= \|\frac{\partial (W f(h_{j-1})) + W_e x^{(t)}}{\partial h_{j-1}}\| \\ &= \|W \frac{\partial (f(h_{j-1}))}{\partial h_{j-1}}\| \\ &\leq \|W^T\| \|diag[f^{'}(h_{j-1})]\| \leq \beta_W \beta_h \end{aligned} \tag{2}$$

则 $\partial h_t / \partial h_k \leq (\beta_W \beta_h)^{t-k}$, 底数大于 1 时梯度爆炸，指数小于 1 时梯度消失。



4.0.1 Gradient Clipping

有效解决梯度爆炸:

```
grad = dE/ dtheta
if norm(grad) >= threshold:
    g = (threshold/norm(grad)) * grad
```

4.0.2 Initialization + ReLus

$\|W\| \cdot \|diag(f'(h_{j-1}))\| \to 1$

I　ReLu

是否只能保证前n个epoch不会梯度消失?

有效解决梯度消失

```
#initialize
    W = np.eye(dimh)
    activation_function := rect(z) := max(z, 0)
```

4.1 具体结构

```
def gen_batch(raw_data, batch_size, num_steps):
    # data_length 文本长度
    # batch_size  分成batch_size份
    # num_steps

    batch_length = data_length / batch_size
    data_x = np.zeros([batch_size, batch_length])
    data_y = np.zeros([batch_size, batch_length])

    for i in range(batch_size):
        data_x[i] = raw_x[batch_length*i: batch_length*(i+1)]
        data_y[i] = raw_y[batch_length*i: batch_length*(i+1)]

    # RNN模型一次只处理num_steps个数据
    epoch_size = batch_length / num_steps

    for i in range(epoch_size):
        x = data_x[:, num_steps*i: num_steps*(i+1)]
        y = data_y[:, num_steps*i: num_steps*(i+1)]
        yield(x, y) #生成器

#n为在样本规模上循环的次数
def gen_batches(n, num_steps):
    for i in range(n):
        yield gen_batch(gen_data(), batch_size, num_steps)

def RNN():
    x = tf.placeholder(tf.int32, [batch_size, num_steps])
    y = tf.placeholder(tf.int32, [batch_size, num_steps])

    init_state = tf.zeros([batch_size, state_size])

    #输入转换为onehot, number_classes = |V|,[batch_size, num_steps, num_classes]
    x_onehot = tf.onehot(x, number_classes)

    #在num_steps上解绑: num_step个[batch_size, num_class]的tensor
    rnn_inputs = tf.unstack(x_onehot, axis=1)

    with tf.variable_scope('rnn_cell') as scope:
        W = tf.get_variable('W', [num_classes + state_size, state_size])
        b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(
                                                0.0))

#循环使用相同cell参数
```

```python
def rnn_cell(rnn_input, state):
    with tf.variable_scope('rnn_cell', reuse=True):
        W = tf.get_variable('W', [num_classes + state_size, state_size])
        b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(
                                                    0.0))
    return tf.tanh(tf.matmul(tf.concat([rnn_input, state],axis=1),W)+b)

state = init_state #(batch_size, state_size)
rnn_outputs = []

# 循环num_steps次，每次输入一个[batch_size, num_class]
for rnn_input in rnn_inputs:
    state = rnn_cell(rnn_input, state)
    rnn_outputs.append(state)
```