Assignment 2

蜡笔大龙猫@2017.03.22

邮箱：houlisha1987@126.com

第二次作业我们要学习的是如何在NLP任务中使用TensorFlow解决问题。我们将会学习使用TensorFlow实现前向传播网络和循环神经网络RNN，并且将它们用在命名实体识别和语言模型任务中。

本次作业需要我们提前安装好TensorFlow和Numpy。

1. TensorFlow Softmax

本题中，我们将实现一个损失函数如下的线性分类器：

$$J_{softmax-CE}(W) = CE(y, sofrmax(xW))$$

这里 x 的行为特征向量。我们将使用TensorFlow自动微分的特性为提供的数据模拟模型。

> TensorFlow的自动求微分：基于梯度的机器学习算法会受益于TF的这一能力。作为TensorFlow用户，你只需要定义预测模型的结构，将这个结构和目标函数结合在一起，并添加数据，TF将自动为你计算相关的微分导数。

(a). 在q1_softmax.py文件中使用TensorFlow实现softmax()函数。注意不要使用tf.nn.softmax或者其他内嵌函数。已有代码只有简单的测试数据。Softmax函数为：

$$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

解：

注意softmax()函数注释部分给出的提示：

(1). 可以使用tf.exp， tf.reduce_max， tf.reduce_sum， tf.expand_dims（为张量增加一个维度）方法；

(2). 很多TF操作有简便的方式，例如 $x * y$ 即可表示两个张量相乘；

(3). 我们在后续实现模型的时候，传入到本函数的特征向量是行向量，所以这里不需要处理列向量的情况。

答案见代码文件。

(b). 在q1_softmax.py文件中使用TensorFlow实现cross_entropy()函数。注意不要使用TensorFlow内嵌交叉熵计算方法。已有代码只有简单测试。交叉熵函数为：

$$CE(y,\hat{y}) = -\sum_{i=1}^{N_c} y_i log(\hat{y}_i)$$

其中，$y \in R^5$是one-hot类别向量， N_c是类别数量。

解：

cross_entropy()函数注释部分给出的提示：

(1). 可以使用的TF方法：tf.to_float，tf.reduce_sum，tf.log；

答案见代码文件。

(c). 仔细研究model.py文件中的Model类。简单阐述TensorFlow计算过程中placeholder variables和feed dictionaries的作用。在q1_classifier.py文件中实现add_placeholders()和create_feed_dict()方法。

> 注意配置变量都存储在Config类中，代码中需要使用这些配置参数。

解：

tf.placeholder 变量用来在TensorFlow计算图中为数据提供入口，即为模型提供输入，在训练的过程中会传入训练数据进去。

feed_dict是一个从tf.placeholder变量（或者它们的名字）到数据（numpy数组、列表等）的python词典映射。一般形式为：

```
feed_dict = {<placeholder>: <tensor of values to be passed for placeholder>,...}
```

举个例子：

```
input1 = tf.placeholder(tf.float32)  # Define tf.placeholder objects for data entry.
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
        print(sess.run([output], feed_dict={input1:[7.], input2:[2.]})) # Feed data
into computation graph.
    ....
```

其他答案见代码文件。

(d). 在文件q1_classifier.py中实现add_model()方法，完成softmax分类。在这个文件的add_loss_op()方法中添加交叉熵损失。使用前面几个小题已经实现的方法，不要使用TF的内嵌函数。

解：

答案见代码文件。

(e). 在文件q1_classifier.py中实现add_training_op()方法。解释TensorFlow的自动求微分如何省掉定义梯度的过程？确保你的程序通过测试。

> 一定要使用Config类中的学习率。

解：

作为TensorFlow用户，你只需要定义预测模型的结构，将这个结构和目标函数结合在一起，并添加数据，TF将自动为你计算相关的微分导数。

add_training_op()方法中创建一个优化器，应用到所有需要训练的参数的梯度。TF的优化器可以参考https://www.tensorflow.org/versions/r0.7/api_docs/python/train.html#Optimizer。在本题中，只要使用tf.train.GradientDescentOptimizer最小化损失即可。

代码答案见代码文件。

2. Deep Networks for NER

这部分，我们将要练习后向传播和训练深度网络，完成命名实体识别的任务，即给定一个词和它的上下文，预测该词属于以下哪个类别：

- Person（PER）人名
- Organization（ORG）机构名
- Location（LOC）地名
- Miscellaneous（MISC）杂项

我们将这个问题归结为分为5个类别的分类问题，在上述4个类别上添加 O 空类别，标识不属于实体的词（大多数词属于这个类别）。

这是一个单隐藏层的神经网络模型，和word2vec类似有一个额外的表达层。不同于前面对窗口取平均或者采样，这里我们目标词和它的上下文邻居词定义为一个窗口，即：

$$x^{(t)} = [x_{t-1}L, x_tL, x_{(t+1)}L] \in R^{3d}$$

这里，输入x_{t-1}, x_t, x_{t+1}分别表示one-hot行向量，L是词嵌入矩阵$L \in R^{|V|*d}$，每行L_i表示词$i = x_t$的词向量。我们通过下面的公式来预测：

$$h = tanh(x^{(t)}W + b_1)$$

$$\hat{y} = softmax(hU + b_2)$$

使用交叉熵评估损失：

$$J(\theta) = CE(y, \hat{y}) = -\sum_{i=1}^{5} y_i log\hat{y}_i$$

计算训练集的损失，我们对每条样本的$J(\theta)$求和或者平均。

本题，我们取 $d = 50$ 作为词向量的长度，窗口大小为3的情况下长度为150。隐藏层维度取100，输出层$\hat{y}$维度为5.

(a). 计算损失$J(\theta)$对模型中所有参数的梯度：

$$\frac{\partial J}{\partial U}\frac{\partial J}{\partial b_2}\frac{\partial J}{\partial W}\frac{\partial J}{\partial b_1}\frac{\partial J}{\partial L_i}$$

其中，$U \in R^{100*5}, b_2 \in R^5, W \in R^{150*100}, b_1 \in R^{100}, L_I \in R^{50}$

后向传播需要你将激活函数（tanh，softmax）表示为它们值的形式，就像作业一求sigmoid的导数那样。这个公式可能对你有所帮助：

$$tanh(z) = 2sigmoid(2z) - 1$$

并且，你应该将梯度表示为传播回每层的"误差向量"，这就意味着链式法则可以写成用括号括起来的矩阵相乘的形式，简化了我们的分析过程。

> 大部分的工作已经在Assignment 1中完成了。

解：

根据讲义中推导的公式有：

输出层：$\hat{y} = softmax(hU + b_2)$　此处误差累积为：$\delta^{(2)} = (\hat{y} - y)$

隐藏层：$h = tanh(x^{(t)}W + b_1)$　　此处误差累积为：$\delta^{(1)} = (\hat{y} - y)U^T \bigodot tanh'(z)$

标记:　$tanh'(x^{(t)}W + b_1) = 4\sigma(2(x^{(t)}W + b_1))(1 - \sigma(2(x^{(t)}W + b_1)))$

所以：

$$\frac{\partial J}{\partial U} = h^T(\hat{y} - y)$$

$$\frac{\partial J}{\partial b_2} = (\hat{y} - y)$$

$$\frac{\partial J}{\partial W} = x^{(t)T}((\hat{y} - y)U^T \bigodot tanh')$$

$$\frac{\partial J}{\partial b_1} = (\hat{y} - y)U^T \bigodot tanh'$$

$$\frac{\partial J}{\partial L_i} = ((\hat{y} - y)U^T \bigodot tanh')W^T = [\frac{\partial J}{\partial L_{(t-1)}}, \frac{\partial J}{\partial L_T}, \frac{\partial J}{\partial L_{(t+1)}}]$$

这里要特别注意的是在激活函数求梯度的时候，需要矩阵点乘$\bigodot$。还需要特别注意的是矩阵的维度。

(b).为了避免参数过多或者参数的高相关性，一种有效的方法是损失函数增加高斯先验，这会使参数权重更接近于0，而且不约束参数的方向。这种方法会更好的提高分类器的泛化能力。

如果最大化交叉熵损失函数的log似然，那么高斯先验就变成了平方项（L2正则化）：

$$J_{reg}(\theta) = \frac{\lambda}{2}[\sum_{i,j} W_{ij}^2 + \sum_{i',j'} U_{i'j'}^2]$$

结合后损失函数为：

$$J_{full}(\theta) = J(\theta) + J_{reg}(\theta)$$

根据这个损失函数，重新计算（a）小题中的梯度。

> Optional (not graded): The interested reader should prove that this is indeed the maximum-likelihood objective when we let Wij ~ N(0,1/λ) for all i,j.

解：

$$\frac{\partial J_{full}}{\partial W} = \frac{\partial J}{\partial W} + \frac{\partial J_{reg}}{\partial W} = \frac{\partial J}{\partial W} + \lambda W$$

$$\frac{\partial J_{full}}{\partial U} = \frac{\partial J}{\partial U} + \frac{\partial J_{reg}}{\partial U} = \frac{\partial J}{\partial U} + \lambda U$$

(c). 神经元的相关性强容易陷入局部最优，为了避免这种情况，可以随机初始化参数。一个最常用的方法叫做Xavier初始化。给定$m \times n$维的矩阵A，从$[-\epsilon, \epsilon]$区间均匀采样A_{ij}，其中：

$$\epsilon = \frac{\sqrt{6}}{\sqrt{m+n}}$$

在文件q2_initialization.py中实现初始化函数xavier_weight_init()，用于初始化权重W和U。

> This is also referred to as Glorot initialization and was initially described in http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

解：

答案见代码文件。

(d). 在文件q2_NER.py中实现NER窗口模型。为了快速开发原型，TF的自动微分会自动求出我们在题(a)和(b)中推导的梯度。

在dev集合上运行代码来评估模型的表现，并且在测试集上预测（在最后的评估阶段关掉调试的设置）。注意测试集中只有假的类别数据。submit提交后会在真实数据上做评估（我们自学的不能提交:(!)

交付成果：

- **在q2_NER.py中实现NER窗口。**
- **简单阐述你的模型的优化参数：正则化，维度，学习率，SGD批处理大小等。提交模型在验证集上的表现。验证集上的损失应该小于0.2.**
- **在测试集上预测的类别列表，每行一条，存储在文件q2_test.predicted中。**

> - 调试模型设置max_epochs=1。并且将__init__方法中调用 load_data的参数debug设置为true。
> - 代码在GPU上运行时间应该在15分钟内，在CPU上运行一个小时内。

解：

答案见代码文件。

3. Recurrent Neural Networks：语言模型

本题，我们要实现第一个用来构建语言模型的循环神经网络。在NLP领域，语言模型是一个很重要的任务，可以用于语音识别、机器翻译和其他系统。给定词序列 $x_1,\ldots,x_t$，语言模型可以预测接下来的词$x_{(t+1)}$，语言模型的形式：

$$P(x_{t+1}=v_j|x_t,\ldots,x_1)$$

这里，v_j是词汇集中的某个词。

我们的任务是实现这样一个循环神经网络语言模型，它的隐藏层使用后向信息对历史 $x_t,x_{t-1},\ldots,x_1$ 建模。用公式表示这个模型，对$t=1,\ldots,n-1$：

$$e^{(t)}=x^{(t)}L$$

$$h^{(t)}=sigmoid(h^{(t-1)}H+e^{(t)}I+b_1)$$

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

$$\overline{P}(x_{(t+1)}=v_j|x_t,\ldots,x_1)=\hat{y}_j^{(t)}$$

这里 $h^{(0)}=h_0\in R^{D_h}$ 是隐藏层的一些初始化向量，$x^{(t)}L$是one-hot行向量$x^{(t)}$和L的点乘，表示当前词的索引。公式里的参数有：

$$L\in R^{|V|*d}\quad H\in R^{D_h*D_h}\ I\in R^{d*D_h}\quad b_1\in R^{D_h}\quad U\in R^{D_h*|V|}\ b_2\in R^{|V|}$$

这里的 L 是词嵌入矩阵， I 表示输入词的矩阵， H 是隐藏层转移矩阵，U 表示输出词的矩阵。b_1 和 b_2是偏置。d 是词向量的维度， $|V|$是词汇集的大小，D_h 是隐藏层的维度。

输出的向量$\hat{y}^{(t)}\in R^{(|V|)}$ 是在词汇集上的所有词的概率分布，不使用正则化优化交叉熵损失：

$$J^{(t)}(\theta)=CE(y^{(t)},\hat{y}^{(t)})=-\sum_{t=1}^{|V|}y_i^{(t)}log\hat{y}_i^{(t)}$$

这里$y^{(t)}$是目标词的one-hot向量（等价于$x_{(t+1)}$）。和第二题一样，这是一个点对点的损失，我们对序列里的所有示例的交叉熵损失求和或者平均来评估模型的表现。

(a). 我们一般用混淆度来评价语言模型的表现，它的公式为：

$$PP^{(t)}(y^{(t)}, \hat{y}^{(t)}) = \frac{1}{\overline{P}(x_{t+1}^{pred} = x_{t+1}|x_t, \dots, x_1)} = \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}}$$

可以理解为正确词的逆概率。阐述如何从交叉熵推导出混淆度（注意$y^{(t)}$是one-hot向量！），以及最小化交叉熵损失的算术均值也就是最小化训练集混淆度的几何均值。 这是个很简短的问题，不太复杂！

对于词汇集$|V|$ 个词，如果你的模型预测完全随机，混淆度是什么样的呢？分别计算 $|V| = 2000$ 和 $|V| = 10000$ 的交叉熵损失，将这些值作为baseline。

(b). 同第2题，计算模型中的参数在单个时间点 t 时的梯度：

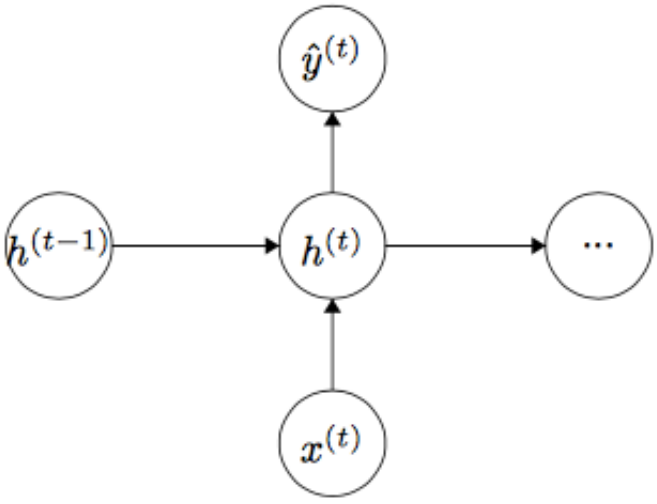
$$\frac{\partial J^{(t)}}{\partial U}, \frac{\partial J^{(t)}}{\partial b_2}, \frac{\partial J^{(t)}}{\partial L_x^{(t)}}, \frac{\partial J^{(t)}}{\partial I}|_{(t)}, \frac{\partial J^{(t)}}{\partial H}|_{(t)}, \frac{\partial J^{(t)}}{\partial b_1}|_{(t)}$$

这里的 $L_{x^{(t)}}$ 是当前词 $x^{(t)}$ 在 L 中的所在列，$|_{(t)}$ 表示参数在t时刻的梯度。（同样的，$h^{(t-1)}$是固定的，现在还不需要反向传播到前一时刻，在(c)小题中将会实现）。

此外，还要计算前一隐藏层的导数：

$$\frac{\partial J^{(t)}}{\partial h^{(t-1)}}$$

(c). 下图是网络在某一时刻的草图。



将这个网络展开3个时间单元，绘制草图。计算后向传播梯度：

$$\frac{\partial J^{(t)}}{\partial L_{x^{(t-1)}}}, \frac{\partial J^{(t)}}{\partial H}|_{(t-1)}, \frac{\partial J^{(t)}}{\partial I}|_{(t-1)}, \frac{\partial J^{(t)}}{\partial b_1}|_{(t-1)}$$

$|_{(t-1)}$表示参数在时刻(t-1)的梯度。因为在前向计算过程中多次使用这些参数，所以我们需要计算每个时刻的梯度。

我们可以使用在Lecture 5中学到的后向传播规则，使用误差项来表示这些导数：

$$\delta^{(t-1)} = \frac{\partial J^{(t)}}{\partial h^{(t-1)}}$$

（在 t-2，t-3时刻会重用这些表达式）。

> 真实的对训练样本计算题都，我们需要将误差向后传递到 t=0 时刻，而在练习中，我们只计算到 $\tau \approx 3-5$ 时刻。