Lecture 4. Word Window Classification

蜡笔大龙猫@2017.03.06

邮箱：houlisha1987@126.com

我们在上节课快结束的时候提到了窗口分类，Lecture 3这节课更详细的介绍了常用分类的背景、窗口分类、更新词向量以实现分类，交叉熵推导经验等，课程的最后5分钟教授简单的介绍了单层神经网络，这部分笔记放在第五节课中。

分类背景知识

分类器我们都比较了解，通常给定一个训数据集，其中包含这样的样本：

$$\{x_i, y_i\}_{i=1}^N$$

- x_i 是输入，例如单词（索引或者向量），上下文窗口，句子，文章等；
- y_i是我们想要预测的类别，例如情感，其他词，命名实体（地名、机构名）， 买卖决策，以及多词序列；

假设我们以简单的直观的 $2d$ 词向量分类为例，使用逻辑回归方法（线性决策边界），通常机器学习分类方法是这样做的：输入 x 是固定的，只需要训练逻辑回归的权重 W 和 仅仅修改决策边界：

$$p(y|x) = \frac{exp(W_{y\cdot}x)}{\sum_{c=1}^{C} exp(W_{c\cdot}x)}$$

训练集 $\{x_i, y_i\}_{i=1}^N$ 的损失函数为：

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -log(\frac{e^{fy_i}}{\sum_{c=1}^{C} e^{fc}})$$

对每个数据对 (x_i, y_i)：$fy = f_y(x) = W_{y\cdot}x = \sum_{j=1}^{d} W_{yj}x_j$

我们将 f 写成矩阵的形式： $f = Wx$ 。

实际使用时，训练数据集上损失函数包括所有参数 θ 的 **正则化Regularization**：

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -log(\frac{e^{fy_i}}{\sum_{c=1}^{C} e^{fc}}) + \lambda \sum_{k} \theta_k^2$$

当我们有很多特征或者如后面所见我们有很深的深度模型的时候，正则化能够避免过拟合。

上面讲述了通常所用的分类方法，接下来我们看看使用词向量来分类有什么不同?

词向量用于窗口分类

上述通用的机器学习方法中参数 θ 通常只包含 W 的列：

$$\theta = \begin{bmatrix} W_{\cdot 1} \\ \vdots \\ W_{\cdot d} \end{bmatrix} = W(:) \in R^{Cd}$$

所以，我们只需要更新决策边界：

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla W_{\cdot 1} \\ \vdots \\ \nabla W_{\cdot d} \end{bmatrix} \in R^{Cd}$$

然而，在深度学习方法中，我们需要同时学习权重 W 和 词向量 x：

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla W_{\cdot 1} \\ \vdots \\ \nabla W_{\cdot d} \\ \nabla x_{aardvark} \\ \vdots \\ \nabla x_{zebra} \end{bmatrix} \in R^{Cd+Vd}$$

从上式可以看到，需要求导的未知参数的范围$Cd + Vd$， V 表示整个词汇集大小，通常是百万级别，如此大的未知参数会导致两个问题：过拟合和失去泛化能力。

上面的这种算法也就是之前课程我们提到过得根据实际任务对词向量的重训练方法，失去泛化能力是什么意思呢？我们还是用之前的同一个例子，假设我们要训练电影评论的情感分类问题，在训练集中有"TV"和“Telly”两个词表示“电视”，测试集中有“television”这个词，当我们重训练词向量的时候发生了什么呢？一方面训练集中的数据在向量空间发生了移动，而测试集中的"television"没有出现在训练集中，所以不会发生移动，这个时候，“TV”，“Telly”和“television”的相对位置就产生了变化，也就失去了他们之间的相似性。（slides上有个图片可以直接的解释这个问题，感兴趣的同学可以自己查看。）

这就给我们带来了启示：

- 如果训练数据集很小，不需要重训练词向量；
- 如果你有足够大的训练集，重训练效果可能会更好；

词向量的数学标记

在讲词向量用于分类之前，我们首先来做一些标记，方面后面公式的理解。

词向量矩阵 L 也叫做查找表，通常在word2vec或者GloVe中是一个 d 行 $|V|$ 列大小的矩阵，其中每一列表示对应索引的词的词向量，也就是后面我们所说的词的特征。

从概念上说，我们额可以通过乘以一个one-hot向量 e 来得到某个词的向量，即
$x = Le \in d * V \cdot V * 1$

窗口分类

我们很少对单个词进行分类，原因是词的多义性。例如，“to seed”的意思可能是播种，也可能是除种，“Hathaway”可能表示城市，也可能表示女明星，这些意思都取决于词的上下文。

所以，就有了这样一个想法：**针对单词结合它的上下文窗口来分类**。

举个例子，NER任务中我们将词分成4个类别：人名、地名、机构名、其他。很多根据窗口分类的方法是这样做的，例如，对窗口中的词的向量取平均值作为特征，很明显，这样就失去了位置信息。

通用的方法是怎样的呢？**通过中心词赋予一个类标，将它周围的所有词的向量首尾拼接，来训练分类器。**

例如：对句子“... museums in Paris are amazing ... .”这个句子对“Paris”以大小为2的窗口分类，那么就产生了一个大小为$5d$的列向量：

$$X_{window} = [x_{museums}\ x_{in}\ x_{Paris}\ x_{are}\ x_{amazing}] \in R^{5d}$$

我们来看一个最简单的窗口分类器：**softmax**

$$\hat{y} = p(y|x) = \frac{exp(W_{y\cdot}x)}{\sum_{c=1}^{C} exp(W_{c\cdot}x)}$$

以交叉熵为损失函数：

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -log(\frac{e^{fy_i}}{\sum_{c=1}^{C} e^{fc}})$$

问题来了，我们如何更新词向量呢?

简短的回答：和前面一样，对每个参数推导。具体的很长的过程我们在PSet #1中已经实现了细节。

连接词向量的更新

更新首尾连接的词向量，有以下技巧需要注意：

1. 定义变量要仔细，并且时刻注意它们的维度。例如：

   $$f = f(x) = Wx \in R^{C}$$

   $$\hat{y} \quad t \quad W \in R^{C*5d}$$

2. **链式法则**很重要！不要忘记哪些变量使用了其他变量。
3. 对softmax推导，首先计算$c = y$时fc的导数，然后计算$c \neq y$时fc的导数。

4. 当对f中的每个元素求导后，不要忘记查看，是否能够创造一个包含所有部分导数的梯度，例如：

$$\frac{\partial}{\partial f} - log\ softmax(fy) = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_y - 1 \\ \vdots \\ \hat{y}_C \end{bmatrix} = \hat{y} - t$$

5. 为了节省后续麻烦，推导结果尽量用向量操作，并且及时定义新的单索引的向量，例如：

$$\frac{\partial}{\partial f} - log\ softmax(fy) = [\hat{y} - t] = \delta$$

6. 当使用链式法则的时候，首先使用显示的求和来部分求导，例如对x_i 或者 W_{ij}求导：

$$\sum_{c=1}^{C} - \frac{\partial og\ softmax(fy(x))}{\partial fc} \frac{\partial fc(x)}{\partial x} = \sum_{c=1}^{C} \delta_c W_c$$

7. 复杂函数的简化，注意变量的维度，尽量用矩阵来简化，例如：

$$\frac{\partial}{\partial x} - logp(y|x) = \sum_{c=1}^{C} \delta_c W_{c\cdot} = W^T \delta$$

8. 如果不清楚就把求和展开写。

第7点中公式最终的维度是多少？回答：5d。x是5d大小，那么对x求导数也是5d大小。

一个重要的提示：

在实现softmax的时候有两个代价很高的操作，分别是矩阵相乘和指数计算。我们在代码实现过程中，应该尽量避免使用for循环计算矩阵的相乘和其他数学计算。你可以用以下代码计算一下for循环的代价：

```
from numpy import random
N = 500
d = 300
C = 5
W = rando.rand(C,d)
wordvectors_list = [random.rand(d, 1) for i in range(N)]
wordvectors_one_matrix = random.rand(d, N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

softmax(=逻辑回归)并不是足够强大的算法。softmax只是在原始空间内给定一个线性的决策边界，对于少量的数据集可能会得到很好的结果，但是对于大量数据，它的作用是有限的！

接下来我们会介绍神经网络，它会更复杂，但是也会产生非线性的决策边界。