Lecture 3. More Word Vectors

Lecture 3 这节课首先复习了上节课学到的word2vec模型，以及使用梯度下降和SGD优化参数，然后介绍了词向量的内部评测和外部评测，参数对于类比评测任务的影响，处理词义的模糊性和窗口分类等。

梯度的更新

回顾一下上节课提到的word2vec的损失函数：

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

概率P定义为：

$$p(w_O|w_I) = \frac{exp(u_{w_O}^T v_{w_I})}{\sum_{w=1}^{W} exp(u_w^T v_{w_I})}$$

其中u 和 v 表示词的输入和输出向量，我们在上节课也推导了v_{w_I} 的梯度，同样也需要对 u 推导。

通常来说，对于每个上下文窗口我们要计算所有用到的参数的更新，例如“I like learning”这句话，当窗口大小为1的时候，第一个窗口我们需要计算参数输入向量 v_{like} ，输出向量u_I 和 $u_{learning}$ 参数的梯度。

> 对于句首和句尾词的处理，通常在句子首尾加上<\s>字符串，例如"<\s> I like learning <\s>"。所以我们用google的word2vec工具训练出来的词向量，第一个总是<\s>。

我们通常用 θ 代表模型中的所有参数的集合，在维度为 d 的词向量中，词典为V ，那么 θ 应该为：

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ \vdots \\ u_{zebra} \end{bmatrix} \in R^{2dV}$$

在整个训练数据上计算化损失函数 $J(\theta)$ 的最小化，需要对所有的窗口计算以下梯度：

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\alpha}{\alpha \theta_j^{old}} J(\theta)$$

$$\theta_j^{new} = \theta_j^{old} - \alpha \nabla_\theta J(\theta)$$

实现的代码为：

```
while True:
    theta_grad = evaluate_gradient(J, corpus, theta)
    theta = theta - alpha * theta_grad
```

这就是**梯度下降**优化方法。然而通常训练集会很大，也许有4亿个单词，更新一次就需要耗费很长的时间训练，所有一般使用**Stochastic Gradient Descent（SGD）**随机梯度下降来计算参数，核心就是每个窗口 t 后更新参数，公式就变为：

$$\theta^{new} = \theta^{old} = \alpha \nabla_\theta J_t(\theta)$$

代码更改为：

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J, window, theta)
    theta = theta - alpha * theta_grad
```

在计算每个窗口时，最多有 $2c-1$ 个词，所以梯度矩阵是很稀疏的，我们只需要更新出现的词的向量。有两种方法：

1. 对每个词向量做hash
2. 只更新输入词向量矩阵和输出词向量矩阵的指定的列（即词在的那列）

重要的一点，如果我们有百万级别的词向量要计算，最好使用分布式方式。

词向量的评测

目前为止，我们已经讨论了使用word2vec、GloVe方法来训练得到词在语义空间的潜在语义向量表达，下面我们就来讨论下如何评价词向量的质量。

评价词向量的方法一般分为两种，内部任务评测和外部任务评测。

内部评测 vs 外部评测

Intrinsic Evaluation内部任务评测通常是指定内部子任务，例如词向量的类比任务。

内部评测的特点有：

- 在指定的内部的任务上做评测
- 评测计算速度快
- 能够辅助我们了解子系统（辅助我们了解word2vec的原理）
- 在评测效果上需要与现实系统正相关（即如果内部任务评测效果好，应该推进现实系统的表现）

我们以要构建问答系统为例，训练该系统有这么几个步骤：

1. 输入表达问题的词。
2. 将它们转化为向量。
3. 将词向量作为输入到复杂的问答系统中。
4. 系统的输出词向量映射到真实的词。

5. 结果组合为答案。

在构建这样一个完美的问答系统过程中，我们首先需要最优的词向量表达，然后将它们应用在下游系统中。在实践过程中，需要调非常多的参数，理想的做法是每次调整word2vec的参数后都要重新训练下游问答系统来查看调参的结果，但是复杂的下游系统可能有很多层和百万级别的参数，这就导致这种理想的做法不现实。这种情况下，我们就需要有一个内部任务来代替。

外部评测就是我们上面说的用下游子系统来评测，特点有：

- 在真实任务上做评测
- 评测计算速度慢
- 不清楚问题在于哪个子系统，或者子系统之间的交互上
- 如果替换子系统能够提高效果，那么这种替换通常是好的

内部评测示例：词向量的类比

> 这小节有很多示例图片，我就不截图了，可以打开Lecture Notes 2.pdf 查看

词向量的类比形式就是 a : b : : c : ?

内部任务评测时，就是找到最大化余弦距离的词，数学公式表达为：

$$d = argmax_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

使用类比评测时，还需要注意训练集的不同方面。例如，在计算美国各州包含的城市时，不同城市具有相同的名字。在city-captial类比中，同一个国家在不同时期首都可能会变化。

上述两个例子都是语义方面的类比。在语法的类比上，可以有“形容词-最高级”、“动词-动词过去式”等形式。

内部评测调整示例

> 同样有很多评测结果的图片，不截图

词向量类比评测任务中，可以调节的参数：

- 词向量维度
- 训练集大小
- 训练集数据源/类型
- 上下文窗口大小
- 上下文窗口对称性

  还能想到其他参数吗?

几个观察的结论：

- 评测效果依赖选择训练词向量的模型/算法
- 训练集越大，效果越好
- 词向量维度特别高或者特别低的极端情况下，效果都不好。一般控制在300以内。

  维度过低，导致高偏差；维度过高，导致高方差。

- GloVe训练时，窗口大小为4时效果较好
- 对称窗口比非对称窗口效果好

内部评测示例：相关性评测

相关性评测也是评价词向量质量的一个简单方法，通常是人为的对两个词的相似度打分（0-10），然后与对应的词向量的余弦相似性比较。

扩展阅读：处理词的模糊性

也许有人疑惑，我们对于词的多义性怎么处理，例如，“run”即是动词，也是名词，根据不同的上下文表现出的词性不同。有一篇论文描述了解决这种情况的一个解决方法：

论文：Improving Word Representation Via Global Context And Multiple Word Prototypes (Huang et al. 2012)

1. 对所有词固定上下文（例如前5后5）
2. 词向量加权平均作为窗口的向量（例如以idf为权重）
3. 对窗口用kmeans聚类
4. 最后，出现在不同类簇的词用下标区别开，例如“run1”“run2”

   LDA可以解决词的多义性，也许有方法将两者结合起来用

外部任务的训练

到目前，我们讨论了几种内部评测任务，现实中，大部分场景是用词向量来完成其他外部任务，接下来我们讨论下几种外部任务：

用公式表示问题

大多数的任务都可以归结为分类问题。例如，给定一个句子，可以将其分类为正、负、中间情绪；NER任务中，给定上下文和中心词，将中心词分类为不同的类别。针对这类问题，我们的输入都是这样的形式：

$$\{x^{(i)}, y^{(i)}\}_1^N$$

其中，$x^{(i)}$表示d维的词向量，$y^{(i)}$是C维的ohe-hot向量表示类别。

在典型的机器学习任务中，我们通常固定输入数据和目标类别，使用优化方法（梯度下降、L-BFGS、牛顿法）训练权重参数。然而，在NLP任务中，我们介绍一种思想，当我们训练外部任务的时候同样对词向量也重新训练。

下面讨论什么时候和为什么要重训练词向量。

词向量的重训练

我们已经知道，外部任务输入的词向量都是经过内部任务评测后效果较好的词向量，但是在子系统任务中，仍然可以对词向量进行重训练来达到更好的效果。然而，重训练也是有风险的。

Tips：词向量重训练时，需要考虑用较大的数据集，最好可以覆盖现有的词向量。否则，效果可能变差。

例如，初始训练“Telly”“TV”“Television”可能在相似的位置。重训练后，新的训练集可能不包含“Television”，而“Telly”“TV”训练到了新的位置，在计算相似性时，就会出现偏差。

Softmax分类和正则化

我们来讨论下softmax分类函数的形式：

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

这里，$W \in R^{C*d}$ ，公式中W的下标$W_{j\cdot}$表示取第j行， $W_{\cdot i}$ 表示取i列。我们需要计算的是词向量 x 属于类别 j 的概率，使用交叉熵作为你损失函数，则损失函数公式为：

$$-\sum_{j=1}^{C} y_i log(p(y_j = 1|x)) = -\sum_{j=1}^{C} y_i log(\frac{exp(W_{j\cdot}x)}{\sum_{c=1}^{C} exp(W_{c\cdot}x)})$$

因为只有一个 y_j 为1，我们假设第k位为1，这个损失函数公式可以简化为：

$$-log(\frac{exp(W_{k\cdot}x)}{\sum_{c=1}^{C} exp(W_{c\cdot}x)})$$

然后，我们就可以将上述公式扩展到整个训练集上：

$$-\sum_{i=1}^{N} log(\frac{exp(W_{k(i)\cdot}x^{(i)})}{\sum_{c=1}^{C} exp(W_{c\cdot}x^{(i)})})$$

这里，$k(i)$ 是一个函数，返回样本 $x^{(i)}$ 对应的正确的类别的下标。

我们来估计下训练模型权重和词向量会有多少个参数需要更新。我们知道，一个简单的线性决策边界需要一个输入 d 维向量和输出 C 中类别的分布的模型，那么，我们就需要更新 $C*d$ 个参数。如果我们对词汇集 V 中每个词向量都更新，那么就需要更新 $|V|$ 个词向量，每个 d 维。所以，一个简单的线性分类器总共需要更新的参数个数为 $C*d+|V|*d$：

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix}$$

这是一个非常巨大的参数，很容易造成过拟合。为了降低过拟合的风险，提出了增加正则项的方法：

$$-\sum_{i=1}^{N} log(\frac{exp(W_{k(i)\cdot}x^{(i)})}{\sum_{c=1}^{C} exp(W_{c\cdot}x^{(i)})}) + \lambda \sum_{k=1}^{C\cdot d+|V|\cdot d} \theta_k^2$$

如果相对权重 λ 调参到合适的值，最小化上述的损失函数，就会避免出现权重特别大的情况，同事模型的泛化能力也表现不错。需要注意的是，正则化对于拥有很多参数的神经网络这种复杂的模型很重要。

窗口分类

目前为止，我们讨论都是基于一个单词的词向量作为输入的外部任务。现实中，自然语言在不同上下文往往有不同的含义，所以我们需要消歧。大多数情况下，我们需要一系列的词作为模型的输入，也就是一个上下文窗口。

通常，小窗口适于解决语法任务，大窗口适于解决语义任务。

在之前定义的softmax模型中，如果使用窗口代替单词，则输入 $x^{(i)}$ 要变为 $x_{window}^{(i)}$

$$x_{window}^{(i)} = \begin{bmatrix} x^{(i-2)} \\ x^{(i-1)} \\ x^{(i)} \\ x^{(i+1)} \\ x^{(i+2)} \end{bmatrix}$$

相应的，在计算损失函数的梯度的时候，我们会得到 ∇ 形式的向量。

实际上，这一步是可以分布式计算的。

非线性分类

现在，需要介绍一下非线性分类器，例如神经网络。在实际应用中经常会需要非线性分类器。我们将在下节课开始介绍在深度学习中表现出色的非线性决策边界—**神经网络**。