Stanfor NLP Lecture(CS224)
Notes (1)

易凯

2017 年 1 月 17 日

授课人 Rechard Sochoer
授课单位 Stanford University
录制时间 Spring 2015
学习起始结点 01/11/2017
阶段完成结点 01/17/2017
个人网站 https://williamyi96.github.io

目录

1 课程基本信息

1.1 写在前面

此为斯坦福大学《Deep Learning with NLP》的课程总结归纳，目前此章节内容的中文翻译可以在 http://blog.csdn.net/han-xiaoyang/article /details/51567822(注意由于版权问题查找时改为下划线) 找到，但是个人认为其翻译质量不高，但是可以作为课程理解的补充。

如果有任何疑问欢迎 GitHub 或者邮箱 (williamyi96@gmail.com) 进行探讨学习。

1.2 基本内容

Keyphrases ：MLP, Word Vectors, Singular Value Decomposition, Skip-gram, Continuous Bag of Words(CBOW), Negative Sampling

此笔记从讲解 nlp 相关概念以及当前 nlp 遇到的问题开始，然后讨论以数字向量为基准讲解词表示的基本概念，最后，讨论了设计词向量的流行方法。

2 自然语言处理简介

2.1 NLP 的目标

为解决某些任务而设计相应的算法去理解自然语言

2.2 NLP 任务难度分层

简单级： 拼写检查，关键词查找，同义词查找

一般级： 从网页、文本中解析信息

困难级： 机器翻译，语义分析，推断 (he 表示的是什么)，问答系统 (QA)

3 词向量

在自然语言处理中，我们将任何一个词都转化为一个向量，这个向量就被称之为词向量 (Word Vector)

3.1 One-hot 向量

概念： 一个除了某一行是 1 其他行都是 0 的列向量。

$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{a} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

特点： 简便直观。而由于任何两个 one-hot 向量之间都是线性无关的，实际上的词汇之间或多或少都存在联系，同时导致其规模较大，因此其在减小规模，减小计算量上存在障碍。

解决的方式最为简单有效的可以使用 SVD(奇异值分解)

4 SVD 基本方法

4.1 奇异值分解

理论描述：

Suppose $\mathbf{M}$ is a $m \times n$ matrix whose entries come from the field K, which is either the field of real numbers or the field of complex numbers. Then there exists a factorization, called a singular value decomposition of $\mathbf{M}$, of the form

$$\mathbf{M} = \mathbf{U\Sigma V}^*$$

where

- $\mathbf{U}$ is a $m \times m$, unitary matrix,
- $\mathbf{\Sigma}$ is a diagonal $m \times n$ matrix with non-negative real numbers on the diagonal, and
- $\mathbf{V}^*$ is a $n \times n$, unitary matrix over K. (If $K = \mathbf{R}$, unitary matrices are orthogonal matrices.) $\mathbf{V}^*$ is the conjugate transpose of the $n \times n$ unitary matrix, $\mathbf{V}$.

The diagonal entries σ_i of $\mathbf{\Sigma}$ are known as the **singular values** of $\mathbf{M}$. A common convention is to list the singular values in descending order. In this case, the diagonal matrix, $\mathbf{\Sigma}$, is uniquely determined by $\mathbf{M}$ (though not the matrices $\mathbf{U}$ and $\mathbf{V}$, see below).

4.2 词-文本矩阵

猜想： 相关的词通常在同一个文本中出现

实践： 遍历上亿的文本，当词汇 i 出现在文本 j 中时，我们在 x_{ij} 位置上 +1

缺陷： 一方面猜想存在不准确性，另一方面生成的矩阵 $V \times M$ 规模太大

4.3 词词共现矩阵

算法步骤： 1. 生成 |V| × |V| 共现矩阵 X
2. 在 X 上使用 SVD 分解，得到 $X = USV^T$
3. 选择 U 的第 K 列得到一个 k 维词向量
4. $\frac{\sum_{j=1}^{k} \delta_j}{\sum_{i=1}^{|V|} \delta_i}$ 表示 k 维向量的数量方差

实例： 如果语料库中有三句话：
1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

那么我们可以列出词词共现矩阵为：

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

通过奇异值分解，我们可以显著地降低词的维度。

缺陷 (问题) 1. 矩阵的维数经常改变 (新词的添加和语料库大小的变化)；
2. 得出的矩阵很稀疏；
3. 矩阵规模仍然很大 (10 的 6 次方)；
4. SVD 的训练算法复杂度为平方级别；
5. 可能存在词频率的剧烈失衡

部分解决方法 1. 忽略诸如“the”，“he”，“has”等的功能词 (function words)
2. 应用 ramp window，使用距离来权衡词词之间的相关性；
3. 使用 Pearson 相关性并将负数设置为 0，而不是仅使用原始计数

5 基于迭代的方法

5.1 基本思想

通过迭代学习参数，来找出概率最大的句子组合，以此来匹配编码单词，从而降低计算复杂度。

5.2 基本语言模型

一元模型和二元模型。

一元模型 (Unigram Model)

$$P(w^{(1)}, w^{(2)}, \cdots, w^{(n)}) = \prod_{i=1}^{n} P(w^{(i)})$$

二元模型 (Bigram Model)

$$P(w^{(1)}, w^{(2)}, \cdots, w^{(n)}) = \prod_{i=2}^{n} P(w^{(i)}|w^{(i-1)})$$

5.3 词袋模型 (Bag of Word)

词袋模型概念： 不考虑句子中的语法、词间关系和顺序，将文本中出现的词使用 one-hot 向量表示的方法

具体实例： 如果两段文本，将两段文本中出现的所有词构成一个该维度的列向量，然后将每个文本中对应的词出现的次数写在相应的位置即可

5.4 连续词袋模型 (CBOW)

CBOW 概念： 从一个构造特定环境的文本中预测中间词语.(Predicting a center word from the surrounding context)

具体实例： 我喜欢 (吃) 妈妈削过皮的苹果。此处通过句意我们可以在该句中添加“吃”

算法步骤： **（具体实践相对而言比较复杂，暂时不对其进行深入的涉及)**

1. We generate our one hot word vectors $(x^{(i-C)}, \ldots, x^{(i-1)}, x^{(i+1)}, \ldots, x^{(i+C)})$ for the input context of size C.

2. We get our embedded word vectors for the context $(u^{(i-C)} = W^{(1)}x^{(i-C)}, u^{(i-C+1)} = W^{(1)}x^{(i-C+1)}, \ldots, u^{(i+C)} = W^{(1)}x^{(i+C)})$

3. Average these vectors to get $h = \frac{u^{(i-C)}+u^{(i-C+1)}+\ldots+u^{(i+C)}}{2C}$

4. Generate a score vector $z = W^{(2)}h$

5. Turn the scores into probabilities $\hat{y} = \text{softmax}(z)$

6. We desire our probabilities generated, $\hat{y}$, to match the true probabilities, y, which also happens to be the one hot vector of the actual word.

5.5 Skip-Gram 模型

模型概念： 通过一个给定的词来预测最可能的环境文本。(Predicting surrounding context words given a center word)

算法步骤：

1. We generate our one hot input vector x
2. We get our embedded word vectors for the context $u^{(i)} = W^{(1)}x$
3. Since there is no averaging, just set $h = u^{(i)}$?
4. Generate $2C$ score vectors, $v^{(i-C)}, \ldots, v^{(i-1)}, v^{(i+1)}, \ldots, v^{(i+C)}$ using $v = W^{(2)}h$
5. Turn each of the scores into probabilities, $y = \text{softmax}(v)$
6. We desire our probability vector generated to match the true probabilities which is $y^{(i-C)}, \ldots, y^{(i-1)}, y^{(i+1)}, \ldots, y^{(i+C)}$, the one hot vectors of the actual output.

5.6 负采样 (Negative Sampling)

负采样本质上是一种带权采样问题。
暂时关于其相关原理掌握得不是很深，可以参见
http://hacker.duanshishi.com/?p=1577
以及
http://blog.csdn.net/itplus/article/details/37998797