word2vec理论与实践

原创 bamtercelboo 深度学习自然语言处理 2018-03-26

收录于话题
#NLP论文解读

24个

> 阅读大概8分钟

导读

本文简单的介绍了Google 于 2013 年开源推出的一个用于获取 word vector 的工具包（word2vec），并且简单的介绍了其中的两个训练模型（Skip-gram，CBOW），以及两种加速的方法（Hierarchical Softmax，Negative Sampling）。

一、word2vec

word2vec最初是由Tomas Mikolov 2013年在ICLR发表的一篇文章 Efficient Estimation of Word Representations in Vector Space [https://arxiv.org/pdf/1301.3781.pdf]， 并且开源了代码，作用是将所有词语投影到K维的向量空间，每个词语都可以用一个K维向量表示。由于它简洁，高效的特点，引起了人们的广泛关注，并应用在很多NLP任务中，用于训练相应的词向量。

1、传统的词表示 — one-hot representation

- 这种方法把每个词表示为一个很长的向量。这个向量的维度是词表大小，其中绝大多数元素为 0，只有一个维度的值为 1，这个维度就代表了当前的词。
- 假如词表是：[气温、已经、开始、回升、了]，那么词的词向量分别可以是[1,0,0,0,0]，[0,1,0,0,0]，[0,0,1,0,0]，[0,0,0,1,0]，[0,0,0,0,1]。这样的表示方法简单容易理解，而且编程也很容易实现，只需要取对应的索引就能够完成，已经可以解决相当一部分NLP的问题，但是仍然存在不足，即词向量与词向量之间都是相互独立的；我们都知道，词与词之间是有一定的联系的，我们无法通过这种词向量得知两个词在语义上是否相似，并且如果词表非常大的情况下，每个词都是茫茫 0

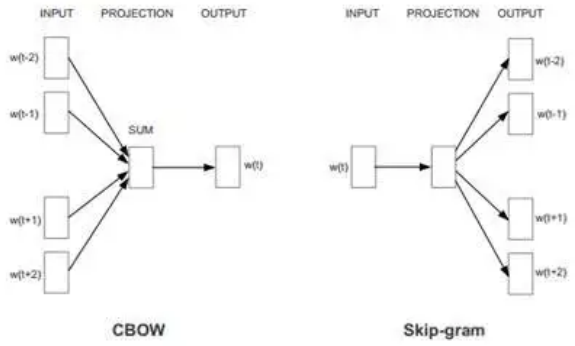
海中的一个 1，这种高维稀疏的表示也有可能引发维度灾难。为了解决上述问题，就有了词向量的第二种表示方法。

2、Distributed representation — word embedding

- word2vec就是通过这种方法将词表示为向量，即通过训练将词表示为限定维度K的实数向量，这种非稀疏表示的向量很容易求它们之间的距离(欧式、余弦等)，从而判断词与词语义上的相似性，也就解决了上述one-hot方法表示两个词之间的相互独立的问题。
- 不过Distributed representation并不是word2vec诞生才有的， Distributed representation 最早是 Hinton 在 1986 年的论文《Learning distributed representations of concepts》中提出的。虽然这篇文章没有说要将词做 Distributed representation，但至少这种先进的思想在那个时候就在人们的心中埋下了火种，到 2000 年之后开始逐渐被人重视。 word2vec之所以会产生这么大的影响，是因为它采用了简化的模型，使得训练速度大为提升，让word embedding这项技术(也就是词的distributed representation)变得实用，能够应用在很多的任务上。

二、Skip-Gram model and CBOW model

- 我们先首先来看一下两个model的结构图。



- 上图示CBOW和Skip-Gram的结构图，从图中能够看出，两个模型都包含三层结构，分别是输入层，投影层，输出层；CBOW模型是在已知当前词上下文context的前提下预测当前词w(t)，类似阅读理解中的完形填空；而Skip-Gram模型恰恰相反，是在已知当前词w(t)的前提下，预测上下文context。
- 对于CBOW和Skip-Gram两个模型，word2vec给出了两套框架，用于训练快而好的词向量，他们分别是Hierarchical Softmax 和 Negative Sampling，下文将介绍这两种加速方法。

三、Negative Sampling

- Negative Sampling（NEG） 是Tomas Mikolov在Distributed Representations of Words and Phrasesand their Compositionality中提出的，它是噪声对比损失函数NCE(Noise Contrastive Estimation)的简化版本，用于提高训练速度和提升词向量质量。

1、 Negative Sampling

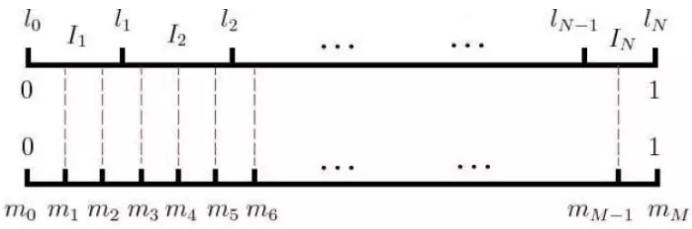
- 比如我们的训练样本，中心词是w，它周围上下文共有2c个词，记为context(w)。由于这个中心词w，的确和context(w)相关存在，因此它是一个真实的正例。通过Negative Sampling进行负采样，我们得到neg（负采样的个数）个和w不同的中心词wi，i=1,2,..neg，这样context(w)和wi就组成了neg个并不真实存在的负例。利用这一个正例和neg个负例，我们进行二元逻辑回归（可以理解成一个二分类问题），得到负采样对应每个词wi对应的模型参数以及每个词的词向量。

2、 How to do Negative Sampling?

- 我们来看一下如何进行负采样，得到neg个负例。word2vec采样的方法并不复杂，如果词汇表的大小为V，那么我们就将一段长度为1的线段分成V份，每份对应词汇表中的一个词。当然每个词对应的线段长度是不一样的，高频词对应的线段长，低频词对应的线段短（根据词频采样，出现的次数越多，负采样的概率越大）。每个词w的线段长度由下式决定：

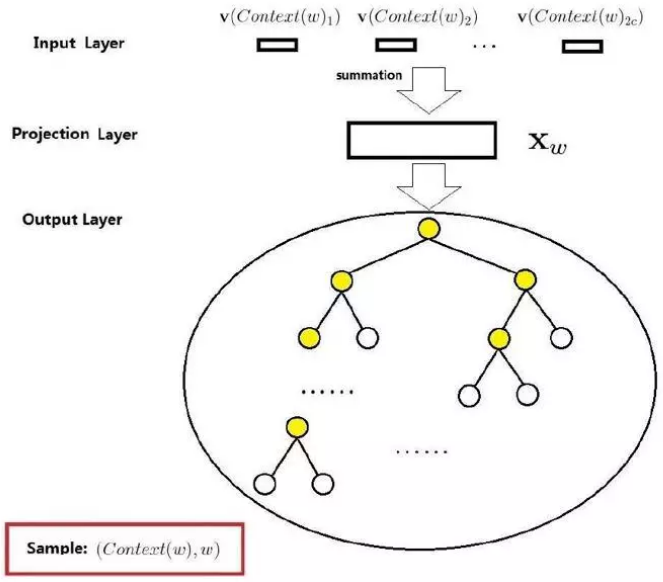
$$\text{len}(w) = \frac{count(w)}{\sum_{u \in vocab} count(u)}$$

- 在采样前，我们将这段长度为1的线段划分成M等份，这里M»V，这样能够保证每个词对应的线段都会划分成对应的小块，而M份中每一份都会落在某一个词对应的线段上(如下图)，采样的时候，我们只需要随机生成neg个数，对应的位置就是采样的负例词。



四、Hierarchical Softmax

- 如下图所示：网络结构很简单，仅仅包含三层网络结构，输入层，投影层，输出层。
- 输入层到投影层是把输入层的所有向量进行加和给投影层，比如，输入的是三个4维词向量：(1,2,3,4)，(9,6,11,8)，(5,10,7,12)，那么我们word2vec映射后的词向量就是(5,6,7,8)，对CBOW模型来说，就是把上下文词向量加和，然而，对于Skip-Gram模型来说就是简单的传值。
- 最后的输出是构建一颗哈夫曼树，如何去构造简单的哈夫曼树。在这里不在累述；在这里，哈夫曼树的所有叶子节点是词表中的所有词，权值是每个词在词表中出现的次数，也就是词频。



- 一般得到哈夫曼树后我们会对叶子节点进行哈夫曼编码，由于权重高的叶子节点越靠近根节点，而权重低的叶子节点会远离根节点，这样我们的高权重节点编码值较短，而低权重值编码值较长。这保证的树的带权路径最短，也符合我们的信息论，即我们希望越常用的词（词频越高的词）拥有更短的编码，一般的编码规则是左0右1，但是这都是人为规定的，word2vec中正好采用了相反的编码规则，同时约定左子树的权重不小于右子树的权重。
- 如何"沿着哈夫曼树一步步完成"呢?
  - 在word2vec中，采用了二元逻辑回归的方法，即规定沿着左子树走，那么就是负类(哈夫曼树编码1)，沿着右子树走，那么就是正类(哈夫曼树编码0)。
- 使用哈夫曼树有什么好处呢?
  - 首先，由于是二叉树，之前计算量为V，现在变成了log2V。
  - 其次，由于使用哈夫曼树是高频的词靠近树根，这样高频词需要更少的时间会被找到，这符合我们的贪心优化思想。

五、Demo

- 在这里提供了几份代码，包括我实现的c++，pytorch版本，以及word2vec源码版本及其源码注释版。
- pytorch-version: https://github.com/bamtercelboo/pytorch_word2vec
- cpp-version: https://github.com/bamtercelboo/word2vec/tree/master/word2vec
- word2vec-source-version：word2vec.googlecode.com/svn/trunk/
- word2vec-annotation-version: https://github.com/tankle/word2vec

六、Experiment Result

1、 Word-Similar Performance

- 我们在英文语料enwiki-20150112_text.txt（12G）上进行了测试，测试采用的是根据这篇论文 Community Evaluation and Exchange of Word Vectors at wordvectors.org 提供的方法与site（http://www.wordvectors.org/index.php），计算词之间的相似度。
- 结果如下图所示：由于 pytorch-version 训练速度慢并且demo还没有完善，所以仅在 Cpp-version 和 word2vec源码（C-version）进行了测试对比。以上对比试验均是在相同的参数设置下完成的。
- 参数设置
  - model: skip-gram, loss: Negative Sampling, neg: 10
  - dim: 100, lr: 0.025, windows size: 5, minCount: 10, iters: 5

No.	Task Name	Word pairs	Cpp-version		C-version	
			Pairs	Correlation	Pairs	Correlation
1	WS-353	353	353	**0.6969**	353	**0.6862**
2	WS-353-SIM	203	203	**0.7587**	203	**0.7508**
3	WS-353-REL	252	252	**0.6358**	252	**0.6315**
4	MC-30	30	30	**0.7996**	30	**0.7876**
5	RG-65	65	65	**0.7908**	65	**0.7837**
6	Rare-Word	2034	1938	**0.4180**	1953	**0.4235**
7	MEN	3000	3000	**0.7307**	3000	**0.7224**
8	MTurk-287	287	287	**0.6873**	287	**0.6843**
9	MTurk-771	771	771	**0.6369**	771	**0.6283**
10	YP-130	130	130	**0.4332**	130	**0.4078**
11	SimLex-999	999	999	**0.3187**	999	**0.3112**
12	Verb-143	143	144	**0.3609**	144	**0.3316**
13	SimVerb-3500	3500	3498	**0.1980**	3498	**0.1978**
14	Sum	—	—	**7.4655**	—	**7.3467**

- 上面结果表明，Cpp-version 和C-version 训练出来的词向量都能达到一样的性能，甚至还比C-version训练出来词向量稍高一点。

2、 Train Time Performance

- 由于上述实验是在不同服务器，不同线程数目的情况下进行训练，所以上述实验的训练时间不存在对比，为了测试方便快速，在12G英文语料 enwiki-20150112_text.txt 上取出大约1G的文件，进行重新训练两份词向量，看一下训练时间，下图是实验结果。

	Cpp-version	C-version
Cpu Info	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz	
Threads	12 threads	
Time	23(min)	24(min)

- 上述实验结果能够看出来：Cpp-version 和C-version 的训练时间相差不大。

References

[1] Efficient Estimation of Word Representations in Vector Space
[2] Learning distributed representations of concepts
[3] Distributed Representations of Words and Phrasesand their Compositionality
[4] Community Evaluation and Exchange of Word Vectors at wordvectors.org
[5] https://blog.csdn.net/itplus/article/details/37998797(word2vec 中的数学原理详解)
[6] http://www.cnblogs.com/pinard/p/7249903.html(word2vec 原理)

推荐阅读：
【干货】神经网络SRU