WORD2VEC TUTORIAL

1. 介绍

word2vec 是 Google 于2013年推出的一个词向量嵌入 (word embedding) 工具。所谓词向量，直观的理解就是，将一个词 (word) 映射为一个向量 (vector)。这类方法有很多，比如:

- one-hot。若有5个词 a, b, c, d, e，则 a 的词向量为 $[1, 0, 0, 0, 0]$。很明显这种词向量只跟字典有关，而忽略了上下文的关系。任意两个词的距离都是一样的，但是 small 和 little 的距离应该比 small 和 man, woman, building 这些名词的距离小。于是有了下面一种方法。
- co-occurrence matrix。需要用 svd 来降维，当 vocabulary 过大时就难以计算。

word2vec 计算出的词向量，具有丰富的 semantic 和 syntactic 信息，而且其计算对于超大量的文本也是可行的，空间复杂度更是只有 $O(|V|)$。

另外，Embedding 其实是从数学借用过来的一个概念，较正式的定义为：

> In mathematics, an embedding is one instance of some mathematical structure contained within another instance, such as a group that is a subgroup. *(From Wikipedia)*

2. 发展

word2vec 的模型可以追溯到[1]，该文章提出了一种基于神经网络的 Language Model (LM)。这里简要介绍一下 LM，它是一类模型的统称。LM 要解决的问题，是计算词序列的概率，即 $P(w_1, \dots, w_n)$。比如，"how are you" 和 "you how are" 是两个不同的词序列，很明显前者出现的概率要比后者大很多。

假设用 w_i^j 表示序列 $w_i, \dots, w_j$，则根据贝叶斯公式，有：

$$P(w_1^T) = \prod_{k=1}^{T} P(w_k | w_1^{k-1})$$

根据 Markov Assumption，词 w_k 的条件概率仅与它之前的若干个词有关，因此可以用 $P(w_k | w_{k-n+1}^{k-1})$ 来代替 $P(w_k | w_1^{k-1})$。于是序列的概率可以这样计算：

$$P(w_1^T) \approx \prod_{k=1}^{T} P(w_k | w_{k-n+1}^{k-1})$$

这样问题就转化为，求某个词在给定它之前的 $n-1$ 个词下的条件概率。在[1]之前，n-gram 是一个很流行的方法。而这篇论文中，作者提出使用神经网络的方法，简称 NNLM。

在后来实际的应用中，NNLM 的效果虽然优于 n-gram，但由于网络最后一层的 softmax 的计算是 $O(|V|)$ 的，所以训练速度过于缓慢。于是，在[2]中，作者提出了一种加速的方法，从而极大地简化了计算。相关的论文再推荐一篇[3]。

事实上，word2vec 正是基于这些研究，同时增加了一些 Google 自己的黑科技，才脱胎而生。

3. 模型

本质上来讲，word2vec 是一个三层的前馈神经网络 (Feedforward Neural Network)。假设字典大小为 $|V|$，则第一层是一个维度为 $|V|$ 的输入层；第二层是一个维度为 D $(D \ll |V|)$ 的隐藏层，且激活函数为 $h(x) = x$；第三层是维度为 $|V|$ 的输出层，激活函数为 softmax。采用极大似然作为目标函数。更具体的细节可以看论文和源代码。

word2vec 的论文发了两篇，分别是[4]和[5]，但读者如果没有知识储备，很难从这两篇论文里读懂些什么。因此，推荐先看完上述的一些文章，了解了 background 再看这两篇。总的来说，这两篇论文里叙述了这样一些改进：

- 使用某个词周围的词来预测中间这个词（或相反），而不是之前所说的用前 $n-1$ 个词预测下一个词。
- 使用 hierarchical softmax (hs) 简化计算。
- 使用 negative sampling (ns) 简化计算。

- 使用 subsampling 限制高频词。
- 定义了两种模型 cbow 和 skip-gram，前者用某词的上下文来预测该词，即求 $P(w_i|\text{context})$，后者正好相反。

模型本身很简单，难点都在于上述的一些改进方法上，尤其是 hs 和 ns。

以 cbow 模型举例，假设 context 为 machine, learning, me, happy，中间的词为 makes。则输入为一个 one-hot 矩阵，context 中四个词所对应的位置上为1，字典中其余词为0。

假设输入层和隐藏层之间的权重矩阵为 $W \in \mathbb{R}^{|V|\times D}$，输入层向量用 $x \in \mathbb{R}^{|V|}$ 表示，隐藏层向量用 $h \in \mathbb{R}^{D}$ 表示。则

$$h = W^T x$$

很明显，h 就是将 W 的若干行相加所得到的向量，而 x 中的 1 则规定了是哪些行相加。因此，你可以将 W 的每一行看作是一个词向量，而隐藏层向量就是 context 对应词向量的和（实际上要取平均）。

通过 hs 和 ns 的方法，可以快速计算出梯度，然后更新 W 矩阵。

时间限制，无法展开更多，具体的内容请查看原论文及代码。

Reference

1. Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin. A Neural Probabilistic Language Model. Journal of Machine Learning Research, 3:1137–1155, 2003.
2. F. Morin, Y. Bengio. Hierarchical Probabilistic Neural Network Language Model. AISTATS, 2005.
3. A. Mnih, G. Hinton. A Scalable Hierarchical Distributed Language Model. NIPS, 2008.
4. T. Mikolov, K. Chen, G. Corrado, J. Dean. Efficient Estimation of Word Representations in Vector Space. ICLR, 2013.
5. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean. Distributed Representations of Words and Phrases and their Compositionality. NIPS, 2013.