

【论文笔记】词向量的训练和语言模型

词向量的训练

要介绍词向量是怎么训练得到的，就不得不提到语言模型。到目前为止我了解到的所有训练方法都是在训练语言模型的同时，顺便得到词向量的。

这也比较容易理解，要从一段无标注的自然文本中学习出一些东西，无非就是统计出词频、词的共现、词的搭配之类的信息。而要从自然文本中统计并建立一个语言模型，无疑是要求最为精确的一个任务（也不排除以后有人创造出更好更有用的方法）。既然构建语言模型这一任务要求这么高，其中必然也需要对语言进行更精细的统计和分析，同时也会需要更好的模型，更大的数据来支撑。目前最好的词向量都来自于此，也就不难理解了。

这里介绍的工作均为从大量未标注的普通文本数据中无监督地学习出词向量（语言模型本来就是基于这个想法而来的），可以猜测，如果用上了有标注的语料，训练词向量的方法肯定会更多。不过视目前的语料规模，还是使用未标注语料的方法靠谱一些。

语言模型简介

语言模型其实就是看一句话是不是正常人说出来的。这玩意很有用，比如机器翻译、语音识别得到若干候选之后，可以利用语言模型挑一个尽量靠谱的结果。在 NLP 的其它任务里也都能用到。

语言模型形式化的描述就是给定一个字符串，看它是自然语言的概率

$P(w_1, w_2, \dots, w_t)$ 。 w_1 到 w_t 依次表示这句话中的各个词。有个很简单的推论是：

$$P(w_1, w_2, \dots, w_t) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2) \times \dots \times P(w_t|w_1, w_2, \dots, w_{t-1})$$

常用的语言模型都是在近似地求 $P(w_t|w_1, w_2, \dots, w_{t-1})$ 。比如 n-gram 模型就是用 $P(w_t|w_{t-n+1}, \dots, w_{t-1})$ 近似表示前者。

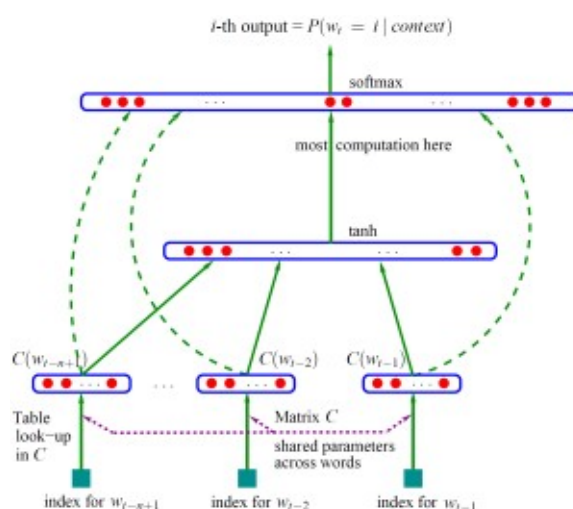
Bengio 的经典之作

词向量的训练最经典的有 3 个工作，C&W 2008、M&H 2008、Mikolov 2010。当然在说这些工作之前，不得不介绍一下这一系列中 Bengio 的经典之作。

用神经网络训练语言模型的思想最早由百度 IDL 的徐伟于 2000 提出。其论文《Can Artificial Neural Networks Learn Language Models?》提出一种用神经网络构建二元语言模型（即 $P(w_t|w_{t-1})$ ）的方法。文中的基本思路与后续的语言模型的差别已经不大。

训练语言模型的最经典之作，要数 Bengio 等人在 2001 年发表在 NIPS 上的文章《A Neural Probabilistic Language Model》。

Bengio 用了一个三层的神经网络来构建语言模型，同样也是 n-gram 模型。



图中最下方的 $w_{t-n+1}, \dots, w_{t-2}, w_{t-1}$ 就是前 $n-1$ 个词。现在需要根据这已知的 $n-1$ 个词预测下一个词 w_t 。 $C(w)$ 表示词 w 所对应的词向量，整个模型中使用的是一套唯一的词向量，存在矩阵 C （一个 $|V| \times m$ 的矩阵）中。其中 $|V|$ 表示词表的大小（语料中的总词数）， m 表示词向量的维度。 w 到 $C(w)$ 的转化就是从矩阵中取出一行。

网络的第一层（输入层）是将 $C(w_{t-n+1}), \dots, C(w_{t-2}), C(w_{t-1})$ 这 $n-1$ 个向量首尾相接拼起来，形成一个 $(n-1)m$ 维的向量，下面记为 x 。

网络的第二层（隐藏层）就如同普通的神经网络，直接使用 $d + Hx$ 计算得到。 d 是一个偏置项。在此之后，使用 \tanh 作为激活函数。

网络的第三层（输出层）一共有 $|V|$ 个节点，每个节点 y_i 表示下一个词为 i 的未归一化 log 概率。最后使用 softmax 激活函数将输出值 y 归一化成概率。最终， y 的计算公式为：

$$y = b + Wx + U \tanh(d + Hx)$$

式子中的 U （一个 $|V| \times h$ 的矩阵）是隐藏层到输出层的参数，整个模型的多数计算集中在 U 和隐藏层的矩阵乘法中。后文的提到的 3 个工作，都有对这一环节的简

化，提升计算的速度。

式子中还有一个矩阵 W ($|V| \times (n-1)m$)，这个矩阵包含了从输入层到输出层的直连边。直连边就是从输入层直接到输出层的一个线性变换，好像也是神经网络中的一种常用技巧（没有仔细考察过）。如果不需要直连边的话，将 W 置为 0 就可以了。在最后的实验中，Bengio 发现直连边虽然不能提升模型效果，但是可以少一半的迭代次数。同时他也猜想如果没有直连边，可能可以生成更好的词向量。

现在万事俱备，用随机梯度下降法把这个模型优化出来就可以了。需要注意的是，一般神经网络的输入层只是一个输入值，而在这里，输入层 x 也是参数（存在 C 中），也是需要优化的。优化结束之后，词向量有了，语言模型也有了。

这样得到的语言模型自带平滑，无需传统 n-gram 模型中那些复杂的平滑算法。Bengio 在 APNews 数据集上做的对比实验也表明他的模型效果比精心设计平滑算法的普通 n-gram 算法要好 10% 到 20%。

C&W 的 SENNA

Ronan Collobert 和 Jason Weston 在 2008 年的 ICML 上发表的《A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning》里面首次介绍了他们提出的词向量的计算方法。和上一篇牛文类似，如果现在要看的话，应该去看他们在 2011 年投到 JMLR 上的论文《Natural Language Processing (Almost) from Scratch》。文中总结了他们的多项工作，非常有系统性。

实际上 C&W 这篇论文主要目的并不是在于生成一份好的词向量，甚至不想训练语言模型，而是要用这份词向量去完成 NLP 里面的各种任务，比如词性标注、命名实体识别、短语识别、语义角色标注等等。

由于目的的不同，C&W 的词向量训练方法在我看来也是最特别的。他们没有去近似地求 $P(w_t | w_1, w_2, \dots, w_{t-1})$ ，而是直接去尝试近似 $P(w_1, w_2, \dots, w_t)$ 。在实际操作中，他们并没有去求一个字符串的概率，而是求窗口连续 n 个词的打分 $f(w_{t-n+1}, \dots, w_{t-1}, w_t)$ 。打分 f 越高的说明这句话越是正常的话；打分低的说明这句话不太合理；如果是随机把几个词堆积在一起，那肯定是负分（差评）。打分只有相对高低之分，并没有概率的特性。

有了这个对 f 的假设，C&W 就直接使用 pair-wise 的方法训练词向量。具体的来说，就是最小化下面的目标函数。

$$\sum_{x \in X} \sum_{w \in D} \max \{0, 1 - f(x) + f(x^{(w)})\}$$

X 为训练集中的所有连续的 n 元短语， D 是整个字典。第一个求和枚举了训练语料中的所有 n 元短语，作为正样本。第二个对字典的枚举是构建负样本。 $x^{(w)}$ 是将短语 x 的最中间的那个词，替换成 w 。在大多数情况下，在一个正常短语的基础上随便找个词替换掉中间的词，最后得到的短语肯定不是正确的短语，所以这样构造的负样本是非常可用的（多数情况下确实是负样本，极少数情况下把正常短语当作负样本也不影响大局）。同时，由于负样本仅仅是修改了正样本中的一个词，也不会让分类面距离负样本太远而影响分类效果。再回顾这个式子， x 是正样本， $x^{(w)}$ 是负样本， $f(x)$ 是对正样本的打分， $f(x^{(w)})$ 是对负样本的打分。最后希望正样本的打分要比负样本的打分至少高 1 分。

f 函数的结构和 Bengio 2003 中提到的网络结构基本一致。同样是把窗口中的 n 个词对应的词向量串成一个长的向量，同样是经过一层网络（乘一个矩阵）得到隐藏层。不同之处在于 C&W 的输出层只有一个节点，表示得分，而不像 Bengio 那样的有 $|V|$ 个节点。这么做可以大大降低计算复杂度，当然有这种简化还是因为 C&W 并不想做一个真正的语言模型，只是借用语言模型的思想辅助他完成 NLP 的其它任务。（其实 C&W 的方法与 Bengio 的方法还有一个区别，他们为了程序的效率用 HardTanh 代替 tanh 激活函数。）

他们在实验中取窗口大小 $n = 11$ ，字典大小 $|V| = 130000$ ，在维基百科英文语料和路透社语料中一共训练了 7 周，终于得到了这份伟大的词向量。

如前面所说 C&W 训练词向量的动机与其他人不同，因此他公布的词向量与其它词向量相比主要有两个区别：

1. 他的词表中只有小写单词。也就是说他把大写开头的单词和小写单词当作同一个词处理。其它的词向量都是把他们当作不同的词处理的。

2. 他公布的词向量并不直接是上述公式的优化结果，而是在此基础上进一步跑了词性标注、命名实体识别等一系列任务的 Multi-Task Learning 之后，二次优化得到的。也可以理解为是半监督学习得到的，而非其他方法中纯无监督学习得到的。

不过好在 Turian 在 2010 年对 C&W 和 M&H 向量做对比时，重新训练了一份词向量放到了网上，那份就没上面的两个“问题”（确切的说应该是差别），也可以用的更放心。后面会详细介绍 Turian 的工作。

M&H 的 HLBL

Andriy Mnih 和 Geoffrey Hinton 在 2007 年和 2008 年各发表了一篇关于训练语言模型和词向量的文章。2007 年发表在 ICML 上的《Three new graphical models for statistical language modelling》表明了 Hinton 将 Deep Learning 战场扩展到 NLP 领域的决心。2008 年发表在 NIPS 上的《A scalable hierarchical distributed language model》则提出了一种层级的思想替换了 Bengio 2003 方法中最后隐藏层到输出层最花

时间的矩阵乘法，在保证效果的基础上，同时也提升了速度。下面简单介绍一下这两篇文章。

Hinton 在 2006 年提出 Deep Learning 的概念之后，很快就来 NLP 最基础的任务上试了一把。果然，有效。M&H 在 ICML 2007 上发表的这篇文章提出了“Log-Bilinear”语言模型。文章标题中可以看出他们其实一共提了 3 个模型。从最基本的 RBM 出发，一点点修改能量函数，最后得到了“Log-Bilinear”模型。

模型如果用神经网络的形式写出来，是这个样子：

$$h = \sum_{i=1}^{t-1} H_i C(w_i)$$
$$y_j = C(w_j)^T h$$

这里的两个式子可以合写成一个 $y_j = \sum_{i=1}^{t-1} C(w_j)^T H_i C(w_i)$ 。 $C(w)$ 是词 w 对应的词向量，形如 $x^T M y$ 的模型叫做 Bilinear 模型，也就是 M&H 方法名字的来历了。

为了更好地理解模型的含义，还是来看这两个拆解的式子。 h 在这里表示隐藏层，这里的隐藏层比前面的所有模型都更厉害，直接有语义信息。首先从第二个式子中隐藏层能和词向量直接做内积可以看出，隐藏层的维度和词向量的维度是一致的（都是 m 维）。 H_i 就是一个 $m \times m$ 的矩阵，该矩阵可以理解为第 i 个词经过 H_i 这种变换之后，对第 t 个词产生的贡献。因此这里的隐藏层是对前 $t-1$ 个词的总结，也就是说隐藏层 h 是对下一个词的一种预测。

再看看第二个式子，预测下一个词为 w_j 的 log 概率是 y_j ，它直接就是 $C(w_j)$ 和 h 的内积。内积基本上就可以反应相似度，如果各词向量的模基本一致的话，内积的大小能直接反应两个向量的 \cos 夹角的大小。这里使用预测词向量 h 和各个已知词的词向量的相似度作为 log 概率，将词向量的作用发挥到了极致。这也是我觉得这次介绍的模型中最漂亮的一个。

这种“Log-Bilinear”模型看起来每个词需要使用上文所有的词作为输入，于是语料中最长的句子有多长，就会有多少个 H 矩阵。这显然是过于理想化了。最后在实现模型时，还是迫于现实的压力，用了类似 n-gram 的近似，只考虑了上文的 3 到 5 个词作为输入来预测下一个词。

M&H 的思路如前面提到，是 Bengio 2003 提出的。经过大牛的实现，效果确实不错。虽然复杂度没有数量级上的降低，但是由于是纯线性模型，没有激活函数（当然在做语言模型的时候，最后还是对 y_j 跑了一个 softmax），因此实际的训练和预测速度都会有很大的提升。同时隐藏层到输出层的变量直接用了词向量，这也就几乎少了一半的变

量，使得模型更为简洁。最后论文中 M&H 用了和 Bengio 2003 完全一样的数据集做实验，效果有了一定的提升。

2008 年 NIPS 的这篇论文，介绍的是“hierarchical log-bilinear”模型，很多论文中都把它称作简称“HLBL”。和前作相比，该方法使用了一个层级的结构做最后的预测。可以简单地设想一下把网络的最后一层变成一颗平衡二叉树，二叉树的每个非叶节点用于给预测向量分类，最后到叶节点就可以确定下一个词是哪个了。这在复杂度上有显著的提升，以前是对 $|V|$ 个词——做比较，最后找出最相似的，现在只需要做 $\log_2(|V|)$ 次判断即可。

这种层级的思想最初可见于 Frederic Morin 和 Yoshua Bengio 于 2005 年发表的论文《Hierarchical probabilistic neural network language model》中。但是这篇论文使用 WordNet 中的 IS-A 关系，转化为二叉树用于分类预测。实验结果发现速度提升了，效果变差了。

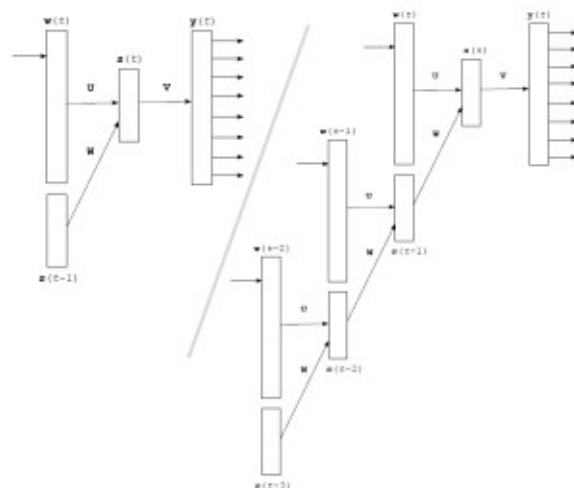
有了前车之鉴，M&H 就希望能从语料中自动学习出一棵树，并能达到比人工构建更好的效果。M&H 使用一种 bootstrapping 的方法来构建这棵树。从随机的树开始，根据分类结果不断调整和迭代。最后得到的是一棵平衡二叉树，并且同一个词的预测可能处于多个不同的叶节点。这种用多个叶节点表示一个词的方法，可以提升下一个词是多义词时候的效果。M&H 做的还不够彻底，后面 Huang 的工作直接对每个词学习出多个词向量，能更好地处理多义词。

Mikolov 的 RNNLM

Mikolov 最早用循环神经网络做语言模型是在 INTERSPEECH 2010 上发表的《Recurrent neural network based language model》里。Recurrent neural network 是循环神经网络，简称 RNN，还有个 Recursive neural networks 是递归神经网络（Richard Socher 借此发了一大堆论文），也简称 RNN。看到的时候需要注意区分一下。不过到目前为止，RNNLM 只表示循环神经网络做的语言模型，还没有歧义。

在之后的几年中，Mikolov 一直在 RNNLM 上做各种改进，有速度上的，也有准确率上的。现在想了解 RNNLM，看他的博士论文《Statistical Language Models based on Neural Networks》肯定是最好的选择。

循环神经网络与前面各方法中用到的前馈网络在结构上有比较大的差别，但是原理还是一样的。



左边是网络的抽象结构，由于循环神经网络多用在时序序列上，因此里面的输入层、隐藏层和输出层都带上了“(t)”。 $w(t)$ 是句子中第 t 个词的 One-hot representation 的向量，也就是说 w 是一个非常长的向量，里面只有一个元素是 1。而下面的 $s(t-1)$ 向量就是上一个隐藏层。最后隐藏层计算公式为：

$$s(t) = \text{sigmoid}(Uw(t) + Ws(t-1))$$

从右图可以看出循环神经网络是如何展开的。每来一个新词，就和上一个隐藏层联合计算出下一个隐藏层，隐藏层反复利用，一直保留着最新的状态。各隐藏层通过一层传统的前馈网络得到输出值。

$w(t)$ 是一个词的 One-hot representation，那么 $Uw(t)$ 也就相当于从矩阵 U 中选出了一列，这一列就是该词对应的词向量。

循环神经网络的最大优势在于，可以真正充分地利用所有上文信息来预测下一个词，而不像前面的其它工作那样，只能开一个 n 个词的窗口，只用前 n 个词来预测下一个词。从形式上看，这是一个非常“终极”的模型，毕竟语言模型里能用到的信息，他全用上了。可惜的是，循环神经网络形式上非常好看，使用起来却非常难优化，如果优化的不好，长距离的信息就会丢失，甚至还无法达到开窗口看前若干个词的效果。

Mikolov 在 RNNLM 里面只使用了最朴素的 BPTT 优化算法，就已经比 n -gram 中的 state of the art 方法有更好的效果，这非常令人欣慰。如果用上了更强的优化算法，最后效果肯定还能提升很多。

对于最后隐藏层到输出层的巨大计算量，Mikolov 使用了一种分组的方法：根据词频将 $|V|$ 个词分成 $\sqrt{|V|}$ 组，先通过 $\sqrt{|V|}$ 次判断，看下一个词属于哪个组，再通过若干次判断，找出其属于组内的哪个元素。最后均摊复杂度约为 $o(\sqrt{|V|})$ ，略差于 M&H 的 $o(\log(|V|))$ ，但是其浅层结构某种程度上可以减少误差传递，也不失为一种良策。

Huang 的语义强化

与前几位大牛的工作不同，Eric H. Huang 的工作是在 C&W 的基础上改进而成的，并非自成一派从头做起。他这篇发表在 ACL 2012 上的《Improving Word Representations via Global Context and Multiple Word Prototypes》试图通过对模型的改进，使得词向量富含更丰富的语义信息。他在文中提出了两个主要创新来完成这一目标：（其实从论文标题就能看出来）第一个创新是使用全文信息辅助已有的局部信息，第二个创新是使用多个词向量来表示多义词。下面逐一介绍。

Huang 认为 C&W 的工作只利用了“局部上下文（Local Context）”。C&W 在训练词向量的时候，只使用了上下文各 5 个词，算上自己总共有 11 个词的信息，这些局部的信息还不能充分挖掘出中间词的语义信息。Huang 直接使用 C&W 的网络结构计算出一个得分，作为“局部得分”。

然后 Huang 提出了一个“全局信息”，这有点类似传统的词袋子模型。词袋子模型是把文章中所有词的 One-hot Representation 加起来，形成一个向量（就像把词全都扔进一个袋子里），用来表示文章。Huang 的全局模型是将文章中所有词的词向量求个加权平均（权重是词的 idf），作为文章的语义。他把文章的语义向量和当前词的词向量拼接起来，形成一个两倍长度的向量作为输入，之后还是用 C&W 的网络结构算出一个得分。

有了 C&W 方法得到的“局部得分”，再加上在 C&W 方法基础上改造得到的“全局得分”，Huang 直接把两个得分相加，作为最终得分。最终得分使用 C&W 提出的 pair-wise 目标函数来优化。

加了这个全局信息有什么用处呢？Huang 在实验中发现，他的模型能更好地捕捉词的语义信息。比如 C&W 的模型中，与 markets 最相近的词为 firms、industries；而 Huang 的模型得到的结果是 market、firms。很明显，C&W 的方法由于只考虑了临近词的信息，最后的结果是词法特征最相近的词排在了前面（都是复数形式）。不过我觉得这个可能是英语才有的现象，中文没有词形变化，如果在中文中做同样的实验还不知道会有什么效果。

Huang 论文的第二个贡献是将多义词用多个词向量来表示。Bengio 2003 在最后提过这是一个重要的问题，不过当时他还在想办法解决，现在 Huang 给出了一种思路。

将每个词的上下文各 5 个词拿出来，对这 10 个词的词向量做加权平均（同样使用 idf 作为权重）。对所有得到的上下文向量做 k-means 聚类，根据聚类结果给每个词打上标签（不同类中的同一个词，当作不同的词处理），最后重新训练词向量。

当然这个实验的效果也是很不错的，最后 star 的某一个表示最接近的词是 movie、film；另一个表示最接近的词是 galaxy、planet。

总结

事实上，除了 RNNLM 以外，上面其它所有模型在第一层（输入层到隐藏层）都是等价的，都可以看成一个单层网络。可能形式最为特别的是 M&H 的模型，对前面的每个词单独乘以矩阵 H_i ，而不是像其它方法那样把词向量串接起来乘以矩阵 H 。但如果把 H 看成 H_i 的拼接： $[H_1 H_2 \dots H_t]$ ，则会有以下等式：

$$[H_1 H_2 \dots H_t] \begin{bmatrix} C(w_1) \\ C(w_2) \\ \dots \\ C(w_t) \end{bmatrix} = H_1 C(w_1) + H_2 C(w_2) + \dots + H_t C(w_t)$$

这么看来还是等价的。

所以前面的这么多模型，本质是非常相似的。都是从前若干个词的词向量通过线性变换抽象出一个新的语义（隐藏层），再通过不同的方法来解析这个隐藏层。模型的差别主要就在隐藏层到输出层的语义。Bengio 2003 使用了最朴素的线性变换，直接从隐藏层映射到每个词；C&W 简化了模型（不求语言模型），通过线性变换将隐藏层转换成一个打分；M&H 复用了词向量，进一步强化了语义，并用层级结构加速；Mikolov 则用了分组来加速。