Deep Learning in NLP （一）词向量和语言模型

2013 年 7 月 29 日 BY LICSTAR · 325条评论

这篇博客是我看了半年的论文后，自己对 Deep Learning 在 NLP 领域中应用的理解和总结，在此分享。其中必然有局限性，欢迎各种交流，随便拍。

Deep Learning 算法已经在图像和音频领域取得了惊人的成果，但是在 NLP 领域中尚未见到如此激动人心的结果。关于这个原因，引一条我比较赞同的微博。

> @王威廉：Steve Renals算了一下icassp录取文章题目中包含deep learning的数量，发现有44篇，而naacl则有0篇。有一种说法是，语言（词、句子、篇章等）属于人类认知过程中产生的高层认知抽象实体，而语音和图像属于较为底层的原始输入信号，所以后两者更适合做deep learning来学习特征。
> 2013年3月4日 14:46

第一句就先不用管了，毕竟今年的 ACL 已经被灌了好多 Deep Learning 的论文了。第二句我很认同，不过我也有信心以后一定有人能挖掘出语言这种高层次抽象中的本质。不论最后这种方法是不是 Deep Learning，就目前而言，Deep Learning 在 NLP 领域中的研究已经将高深莫测的人类语言撕开了一层神秘的面纱。

我觉得其中最有趣也是最基本的，就是“词向量”了。

将词用“词向量”的方式表示可谓是将 Deep Learning 算法引入 NLP 领域的一个核心技术。大多数宣称用了 Deep Learning 的论文，其中往往也用了词向量。

本文目录：

0. 词向量是什么
1. 词向量的来历
2. 词向量的训练
    2.0 语言模型简介
    2.1 Bengio 的经典之作
    2.2 C&W 的 SENNA
    2.3 M&H 的 HLBL
    2.4 Mikolov 的 RNNLM
    2.5 Huang 的语义强化
    2.999 总结
3. 词向量的评价
    3.1 提升现有系统
    3.2 语言学评价
参考文献

0. 词向量是什么

自然语言理解的问题要转化为机器学习的问题，第一步肯定是要找一种方法把这些符号数学化。

NLP 中最直观，也是到目前为止最常用的词表示方法是 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。

举个栗子，

“话筒”表示为 [0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 …]

“麦克”表示为 [0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 …]

每个词都是茫茫 0 海中的一个 1。

这种 One-hot Representation 如果采用稀疏方式存储，会是非常的简洁：也就是给每个词分配一个数字 ID。比如刚才的例子中，话筒记为 3，麦克记为 8（假设从 0 开始记）。如果要编程实现的话，用 Hash 表给每个词分配一个编号就可以了。这么简洁的表示方法配合上最大熵、SVM、CRF 等等算法已经很好地完成了 NLP 领域的各种主流任务。

当然这种表示方法也存在一个重要的问题就是“词汇鸿沟”现象：任意两个词之间都是孤立的。光从这两个向量中看不出两个词是否有关系，哪怕是话筒和麦克这样的同义词也不能幸免于难。

Deep Learning 中一般用到的词向量并不是刚才提到的用 One-hot Representation 表示的那种很长很长的词向量，而是用 **Distributed Representation**（不知道这个应该怎么翻译，因为还存在一种叫“Distributional Representation”的表示方法，又是另一个不同的概念）表示的一种低维实数向量。这种向量一般长成这个样子：[0.792, −0.177, −0.107, 0.109, −0.542, …]。维度以 50 维和 100 维比较常见。这种向量的表示不是唯一的，后文会提到目前计算出这种向量的主流方法。

（个人认为）Distributed representation 最大的贡献就是让相关或者相似的词，在距离上更接近了。向量的距离可以用最传统的欧氏距离来衡量，也可以用 cos 夹角来衡量。用这种方式表示的向量，“麦克”和“话筒”的距离会远远小于“麦克”和“天气”。可能理想情况下“麦克”和“话筒”的表示应该是完全一样的，但是由于有些人会把英文名“迈克”也写成“麦克”，导致“麦克”一词带上了一些人名的语义，因此不会和“话筒”完全一致。

1. 词向量的来历

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，（甚至我很无厘头地猜想那篇文章是为了给他刚提出的 BP 网络打广告，）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。

Distributed representation 用来表示词，通常被称为“Word Representation”或“Word Embedding”，中文俗称“词向量”。真的只能叫“俗称”，算不上翻译。半年前我本想翻译的，但是硬是想不出 Embedding 应该怎么翻译的，后来就这么叫习惯了-_-||| 如果有好的翻译欢迎提出。（更新：@南大周志华 在这篇微博中给了一个合适的翻译：词嵌入）Embedding 一词的意义可以参考维基百科的相应页面（链接）。后文提到的所有“词向量”都是指用 **Distributed Representation** 表示的词向量。

如果用传统的稀疏表示法表示词，在解决某些任务的时候（比如构建语言模型）会造成维数灾难[Bengio 2003]。使用低维的词向量就没这样的问题。同时从实践上看，高维的特征如果要套用 Deep Learning，其复杂度几乎是难以接受的，因此低维的词向量在这里也饱受追捧。

同时如上一节提到的，相似词的词向量距离相近，这就让基于词向量设计的一些模型自带平滑功能，让模型看起来非常的漂亮。

2. 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。

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。

2.0 语言模型简介

插段广告，简单介绍一下语言模型，知道的可以无视这节。

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。

顺便提一句，由于后面要介绍的每篇论文使用的符号差异太大，本博文里尝试统一使用 Bengio 2003 的符号系统（略做简化），以便在各方法之间做对比和分析。

2.1 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。（感谢 @余凯_西二旗民工 博士指出。）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了。

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》。当然现在看的话，肯定是要看他在 2003 年投到 JMLR 上的同名论文了。

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。如图1。

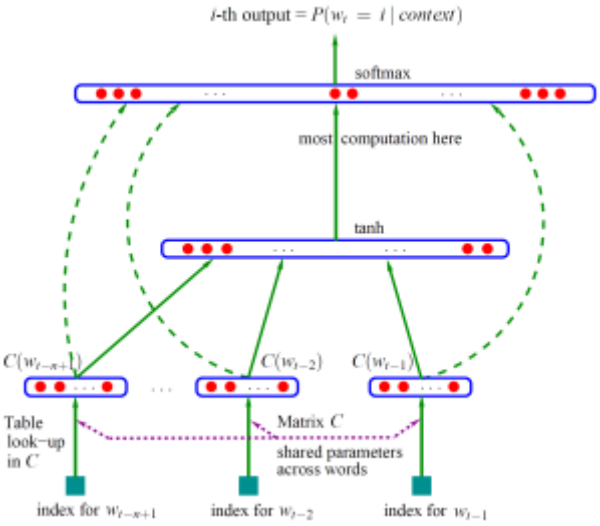

图1（点击查看大图）

图中最下方的 $w_{t-n+1}, \ldots, w_{t-2}, w_{t-1}$ 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 $C(w_{t-n+1}), \ldots, C(w_{t-2}), C(w_{t-1})$ 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%。

在结束介绍 Bengio 大牛的经典作品之前再插一段八卦。在其 JMLR 论文中的未来工作一段，他提了一个能量函数，把输入向量和输出向量统一考虑，并以最小化能量函数为目标进行优化。后来 M&H 工作就是以此为基础展开的。

他提到一词多义有待解决，9 年之后 Huang 提出了一种解决方案。他还在论文中随口（不是在

Future Work 中写的）提到：可以使用一些方法降低参数个数，比如用循环神经网络。后来 Mikolov 就顺着这个方向发表了一大堆论文，直到博士毕业。

大牛就是大牛。

2.2 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。这篇 JMLR 的论文题目也很霸气啊：从头开始搞 NLP。他们还把论文所写的系统开源了，叫做 SENNA（主页链接），3500 多行纯 C 代码也是写得非常清晰。我就是靠着这份代码才慢慢看懂这篇论文的。可惜的是，代码只有测试部分，没有训练部分。

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是太合理；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的 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。

关于这篇论文其实还是有些东西可以吐槽的，不过训练词向量这一块没有，是论文其他部分的。把吐槽机会留给下一篇博文了。

2.3 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 $y_j = \sum_{i=1}^{n-1} C(w_j)^T H_i C(w_i)$。$C(w)$ 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。

————————两篇文章中间是不是应该有个分割线？————————

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 $|V|$ 个词一一做比较，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。

2.4 Mikolov 的 RNNLM

前文说到，Bengio 2003 论文里提了一句，可以使用一些方法降低参数个数，比如用循环神经网络。Mikolov 就抓住了这个坑，从此与循环神经网络结下了不解之缘。他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 在一直在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。网络结构大致如图2。

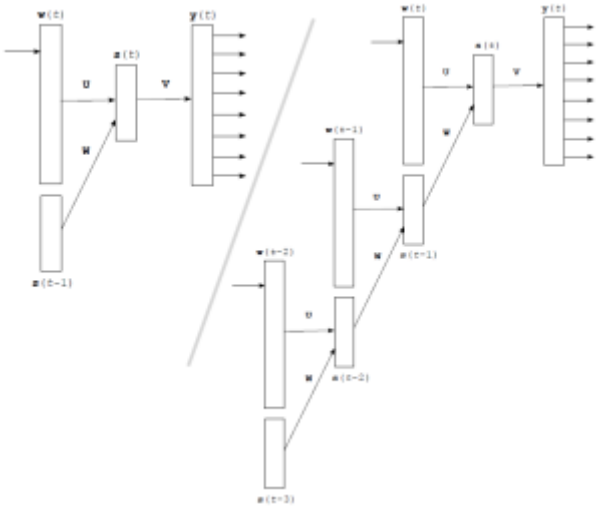

图2（点击查看大图）

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。

Mikolov 的 RNNLM 也是开源的（网址）。非常算法风格的代码，几乎所有功能都在一个文件里，工程也很好编译。比较好的是，RNNLM 可以完美支持中文，如果语料存成 UTF-8 格式，就可以直接用了。

最后吐槽一句，我觉得他在隐藏层用 sigmoid 作为激活函数不够漂亮。因为隐藏层要和输入词联合计算得到下一个隐藏层，如果当前隐藏层的值全是正的，那么输入词对应的参数就会略微偏

负，也就是说最后得到的词向量的均值不在 0 附近。总感觉不好看。当然，从实验效果看，是我太强迫症了。

2.5 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打分。

有了 C&W 方法的得到的“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。

这篇文章还做了一些对比实验，在下一章评价里细讲。

2.999 总结

//博主道：本节承上启下，不知道应该放在第 2 章还是第 3 章，便将小节号写为 2.999。

讲完了大牛们的各种方法，自己也忍不住来总结一把。当然，为了方便对比，我先列举一下上面提到的各个系统的现有资源，见下表。对应的论文不在表中列出，可参见最后的参考文献。

名称	训练语料及规模	词向量	特点	资源
C&W	English Wikipedia + Reuters RCV1 共 631M + 221M 词	130000 词 50 维	不区分大小写；经过有监督修正；训练了 7 周	测试代码、词向量 [链接]
C&W - Turian	Reuters RCV1 63M 词	268810 词 25、50、100、200 维	区分大小写；训练了若干周	训练代码、词向量 [链接]
M&H - Turian	Reuters RCV1	246122 词 50、100 维	区分大小写；用GPU训练了7天	词向量 [链接]
Mikolov	Broadcast news	82390 词 80、640、1600 维	不区分大小写；训练了若干天	训练、测试代码、词向量 [链接]
Huang 2012	English Wikipedia	100232 词 50 维	不区分大小写；最高频的6000词，每词有10种表示	训练、测试代码、语料及词向量 [链接]

显示第 1 至 5 项结果，共 5 项

Turian 的工作前面只是提了一下，他在做 C&W 向量与 H&M 向量的对比实验时，自己按照论文重新实现了一遍他们的方法，并公布了词向量。后来 C&W 在主页上强调了一下：尽管很多论文把 Turian 实现的结果叫做 C&W 向量，但是与我发布的词向量是不同的，我这个在更大的语料上训练，还花了两个月时间呢！

Turian 公布的 H&M 向量是直接请 Andriy Mnih 在 Turian 做好的语料上运行了一下 HLBL，所以没有代码公布。同时 Turian 自己实现了一份 LBL模型，但是没有公布训练出来的词向量。（这是根据他主页上描述推测的结果，从 Turian 的论文中看，他应该是实现了 HLBL 算法并且算出词向量

的。）

RCV1 的词数两篇文章中所写的数据差距较大，还不知道是什么原因。

Holger Schwenk 在词向量和语言模型方面也做了一些工作，看起来大体相似，也没仔细读过他的论文。有兴趣的读者可以直接搜他的论文。

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。

每种方法真正的差别看起来并不大，当然里面的这些创新，也都是有据可循的。下一章就直接来看看不同模型的效果如何。

3. 词向量的评价

词向量的评价大体上可以分成两种方式，第一种是把词向量融入现有系统中，看对系统性能的提升；第二种是直接从语言学的角度对词向量进行分析，如相似度、语义偏移等。

3.1 提升现有系统

词向量的用法最常见的有两种：

1. 直接用于神经网络模型的输入层。如 C&W 的 SENNA 系统中，将训练好的词向量作为输入，用前馈网络和卷积网络完成了词性标注、语义角色标注等一系列任务。再如 Socher 将词向量作为输入，用递归神经网络完成了句法分析、情感分析等多项任务。

2. 作为辅助特征扩充现有模型。如 Turian 将词向量作为额外的特征加入到接近 state of the art 的方法中，进一步提高了命名实体识别和短语识别的效果。

具体的用法理论上会在下一篇博文中细讲。

C&W 的论文中有一些对比实验。实验的结果表明，使用词向量作为初始值替代随机初始值，其效果会有非常显著的提升（如：词性标注准确率从 96.37% 提升到 97.20%；命名实体识别 F 值从 81.47% 提升到 88.67%）。同时使用更大的语料来训练，效果也会有一些提升。

Turian 发表在 ACL 2010 上的实验对比了 C&W 向量与 M&H 向量用作辅助特征时的效果。在短语识别和命名实体识别两个任务中，C&W 向量的效果都有略微的优势。同时他也发现，如果将这两种向量融合起来，会有更好的效果。除了这两种词向量，Turian 还使用 Brown Cluster 作为辅助特征做了对比，效果最好的其实是 Brown Cluster，不过这个已经超出本文的范围了。

3.2 语言学评价

Huang 2012 的论文提出了一些创新，能提升词向量中的语义成分。他也做了一些实验对比了各种词向量的语义特性。实验方法大致就是将词向量的相似度与人工标注的相似度做比较。最后 Huang 的方法语义相似度最好，其次是 C&W 向量，再然后是 Turian 训练的 HLBL 向量与 C&W 向量。这里因为 Turian 训练词向量时使用的数据集（RCV1）与其他的对比实验（Wiki）并不相同，因此并不是非常有可比性。但从这里可以推测一下，可能更大更丰富的语料对于语义的挖掘是有帮助的。

还有一个有意思的分析是 Mikolov 在 2013 年刚刚发表的一项发现。他发现两个词向量之间的关系，可以直接从这两个向量的差里体现出来。向量的差就是数学上的定义，直接逐位相减。比如 $C(\text{king}) - C(\text{queen}) \approx C(\text{man}) - C(\text{woman})$。更强大的是，与 $C(\text{king}) - C(\text{man}) + C(\text{woman})$ 最接近的向量就是 $C(\text{queen})$。

为了分析词向量的这个特点， Mikolov 使用类比（analogy）的方式来评测。如已知 a 之于 b 犹如 c 之于 d。现在给出 a、b、c，看 $C(a) - C(b) + C(c)$ 最接近的词是否是 d。

在文章 Mikolov 对比了词法关系（名词单复数 good-better:rough-rougher、动词第三人称单数、形容词比较级最高级等）和语义关系（clothing-shirt:dish-bowl）。

在词法关系上，RNN 的效果最好，然后是 Turian 实现的 HLBL，最后是 Turian 的 C&W。（RNN-80:19%；RNN-1600:39.6%；HLBL-100:18.7%；C&W-100:5%；-100表示词向量为100维）

在语义关系上，表现最好的还是 RNN，然后是 Turian 的两个向量，差距没刚才的大。（RNN-80:0.211；C&W-100:0.154；HLBL-100:0.146）

但是这个对比实验用的训练语料是不同的，也不能特别说明优劣。

这些实验结果中最容易理解的是：语料越大，词向量就越好。其它的实验由于缺乏严格控制条件进行对比，谈不上哪个更好哪个更差。不过这里的两个语言学分析都非常有意思，尤其是向量之间存在这种线性平移的关系，可能会是词向量发展的一个突破口。