

深度学习下的自然语言处理

CS224N

学习笔记总结

作者：徐泽南

目录

前言	III
第 1 课 背景介绍.....	1
1.1 什么是自然语言处理.....	1
1.1.1 自然语言处理涉及的层次.....	1
1.1.2 自然语言处理的应用.....	3
1.1.3 人类语言特殊之处.....	3
1.2 什么是深度学习.....	5
1.2.1 传统机器学习特点.....	5
1.2.2 深度学习特点.....	6
1.2.3 为什么学习探索深度学习.....	8
1.2.4 深度学习带来的突破.....	8
1.3 为什么 NLP 难	10
1.4 深度学习与自然语言处理的结合.....	10
1.4.1 词向量简谈.....	10
1.4.2 NLP 形态级别层次简谈	11
1.4.3 NLP 句法分析简谈	12
1.4.4 NLP 语义层面简谈	12
1.5 自然语言处理的应用简谈.....	13
第 2 课 word2vec 入门.....	17
2.1 传统方法处理词语.....	17
2.1.1 Discrete Representation	17
2.2 Distributional representations	18
2.2.1 Skip-gram prediction	20
第 3 课 word2vec 高级 GloVe.....	25
3.1 传统 SG 模型的缺点以及解决方案	25
3.1.1 传统 SG 模型的缺点	25
3.1.2 Negative Sampling.....	26
3.1.3 改进后的 SG	27
3.2 co-occurrence 方法.....	28
3.3 GloVe.....	30
3.4 词向量评估方法.....	32
3.4.1 外部任务评价.....	32
3.4.2 内部任务评价.....	33
3.4.3 内部任务调参.....	37
第 4 课 神经网络入门.....	41
4.1 分类问题.....	41
4.1.1 分类问题介绍.....	41
4.1.2 分类问题损失函数.....	42
4.1.3 分类问题参数更新.....	44
4.2 简单的分类任务（命名实体识别）	46
4.3 简单的神经网络.....	51

课外知识补充（部分完成）	52
课外第一课 神经网络入门.....	53
1.1 神经网络是什么.....	53
1.2 神经网络简单应用问题.....	54
1.3 神经网络实现逻辑与和逻辑或功能.....	55
1.4 神经网络稍复杂应用问题.....	56
1.5 神经网络表达力与过拟合	57
1.6 参数更新 BP 算法	58

前言

本课程是旧版的 CS224d (Deep Learning for Natural Language Processing) 与旧版 CS224n (Natural Language Processing) 的合并版本新 CS224n, 一共 18 门课。这门课之前叫做 Deep Learning for Natural Language Processing, 之后改名为目前的 Natural Language Processing with Deep Learning。两者命名感觉前者似乎更突出自然语言处理的地位, 而后者则是将二者地位平等对待, 笔者觉得后者命名更好, 毕竟深度学习是一个学习工具, 它既可以用于自然语言处理, 也可以用于图像处理, 后者的命名更好的体现了二者的关系。

笔者的这份读书笔记是完全按照课程内容展开(除此之外也额外附加一些视频里面没有的), 以课程的时间安排来进行文章段落的安排。此读书笔记的目的在于记录这门课程的相关知识, 以便自己日后复习或者给其他有需要的人进行学习。

注, 文中所用截图均取自 CS224d 的视频中, 此后不在重复申明。(此读书笔记不可用于商用, 秉承着知识分享的目的)。

第 1 课 背景介绍

1.1 什么是自然语言处理

自然语言处理其实是一门与计算机科学，人工智能技术和语言学的一门交叉学科。顾名思义，现阶段我们处理语音或者是文本挖掘都需要使用到计算机这一工具(如图 1.1a 所示)，自然这就涉及计算机科学；想让计算机理解人类的语言，并对此作出一定的反应，这一定程度上使得机器具备了智能，所以这又涉及到人工智能；语言学就更不用说明了，计算机研究语言的前提就是人类先研究出语言的规律，再此基础上教会计算机，这自然涉及语言学。

自然语言处理的目标是什么？至少现阶段，我们的目标是让计算机去一定程度上理解我们的自然语言，根据我们的语意去完成一些有意义的任务。例如餐厅订座，帮忙定闹钟，又或者是有问有答的类似我们常见的苹果系统里的 Siri 等（如图 1.1）。要让计算机完全理解和组织语言与我们对话是非常困难的一件事情，完美的语言理解类似于实现完全的人工智能。

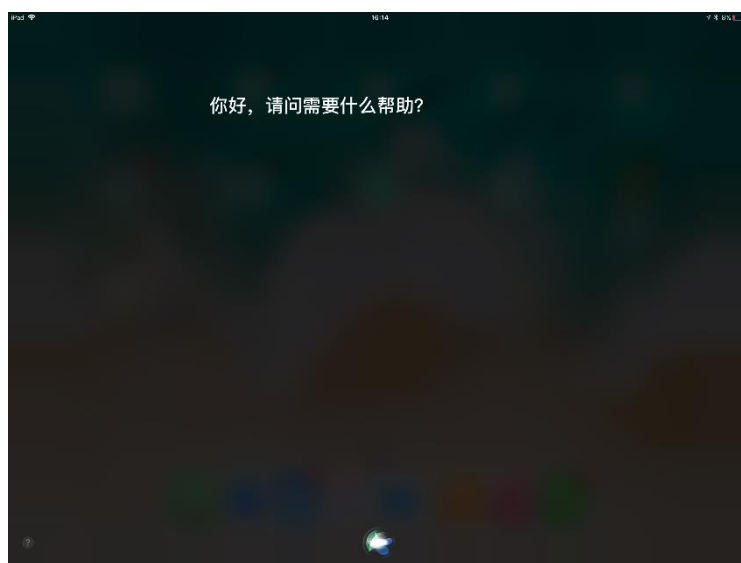


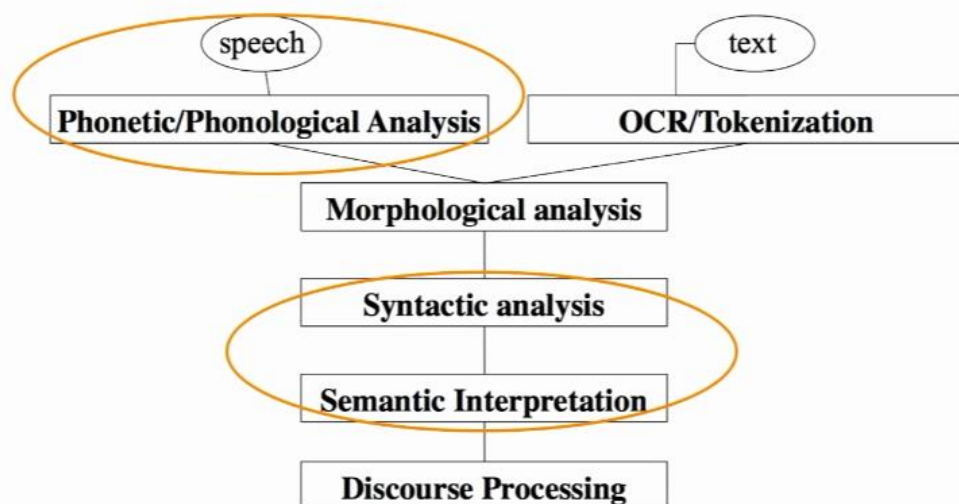
图 1.1 常见的问答系统 Siri 示意图

1.1.1 自然语言处理涉及的层次

自然语言处理涉及到不同的层次问题（如图 1.2），第一层自然是我们的输入层次。我们的输入有两个大的来源，一个是语音，一个是文本。对于语音首先我们还需要进行语音

分析，将声波转换成对应的文字或其他方便接下来的分析。第二个输入是文本输入，当通过扫描仪或数码相机扫描纸张上的文字时，我们还需要进行 OCR (Optical Character Recognition 光学字符识别)，当输入已经是文本的时候，我们可能还需要进行 Tokenization 分词操作 (英文的分词较为简单，直接一个单词一个分词，但是中文的分词较为复杂，例如超人你需要将这个划分为一个词，而不能划分为两个词例如超和人)。接下来的一层是形态分析 (Morphological analysis)，什么意思呢，形态学指词的内部结构，包括曲折变化啊和构词法，后面的构词法可以理解成，例如 hard 加上 ly 变成 hardly，这就是构词法，通过一个单词加上其他部分成为一个新的单词。再下一层是句法分析 (understand the structure of sentence that syntactic analysis)，例如有一句话我坐在沙滩上，那么我是主语，坐是谓语，诸如此类是句法分析。紧接着是语义分析，因为我们知道一个句子即便我们知道句子里面每一个单词是什么意思，我们也不一定能够完全理解这个句子的含义，很多时候一个句子的含义潜藏在语义信息或者上下文里，这就需要用到语义分析。最后一层应该是涉及到对话，这里我们就略过了。之后我们的课程重点关注画圈的三个部分。

NLP Levels



So in this class, where we're gonna
spend most of our time is in that middle

Stan
Univ

图 1.2 自然语言处理涉及到的层次示意图

1.1.2 自然语言处理的应用

接下来我们按照应用的从简单到复杂的程度列举一下自然语言处理的应用，拼写检查，关键字搜索等，文本挖掘例如从互联网上挖掘出产品价格，定位，人名或者公司名等，文本分类例如长文本的情感分类等，依次还有机器翻译，对话系统，复杂的问答系统等。



(a) 拼写检查

(b) 关键字搜索以及自动纠错



(c) 机器翻译

图 1.3 自然语言处理的简单应用图示

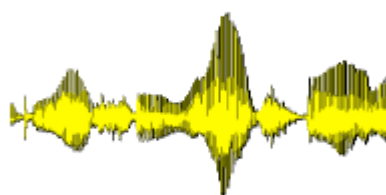
目前工业界自然语言处理的应用也是五花八门，包括网络广告匹配等，自动或者辅助翻译等，市场或者金融情感分析等（这里视频没有解释，估计是市场繁荣或者萧条之类的情感意思），语音识别，人机对话等。

1.1.3 人类语言特殊之处

视频中提到，人类语言系统实际上可以看成是“故意”构建某种信号去表达说话者的意思，这种信号跟自然界某些杂乱无序的信号不一样，这种信号是“故意”为之的。我们可以把人类语言理解成某种加密的信号（毕竟即便是人类之间也存在 A 听不懂 B 的语言，C 却可

以听懂 B 的语言这类情况)，而这种加密的信号小孩子学习起来却非常快。这一点让人非常吃惊（文中用 *amazingly*）。很多时候很小的一个表达式就可以表达强烈的信号，例如（*I Looooooooove it*, 或者 *Whooooomppaaa*），基于此文中提出语言这种符号绝对不是传统逻辑 AI 的产物，毕竟传统 AI 估计都是循规循矩，绝对说不出上文那种 *I Love it* 的感觉。

人类语言信号的种类有很多种，一下简单介绍三种，声音（如图 1.4a 所示），肢体语言（如图 1.4b 所示），又或者是图片与手写的文字（如图 1.4c 所示）。



（a）声波



（b）肢体语言



（c）一种还未知的文字

图 1.4 人类语言信号种类示意图

另外补充一点，视频中提到人类大脑我们认为是一种连续激活的一种模式，而并不是离散的，因为它处理的信号（语言）是一种连续性的声音或者视觉信号，所以我们设计实验的时候也设计成一种连续的模式。同时由于人类词汇量的巨大，这使得实验实现的时候很容易

导致稀疏性问题（sparsity）。（关于稀疏性问题，对于具有一定自然语言处理的同学自然很容易理解，这里举一个例子是为了帮助刚进入这一领域的同学理解稀疏性是什么意思。在我们做 LDA 主题模型的时候，我们会使用一个 tf（term frequency）来表示一个句子，例如如果总体词汇量有 1000，我们现在需要表达一个只有 6 个单词的句子，假设这个句子有 6 个不一样的单词，那么这个句子的向量表达就是一个长度 1000 其中有六个位置为 1 其余位置为 0 的向量，当然如果有 2 个一样的单词，那么向量就是长度 1000,5 个位置为 1,1 个位置为 2 的向量，很明显词汇量越大，这个向量就越长，但是里面的 1 却很少，大部分都是 0，这就带来了极大的稀疏性的问题，这个向量是非常稀疏的。）

1.2 什么是深度学习

深度学习首先是机器学习的一个子集。那么有了机器学习为什么又要提出深度学习呢？这里我们要看看传统的机器学习有什么特点。

1.2.1 传统机器学习特点

首先传统的机器学习解决一个问题需要专家对一个问题理解的非常透彻，提出一定的特征，然后人为的将特征提取出来交给计算机，例如一句话可能从句法，或者情感词，或者动词形容词上提取出了很多特征（如图 1.5 展示的是一个单词可以提取的特征，**目前笔者对该图第二栏 NER（Named Entity Recognition）命名实体识别的表示并不是很懂，✓可以理解成选用该特征，all 是指所有的 n-gram 都计算？size 4 是指 window 的大小？后续如果有人看懂欢迎告知**），然后让计算机给这些个特征赋予一定的权重并进行调整，最终使得误差越来越小。

Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

图 1.5 人工特征示意图

视频中给了一个例子，以前的 google search 为了提高一点精确度，通常需要 some bunch of（几堆人）工程师几个月研究提取一个新的特征，然后加进去让计算机继续计算权重，最终提高一点点精度。这就使得，人对某一个问题越来越熟悉，而计算机学到的只是计算调整权重而已。（相当于是对一个高纬度的方程一直求解而已，如图 1.6 所示）

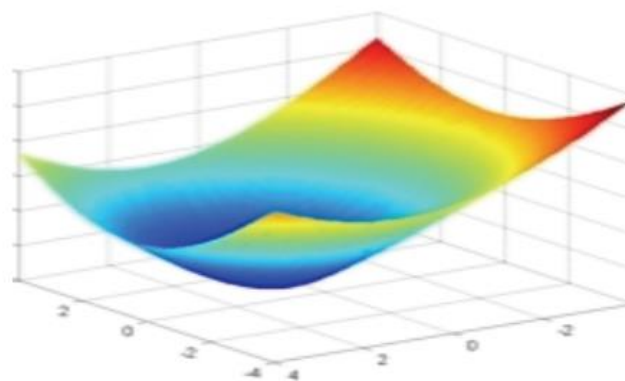


图 1.6 计算机实际上的工作

这就使得传统的机器学习方法可能 90%都是人工提取特征，最后 10%参数优化教给了计算机而已。（如图 1.7 所示）换句话说虽然叫机器学习但是感觉最终机器什么都没有学到！

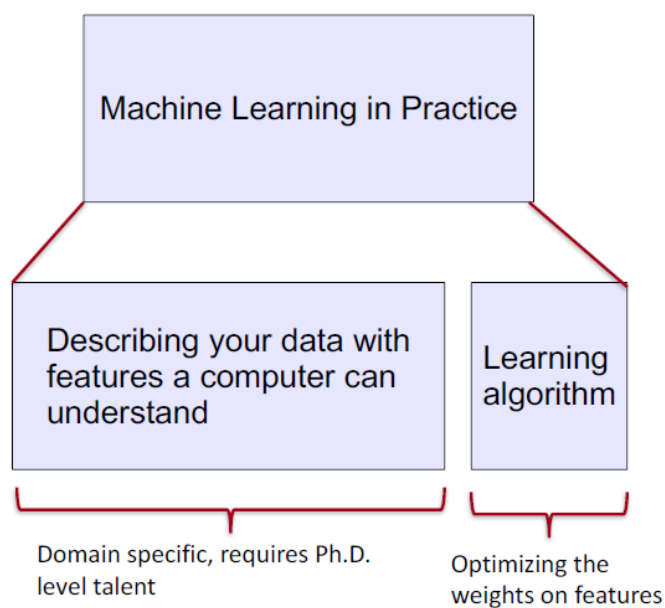


图 1.7 实际上机器学习任务分配

1.2.2 深度学习特点

相比较机器学习，在深度学习中，虽然也需要人工帮忙设计特征，但是这里的设计与传

统机器学习的设计不一样，传统机器学习通常是直接告诉计算机，特征怎么计算，而深度学习是告诉机器，有几个特征，特征间会有什么关系。（例如传统的机器学习如果有句法特征，会详细告诉计算机如何根据句子的某些词经过如何的计算得到，但是对于深度学习，我们会告诉计算机，一个句子可以提取出一个初步特征，初步特征又可以提取出高度特征，最后高度特征可以结合给句子分类，但是这些特征具体是什么我们不知道，只是告诉计算机每个句子的正确结果，然后计算机自己根据梯度迭代的方法去提取特征，最终得到的特征就算输出来但是人基本也看不懂，只有计算机可以看得懂）。我们先人工搭建深度学习的网络结构，然后将原始的（raw）（例如声音，字母符号或者单词）数据输入进去，等计算机自己处理加工提取特征即可。一个简单的深度学习网络结构如图 1.8 所示。

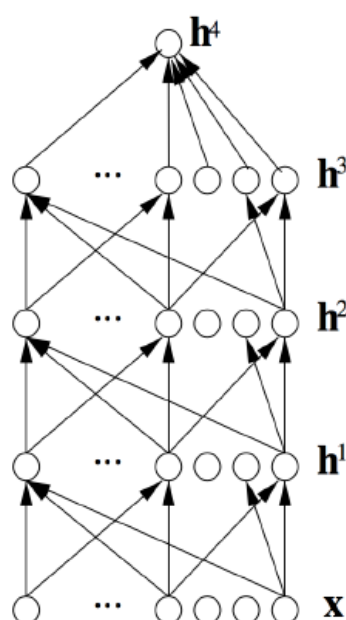


图 1.8 一个简单的深度学习网络示意图

虽然深度学习这个术语还包括其他很多模型，例如使用概率图等模型，但是目前大多数情况下都是指代各种各样多层的神经网络。稍微了解神经网络的人可能会说，这只不过是一些神经元的堆砌，或者说神经网络其实只是另一种计算的方式，毕竟最终还是通过损失函数（Loss Function）来进行迭代更新，感觉事实上好像也并没有多智能。（反驳这句话可以说，人也是细胞的堆砌，有什么了不得的呢？继续讨论还可以涉及到 $1+1>2$ 之类的话题，这就不在我们讨论范围内）。

本课程主要学习现阶段在 NLP 领域表现比较好的模型方法，而并不会讲解历史上 NLP 问题的处理办法，如果你非常想要了解，视频中给了一篇论文叫 *Deep Learning in Neural*

Networks: An Overview, 视频中给的评价是 I'll warn you it's a pretty dry and boring history, 如果这样还想看论文的话可以继续。

1.2.3 为什么学习探索深度学习

根据我们上面的讨论我们可以很明显看到,手工设计特征通常会设计的具有特别强的专用性,(在这个任务可以这样,其他任务就不行了)并且非常耗时。往往一大群工程师几个月才能设计出一个提高性能非常小的特征。而深度学习学习特征具备很好的泛化能力,换一个任务很快就又可以计算出适合的特征,即便你是从语音换到视觉,那么只需要稍微修改一下特征的数量或者关系(修改网络结果),计算机很快就又可以提取出适合这个任务的特征。同时深度学习对于有监督(相当于就是数据都提前打了标签的,可以根据标签学习)与无监督(相当于是数据是没有标签的,需要自己发掘数据间的关系)都可以进行学习,而人类对于无监督往往束手无策。(例如对于人类基因组计划,让人类去发现基因组合之间的关系是非常困难的,但是将基因数据输入给计算机,计算机通过大量快速的计算往往可以基因数据中的一些潜在的组合关系)

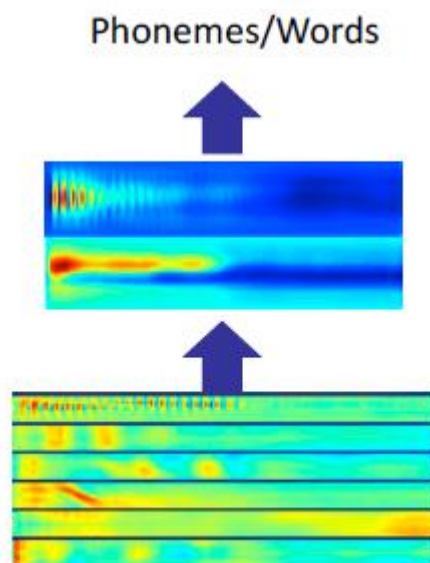
同时也是因为这个适合深度学习的时代。深度学习技术直到 2010 年左右才开始明显比其他机器学习方法优越,为什么是这个时间呢?那是因为这个时间是计算机高速发展的时间,深度学习其实伴随着巨大的计算量,这是最限制深度学习发展的。早些时候的计算机发展速度较慢,计算能力较差,很多时候可能根本无法开展深度学习任务(可能内存不够)或者即便可以计算但是也伴随着计算能力低下时间消耗巨大等缺点。近些年来随着多核计算机(CPU 或者 GPU)的出现,给深度学习带来了可能实现的硬件条件,再此基础上一些更好更高效的算法的出现使得深度学习面临更好的发展。接下来我们逐一讲解深度学习相比传统学习带来的正确率上的巨大突破。首先是在语音和图像中,接着是 NLP。

1.2.4 深度学习带来的突破

深度学习带来的第一个巨大型的突破就是在语音识别上(如图 1.9a 所示)(论文 *Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition Dahl et al. (2010)*),相比传统方法直接降低了 30%左右的错误率(如图 1.9b 所示)。笔者这里给大家推荐一个链接,跟语音识别有关,使用到了 RBN(循环玻尔兹曼机)。

<http://www.hankcs.com/ml/hinton-deep-neural-nets-with-generative-pre-training.html#h3-11>,

这里详细讲解了 RBM 用到语音识别。



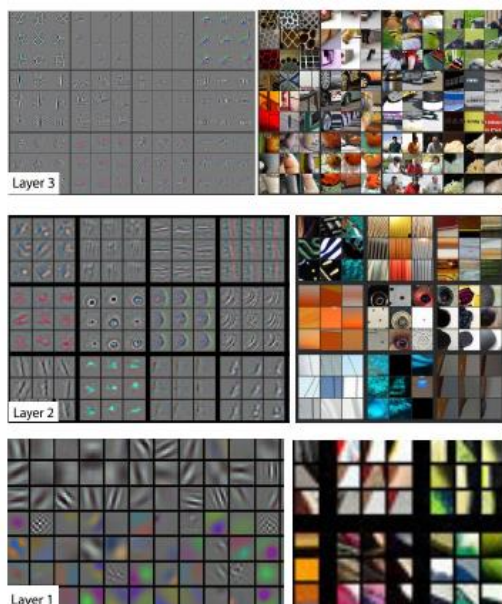
(a) 引入深度学习的语音识别分析图

Acoustic model	Recog WER	RT03S FSH	Hub5 SWB
Traditional features	1-pass -adapt	27.4	23.6
Deep Learning	1-pass -adapt	18.5 (-33%)	16.1 (-32%)

(b) 深度学习与传统特征方法语音识别错误率比较

图 1.9 深度学习下的语音识别

深度学习带来的第二个巨大突破是在计算机视觉上的（如图 1.10a 所示），（论文 *ImageNet Classification with Deep Convolutional Neural Networks* by Krizhevsky, Sutskever, & Hinton, 2012, U. Toronto.）。主要是用于图像识别（如图 1.10b 所示），相比传统方法直接降低了 37%左右的错误率。



(a) 引入深度学习的图像识别分析图



(b) 图像识别

图 1.10 深度学习下的计算机视觉

1.3 为什么 NLP 难

自然语言跟一般的程序语言不通，自然语言的表达是带有一定歧义性的，并且自然语言由于我们说话的语速问题（不像计算机可以秒传 MB 甚至是 GB 的数据），我们短短一句话里面可能会包含大量的信息，考虑到说话者和对象都具备一定的知识储备，所以很多时候我们说话是会省略掉大量信息，而这一段话语会在听者脑海中根据常识或者上下文将缺失的信息补充完全。接着视频中举了一个有意思的带有歧义的句子。The Pope's baby steps on gays，这里如果我们将 The Pope's baby 定为主语，那么这句话相当于就是 A 用脚踩了 B，如果我们将 The Pope's 定为主语，将 baby steps 看成一个词组（意思是一小步），那么这句话可以翻译成教皇在 B 问题上迈出了一小步。具体是那种意思需要结合上下文以及我们需要知道英文的一些词组。这些对于机器来说学习就非常困难。

1.4 深度学习与自然语言处理的结合

结合自然语言处理的思路以及目标，我们引入深度学习的方法来解决自然语言处理的困难。这些年我们在如下方面上有巨大的突破。从层次上看，我们在语音层，单词层面，句法层面，语义层面上有巨大突破；工具上我们在词性标注，命名实体识别，词语语法分析等提出了效果较好的工具来帮助实现；应用上有机器翻译，情感分析，问答系统。这里我们需要介绍一下 word vector，毕竟这是自然语言处理与深度学习可以结合的基础。

1.4.1 词向量简谈

词向量就是将每一个单词用向量的形式表示，这个向量极少数情况下可以有 25 维，但是一般情况下都是 300 维度居多，维度越多就越可以更好的表达一个单词的含义。（维度越多，就可以包含更多的细节）（有些人可能会问，假设一个 300 维的向量，那么每一维都代表什么意思，这里并没有说每一个维度都有确定的意思，比如第一个维度代表了学科，第二个维度代表情感之类，这些是没有的，你可以理解成，计算机也许知道每一个维度代表什么意思，但是人类是不知道的，因为每个单词最终的向量生成是计算机计算得到的）。词向量空间有这么一个特点，两个单词的语义越接近，那么两个向量的空间距离更近，如图 1.11 所示。注意由于人类对高维空间是很难有直观的感受的，所以图 1.11 是将向量进行了降维，

降到维度为 2，（常见的降维方法可以是 PCA，Principal Component Analysis，主成分分析，注意降维必然带来信息的损失，只是每种降维方法损失的不同而已）。我们观察可以看到图 1.11 中 go 和 come 距离是接近的，was 跟 were 也是接近的。



图 1.11 降维后的向量空间

1.4.2 NLP 形态级别层次简谈

前面我们说过形态级别就是类似于一个 `uninterested` 单词，它是由三个 morphemes（词素）构成（`un` 和 `interest` 以及 `ed`）。这只是形态学的一部分，但是这里我们主要谈论关于单词的构成。我们考虑每一个词素都是一个向量，然后通过一个神经网络来将多个词素合并构成新单词并得到新单词的向量表达，如图 1.12 所示。

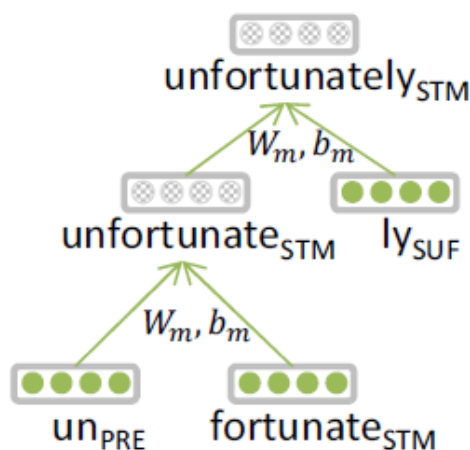


图 1.12 神经网络合成词素构成新单词示意图

1.4.3 NLP 句法分析简谈

神经网络可以精确地确定一个句子的构成，从而对翻译句子特别有帮助。这里列举图 1.13 帮助演示，其中（NNS，VBP 等单词目前我也还没看懂，图大概表达了神经网络用于句子结构分析，关于此图具体的讲述就不在这里进行，里面的隐藏层，以及最后的 softmax 函数之后大家都会遇到）。

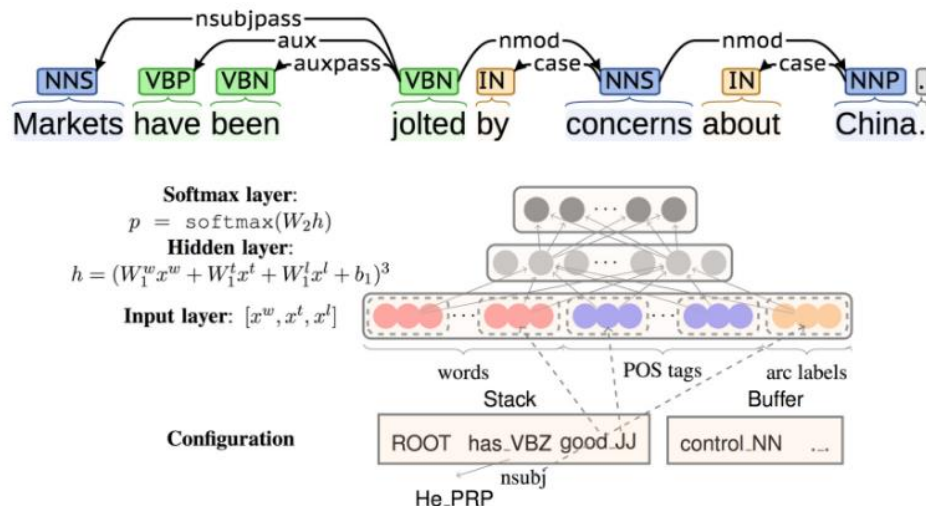


图 1.13 神经网络分析句子构成示意图

感兴趣的同学这里列举一篇论文 *A Fast and Accurate Dependency Parser using Neural Networks*，以及两个链接，基于神经网络的高性能依存句法分析器

<http://www.hankcs.com/nlp/parsing/neural-network-based-dependency-parser.html>

移植的 LTP 句法分析器

<http://hanlp.hankcs.com/?sentence=hello%20xzn>

1.4.4 NLP 语义层面简谈

传统的语义层面需要大量的工程师手写一些复杂的微积分函数（大多是概率函数），如图 1.14 所示。

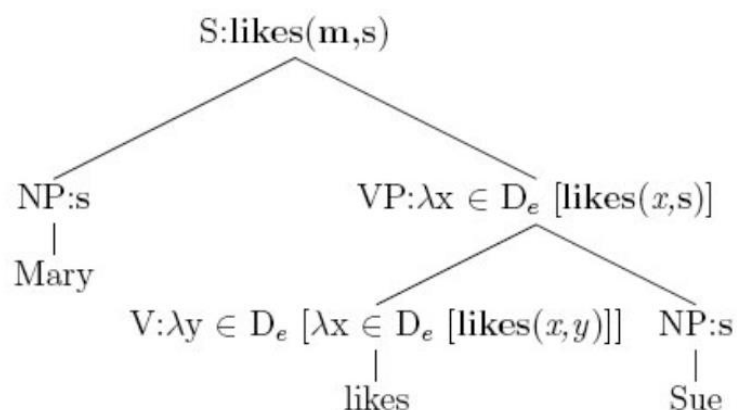


图 1.14 传统方法分析语义层面示意图

在深度学习中，每一个单词每一个短语甚至是逻辑表达式都可以看成是一个向量，神经网络可以将两至多个向量组合成一个向量。如图 1.15 所示

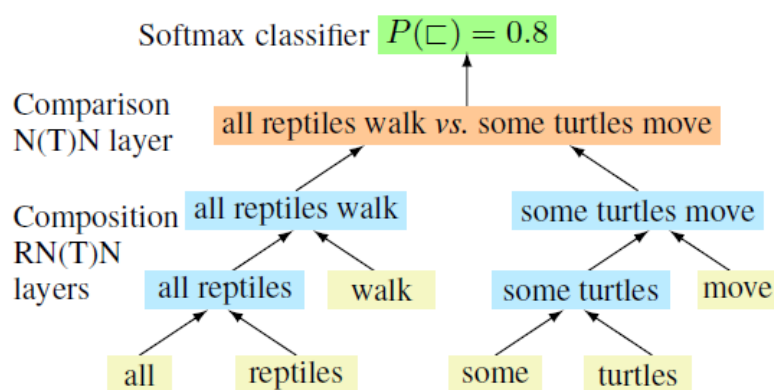


图 1.15 神经网络分析语义层面示意图 (Bowman et al. 2014)

1.5 自然语言处理的应用简谈

自然语言处理的应用有很多，这里我们快速讲一下，首先是情感分析，传统的方法可以是请人工做一个情感词典，然后对每个句子使用词带模型（就是不考虑词的顺序，只考虑你这里面有那些词）查找情感词典找到对应的情感然后加和或者其他方式来进行判断。深度学习可以直接使用 RNN 来解决。如图 1.16 所示。

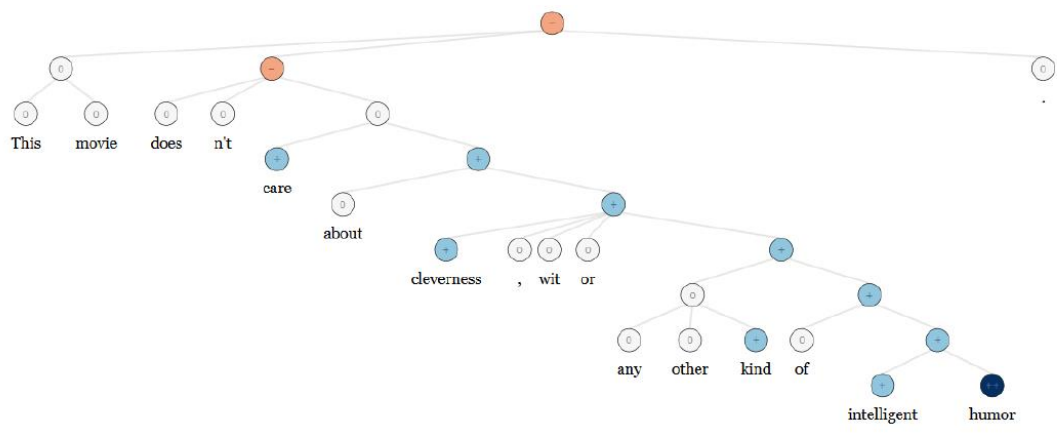


图 1.16 深度学习用于情感分析

另一个应用是问答系统,传统的方法需要工程师去根据已有的人类知识来手动寻找特征,然后通过一些规则表达式来进行判断(如图 1.17)。

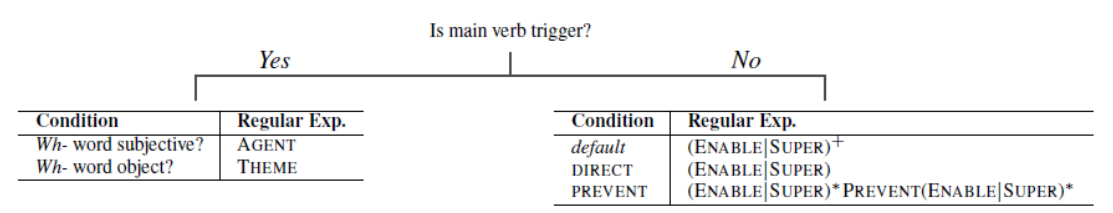


图 1.17 传统方法下的问答系统图示

深度学习则具备更多的优点,由于向量存储了一定程度上的信息,所以通过处理向量可以得到一个很好的效果。(如图 1.18)

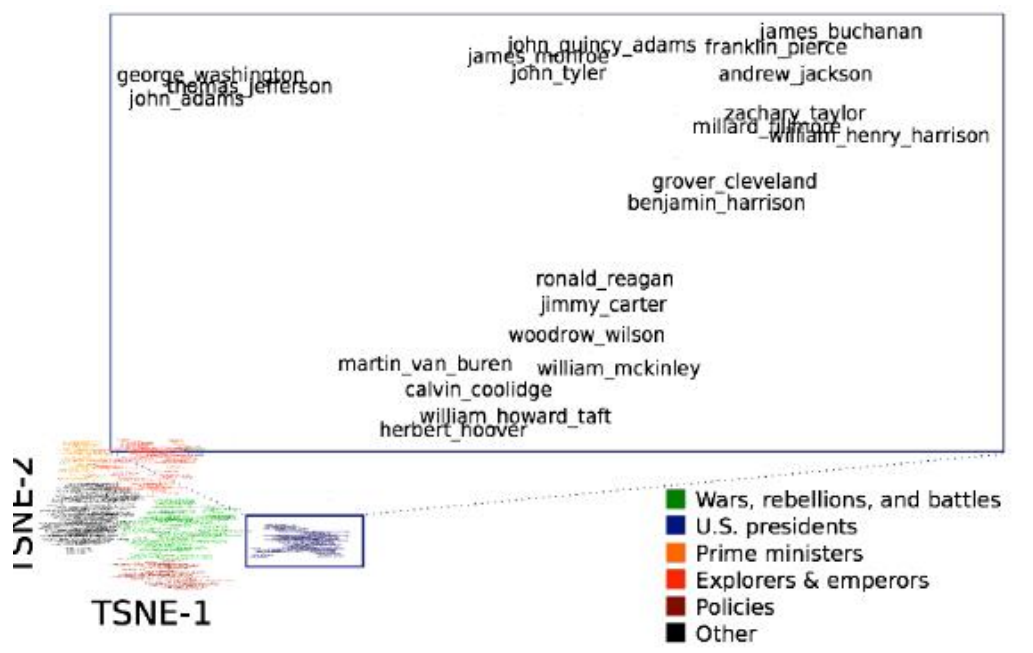


图 1.18 深度学习下的问答系统图示

还有一个例子是谷歌的 Gmail 自动回复(如图 1.19),他会读取你收到的邮件,然后给

出三个可能的回复,而这三个回复通常情况下效果非常好,你可以从其中找到你需要的答复。

(虽然这涉及邮件私密等,但是这个问题不在这里讨论)

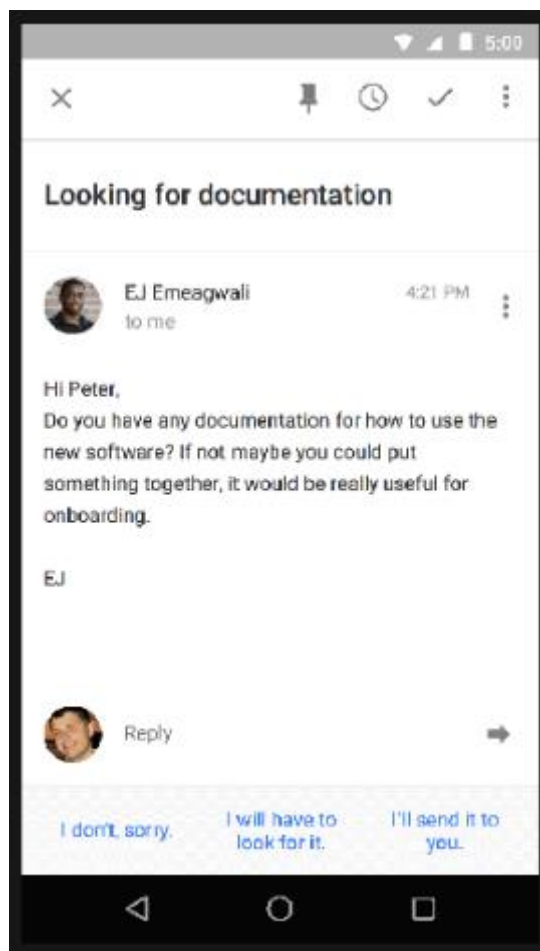


图 1.19 Gmail 自动回复示意图

Gmail 的实现主要用到了 RNN 循环神经网络,示意图如图 1.20 所示,这里我来讲解一下这个图是什么意思,首先经过训练之后 RNN 具备了生成答案的能力,这时候我们首先要指定一个输入,图中就是输入 the,然后神经网络计算在出现 the 的条件下出现其他词的概率,然后发现条件概率 $p(\text{cat}|\text{the})$ 是最大的,那么第二个单词输出 cat,以此循环,第三次发现条件概率 $p(\text{is}|\text{cat}, \text{the})$ 是最大的,那么输出 is,一直这样直到将答案输出出来。

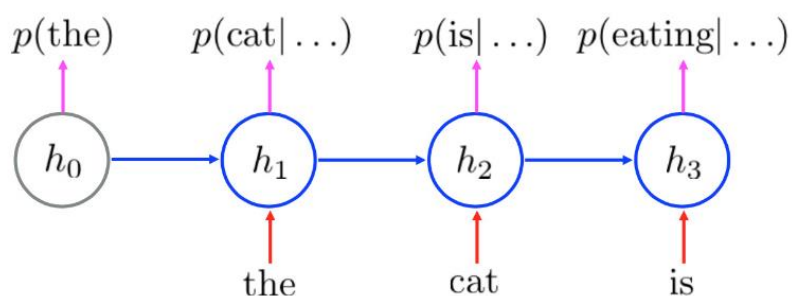


图 1.20 Gmail 原理 RNN 实现示意图

接下来我们谈谈机器翻译，传统方法在很多层级上做了尝试，例如直接翻译，或者通过语义分析翻译，句子架构辅助翻译等，这样往往使得翻译的过程变得非常复杂。视频中还提到人们试图找到一种 Interlingua（国际通用化语言），这个目的是为了假设有 AB 两种语言，我们可以先将 A 翻译成 Interlingua，然后再将 Interlingua 翻译成 B 语言，相当于 Interlingua 作为一个翻译的桥梁，世界上任何语言都先翻译成 Interlingua，然后再将 Interlingua 翻译成其他语言实现语言之间的翻译。示意图如图 1.21 所示。

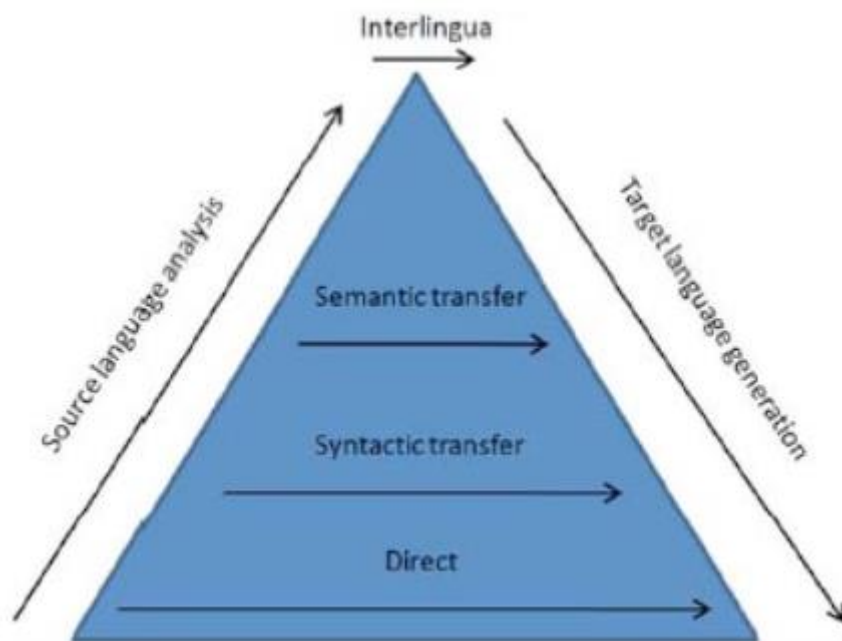


图 1.21 传统方法机器翻译示意图

结合了深度学习的机器翻译则是将单词映射成向量然后进行处理，如图 1.22 所示是多层循环 RNN 用于机器翻译。

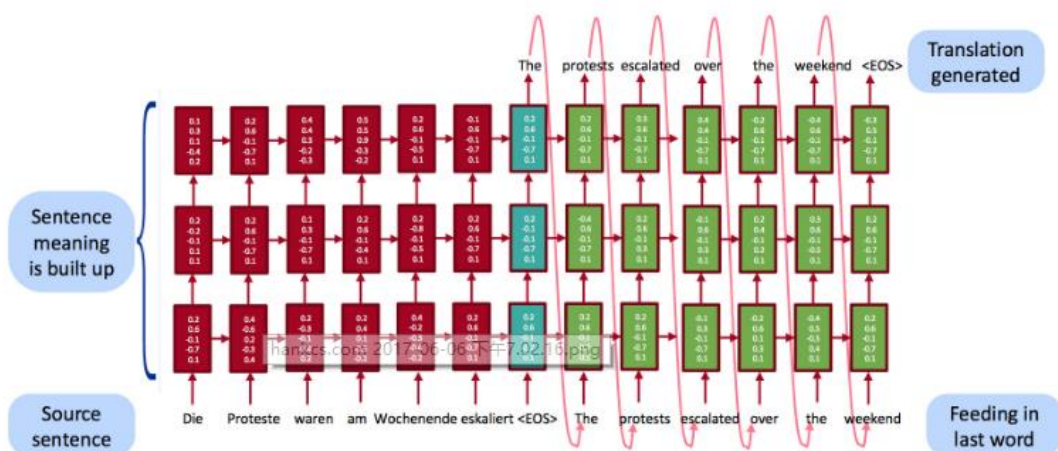


图 1.22 深度学习机器翻译示意图

第 2 课 word2vec 入门

2.1 传统方法处理词语

在开始这一课之前，我们先来看看什么是意思（meaning），我们可以说意思是潜藏在单词或者词组中，也可以说人类通过使用单词或者符号表达了意思，还可以说意思是人类通过某种其他形式表现出来的例如写作，艺术等。语义学上意思就是就是指代一种符号。刚看完上面这段话估计要晕了，谈论这个是为了什么。因为自然语言处理要求计算机理解语言里面的意思，所以我们才讨论了，既然需要计算机理解意思，那么我们应该让计算机从什么地方理解呢？如果说意思潜藏在单词中，是不是让计算机对着单词学习就可以了？这里我们就要开始引出今天的课程，计算机如何处理词语的意思。首先我们先讲传统方法。

2.1.1 Discrete Representation

传统方法是维护一个大的分类系统，例如 WordNet 那样维护了一个词的 **hypernyms**（上位词，是一种概念上更广的主题词，例如“花”是“鲜花”的上位词，“植物”是“花”上位词），以及同义词等。我们可以通过这样的 **Net** 直接查询得到每个单词的上位词或者同义词等。（如图 2.1 所示）

<pre>from nltk.corpus import wordnet as wn panda = wn.synset('panda.n.01') hyper = lambda s: s.hypernyms() list(panda.closure(hyper))</pre>	(here, for good):
<pre>[Synset('procyonid.n.01'), Synset('carnivore.n.01'), Synset('placental.n.01'), Synset('mammal.n.01'), Synset('vertebrate.n.01'), Synset('chordate.n.01'), Synset('animal.n.01'), Synset('organism.n.01'), Synset('living_thing.n.01'), Synset('whole.n.02'), Synset('object.n.01'), Synset('physical_entity.n.01'), Synset('entity.n.01')]</pre>	<pre>S: (adj) full, good S: (adj) estimable, good, honorable, respectable S: (adj) beneficial, good S: (adj) good, just, upright S: (adj) adept, expert, good, practiced, proficient, skillful S: (adj) dear, good, near S: (adj) good, right, ripe ... S: (adv) well, good S: (adv) thoroughly, soundly, good S: (n) good, goodness S: (n) commodity, trade good, good</pre>

图 2.1 其中一种 WordNet 查询示意图

这样一种 Discrete Representation（离散的）的表达方式一定程度上还是损失了一些信息，例如 **adept**, **expert**, **good**, **skillful** 等这些词其实意思还是有些许差别的。并且这种

方法使得新出现的词汇很难以更新加入。由于这种方法需要大量的人力去人工打上标签建造，费时费力还带有主观差别，并且很难去计算词之间的相似度。接下来我们讲述如何计算相似度，为什么难以计算相似度。

大多数的规则学家或者统计 NLP 学家都将单词看成一个 **atomic symbols**（原子符号，意思就是单词是最小的不可分割的带有意思的元素，彼此独立），例如 **hotel**, **conference**, **walk**。（单一的单词），实际上，这些单词在向量空间中就是一个有很多个 0，一个 1 的向量（也就是我们称为 **one-hot** 的向量，视频里还叫做 **localist representation**，个人理解这个向量真的很局部，只有一个地方是 1，也可以理解成，意思很有局限性，所以叫 **local**，**anyway**，每个人可以有不同理解），例如假设语料库一共有 1 万个单词，**hotel** 是第 5 个单词，那么只有第五位置上是 1，其余位置都是 0，**hotel** 的向量为 $[0,0,0,0,1,0,0 \dots]$ ，而向量的长度事实上跟我们的语料库有关，**Google** 维护的 1TB 的语料库甚至有 1300 万的词汇量，如果用 **one-hot** 来代表每一个单词，那么每一个单词就有 1300 万的维度，并且只有一个 1 其余都是 0。

当单词变成向量形式后，我们可以通过向量间的运算得到两个向量的距离，继而反应向量的相似度。这里有两个例子，“**Dell notebook battery size**”和“**Dell laptop battery capacity**”两个句子意思是很接近的，“**Seattle motel**”和“**Seattle hotel**”意思也非常接近，但是两个向量的点积却为 0，如图 2.2 所示，我们无法根据两个向量的点积判断是否相似（因为所有向量相互点积都为 0）。由此我们需要另外一种编码方式。

$$\begin{array}{l} \text{motel } [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]^T \\ \text{hotel } [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T = 0 \end{array}$$

图 2.2 motel 和 hotel 向量点积

2.2 Distributional representations

视频里给了 J.R.Firth 的一句话，“**You shall know a word by the company it keeps**”，意思可以理解成，由于单词不是单独存在，是依靠上下文的，而上下文的单词又会相互关联，所以一个单词最好的理解方式就是放在句子中，反过来我们也可以从句子中提取出一个单词的意思。这也是我们现在用的思想，通过邻居词的意思来生成对应词的向量表达。如图 2.3 所示，**banking** 周围的词反过来可以提取出 **banking** 的意思，如果我们将 **banking** 从句子中挖掉，那么很大概率我们可以根据上下文将 **banking** 填进去。

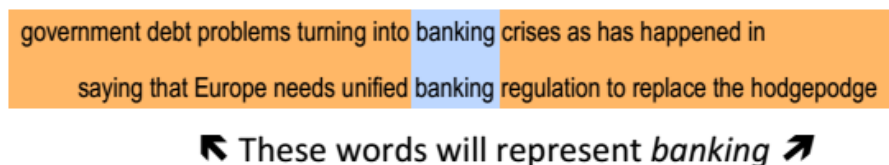


图 2.3 Distributional representations 理论基础示意图

我们可以通过两种手段来训练我们的向量，一种是根据词的上下文单词来预测这个词（CBOW 模型），一种是根据词来预测该词上下文的单词（SG 模型）。我们根据这两种模型会得到一些 dense vector（与 one-hot 比较而言是 dense 的），长相可能如图 2.4 所示。

$$\text{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

图 2.4 linguistic 的一种可能的向量形式

无论是哪一种模型，我们最终都是要预测 center word（中心词 w_t ）和 context words（上下文词）之间的关系。

表达式

$$p(\text{context}|w_t) \text{ or } p(w_t|\text{context})$$

前者是根据中心词计算上下文词的概率，后者是根据上下文词计算中心词的概率。

损失函数

$$J = 1 - p(w_{-t}|w_t)$$

其中这里的 w_{-t} 表示除去 t 的上下文，如果模型效果很好，那么得到的上下文跟中心词的联系就非常大，那么预测出来的概率 p 接近 1，从而 J 的值接近 0。所以我们的任务就是在一个特别大的语料中选取数据进行训练，然后使得损失函数越来越小即可。

虽然以前就有类似的研究，但是一直没有引起足够的重视。研究有：

Learning representations by back-propagating errors (Rumelhart et al., 1986)

A neural probabilistic language model (Bengio et al., 2003)

NLP (almost) from Scratch (Collobert & Weston, 2008)

A recent, even simpler and faster model: word2vec (Mikolov et al. 2013)

我们使用的训练模型主要是两种 Skip-grams (SG) 和 Continuous Bag of Words (CBOW)，主要的训练方法有 Hierarchical softmax (多层次 softmax) 和 Negative sampling (负样本抽样)，但是这门课主要讲 Naïve softmax (就是最简单的 softmax)。

2.2.1 Skip-gram prediction

首先我们引入视频中的图片来简单介绍 SG 的原理，如图 2.5 所示。

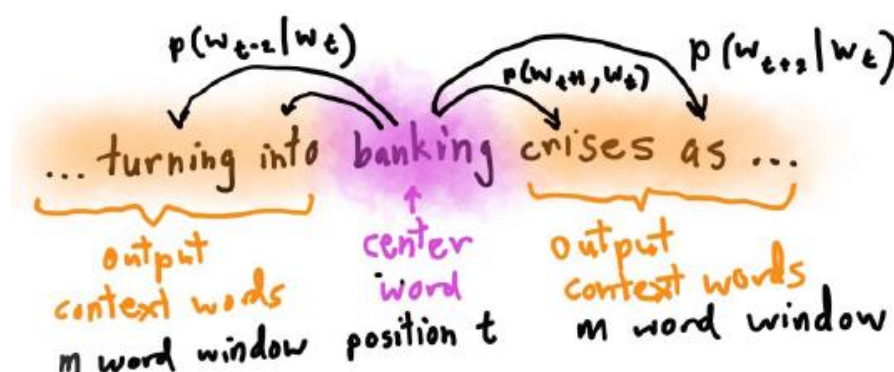


图 2.5 SG 原理介绍

SG 的原理就是通过中心词 (banking) 来预测中心词周围的词 (turning into crises as) 的概率。这里我们需要简单介绍几个概念，首先是 window，window 是超参数，不是机器学习的，是我们人为一开始指定的，图中 window=2，也就是考虑中心词左边两个和右边两个加上中心词共 $2*2+1=5$ 个词，其中中心词作为输入，输出是周围的四个词。由于我们是词袋模型，所以我们不考虑词的具体次序 (比如谁在前面，谁在后面)，只要是周围的我们都一视同仁。

实现细节上，我们目标是最大化概率函数，该概率函数代表了所有位置上的中心词预测周围词的乘积最大。

$$J(\theta) = \prod_{t=1}^T \prod_{-m \leq j \leq m \text{ and } j \neq 0} p(w_{t+j} | w_t; \theta)$$

该公式很好理解，首先第一个累乘是针对中心词的，第二个是对于每个中心词，我们累

乘他对周围词预测的概率。一般对于累乘，我们习惯加上一个对数，由于这里是要求目标函数最大值，且目标函数取值范围在(0,1),所以我们选取负对数作为损失函数

$$\begin{aligned} J'(\theta) &= -\frac{1}{T} \log(J(\theta)) = -\frac{1}{T} \log \left(\prod_{t=1}^T \prod_{-m \leq j \leq m \text{ and } j \neq 0} p(w_{t+j}|w_t; \theta) \right) \\ &= -\frac{1}{T} \sum_{t=1}^T \log \left(\prod_{-m \leq j \leq m \text{ and } j \neq 0} p(w_{t+j}|w_t; \theta) \right) \\ &= -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m \text{ and } j \neq 0} \log(p(w_{t+j}|w_t; \theta)) \end{aligned}$$

其中 θ 是模型的参数，后期我们会训练更新。

这里简单提及一下，其实 Loss function=Cost function=Objective function，这些专业术语是一样的意思。并且常用的损失函数为交叉熵形式（交叉熵其实有很多类似 $p \cdot \log(p)$ 的项组合而成，每一种情况的概率 p 乘上 $\log(p)$ 的值，具体的后面会继续阐述）。

上面我们公式得到了损失函数，其中最重要的就是里面的 $p(w_{t+j}|w_t; \theta)$ 到底等于多少。这里我们需要引入一个 softmax 函数，softmax 函数是从实数空间到概率分布的标准映射方式，通过指数的形式将实数映射成正数，然后通过分母的和进行归一化。Softmax 公式如下

$$p_i = \frac{e^{p_i}}{\sum_{t=1}^N e^{p_t}}$$

一个简单的例子，假设我们有一组数(0.1, 0.3, -0.4, -0.1, 0.5)，那么他们的以 e 为底的幂是

$$\begin{aligned} e^{0.1} &= 1.10517, & e^{0.3} &= 1.34986, & e^{-0.4} &= 0.67032, \\ e^{-0.1} &= 0.904837, & e^{0.5} &= 1.64872 \end{aligned}$$

对应的概率为(0.19461, 0.237697, 0.118037, 0.159333, 0.290324)。这里我们可以看到，由于 softmax 对比较大的数会比较友好，因为根据 e 为底，大的数会扩大更多，这样最后的概率也会越大，而小的数虽然也会放大，但是放大的比例没有大数的多，所以导致小数最后的概率实际上会变小（是对比不用 softmax 直接求和算概率的情况），所以某种程度上 softmax 具有选出最大的，将小数的影响削弱的效果，所以这也是 softmax 名称的由来。

接下来我们回到原来的问题，如何计算 $p(w_{t+j}|w_t; \theta)$ ，这里我们先直接给出公式

$$p(w_j|w_t) = \frac{e^{(w_j^T w_t)}}{\sum_{j=1}^V e^{(w_j^T w_t)}}$$

其中 w_j 是输出词（也即中心词附近的词）， w_t 是中心词，这里是通过中心词得到周围词的概率分布，看到这里这么多公式可能你已经晕了，我们现在就用一个实例来讲解一遍。如图 2.6 所示。我们接下来根据这个图开始讲解。我们根据图中上面标注的 $V \times 1$ ，或者 $d \times v$ 来进行阶段定位。（例如我下面说，现在进入到 $d \times v$ 阶段）

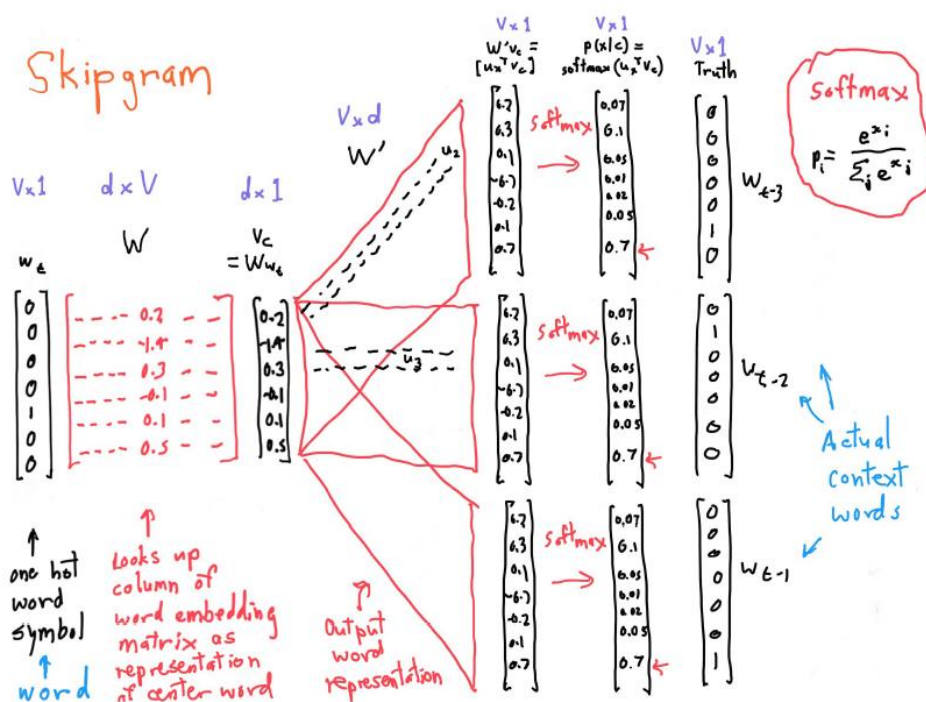


图 2.6 SG 模型细节示意图

首先我们需要选定一个中心词，假设现在有一个例句，I love china very much，首先我们转成 one-hot 向量，那么比如 love 就是 $x=[0, 1, 0, 0, 0]$ ，之后我们系统有一个 W 矩阵（其实这个句子就是我们未来的词向量矩阵），我们可以观察一下，首先我们需要执行 $W \times x$ ，我们仔细观察是不是可以发现，其实 x 相当于就是将 W 的第二列取出来。这里我们再举一个简单的例子，假设有

$$\begin{pmatrix} 1 & 4 & 9 \\ 2 & 5 & 8 \\ 3 & 6 & 7 \end{pmatrix} * (0,0,1)^T = (9, 8, 7)^T$$

$$\begin{pmatrix} 1 & 4 & 9 \\ 2 & 5 & 8 \\ 3 & 6 & 7 \end{pmatrix} * (0,1,0)^T = (4, 5, 6)^T$$

$$\begin{pmatrix} 1 & 4 & 9 \\ 2 & 5 & 8 \\ 3 & 6 & 7 \end{pmatrix} * (1,0,0)^T = (1,2,3)^T$$

观察可以发现，第几个元素是 1，相当于就是将原来矩阵里面第几列取出来（后面我们会直接称 W 这个矩阵为 **look up** 矩阵，因为对于一个 **one-hot** 向量，我们可以根据 1 的位置直接将矩阵里面某一列全部取出来而不必再矩阵运算，所以是不是可以理解成 W 矩阵其实存储的就是所有单词的向量，例如如果有 1000 个单词，每个单词向量是 300 维，那么 W 矩阵就是 $300*1000$ 的，矩阵第一列就是第一个单词的词向量，第二列就是第二个单词的词向量，也就是说 W 矩阵其实就是词向量矩阵，注意 W 矩阵会在迭代中一直更新里面的参数，相当于每个单词的向量形式一直在进行更新）。现在我们将 **one-hot** 跟矩阵相乘得到矩阵某一列后记为 v （这一列其实就是这个单词的向量形式）就进入图 2.6 中仔细观察 $d*1$ 那一列了。之后我们还需要乘以一个 w' 矩阵（该矩阵也会一起更新），有了刚才 W 矩阵其实就是词向量矩阵的理解，这里我们还需要理解 w' 其实里面装的也是所有单词的向量，只不过这个向量我们成为上下文向量，相当于每一个单词都有两个向量。我们经过第一步得到 v 向量之后，还需要跟 w' 相乘，根据矩阵乘法得到了图 2.6 中第二个 $v*1$ 的阶段。这里我们可以这样理解，首先两个向量点积得到的是一个具体的数值，我们假设单词 a 的向量*单词 b 的上下文向量得到的值是 b 是 a 的上下文的可能性（需要经过 **softmax** 才是概率），比如单词 **aware** 的向量乘上 **of** 的上下文向量=2，意思就是 **aware** 的上下文是 **of** 的可能大小是 2（如果还有其他数据，那么经过 **softmax** 之后，2 可能变为 0.5，这时候意思就是 **aware** 的上下文是 **of** 的概率是 0.5）。那么对于每一个单词我们需要计算出它跟语料库里面所有单词是上下文的概率，所以我们就需要将每一个单词跟其他所有单词的上下文向量进行相乘得到一个结果，为了方便我们将其他所有单词的上下文向量放在一个矩阵里，这样矩阵运算得到一个一维数组，一维数组里面每一个数据就代表了单词跟某一个单词是上下文的的可能性，然后将该数组进行 **softmax** 就得到了是上下文的概率。这时候就进入到图 2.6 中第三个 $v*1$ 的阶段。这时候当有了概率之后，假设 **window** 选取为 2，也就是要选出周围四个单词，那么我们从概率中选出最大的 4 个即可，然后只需要跟原文判断，这 4 个是否是正确答案，然后进行迭代更新即可。最后更新完成后，我们就得到两个矩阵，一个 W 保存了每个单词的向量形式， w' 保持了每个单词的上下文向量形式，一般情况下我们直接选取 W 矩阵的内容即可，有些时候有些研究者会将两个矩阵中一个单词的两种向量形式叠加/2 当做每一个单词的向量形式。

现在我们讲解了模型的正向传播，接下来我们要开始逆向推导求导数，这样才可以进行更新。我们一步步来进行求导。

$$\begin{aligned}
\frac{\partial \log(p(w_j|w_t))}{\partial w_t} &= \frac{\partial \log\left(\frac{e^{(w_j^T w_t)}}{\sum_{j=1}^v e^{(w_j^T w_t)}}\right)}{\partial w_t} = \frac{\partial \log(e^{(w_j^T w_t)})}{\partial w_t} - \frac{\partial \log\left(\sum_{j=1}^v e^{(w_j^T w_t)}\right)}{\partial w_t} \\
&= \frac{\partial w_j^T w_t}{\partial w_t} - \frac{1}{\sum_{j=1}^v e^{(w_j^T w_t)}} \frac{\partial \sum_{j=1}^v e^{(w_j^T w_t)}}{\partial w_t} \\
&= w_j - \frac{1}{\sum_{j=1}^v e^{(w_j^T w_t)}} \frac{\sum_{j=1}^v \partial e^{(w_j^T w_t)}}{\partial w_t} \\
&= w_j - \frac{1}{\sum_{j=1}^v e^{(w_j^T w_t)}} \frac{\sum_{j=1}^v e^{(w_j^T w_t)} \partial w_j^T w_t}{\partial w_t} = w_j - \frac{\sum_{j=1}^v e^{(w_j^T w_t)} w_j}{\sum_{j=1}^v e^{(w_j^T w_t)}} \\
&= w_j - \sum_{j=1}^v \frac{e^{(w_j^T w_t)}}{\sum_{j=1}^v e^{(w_j^T w_t)}} w_j = w_j - \sum_{j=1}^v p(w_j|w_t) w_j
\end{aligned}$$

当我们计算出梯度之后，就可以根据梯度上升或者梯度下降的方法继续更新了，如图 2.7 所示，图 2.8 所示

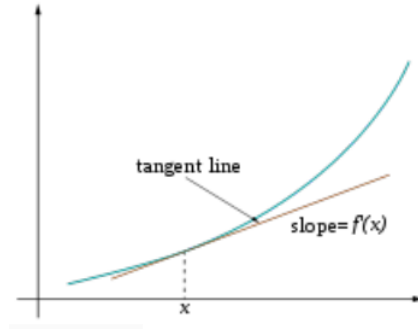


图 2.7 梯度示意图

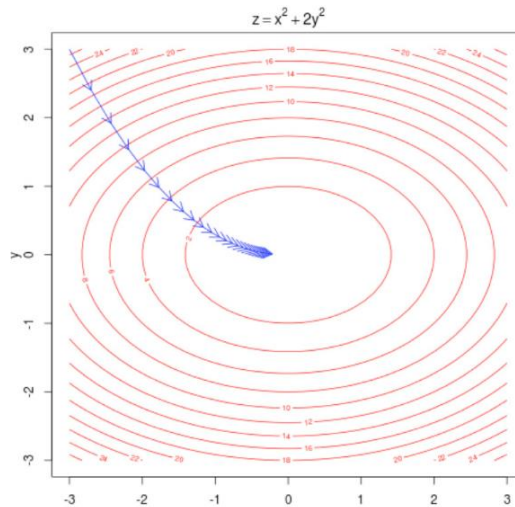


图 2.8 通过梯度寻找极值示意图

第 3 课 word2vec 高级 GloVe

3.1 传统 SG 模型的缺点以及解决方案

这一节课我们首先回顾一下上节课讲到的 SG 模型的缺点以及一些改进的办法，之后我们通过比较其他生成词向量的方法的优缺点，最终我们提出结合了 SG 方法优点以及其他生成词向量方法优点的 GloVe 方法。

3.1.1 传统 SG 模型的缺点

首先我们需要回顾一下上一节课的 SG 模型内容，SG 模型虽然也会遍历整个语料库，并且对每个单词基于一定大小的窗口进行迭代计算，通过 softmax 函数计算概率（如图 3.1 所示），然后针对每个窗口使用梯度下降更新模型参数。我们观察图中的分母可以注意到，每次我们都需要将语料中所有的单词计算一遍这样才可以累加得到分母，当我们的单词大小达到数百万的时候这个代价会非常大。

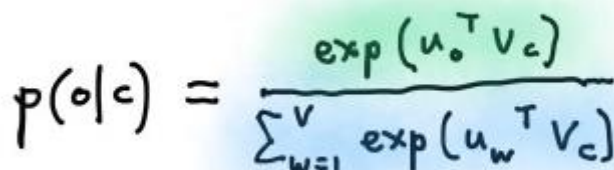

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

图 3.1 SG 模型 softmax 函数示意图

传统 SG 模型还有一个缺点。假设窗口大小为 m ，那么每次最多只有 $2m+1$ 个单词（每一个单词的左边 m 个加上右边 m 个以及它自身），这就使得我们的参数更新矩阵 $\nabla_{\theta} J_t(\theta)$ 特别稀疏。如图 3.2 所示，只有 $2m+1$ 个位置不为 0，其余位置均为 0（计算梯度的时候我们是对整个参数矩阵进行运算，因为其余位置这次迭代不更新，所以结果为 0）。

$$\nabla_{\theta} J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

图 3.2 SG 参数更新矩阵示意图

由于每一次我们都只会更新这次迭代出现的向量，所以要么你就维持一个大型的稀疏矩阵用来只更新 \mathbf{w} 和 \mathbf{w}' 中特定的某一行，或者给每个单词一个哈希映射（笔者上网查资料也搞不懂哈希如何解决这个问题）。

3.1.2 Negative Sampling

为了解决上节我们提到的 softmax 计算困难以及反向传播的时候更新矩阵过于稀疏，这里我们提出一个 Negative Sampling（负采样）的方法。那什么是负采样呢？首先我们需要介绍什么是负样本，例如对于 SG 模型，假设我们有语句“I love China very much”，窗口大小=1，那么对于“China”来说，他的上下文词有“love”和“very”，那么我们称（“China”，“love”）和（“China”，“very”）为正样本（某种程度上也即该 pair of words 是我们需要的输入和输出形式，也即中心词和上下文词的组合），那么类似（“China”，“I”）或者（“China”，“much”）就是负样本，这些样本是不满足我们的中心词和上下文词的组合的。负采样相当于就是采样出一些“噪音”。我们继续观察可以看出，其实负采样重点就是采样出不是中心词的上下文词即可（有了单词，我们直接跟中心词组合成一个 pair 即可）。基于一个单词如果在文本中出现的次数越多，那么他就有更大的概率被采样得到，但是我们不希望出现次数特别小的词被采样的概率太小，因为这样可能使得有些稀有的词总是采样不出来。所以我们使用下述的公式来平滑一下采样的概率。

$$p(w_i) = \frac{counter(w_i)^{\frac{3}{4}}}{\sum_{t=1}^T counter(w_t)^{\frac{3}{4}}}$$

其中 $3/4$ 是一个经验值，并不是根据什么数学理论计算出来的。其中 $counter(w_i)$ 是指单词 w_i 在语料库中出现的次数。有了上述概率，我们每次可以直接从 0~1 生成一个数，看这

个数落在哪个区间就采样出哪个数（注意如果采样到了正样本，那么跳过再采样即可）。

3.1.3 改进后的 SG

有了负采样之后，我们的核心思想是，每一次计算中心词和上下文词的得分，再加上一些噪声。（也即计算中心词和负采样得到的词的得分）

$$\mathcal{J}(\theta) = \frac{1}{T} \sum_{t=1}^T \mathcal{J}_t(\theta)$$

基于这样的思路我们使用 sigmoid 函数替换 softmax 函数。其中 sigmoid 函数公式如下

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid 函数的函数图像如图 3.3 所示

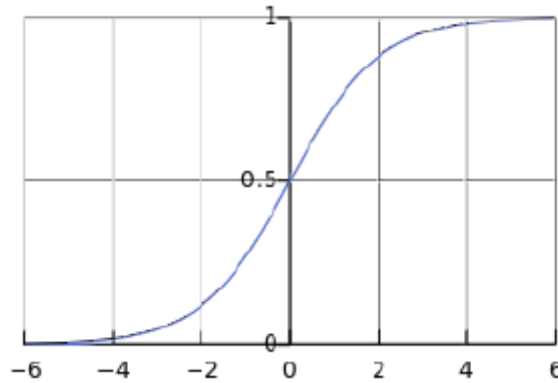


图 3.3 Sigmoid 函数图像示意图

我们可以看到 sigmoid 函数的值域是 (0,1)，我们可以理解成将输入改换成概率，当输入越大的时候，输出的概率越接近 1，当输入越小的时候，输出的概率越接近 0。

我们重新得到 loss function 为

$$\mathcal{J}_t(\theta) = \log(\sigma(u_o^T v_c)) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log(\sigma(-u_j^T v_c))]$$

其中前者为中心词和上下文词的得分，第二部分为中心词和负采样词的得分，那么为什么第二个部分里面有一个‘-’号，那是因为我们希望使得整个 loss function 最大，我们就不仅仅要使得第一部分变大，我们也要使得第二部分变大，但是第二部分，我们希望中心词和负采样词偏离越大越好，也即两个向量点积越偏向于-1 越好，所以加上符号，使得两个向量越

偏离结果越大。

3.2 co-occurrence 方法

现在我们介绍一下其他的生成词向量的方法。首先我们需要引入一个 **cooccurrence**（共现）的概念，如果一个单词出现，另一个单词也出现，那么那就称为共现。对于语料库我们直接将每一个单词的 **window size** 里面的单词记录下共现的次数组成一个共现矩阵。考虑到 SG 模型其实也是考虑到共现的情况的，因为每一次更新的时候都会使用到中心词周围的词，但是这种共现是局部的，只针对每次更新的窗口内的，如果将全局的共现情况考虑进去呢？

（例如‘I’和‘love’全局一共共同出现 5 次，但是每次 SG 更新的时候，在特定的窗口内‘I’和‘love’只共现了 1 次，所以 SG 在单次更新的时候并没有考虑到全局的情况）

其实很早以前就有类似的基于共现矩阵的研究。区别在于统计的方式不同，如果在窗口级别上统计的话，可以得到相似的词，但是如果在文档级别上的话，可以得到相似的文章 LSA。（Latent Semantic Analysis 潜在语义分析）

现在我们举一个基于窗口（**window=1**）统计共现矩阵的例子，如图 3.4 所示

- I like deep learning.
- I like NLP.
- I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

图 3.4 基于窗口共现矩阵统计例子

由于‘like’在‘I’右边出现两次，所以第一行第二列的值=2，由于‘deep’在‘like’周围出现一次，所以第二行第四列=1。（注意图中是窗口=1 的情况，通常情况下会在 5~10 左右，窗口图中我们是选取中心词左边与右边都看的情况，有些时候存在只看左边或者只看右边的情况）。

简单的共现矩阵很明显存在如下问题，首先维度太高，如果有 n 个单词那么每个单词就是 n 维的，并且共现矩阵大小为 $n*n$ ，而里面绝大部分都是 0，这又导致了稀疏性问题。同

时模型鲁棒性较低,当有新的词进来之后,我们还需要重新计算一遍并且改变所有词的维度。

我们可以通过降低向量维度的办法来解决这个问题。(直观上我们可以这样理解,由于原来的向量特别稀疏,所以我们可以使用某些压缩的办法将保留向量绝大部分信息的基础上压缩到低维度,这样低维度的向量就是一种饱含价值的向量, **dense vector**, 这个词也直观的描述了低维后的向量)。

向量的降维我们常用的有 PCA (Principal Component Analysis, 主成分分析), 以及 SVD。

(Singular Value Decomposition 奇异值分解, 如图 3.5)

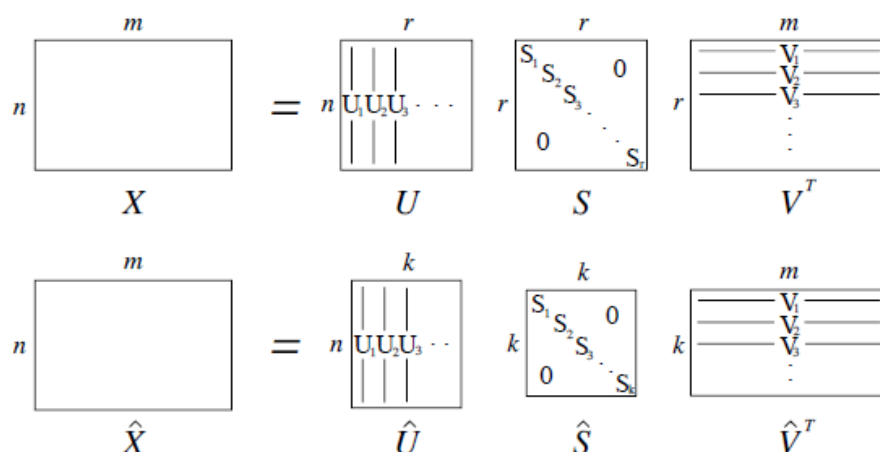


图 3.5 SVD 过程示意图

这里我们不会详细描述这两个算法实现细节。简而言之通过降维方法后,虽然我们损失了一定的精确度,但是我们将向量最主要的信息都保留在了低维的向量中,我们可以直接使用这个低维向量来当做我们 **word2vec** 的结果。

但是传统的 **co-occurrence** 方法存在一些问题,例如有一些 **stop words** 停用词(**the, he, has**) 在文本中出现的次数太多,这就导致共现矩阵中这样的词对应的数值也会较大,并且我们没有考虑离中心词较远的词跟较近的词要区别对待,同时 **SVD** 伴随矩阵分解计算量会随着单词数量的增加而增加,例如对于 $n \times m$ 的矩阵复杂度是 $O(mn^2)$, 并且 **SVD** 也不方便处理新词或者新文档。解决办法如下:

①我们要么采取忽略他们的做法,要么定一个阈值,例如当共现次数超过 100 的时候也记做 100 即可。

②我们可以根据单词与中间词的距离衰减权重,例如距离中间词最近的词共现矩阵对应位置+1, 离中间词距离为 2 的词共现矩阵对应位置记录 0.5, 离中间词距离为 n 的 (前提是 $n < \text{window}$) 共现矩阵对应位置记录 $\frac{1}{2^{n-1}}$ 。衰减权重的方法因人而异,这里是笔者举例的方法。

③还可以选用 **Pearson Correlations**（皮尔逊相关系数，[这个笔者不懂](#)）代替词频，将负数设置为 0。

措施虽然简单，但是效果还不错，视频中给了一些向量举例图，如图 3.6。

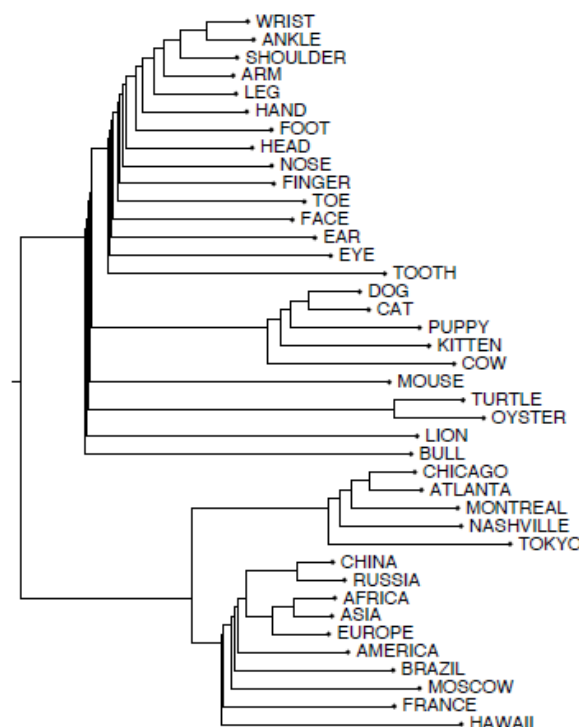


图 3.6 高维空间向量距离远近可视图

3.3 GloVe

我们现在学习了基于统计的方法，以及基于预测的方法。其中基于统计的方法（LSA，HAL，COALS，Hellinger-Pca，笔者对这几个方法也不是很了解，大概就是用 3.2 那样的统计共现矩阵然后降维的思路），这一类的基于统计的方法有以下优点：

- ① 训练速度非常快，过一遍文档统计一下然后直接进行降维操作即可。
- ② 很有效的使用到了全局的统计数据。

但是其主要目的是用来捕捉单词相似度并且很难被扩展到大规模的语料，毕竟大规模矩阵的降维操作会异常复杂。

基于预测的方法主要有（NNLM，HLBL，RNN，Skip-gram，CBOW），这些方法主要就是通过中心词预测上下文或者根据上下文预测中心词这样的思路。这一类的基于预测的方法有以下优点：

在其他任务上可以有更好的表现（即指生成的向量更好）

不仅仅可以捕捉到单词相似性还可以计算一些更加复杂的模式

但是基于预测的方法都会使用到窗口进行滑动，窗口需要遍历整个数据集，并且由于窗口是局部的，所以欠缺考虑全局信息。有没有一种方法可以将两者的优点结合起来呢？答案是有的，这就是 GloVe 模型。我们这里直接给出 GloVe 模型的公式以及具体含义，关于 GloVe 模型如何得来就不在这里做非常详细的解释。

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^w f(P_{ij})(u_i^T v_j - \log(P_{ij}))^2$$

其中假设共现矩阵 C 为 $n \times n$ 的，那么有

$$P_{ij} = \frac{C_{ij}}{\sum_{d=1}^n C_{id}}$$

意思就是 P_{ij} 相当于将矩阵第 i 行第 j 列的值除以矩阵第 i 行所有元素的和，其实矩阵第 i 行代表的就是第 i 个单词的跟其他所有单词共现的情况，这个概率算出来的是当 i 单词出现之后， j 单词有多大的概率跟 i 单词共现。前面的 f 函数相当于一个 \max 函数，当指大于多少之后就固定为 1，函数图像如图 3.7 所示

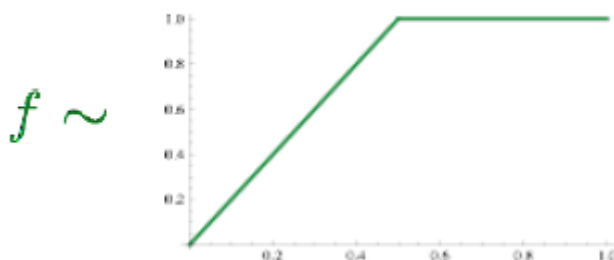


图 3.7 GloVe 的 f 函数

看 GloVe 的代价函数我们就可以注意到有两个向量 U 和 V ，这个有点类似于我们 SG 和 CBOW 中的一个单词具有词向量和上下文向量两种形式，这样的类比虽然不准确但是也有助于我们理解 GloVe 中的两个需要更新的 U 和 V 矩阵。这里可能就会问，既然两个向量都会进行更新，那么哪一个才是我们最终得到的单词的向量，在 SG 和 CBOW 中我们通常取第一个词向量的结果作为最终的单词词向量而不是使用上下文向量，但是这里我们一般是将两个矩阵加起来作为最终得到的单词的向量。

$$X_{final} = U + V$$

GloVe 函数并没有用到神经网络来进行训练，但是其具有如下优点

- ① 训练特别快
- ② 对特别大的语料库也可以适应
- ③ 即便对小的语料库或者小向量也有很好的表现

3.4 词向量评估方法

根据前面的内容我们可以根据 SG 或者 GloVe 模型生成对应的词向量，现在我们就来讲词向量的评估方法。这里我们引入一个示例，假设我们要设计一个问答系统，首先第一步我们肯定预处理训练词向量，然后以得到的词向量为基础继续训练我们的问答系统神经网络。我们需要知道一个好的词向量会使得我们最终训练出来的问答系统有更好的效果，但是很多时候假设我们手头上有两份词向量的表格，我们到底用哪一份词向量表格进行训练呢？一个答案是为什么不两个都训练？由于很多时候我们的问答神经网络可能具有非常多的节点，以及有非常大量的训练集，很有可能跑一次就需要十天半个月，两份都跑一遍需要特别长的时间。而且很多时候我们得到的词向量表格又不止一份，全部训练一遍不现实。这时候我们希望有一些方法来提前测试一下我们的词向量是否效果较好，可以使用一些简单的任务（下文会提到）来进行测试（这种方法就叫做内部任务评价），假设我们的问答系统模型是没有问题的，那么最终我们可以观察训练出来的问答模型的效果来反应原本使用的词向量的效果，例如 A 词向量表训练出来的问答系统效果明显高于 B 词向量表训练出来的，那么我们就可以说 A 词向量表比 B 词向量表效果更好。（这种方法就叫做外部任务评价）

3.4.1 外部任务评价

外部任务评价其实就是将词向量直接使用到实际任务中，看实际任务的效果进行评价。这些任务通常会比较复杂，而且他们的计算量可能较大。例如我们上面提到的任务，在预训练了词向量之后我们直接使用词向量来进行问答系统的训练，（这里假设我们问答系统的模型是正确且高效的）通过最终训练出的问答系统的效果来反推回词向量的效果的就是外部任务评价。但是实际上，很多时候我们并不知道我们的外部模型是否高效，所以很多时候优化外部系统的时候，我们并不知道是哪一个子系统出了问题，例如到底是底层的词向量不够好还是我们的问答模型不够好，所以一般这个时候我们还需要进一步进行内部任务评价。外部

任务评价的特点如下：

- ① 在一个实际任务中进行评价
- ② 可能需要很长的时间才能计算精确度
- ③ 当出现问题的时候，不清楚到底是具体哪一个子系统的问题亦或是子系统间相互作用引起的问题
- ④ 如果我们用另一种模型替换掉原来模型里面的一个子模型发现准确率提高，那么很有可能替换掉的子模型就是有问题的子模型。

我们都知道大多数的 NLP 外部任务都可以被描述成分类任务，例如给一个句子，我们进行情感分类等。类似的在命名实体识别中，我们需要找出上下文的中心词所属的类别。例如输入为“Jim bought 300 shares of Acme Corp. in 2006”，我们期望分类完成后的输出是，“”[Jim]人名 bought 300 shares of [Acme Corp.]机构名 in [2006]时间。”而视频中就举了一个命名实体识别的外部任务评价结果图。（关于外部任务中命名实体识别我们就不做过多叙述，视频中给了图 3.8 作为命名实体识别结果的例子）

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	88.7	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe		93.2	82.9	82.2

图 3.8 外部任务评价命名实体识别示例

3.4.2 内部任务评价

相对于外部任务评价需要应用到实际中，往往周期较长而且涉及因素较多，内部任务评价显得更快，但是内部任务评价较高的不一定实际效果真的好。视频中说曾经有人花了几年时间提高了生成向量的内部评价任务的得分，但是在实际任务中并没有体现出向量的好处。就像我们前面那个例子说的，我们不知道到底应该使用哪个向量表来训练我们的问答系统，这时候我们可以使用一些简单的内部任务对我们的向量表进行测试，看看哪些表对内部任务

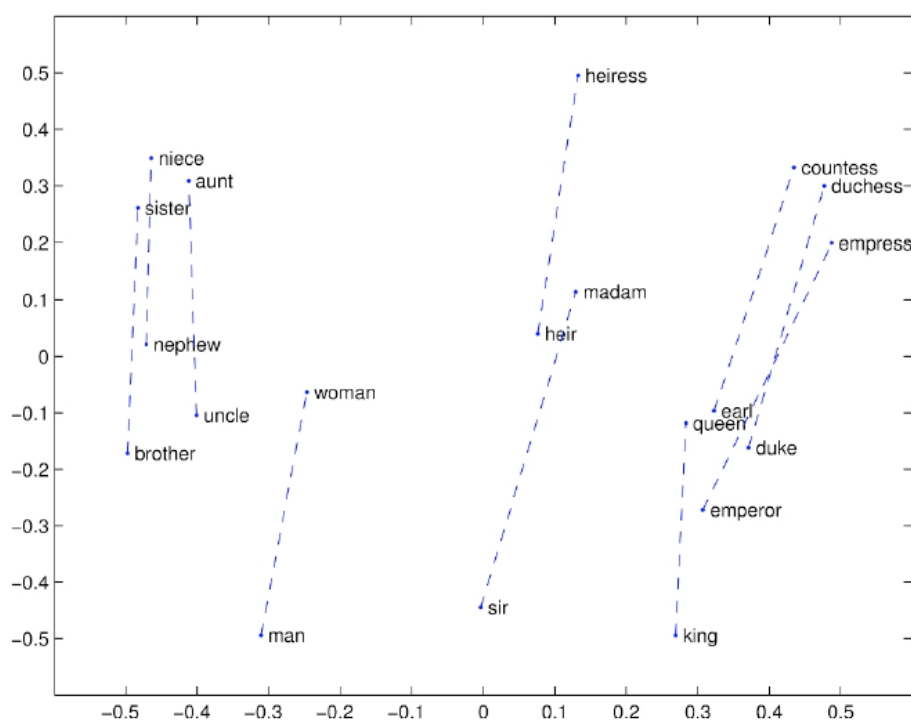
的表现更好，虽然说并不一定内部任务表现好的实际效果一定好，不过毕竟实际任务影响因素太多，如果我们词向量这一块就选择了表现效果特别不好的词向量，那么对于模型后期的修改应该还是会带来一定困扰的。内部任务评价的特点有：

- ① 内部任务通常是在一个特定的子任务中进行评估
- ② 计算速度特别快
- ③ 可以帮助我们理解系统（??? 笔者这里看不懂）
- ④ 并不一定内部任务表现好的外部任务表现就一定好，还是需要看实际效果

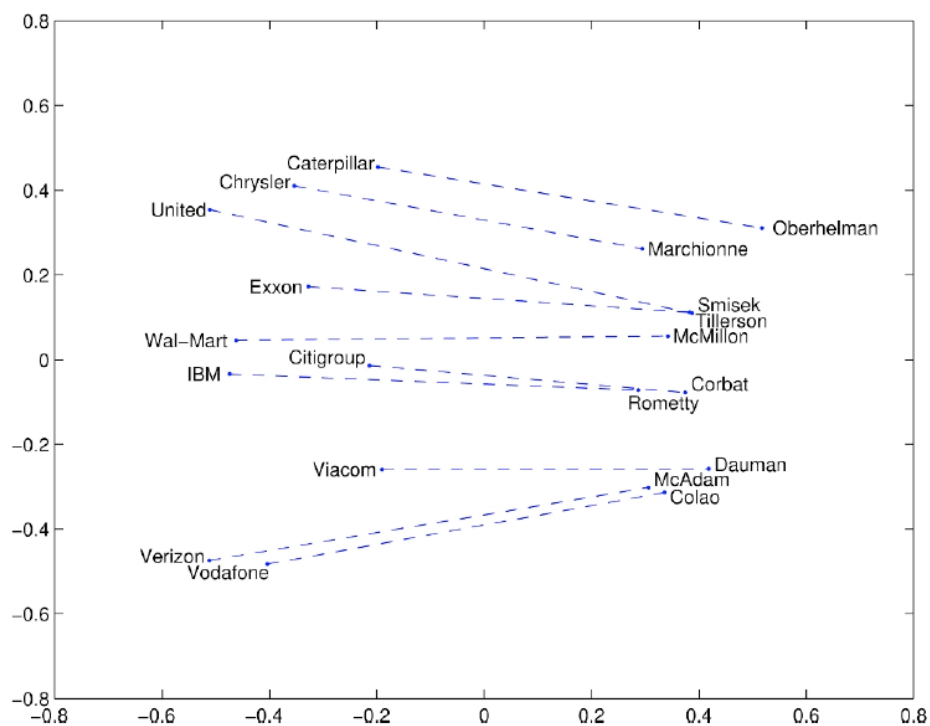
常见的内部评价任务有词向量类比，这类任务具有如下形式，例如“男人-女人=国王-?” 我们希望这里的答案是女王，也就是“A 对于 B 来讲就相当于 C 对于哪个词? ”。对于 a:b->c:? 我们希望通过向量的余弦相似度来进行比较。

$$d = \underset{i}{\operatorname{argmax}} \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

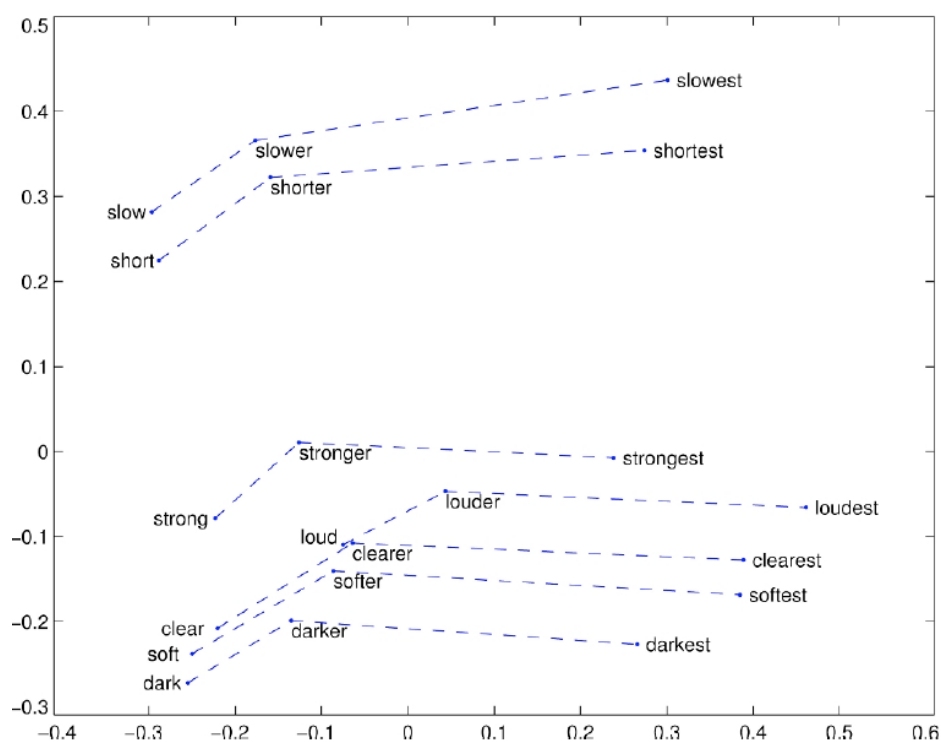
公式的意思是很明显的，最好情况下我们希望 $x_b - x_a = x_c - x_i$ ，那么有 $x_b - x_a + x_c = x_i$ ，在通常情况下等式一般是不成立的，但是等式左边的向量与等式右边的向量夹角应该较小，所以我们希望使得他们的余弦相似度越大，带入余弦计算公式就有了上述公式了。我们使用这个内部任务可以得到很多有趣的结果。（如图 3.9 所示）其他一些有趣的类比结果就不在本文中展示了。



(a) 男人-女人类比模式结果图



(b) 公司-CEO 类比模式结果图



(c) 形容词高级最高级类比模式结果图

<i>Expression</i>	<i>Nearest token</i>
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

(d) 国家-首都类比模式结果图

图 3.9 内部任务一些有趣的结果

但是内部任务有时候也需要谨慎，要考虑到训练文集的各个方面，例如如图 3.10 所示的类比，由于美国有很多不同的城市有相同的名字，例如美国有很多个地方叫 Phoenix，所以图中的 Arizona 不一定是正确答案。

```

: city-in-state
Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

```

图 3.10 City-in-State 类比图

还有一种情况是例如国家的首都其实受惠改变的，例如 1997 年之前的 Kazakhstan 的首都是 Almaty，之后才变成 Astana，所以如果训练的语料比较陈旧就容易出现这样的问题。如图 3.11 所示

: capital-world
 Abuja Nigeria Accra Ghana
 Abuja Nigeria Algiers Algeria
 Abuja Nigeria Amman Jordan
 Abuja Nigeria Ankara Turkey
 Abuja Nigeria Antananarivo Madagascar
 Abuja Nigeria Apia Samoa
 Abuja Nigeria Ashgabat Turkmenistan
 Abuja Nigeria Asmara Eritrea
 Abuja Nigeria Astana Kazakhstan

图 3.11 Capital-World 类比图

视频最后还提到了一个内部任务的例子，就是先通过人为的对两个词的相似度进行打分（如图 3.12），图中是 0~10，然后再与两个词的向量距离进行比较，然后用该数据集对一些模型进行测试。主要方法就是通过对某个单词相似度进行排序后得到最相关的词语，然后就可以量化评测。比如说人为打分发现 A 词最相似的是 B 词，第二相似的是 C 词，然后根据词向量中 A 向量距离最近的向量与距离第二近的向量等方法来进行量化评测。

Word 1 Word 2 Human (mean)

tiger cat 7.35
 tiger tiger 10.00
 book paper 7.46
 computer internet 7.58
 plane car 5.77
 professor doctor 6.62
 stock phone 1.62
 stock CD 1.31
 stock jaguar 0.92

(a) 人为打分情况

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW [†]	6B	57.2	65.6	68.2	57.0	32.5
SG [†]	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	75.9	83.6	82.9	59.6	47.8
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

(b) 部分模型测试结果

3.12 另一种内部任务示意图

3.4.3 内部任务调参

我们使用内部任务的时候可能会需要考虑一些超参数，例如词向量的维度，语料库的大小，窗口的大小等问题，我们现在就来讨论一些词向量生成技术中（例如 SG 和 GloVe）中可能用到的调参细节，我们先看一些同等条件下，词向量生成方法的表现，如图 3.13 所示。

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	<u>67.5</u>	<u>54.3</u>	<u>60.3</u>
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	<u>64.8</u>	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	<u>80.8</u>	61.5	<u>70.3</u>
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW [†]	300	6B	63.6	<u>67.4</u>	65.7
SG [†]	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	<u>81.9</u>	<u>69.3</u>	<u>75.0</u>

图 3.13 不同参数和数据集情况下模型的效果比较

通过观察我们大致可以看到

- ① 不同的模型精度是有显著差异的，所以针对不同特点的语料我们要选择适合的模型才能事半功倍
- ② 数据集越大，精度往往越高。虽然在 1.6B 和 6B 的比较上 Glove 似乎在语义上得分有所降低，但是其总得分也是提高了的，而且对比其他模型，数据集越大，往往精度也是越高。
- ③ 过低维度和过高维度的精确度都会降低。这是因为对于过低的维度，捕捉到的词的信息欠缺，这时候是欠拟合，而过高的精确度会捕捉到过多的噪声，这时候是过拟合。

我们针对 GloVe 还有如下分析，见图 3.14 所示

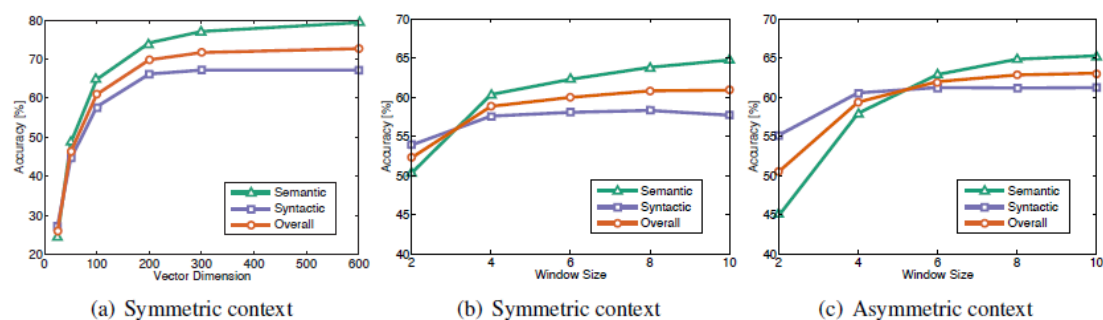


图 3.14 GloVe 性能分析比较图

结合上图我们可以分析得到以下两点：

① 词嵌入在 300 之后几乎趋于平稳，为了那一点点的精确度就设置过高的维度一来将会大量消耗我们的内存，二来也使得我们的计算效率的下降，所以我们取舍一些最终选择 300 维度是最适合 GloVe 的。

② 综合图 3.14 中 BC 两幅图，我们可以看到一般情况下 window=8 是最好的

在训练时间上，如图 3.15 我们可以看到对于 GloVe 来说训练时间一般越长越好

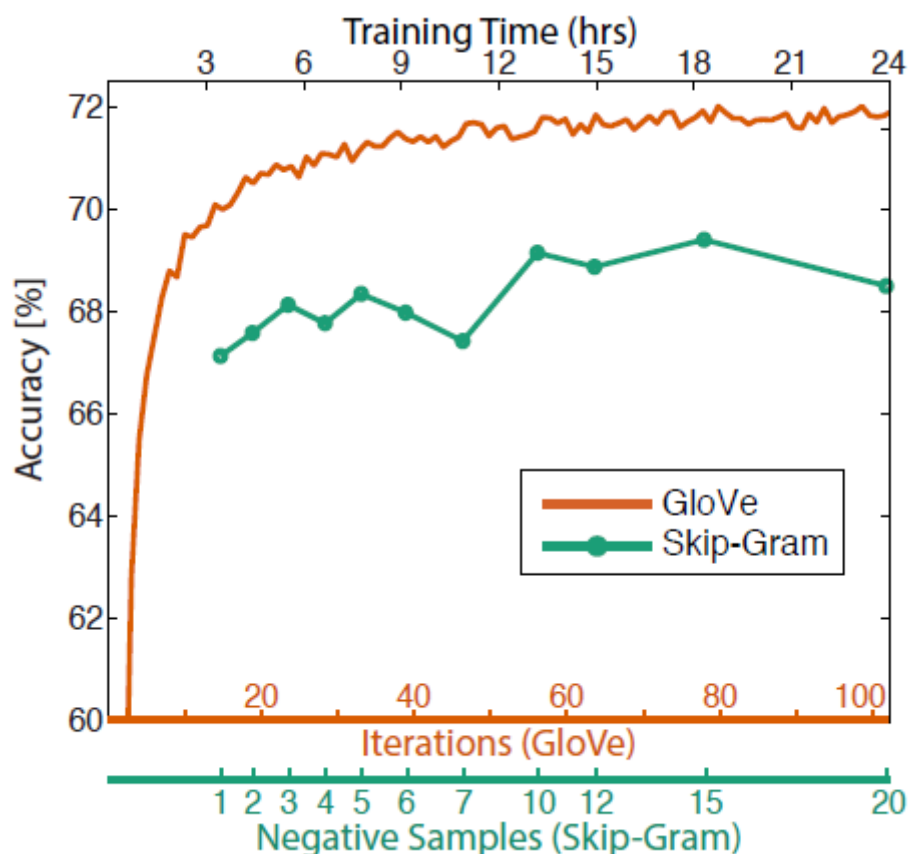


图 3.15 GloVe 训练时间比较图

在训练文集上（如图 3.16 所示），虽然总体趋势上随着训练文本的增加精确度会增加，但是似乎总觉得有一个阈值，当训练集多于某个值少于某个值时精确度会出现一定的下降。

这是笔者的感觉。

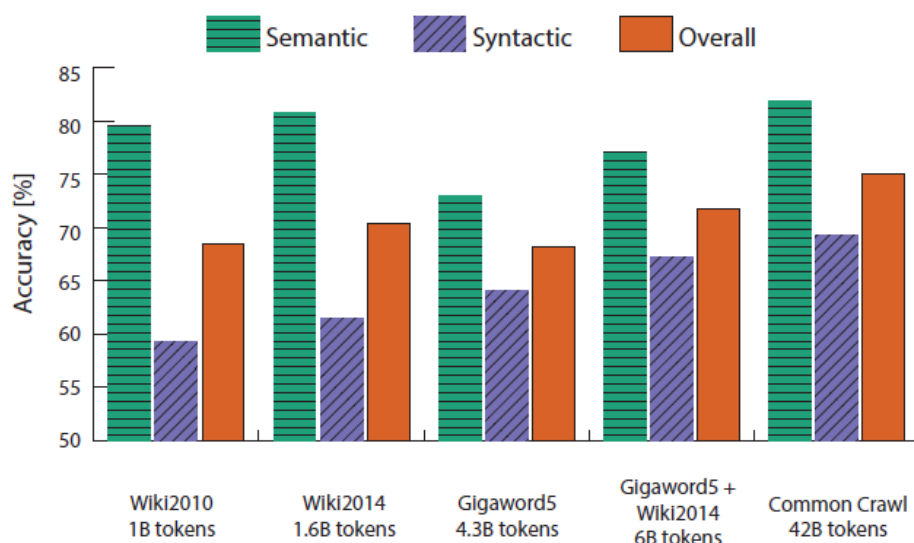


图 3.16 GloVe 训练文本大小效果比较图

视频里还提到了一个一词多义问题的现象，也即一个词放在不同位置可能有不同的意思，这里中心思想就是通过聚类，然后在不同的聚类中选定窗口分别重新训练单词。如图 3.17 所示。（视频这里具体方法讲的很模糊，[笔者也不是很了解](#)，感兴趣的同学可以看论文 *Improving Word Representations Via Global Context And Multiple Word Prototypes* (Huang et al. 2012)）

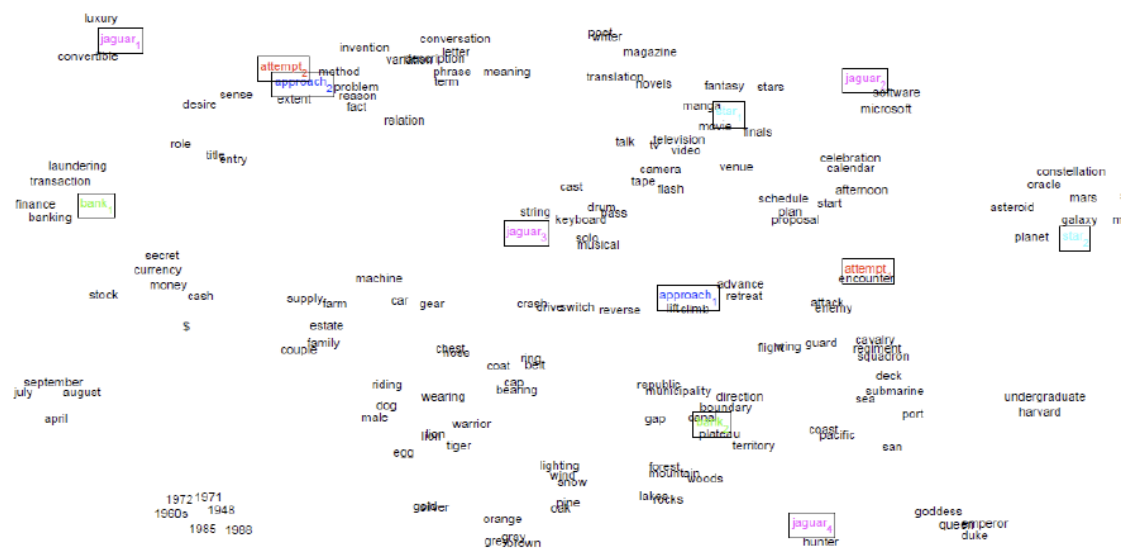


图 3.17 通过聚类等方式消除一词多义示意图

第 4 课 神经网络入门

这节课我们通过引入一个简单的分类问题,介绍训练数据以及标签,然后引出神经网络,并讲解后向传播算法。关于神经网络更详细的内容可以查看附录。

4.1 分类问题

视频首先讲解了分类问题是什么,然后提出了 **loss function**,并根据传统机器学习和深度学习的不同提出深度学习需要加上一个正则化项,由于有不同的 **loss function** 引出了传统机器学习和深度学习更新时的参数的不同,并讨论了一下深度学习什么时候应该更新词向量。现在我们就来逐步讲解。

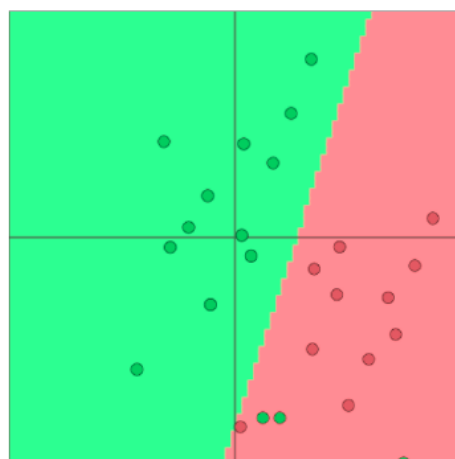
4.1.1 分类问题介绍

常见的分类问题有文本情感分类,命名实体识别等,常见的训练集如下

$$\{x^{(i)}, y^{(i)}\}_1^N$$

其中 $x^{(i)}$ 是一个 d 维向量, $y^{(i)}$ 是一个 c 维向量,这里我做一个详细的解释,其中 $x^{(i)}$ 就是我们的输入数据,假设词向量是 300 维,一般情况下这个 $x^{(i)}$ 也是 300 维的,如果说有时候需要将 5 个单词拼接起来输入(也即将五个向量首位拼接),那么这时候 $x^{(i)}$ 就是 $300 \times 5 = 1500$ 维的。 c 维是对应的分类的结果,例如对于一个命名实体,我们可能有以下几种情况,第一种不是命名实体,第二个是名字,第三个是日期,第四个是地点,那么上面一共存在 4 种情况,那么这时候 $y^{(i)}$ 向量就是一个 4 维向量,例如可能是 $(0.2, 0.3, 0.4, 0.1)$,加起来和为 1,里面每一个代表了输入是哪一种情况的概率,例如对于上面这个向量表示输入不是命名实体概率是 0.2,输入是名字的概率是 0.3,以此类推。上述公式中 N 是总数,代表了训练样例的总个数。

对于一些简单的机器学习,可能直接使用 2 维的词向量来进行分类,或者使用一些简单的逻辑回归或者线性分解或者 **SVM** 等进行分类,如图 4.1 所示



Visualizations with ConvNetJS by Karpathy!
<http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

图 4.1 分类问题示意图

4.1.2 分类问题损失函数

传统大多数机器学习，都假设词向量是固定的（也即词向量在后期的迭代过程中是不会更新的），然后训练逻辑回归权值矩阵 W ，这相当于只修改上图中的决策边界（就是将绿色和红色分隔开的那条界限）那么传统的预测公式可以是

$$P(y|x) = \frac{e^{(W_y * x)}}{\sum_{c=1}^C e^{(W_c * x)}}, \text{ where } W \in R^{C * d}$$

其中 y 可以是标签的值，例如我们的 x 应该是命名实体中的名字，那么我们就将 x 乘上 W 中名字的列向量得到一个值（注意到最终结果有几种可能，那么 W 就有几行，其中每一行与向量 x 相乘都可以得到一个值，这个值代表向量 x 与这个结果的相似度，最后全部相似度用 softmax 取概率），然后在将 x 与 W 中其他列向量相乘依次得到对应的值，最后对结果用 softmax 求概率即可。

对于每一次的训练，我们的目标就是最大化正确标签的概率，也即如果 x 是属于命名实体中名字的类别，那么我们希望预测出来名字这个的概率是最大的，而这等价于最小化正确标签的负 log 概率值，公式如下（其中 f_y 代表 W 的第 y 行跟 x 相乘得到的值）：

$$-\log p(y|x) = -\log\left(\frac{e^{f_y}}{\sum_{c=1}^C e^{f_c}}\right)$$

这个代价函数其实就是交叉熵函数。交叉熵函数定义如下：

$$H(p, q) = - \sum_{c=1}^C p(c) \log q(c)$$

现在我们来一点点阐述为什么上述两个公式是一样的，首先由于标签 c 向量是一个 one-hot 向量，例如假设 x 的标签是命名实体中的名字，而名字是第二种情况，那么 c 向量就是 $(0, 1, 0, 0)$ ，那么交叉熵函数是一个累加的函数，所以就有

$$\begin{aligned} H(p, q) &= -(p(1) \log q(1) + p(2) \log q(2) + p(3) \log q(3) \\ &\quad + p(4) \log q(4)) \\ &= -(0 * \log q(1) + 1 * \log q(2) + 0 * \log q(3) + 0 \\ &\quad * \log q(4)) = -1 * \log q(2) = -\log\left(\frac{e^{f_2}}{\sum_{c=1}^C e^{f_c}}\right) \end{aligned}$$

所以其实二者是一样的。其中视频中又提到了 KL 散度（其实 KL 散度就是相对熵，是跟交叉熵有关的），关于二者的关系我会在附录中详细说明。这里就不做过多描述。

同样道理对整个数据集我们有 loss function

$$J(\theta) = \frac{-1}{N} \sum_{i=1}^N \log\left(\frac{e^{f_{yi}}}{\sum_{c=1}^C e^{f_c}}\right)$$

在这里我们还要引入一个正则化项，这个正则化项是针对我们的神经网络的参数的，我们可以这样想，经过上述的 loss function，神经网络会一直训练参数，直到使得上述 loss function 越来越小，但是这样可能导致过拟合现象（也即我们的神经网络为了拟合这些数据，它的神经结点的参数可能会千奇百怪，这样导致我们的模型的泛化能力不强，那么我们直接给我们的神经网络的参数给一个惩罚项，这样使得神经网络在更新的时候不能随心所欲更新参数，因为参数现在具有惩罚项，这样就一定程度上限制了神经网络的参数更新，这就使得模型不会过拟合全部数据，从而提高泛化能力）视频中给的正则化是平方正则化项，那么现在我们的 loss function 改写为

$$J(\theta) = \frac{-1}{N} \sum_{i=1}^N \log\left(\frac{e^{f_{yi}}}{\sum_{c=1}^C e^{f_c}}\right) + \lambda \sum_k \theta_k^2$$

其中 λ 是一个超参数，越大那么对神经网络的限制能力越大，但是模型往往会欠拟合，越小那么对神经网络限制能力越小，但是模型往往会过拟合。

实际训练的时候我们往往会有如下图 4.2

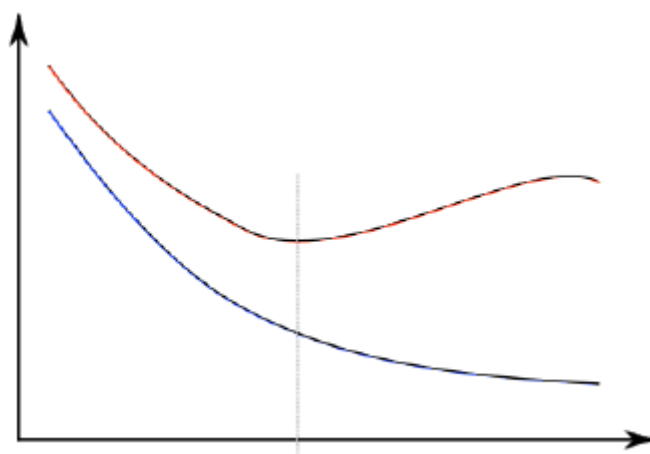


图 4.2 模型训练错误率与测试错误率的比较

其中 x 轴代表模型的能力，越靠近右边模型能力越强，或者也可以代表迭代的次数，越靠近右边模型迭代次数越多， y 轴代表错误率，其中红色的线代表测试错误率（也代表泛化能力），蓝色的线代表训练错误率，可以看到模型能力越强，或者训练次数越多，那么训练错误率越低，但是这有可能导致过拟合，反而使得模型泛化能力下降导致测试错误率上升。

4.1.3 分类问题参数更新

在优化问题上，传统的 ML (Machine Learning) 通常只会更新 W 矩阵中的参数，如图 4.3 所示。（对词向量不会进行更新）

$$\theta = \begin{bmatrix} W_{.1} \\ \vdots \\ W_{.d} \end{bmatrix} = W(:,) \in \mathbb{R}^{Cd} \quad \nabla_{\theta} J(\theta) = \begin{bmatrix} \nabla_{W_{.1}} \\ \vdots \\ \nabla_{W_{.d}} \end{bmatrix} \in \mathbb{R}^{Cd}$$

(a) 传统 ML 模型的参数
(b) 传统 ML 模型更新矩阵

图 4.3 传统 ML 模型更新的内容

但是在深度学习中，我们还会重新训练词向量（如图 4.4 所示），目的是为了使得词向量更贴合我们的模型（当然我们一开始还是会先用 SG 或者 GloVe 等模型训练出词向量，之后在模型训练时继续训练词向量）。

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \nabla_{W.1} \\ \vdots \\ \nabla_{W.d} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+Vd}$$

Very large!

Overfitting Danger!

图 4.4 深度学习模型更新的内容

在模型中继续训练词向量的目的是为了使词向量更适合我们的模型,但是这个是要分情况的,如果我们训练的语料库特别小,这种情况下一般还是不要重新训练词向量(因为原本的字向量可能是基于上亿的大量的数据训练出来的,具有非常好的泛化能力,基于我们的语料库重新训练一定程度上都会损失原本词向量的泛化能力,但是如果我们的语料库特别充足,那么在损失一定泛化能力下增强词向量与我们模型的契合程度是很有必要的,但是如果语料库较小,那么很有可能会使得向量的泛化能力损失太多,得不偿失)。例如有下面例子。原本我们词向量是将 **telly**, **TV**, **television** 都划分到一类(如图 4.5a 所示,全都是电视的意思),但是我们的训练语料库只有 **telly** 和 **TV** 两个词,没有 **telesion**, 那么训练完成后可能会出现如 4.5b 所示的结果,这很大程度上是由于我们的训练语料库太小所导致的。

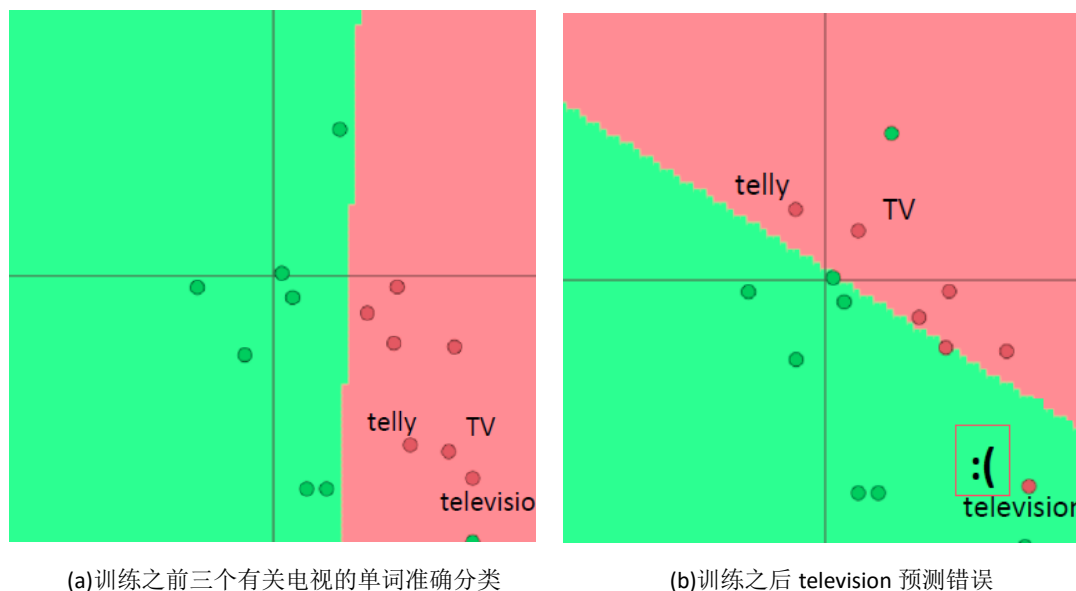


图 4.5 小训练集上重新训练词向量可能的结果

由此我们得出的结论是,如果你的训练集特别小,那么不要重新训练词向量,如果你的数据集特别大,那么推荐重新训练词向量,因为这样会使得词向量更适合你的任务。

4.2 简单的分类任务（命名实体识别）

在开始讲解之前，我们首先需要介绍一个概念，词向量矩阵 L （如图 4.6 所示，通常是通过 word2vec 或者 GloVe 等方法训练出来的词向量矩阵）通常我们称之为 Lookup Table 查找表，原理我相信前面的课也讲了，我们将一个单词的 one-hot 向量乘上这个矩阵就得到了对应的词向量，由于 one-hot 只有一个位置是 1（假设是第 n 个位置）其余位置是 0，所以为了简化运算，我们可以直接将 L 矩阵中第 n 列取出来即可，所以相当于是一个查找的过程，根据第几个位置是 1 找到矩阵第几列即可。

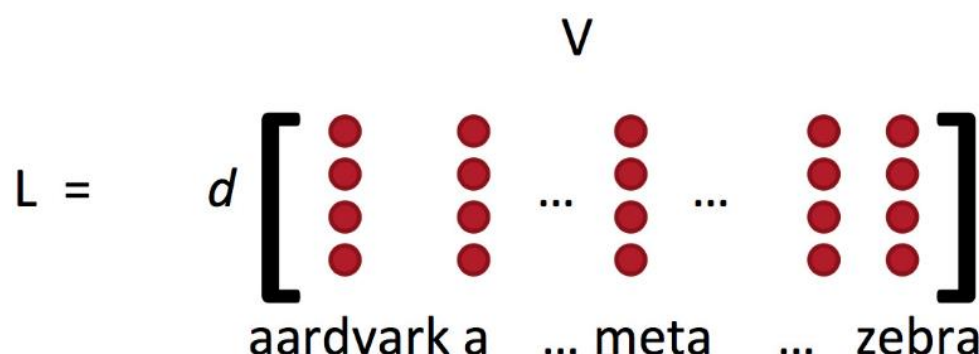


图 4.6 Lookup Table 示意图

视频还补充到 word vectors = word embeddings = word representations，这三个词组通常意思是一样的，用哪个都可以，笔者喜欢中间的，视频说最后一个最经常用。

单词分类很多时候虽然是给单个的单词分类，但是却不能完全基于单个单词，因为单个单词很多时候会带来歧义，例如‘To sanction’既可以表示‘to permit’也可以表示‘to punish’，‘to seed’既可以代表‘to place seeds’也可以代表‘to remove seeds’，而在命名实体识别中‘Paris’大多数情况下代表巴黎，是命名实体中的地名，但是也有‘Paris Hilton’人名的情况，所以单词的分类还需要结合上下文。由于只输入一个单词的时候会有歧义，我们决定结合上下文，选取一个窗口，每次不仅输入中心词还输入中心词窗口内的词，例如我们取句子‘museums in Paris are amazing’，window=2，预测的词为 Paris，那么我们将‘museums in Paris are amazing’五个词当做输入，但是如果将五个词的向量叠加求平均的话，就会损失五个词的位置信息，例如‘museums’无论是在‘Paris’的左边第二个还是右边第一个，最终得到的向量都一样，所以既要考虑上下文又要考虑上下文词的位置信息，我们就将五个词的向量首尾拼接起来。（如图 4.7 所示）所以在单个单词维度为 d ，window=2 的情况下，最终我们的输入是一个 $5*d$ 长度的向量。

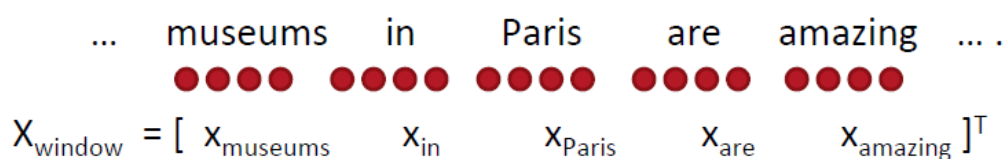


图 4.7 向量首尾拼接示意图

接下来我们继续使用跟之前一样的 softmax 分类法

$$\hat{y}_y = p(y|x) = \frac{e^{W_y * x}}{\sum_{c=1}^C e^{(W_c * x)}}$$

同样使用交叉熵作为损失函数

$$\mathcal{J}(\theta) = -\frac{1}{N} \sum_{i=1}^N \log\left(\frac{e^{f_{yi}}}{\sum_{c=1}^C e^{f_c}}\right)$$

接着使用跟之前一样的求偏导来进行更新即可。

视频里提到求导有以下几个 tips，在最后我们给出这个 loss function 的详细求导步骤：

- ① 非常明确你的变量名称是什么意思，以及他们的维度都要心里有数（因为很多时候这里的求导不再是单一的对某个变量求导，而是对整个矩阵求导，很多时候会涉及到是否转置，所以很多时候一定要千万注意维度）
- ② 链式法则，下面的求导我们就可以看到很明显的链式法则，如果发现 x 跟多个变量有关系，还需要依次对每个变量求导（例如下面的对 x 求导需要拆分成 n 个，每一个分别求导）
- ③ 对于 softmax 这样的函数求导，需要分两部分分别计算，先计算对于 f_c ($c=y$) 的计算一次导数，再对 f_c ($c \neq y$) 的计算导数，你会发现 $c \neq y$ 的导数算出来是有规律的，那么可以直接得到所有 $c \neq y$ 的导数（这里可能听不懂，下面计算过程会详细说明）
- ④ 经过 3 步骤之后，你看看是否可以将所有导数直接写出来，写成一个矩阵，如果还不行，肯定是第 3 步骤漏了什么地方
- ⑤ 将第 4 步骤的向量跟标签 one-hot 向量看是否可以建立联系
- ⑥ 计算的时候尽量把求导结果变成矩阵形式
- ⑦ 中间计算的时候，通过熟悉里面变量的维度，看看能否进行简化

现在我们开始计算：首先我们先明确几个变量的意思

\hat{y} 是 softmax 概率输出向量中正确标签的概率值

t 是真实概率分布，其实就是一个其余部分为 0，正确答案为 1 的向量

$f = f(x) = Wx \in R^c$, $f_c = f$ 向量中第 c 个元素的值 (f 向量就是还没 softmax 的向量)

$$\begin{aligned} \frac{\partial -\log(\text{softmax}(f_y(x)))}{\partial x} &= \sum_{c=1}^c -\frac{\partial \log(\text{softmax}(f_y(x)))}{\partial f_c} \frac{\partial f_c(x)}{\partial x} \\ &= -\frac{\partial \log(\text{softmax}(f_y(x)))}{\partial f_y} \frac{\partial f_y(x)}{\partial x} \\ &\quad - \sum_{c=1 \text{ and } c \neq y}^c \frac{\partial \log(\text{softmax}(f_c(x)))}{\partial f_c} \frac{\partial f_c(x)}{\partial x} \end{aligned}$$

首先我们应用链式法则，将上述求导分开不同部分求导的总和，由于损失函数可以看成 $z = g(f_1(x), f_2(x), \dots, f_c(x))$ 的形式，所以我们有

$$\begin{aligned} \frac{\partial z}{\partial x} &= \frac{\partial z}{\partial f_1(x)} \frac{\partial f_1(x)}{\partial x} + \frac{\partial z}{\partial f_2(x)} \frac{\partial f_2(x)}{\partial x} + \dots + \frac{\partial z}{\partial f_c(x)} \frac{\partial f_c(x)}{\partial x} \\ &= \sum_{c=1}^c \frac{\partial z}{\partial f_c(x)} \frac{\partial f_c(x)}{\partial x} \end{aligned}$$

这就是公式第一步转化成求和的由来，接下来我们继续来说为什么求和又要分成两部分计算，一部分是 $c=y$ ，一部分是 $c \neq y$ 。首先我们来计算 $c=y$ 的情况

$$\begin{aligned} -\frac{\partial \log(\text{softmax}(f_y(x)))}{\partial f_y} &= -\frac{\partial \log\left(\frac{e^{f_y}}{e^{f_1} + e^{f_2} + \dots + e^{f_c}}\right)}{\partial f_y} \\ &= -\frac{e^{f_1} + e^{f_2} + \dots + e^{f_c}}{e^{f_y}} * \frac{\partial \frac{e^{f_y}}{e^{f_1} + e^{f_2} + \dots + e^{f_c}}}{\partial f_y} \\ &= -\frac{e^{f_1} + e^{f_2} + \dots + e^{f_c}}{e^{f_y}} * \frac{e^{f_y}(e^{f_1} + e^{f_2} + \dots + e^{f_c}) - e^{2f_y}}{(e^{f_1} + e^{f_2} + \dots + e^{f_c})^2} \\ &= \frac{e^{f_y}}{e^{f_1} + e^{f_2} + \dots + e^{f_c}} - 1 = \hat{y}_y - 1 \end{aligned}$$

接下来我们来计算 $c \neq y$ 的情况

$$\begin{aligned}
-\frac{\partial \log(\text{softmax}(f_y(x)))}{\partial f_c} &= -\frac{\partial \log\left(\frac{e^{f_y}}{e^{f_1} + e^{f_2} + \dots + e^{f_c}}\right)}{\partial f_c} \\
&= -\frac{e^{f_1} + e^{f_2} + \dots + e^{f_c}}{e^{f_y}} * \frac{\partial \frac{e^{f_y}}{e^{f_1} + e^{f_2} + \dots + e^{f_c}}}{\partial f_c} \\
&= -\frac{e^{f_1} + e^{f_2} + \dots + e^{f_c}}{e^{f_y}} * \frac{0 - e^{f_y} e^{f_c}}{(e^{f_1} + e^{f_2} + \dots + e^{f_c})^2} \\
&= \frac{e^{f_c}}{e^{f_1} + e^{f_2} + \dots + e^{f_c}} = \hat{y}_c
\end{aligned}$$

所以我们发现 $c=y$ 与 $c \neq y$ 计算的结果不一样，所以才需要分开计算，而对于所有 $c \neq y$ 算出来形式一样，结合 t 向量只在正确标签位置为 1 其余位置为 0，所以我们可以有

$$\frac{\partial -\log(\text{softmax}(f_y(x)))}{\partial f} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_y - 1 \\ \vdots \\ \hat{y}_c \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_y \\ \vdots \\ \hat{y}_c \end{bmatrix} - \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = [\hat{y} - t] = \delta$$

之后对于 $\frac{\partial f_c(x)}{\partial x}$ 我们知道其实每一个 f_c 值都是由 W 矩阵第 c 行跟 x 向量相乘的结果，求导约等于就是求 ax 的导数，其中 a 是常数，那么结果自然是 a 也就是求导之后的结果是 w 第 c 行，所以现在求导结果可以统一写成。

$$-\frac{\partial \log(\text{softmax}(f_y(x)))}{\partial f_y} = \delta \frac{\partial f_c(x)}{\partial x} = \sum_{c=1}^C \delta_c W_c^T = W^T \delta \in R^{5d}$$

我们得到的 $\nabla_x J = W^T \delta \in R^{5d}$ 就是我们更新的梯度，注意到这个更新 list 是 $5d$ 维度的，刚好对应我们的输入是 $5d$ ，所以相当于是给输入的每一个维度都有一个更新的梯度，为了帮助理解我们可以直接将 $\nabla_x J$ 五等分，每一部分更新一个单词的词向量，如图 4.8 所示

Let $\nabla_x J = W^T \delta = \delta_{x_{window}}$

With $x_{window} = [x_{museums} \quad x_{in} \quad x_{Paris} \quad x_{are} \quad x_{amazing}]$

We have

$$\delta_{window} = \begin{bmatrix} \nabla_{x_{museums}} \\ \nabla_{x_{in}} \\ \nabla_{x_{Paris}} \\ \nabla_{x_{are}} \\ \nabla_{x_{amazing}} \end{bmatrix} \in \mathbb{R}^{5d}$$

图 4.7 更新矩阵示意图

这样考虑了上下文信息的模型就有助于我们的命名实体识别，例如如果存在单词‘in’，那么模型可能会学会‘in’单词后面紧跟着的中心词很有可能就是一个命名实体中的地点。

有了对词向量的求导之后，同理我们再对 softmax 函数中的权重 W 求导，具体求导过程就不在这里详细展开。但是给大家一个提示（刚才我们对 x 求导的时候 x 是一个向量，但是现在对 w 求导， w 是一个矩阵，这该怎么办？很简单，我们每次对矩阵的一行或者一列单独求导，每次对某一行求导的时候将其他行以及输入 x 看成常量即可，这里要注意哪一行是对应最后正确结果的，这个需要单独求导，其他行的求导得到的结果应该形式类似，这个求导较为简单就不在这里具体展示）。

之后我们将对 x 的求导以及对矩阵 W 的求导的结果合并在一起，会有如下的完整的更新矩阵，如图 4.8 所示。

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \nabla_{W.1} \\ \vdots \\ \nabla_{W.d} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+Vd}$$

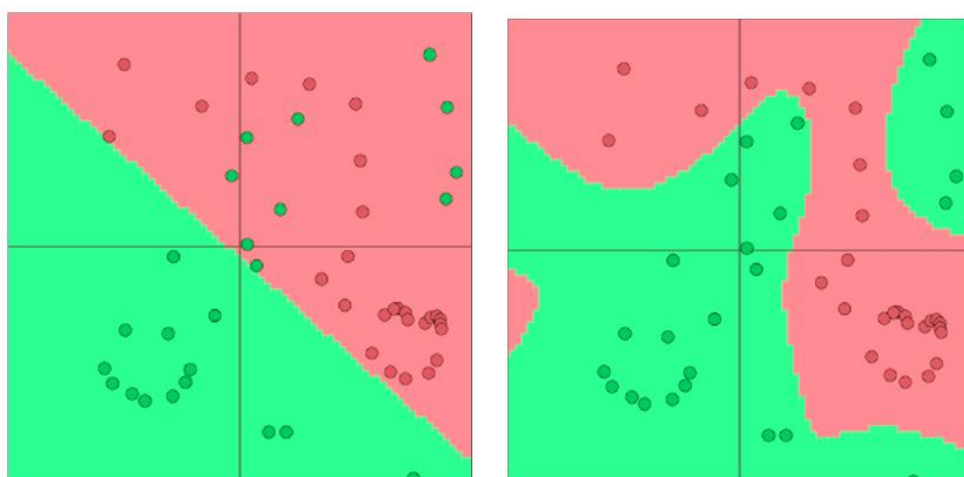
图 4.8 完整的更新矩阵示意图

上述整个过程中有两步的运算代价特别大，其中一步是矩阵运算 $f=Wx$ ，还有一个是指数 e 运算，但是大型矩阵计算却非常高效，而循环现实代码实现起来并没有矩阵计算那么高

效。

4.3 简单的神经网络

上一节我们是使用 **Softmax** 来进行逻辑回归的，但是 **Softmax** 只能提供线性决策边界，如果数据量比较小也许可以提供不错的分类但是对于大数据就存在一定的局限性。如图 4.9a 所示，由此我们想有没有一些方法或者模型可以提供非线性分类从而更好地完成任务呢？这就要引出神经网络了，如图 4.9b 所示。（注意，课外知识补充部分第一章为详细的介绍神经网络的，以及附上了一个反向传播求导的实例，关于这一章更详细的内容感兴趣可以看课外知识补充部分第一章）



(a) Softmax 对于复杂大数据任务表现不好 (b) 神经网络的非线性决策边界更好分类任务

图 4.9 非线性边界与线性边界的比较

接下来我们开始介绍神经网络

。

课外知识补充（部分完成）

Negative sampling 1 亿

CBOW 模型

Hir softmax

GloVe <https://blog.csdn.net/codertc/article/details/73864097>

交叉熵和 KL 散度 <https://www.zhihu.com/question/41252833>

课外第 1 课 神经网络入门

本第一节课截屏以及内容是基于小象学院网络课程寒小阳的《深度学习与神经网络》，图片就不再注明出处。

1.1 神经网络是什么

计算机中的神经网络是模仿人的生理结构神经网络所提出的，接下来我们将从神经网络的结构，功能具体实现以及一个具体事例来进行学习。

目前神经网络主要包括输入层，隐藏层和输出层，其中输入层和输出层是一定有的，一个神经网络可以没有隐藏层，而绝大多数复杂的功能都是在隐藏层中实现的，隐藏层的深度越深，神经网络的能力（capacity）就越大。如图 1.2 所示的神经网络，红色部分是输入层，紧接着是两个隐藏层，最后接了一个绿色的输出层。其中如果只有少量的隐层那么我们称之为浅层神经网络，增加更多的隐藏层后我们称之为深度神经网络。

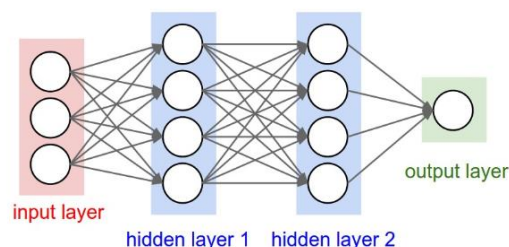


图 1.2 神经网络简单结构示意图

我们可以想到，对于每一个神经元，其与其他神经元的连接方式都是不同的，其余神经元对该神经元做出的响应也是不同的，例如，假设某一个神经元发出一个危险信号给周围的神经元，或许只有一个神经元会做出反应接着传递这个危险信号，而其他神经元可能会选择忽略甚至只对此信号做出较少反应；再举个例子，不同神经元之间的连接长度不同，相同的刺激经过不同的路径到达目的神经元之后的强度也会不同。上述事例说明了每个神经元与其相邻的神经元之间都是有不同的权重的，对于神经元之间的传递我们可以用如图 1.3 来进行示意

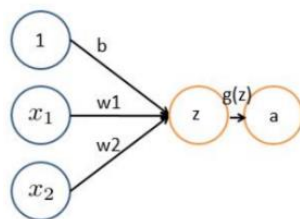


图 1.3 神经元间连接方式

可以看出对于每一个神经元 x ，其与 z 都会有一个权重，对于 z 来说，最终会得到的输入为

$$z = w_1x_1 + w_2x_2 + b$$

其中 b 为偏移量，我们可以认为每个神经元得到的输入除了包括其余神经元的加权输入的总和还会包括每个神经元自己的特殊习惯（对输入进行一定的偏移），得到了输入后每个神经元还需要进行一个激活函数处理。直观上解释很简单，当输入的“刺激”还达不到某个神经元的求时，它完全可以把“刺激”直接截断掉。所以对于每个神经元，会通过自己的激励函数对输入进行处理然后作为自己的输出传递给下一个神经元。

常用的激活函数本节课我们只介绍两种，一种是 sigmoid 函数，一种是 sigmoid 的变形双 S 函数，如图 1.4 所示

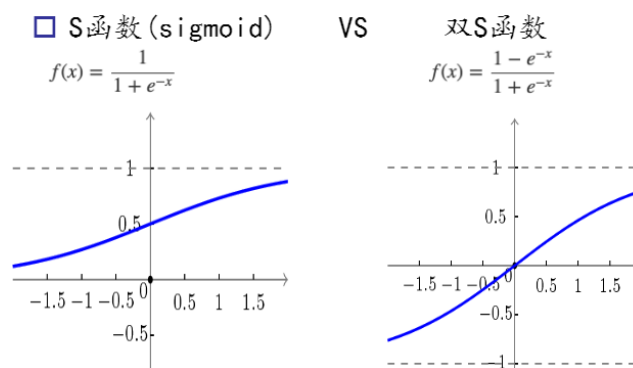


图 1.4 两种常见的激活函数

1.2 神经网络简单应用问题

现在我们考虑一个分类问题，假设一个平面上存在可被一条线分隔开来的两个类别的点，如图 1.5 所示，根据 PLA 学习算法我们可以知道，机器通过学习是一定可以找到一条直线将这个线性可分集分成两部分

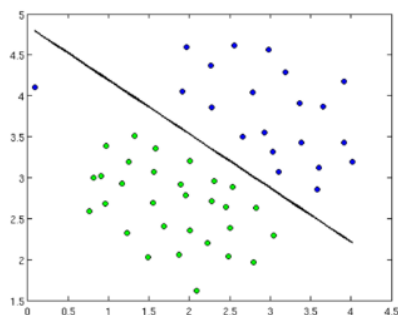


图 1.5 一个线性可分集

换句话说我们只需要用没有隐藏层的神经网络即可得到这样一条直线，假设直线的解析式为

$$z = k_1x_1 + k_2x_2 + b$$

那么我们构造一个包括一个输入和一个输出的神经网络即可，如图 1.6 所示

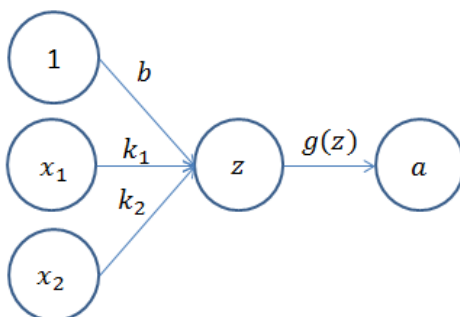


图 1.6 包含一个输入和一个输出的神经网络

这个神经网络激活函数我们采取 sigmoid 函数，最后我们判断当结果 ≥ 0.5 判断为一类，结果 < 0.5 判断为另一类即可。如此我们就用最简单的神经网络模拟了一个感知机。

1.3 神经网络实现逻辑与和逻辑或功能

上一小节我们通过神经网络简单的实现了感知机的功能，这里我们要介绍如何用神经网络实现“逻辑与”和“逻辑或”，我们现在考虑如图 1.7 所示神经网络的输入与输出

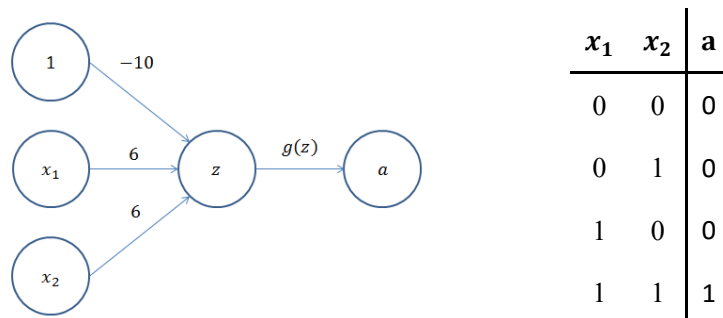


图 1.7 神经网络实现“逻辑与”功能以及对应的输入输出

我们可以看到我们通过改变参数的值成功的实现了“逻辑与”的功能，相对应的“逻辑

或”的功能也可以通过改变参数的值得到，如图 1.8 所示神经网络的输入与输出

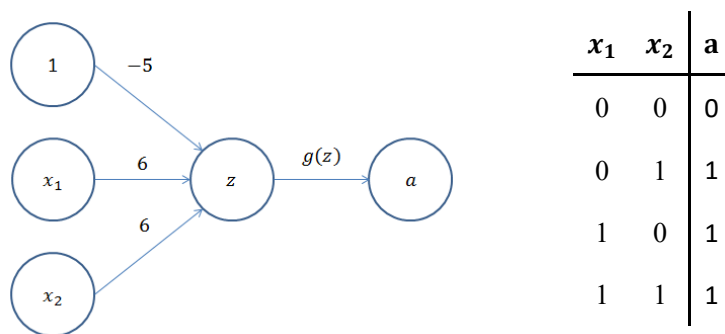


图 1.8 神经网络实现“逻辑或”功能以及对应的输入输出

1.4 神经网络稍复杂应用问题

现在我们考虑线性不可分集，如图 1.9 所示的线性不可分集

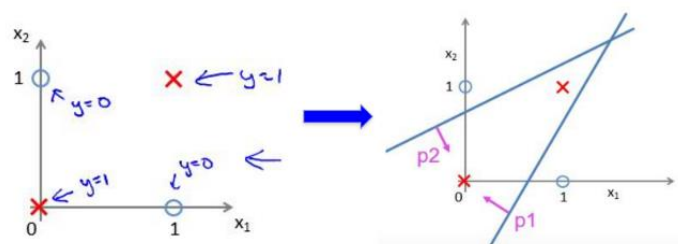


图 1.9 线性不可分集

假设我们定义落在直线一以下与直线二以上公共的区域点记为一个类其余都当做另一个类，假设直线一与直线二有解析式

$$y_1 = k_1x_1 + k_2x_2 + b_1, y_2 = k_3x_1 + k_4x_2 + b_2$$

那么我们可以通过如下图 1.10 所示有一隐藏层的神经网络实现上述功能。

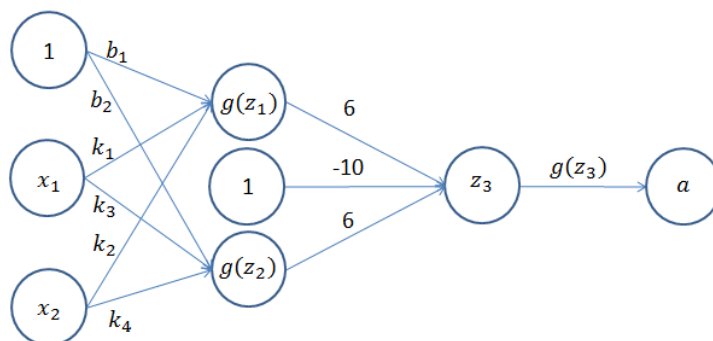


图 1.10 含有一层隐藏层的神经网络

第一层为输入层，第二层有两个神经元，通过各自不同的参数，第一个神经元实际上就

是得到了直线一的解析式，第二个神经元实际上就是得到了直线二的解析式，第三层是用了一个与操作将第二层神经元做了个与操作，然后将结果输出。有次我们可以看到，通过加深神经网络的层数，我们的神经网络的能力大大增强，如果我们将第二层（也即隐藏层第一层的神经元的个数增加）那么我们第三层实际上可以得到很多条直线共同围成的一个区域，我们甚至只用了一个隐藏层就可以完成如图 1.11 所示的分类问题，

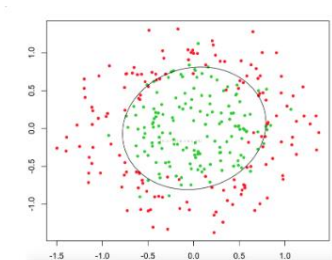


图 1.11 一个较为复杂的分类问题

通过用非常多的直线模拟一个圆，继而围成一个圆域来进行分类问题。关于神经网络的隐藏层的层数我们还可以用图 1.12 来做一个理解

结构	决策区域类型	区域形状	异或问题
无隐层 	由一超平面分成两个		
单隐层 	开凸区域或闭凸区域		
双隐层 	任意形状（其复杂度由单元数目确定）		

图 1.12 神经网络隐藏层的数量与其对应的能力

随着隐藏层的增加，我们可以很轻易的对上一层的结果进行一个组合，越深的层相当于在一个更高维度的一个角度对结果进行处理。

1.5 神经网络表达力与过拟合

从上一个小节我们可以看到，理论上来说单隐层神经网络可以逼近任何连续函数（只要隐层的神经元个数足够多），但是对于一些分类数据，三层神经网络效果优于两层神经网络，但是如果把层数不断增加到 4 层 5 层，对最后结果的帮助却没有那么大的跳变。但是增加神经网络的“容量”会增大神经网络的能力（capacity），神经网络的空间表达能力会变强，但是神经网络的空间表达能力太强往往会导致过拟合的问题。如图 1.13 所示。

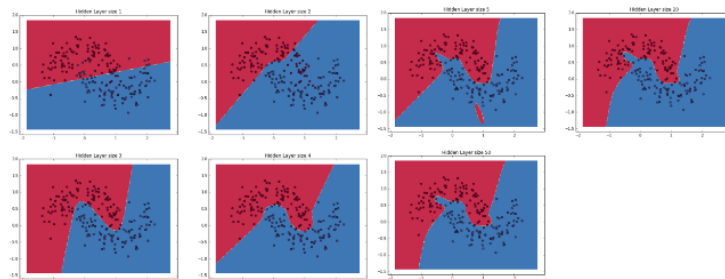


图 1.13 过拟合问题

过拟合会导致神经网络对训练样本的正确率接近 1，但是在测试样本上表现极差，这是我们需要避免的情况，在以后的课中可以通过正则化或者 dropout 来减少过拟合的情况。正则化的思路主要是通过约束神经网络的能力使其无法过拟合，dropout 主要是通过直接废除神经网络的能力使其无法过拟合。

1.6 参数更新 BP 算法

现在我们了解了通过增加隐藏层或者增加神经元的个数可以提高神经网络的能力，但是神经网络最关键的地方是其参数的取值，现在我们要考虑如何去更新神经网络的神经元的参数。这里我们使用 BP 算法（误差反向传播）。BP 算法大致思路如下，假设我们给定一组输入得到了一组输出，我们将这组输出与标准输出计算误差，然后使用随机梯度下降算法（SGD）想办法使得我们的输出与标准输出的误差最小，其中我们将误差函数对每一个需要更新的参数进行求导，然后按照一定的学习率来更新这个参数，经过一定次数的迭代或者当误差下降到一个可以接受的值时算法停止，这时候我们就得到了我们希望得到的参数。

现在我们来一步步详细解释，以三层的感知器为例，如图 1.14 所示

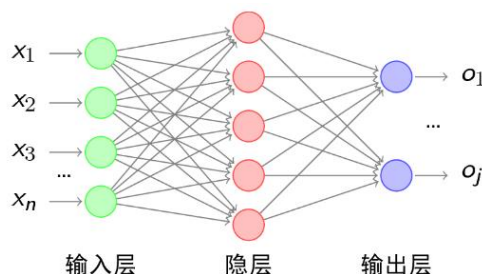


图 1.14 三层感知器

我们假设正确答案为 d ，那么

输出层误差计算

$$E = \frac{1}{2}(d - o)^2 = \frac{1}{2} \sum_{k=1}^n (d_k - o_k)^2$$

误差展开到隐藏层

$$E = \frac{1}{2} \sum_{k=1}^n [d_k - f(net_k)]^2 = \frac{1}{2} \sum_{k=1}^n [d_k - f(\sum_{j=0}^m \omega_{jk} y_j)]^2$$

误差展开到输入层

$$E = \frac{1}{2} \sum_{k=1}^n [d_k - f(\sum_{j=0}^m \omega_{jk} f(net_j))]^2 = \frac{1}{2} \sum_{k=1}^n [d_k - f(\sum_{j=0}^m \omega_{jk} f(\sum_{i=0}^m v_{ij} x_i))]^2$$

光看公式可能会晕，现在我们引入一个实例来进行计算

假设有如图 1.15 神经网络，对应的参数已在图中给出

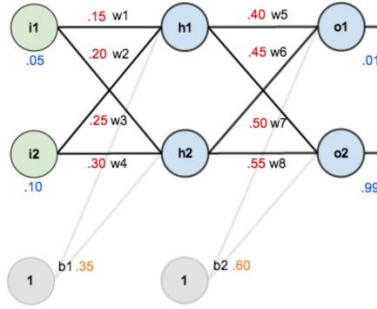


图 1.15 三层神经网络示意图

其中参数初始化为随机的，图中为了方便采取 0.05 为步长进行初始化

现在我们进行前向运算

$$net_{h1} = w_1 i_1 + w_2 i_2 + b_1 * 1 = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

$$net_{o1} = w_5 out_{h1} + w_6 out_{h2} + b_2 * 1 = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-1.105905967}} = 0.75136507$$

$$out_{o1} = 0.772928465$$

误差计算

$$E_{total} = \sum \frac{1}{2} (target - output)^2$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

反向传播示意图见图 1.16 所示

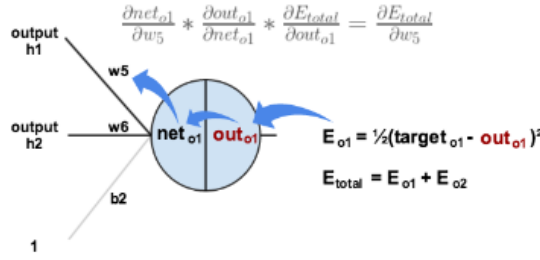


图 1.16 反向传播示意图

$$E_{\text{total}} = \frac{1}{2}(target_{o1} - output_{o1})^2 + \frac{1}{2}(target_{o2} - output_{o2})^2$$

$$\frac{\partial E_{\text{total}}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - output_{o1})^{2-1} * (-1) + 0 = output_{o1} - target_{o1}$$

$$\frac{\partial E_{\text{total}}}{\partial out_{o1}} = 0.75136507 - 0.01 = 0.74136507$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$net_{o1} = w_5 out_{h1} + w_6 out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{1-1} + 0 + 0 = out_{h1} = 0.593269992$$

$$\frac{\partial E_{\text{total}}}{\partial w_5} = (output_{o1} - target_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

$$= 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

$$\text{new } w_5 = w_5 - \eta * \frac{\partial E_{\text{total}}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$\text{new } w_6 = 0.408666186$$

$$\text{new } w_7 = 0.511301270$$

$$\text{new } w_8 = 0.561370121$$

再往下的更新就不计算了越来越复杂。上式中需要注意 η 为学习率，即每次向最陡峭的位置走多大步。