智能问诊

摘要：随着时间的推进，人们越来越感受到整个国家的电子行业的兴起，其中电商，云技术，还有前几年还未出现的智能手机，现在也是满大街都是，最让人瞩目的应该新起之秀--人工智能。首先，人工智能是数学与哲学的结合体，这么说可能不太懂，人工智能研究的的人类的思维方式和对自然语言的处理，而数学就是将自然语言以数学形式在机器上运行，如同高级语言向二进制的转变，使用数学将自然语言变成机器能懂的形式，比如最基本的词向量，这个关键词很重要，后面章节会经常涉及这个。本次核心部分，也就是深度学习用的Google的Word2vec无监督深度学习算法，这个算法相比其他的深度学习算法实现上要简单，训练的时候有更高的效率，对于结果的分析准确度也很高。其选择的是高维转低维用的是PCA主成分分析技术，其能够降低训练成本，不会产生维数灾难，而且提炼了词与词之间的关系。输出矩阵用的是softmax函数，词与词不会产生极端的没有联系的情况。总体来说，训练出来的model完全符合预期的model。

关键词：深度学习；Word2vec；cbow；词向量；

1. 绪论
2. 引言

随着人工智能的推广，开始走入了人们的日常生活中，比如，火热的小爱同学和天猫精灵，也因此以前的只有很少一部分的研究人才到现在有了大量的技术和技术人员，又因此人工智能飞速成长，做出了更好的产品，这一套的良性循环，无不说明人工智能正在发展并崛起，而且以后会影响着人们生活的方方面面。

而智能问诊也是人工智能激流中的一员，他可以像医生一样对病人的病症进行分析，而得出病人得的是什么病，而且它的学习不需要像医生那样需要十几年甚至更多年的学习，词汇量也是远超医生的词汇量，不会产生人类的一段时间不接触某样东西，对这个东西的记忆就会淡忘的情况。如果智能问诊做到对每个细节都把握好，比如可以对人体进行扫描，提取体液，进行方方面面的分析，而不是以简单的病人的口诉病症去分析，那么其准确度绝对完美，甚至超越医生。本次奈何自己能力不够，人体扫描技术目前市场也没有，但坚信，此技术以后只要不断完善，绝对能够给人类带来巨大的贡献。

1. 智能问诊
2. 智能问诊概述

目前智能问诊主要核心部分对病人口诉病症进行分析然后返回反馈信息。其包括唤醒服务，能够只要开机状就可以不断的提供服务，只要喊一下它的名字，就可以完成唤醒，下句话只要说明自己的症状就OK，比如我怎么怎么样或者他怎么怎么样，然后等待系统反馈，反馈完成，一次服务就算完成，进入休眠状态，等待唤醒。

目前其目的是完成急诊医生完成的工作，以后会比医生做的更好，能够不出现误诊情况，而且迅速准确，其可以很容易产生大量终端，可以解决很多病人已经在医院预约好却还一直处于等待看病的情况。

1. 智能问诊构建

开发语言是Python，语言及其灵活，开发库多。唤醒使用snowboy，可以选择公用的唤醒词，也可以自己声音录制唤醒词，区别是设置自己的别人唤醒不了，因为人与人音色不同。深度学习使用Word2vec，训练模型为cbow。文字转语音使用的是pyttsx，声音转byte使用的是Google的SpeechRecognition，byte转文字使用的是讯飞的一个HTTP接口。

1. 智能问诊系统操作

操作及其简单，目前设置的唤醒词是小老弟（个人觉得好玩用了这个词），只要喊声小老弟，系统就被唤醒，然后说出病症，系统就会分析并返回结果。

1. 项目核心技术及原理讲解
2. Python

Python语言今年报道已经超过了Java，凡事皆有原因，Python的语法简练，语言灵活，尤其在自然语言方面，强大到无可替代的地步，其解释器不断的改进，性能也完全够用，而且如果真的要比性能，调个c++编写的demo，绝对不慢，所以不过多讨论性能。在开发速度上也是远超其他语言，不像Java 的臃肿，c++的晦涩复杂，比如知乎就是Python开发的。跨平台性，其是解释性语言，跨平台性毫无疑问。而且动态语言，数据类型在运行时确定，也就不用预先去定义类型，方便。所以此系统最终使用了Python作为开发语言。

1. Word2vec无监督深度学习

word2vec中Word词的意思，vec是vector的缩写，向量的意思，放在一起就是词向量，所以我在上面的关键词里写了词向量。

1. 原理简介
2. 数据源构建

比如我们自然语言说一句话“我周末就可以回家啦”，我们现在对这段话分析，我这里用jieba进行分词分出来['我', '周末', '就', '可以', '回家', '啦']得到这样一组词，那我们如果想用Word2vec训练出来的model，至少需要选个词作为目标词汇，我们这里选择周末这个词，那怎么样提炼语境呢，需要控制俩个参数，第一个是skip\_window，如果设置为2，那么此算法会选择以周末为中心选择左边俩个词和右边俩个词，这个是确定窗口大小，不过这个时候会有三组训练数据出现，（‘周末’，‘我’），（‘周末’，‘就’），（‘周末’，‘可以’），有时候我们不需要全都要，于是还有个参数num\_skips用来控制选择多少个训练数据。

1. 概率model形成

这个是一次拿去数据，而算法会多次去拿数据，那我们就可以得到很多类似于上面的数据源，最终，神经网络基于这些数据源得到得到一个概率分布，这个概率分布就是我们的model，比如拿我们刚刚拿出的一组数据源（‘周末’，‘可以’），周末作为input 词，另一个作为output词，输入input词，整个model会去计算所有output词的可能性，因为上面的数据源，我们已经知道有个“可以”是output里的一员，现在假设数据源里只有“可以”这个output，那此时model会计算出output是“可以”还有其可能性为百分之百。

1. 词向量
2. one-hot

one-hot，历史上伟大的一步，从上面我说的数据源，可以很容易看出，机器，他是处理不了那些自然语言的，于是one-hot出现了，它的作用是把所有分好的词标好号，还是上面那句话，里面总共有六个词，于是得到一个一组编号，那我们还是拿“周末”来用，假设周末编号为2，则one-hot表示就是[0,1,0,0,0,0]，按规律来排，“我”的表示就是[1,0,0,0,0,0]，其他的词以此类推。目前解决了第一个问题，计算机如何存储自然语言。

1. pca

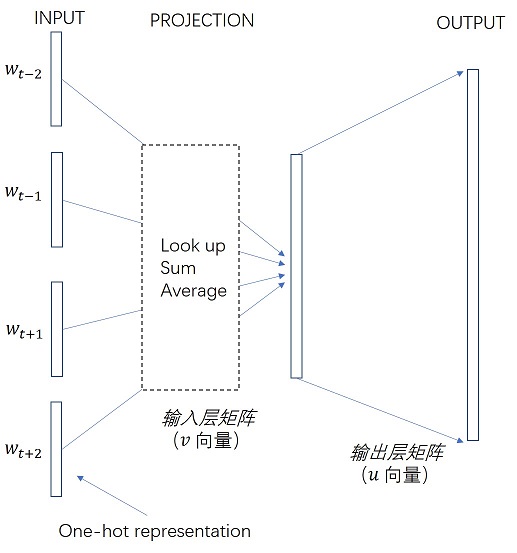
主成分分析（pca），上一步拿到的词向量是完全不行的，因为词库可能十几万词汇甚至更多，首先会产生维数灾难问题，其次，拿到的词向量之间毫无关系，那每个都去分析也是孤立的。于是这里就用到了降维，降维是对高维特征数据处理然后得到低维特征数据。比如现在有很多组词向量，都是one-hot，n维，这里称这些词向量维矩阵A，pca会在矩阵A里找到相互正交的坐标轴，第一步，首先找到矩阵A里面方差最大的，以此方向作为第一个坐标轴，然后第二个坐标轴的标准是这个坐标轴首先与第一个坐标轴相互正交，然后其确定后，平面数据中方差一定是最大的，然后继续找第三个坐标轴，此坐标轴和第二个坐标轴的关系，如同第二个和第一个关系一样，以此类推，找到n个坐标轴，通过公式计算，会发现后面的方差会越来越小，几乎接近于零，于是就把接近于零的坐标轴忽略掉，假设只剩下k个方差较大的坐标轴，那么我们此时就算是完成工作，最终把n维的矩阵A，降维到k维的矩阵，我们称之为矩阵B。其中计算公式有俩种，基于特征值分解协方差矩阵实现PCA算法、基于SVD分解协方差矩阵实现PCA算法，实现方式不同，基本原理都一样。这时pca作用还是很大概，不清晰，没有落到具体的情景，我单独写了pca应用具体场景介绍，本地方写的是pca的原理。

1. pca应用场景

通过上面知道了把n维的one-hot转成k维的一般向量。一般向量表示方式是[ [0.3], [0.4], [-0.3], [-0.4] ]这个样子。我现在使用上面一段话“我周末就可以回家啦”，然后现在新增一段话，“俺周末就可以回家啦”，我们人类一看就知道“俺”这个词和“我”这个词他是相近的，这个如何完成原因暂时不知，不过可以让机器完成这个认知就是使用的pca，现在使用pca对第一段话进行降维，我们会到一个平面，包括坐标轴和其所位于的点，然后对第二段话也使用pca降维，得到坐标轴和其所有点的位置，通过对比俩个平面内词分布情况我们人类可以直观看出“俺”和“我”位置是几乎在一起的。不过机器的操作有所不同，假设现在以“我”这个词作为input词，机器会把这个input词和其他所有词的概率都会求出来，然后将数据送到输出层，其中计算要用到softmax函数，存储使用哈夫曼树。这里我们很容易看出“我”和“俺”这俩个词之间被计算的概率会很大，其实就类似于人类的“我”就是“俺”的认知了。这里的pca照应上面的概率model形成。在这里说一下pca和Word2vec的关系，pca是Word2vec的一个完成工作的一个工具。

1. Word2Vec原理解析（cbow模式）
2. cbow模式的层次结构图

前面介绍了这么多基础知识，现在开始讲核心部分。Word2vec有俩种训练模式，分别是cbow和skip-gram，这里使用的是第一种，cbow是多个词汇然后找出相近词，skip-gram是一个词汇找去多个相似的词。cbow的本次模块涉及的结构如下图所示



上图第一层一般被称为输出层，以中间单词往外扩展，至于扩展多少个词， 在实际操作中是以skip\_window参数控制，在上面的数据源构建里有提 到，而且每个词向量都是以one-hot表示。这些词向量之间也没有顺序， 这个模型又被称为词袋模型。中间的projection叫隐藏层，因为其数据是 通过查表得到的，首先他需要初始化一个矩阵，称为W，矩阵的维度数 由参数size控制，这个和上面在词向量里的pca里的k对应，这个如果不 懂，可以去看一下线性代数里矩阵的知识点，我这里就不讲解了。一般实 际使用的时候设定100或者200，我设定的100，这个矩阵行数就是总次 数，也是和矩阵相关知识点有关系，不懂看书。最后一层是输出层，使用 哈夫曼树作为存储，也就是最后的概率model，上面有说过概率model。

1. 当前窗口一次移动完成的动作

从上图确定，设定当前中心词是wi，输入的是其上下文词的词向量，编码是one-hot，其词向量分别是wi-1，wi-2，wi+1，wi+2，这四个。我在上面cbow模式的层次结构图里说的w矩阵用到了，为|V|\*K，V是其行数，大小是整个词典的词数量，后面K是列数，大小上面已经说过了，然后现在进行lookup操作，第一步：

https://img-blog.csdn.net/20180226203951164，

将one-hot编码的词向量与w矩阵相乘，最后得到了对于wi那一行的向量，这个向量就是上面降维后的向量，这里称之为稠密向量，但是他只是对应其中一个i，所以现在进行第二步：

https://img-blog.csdn.net/20180226210403460

Pmiddle是我们刚刚拿到的只是对应某一行的稠密向量Vi，Wout是输出矩阵，Pout为输出向量，这个pout中的每个维度的值实际就是本次的input 词与其他词相近的概率，大小1\*|V|，然后使用softmax对Pout词向量计算就可以得到一次操作后中心词与其他词汇相似的概率。这里可能还不太清晰，因为我pout每个维度的值已经是与每个词相似的概率了，为什么还要做一次softmax，实际上每次算出概率他所有词汇相似概率可能不是1，输出矩阵严格要求，中心词汇和其他每个词汇相似概率相加为1，所以要softmax一下，其实这里不仅仅得到概率和为1,这个就和softmax原理有关了，内容比较多，而且和论文关系不大，所以这里只提一下。

1. 细讲步骤

1、查表（lookup）

查表操作很简单，就是下面操作：

https://img-blog.csdn.net/2018030521525918

这里简单描述，首先词量大小为V，为了衔接上面的V，于是https://img-blog.csdn.net/20180226203226901这个矩阵为（1 x V），后面的W矩阵为（V x k），这里的k也是衔接上面的k，俩矩阵相乘得到的https://img-blog.csdn.net/20180305215345132大小为（1 x k）。继续上面操作，得到如下公式：

https://img-blog.csdn.net/20180305215831475

https://img-blog.csdn.net/20180305215831475

https://img-blog.csdn.net/20180305215901838

https://img-blog.csdn.net/20180305220015999

2、求和并平均（sum and average）

求和并平均操作很简单，就是拿上一步中心词汇对应的上下文词汇计算出来的词向量相加然后求平均，这个值是下面条件概率里的条件的概率，其值从整个结构上是投影层的输出，计算公式如下：

https://img-blog.csdn.net/20180305220616169

这里其实有个点，不知道你们有没有想过，既然作为上下文词，假设规定后一个词发生的概率是在前一个词已经发生的情况下的概率，这个也就是条件概率，但是还有另外一个概率，就是我前一个词发生的概率是在后一个词已经发生过的情况下的概率，你会发现这样相加是有问题的，所以这时候就是cbow的定义了，cbow是不考虑前后文的，它认为每个词都是作为条件概率里的条件。（这里还有一个点，求词出现概率，为什么是条件概率，想要知道可以看一下隐马尔可夫模型）

3、输出（output）

以下是最初定义中心词概率，中心词在此次上下文的情况下的概率，公式如下：

https://img-blog.csdn.net/2018030522110795

经过lookup和sum and average之后，其生成以下公式：

https://img-blog.csdn.net/20180305221712620

这里有个点，就是在我们给出上下文时，我们实际上可以算出此词典中所有词在本次上下文中的概率，不过一般概率一般比不上中心词，不过依然有，所以一般如下公式去表示：

https://img-blog.csdn.net/20180305222253808

实际上目前上面的这个公式什么都做不了，所以要对此公式进行计算，最终得到以下公式，softmax就是在这个步骤里使用的。具体计算公式如下：

https://img-blog.csdn.net/20180305223947691

这里的http://latex.codecogs.com/gif.latex?V就是上面的http://latex.codecogs.com/gif.latex?V，  https://img-blog.csdn.net/20180305220152578也一样。这里可能不太懂的应该是这个https://img-blog.csdn.net/20180305224132414。在窗口移动那个模块里有https://img-blog.csdn.net/20180305224206230，https://img-blog.csdn.net/20180305224132414是其中的一行，你会发现u和v它是同一个，但实际它们是被看做俩种向量，权威解释是一个是输入矩阵里的行，一个是输出矩阵里的行。

4、优化

在做这步之前，先给一段训练文字，设其长度等于T，由上几步操作可以很轻易的看出以下格式：

https://img-blog.csdn.net/20180312220217533

然后我们可以从这段文字里获取上面格式的数据，i就是现在位于文本的坐标。文本如果要是中文要分词。因为现在要优化，我们这里使用似然函数，因为要最大化优化，所以产生这一案文的可能性如下：

https://img-blog.csdn.net/20180312220618697

这里有个点，比如我i为1，或者i为T的时候，https://img-blog.csdn.net/20180312220722543是有问题的，所以这些特殊值需要被其他的字符代替。而且最大化似然函数就是最小化损失函数，所以又有如下公式：

https://img-blog.csdn.net/20180312221114498

然后再根据上面的https://img-blog.csdn.net/20180305223947691

公式可得

https://img-blog.csdn.net/20180312221926939

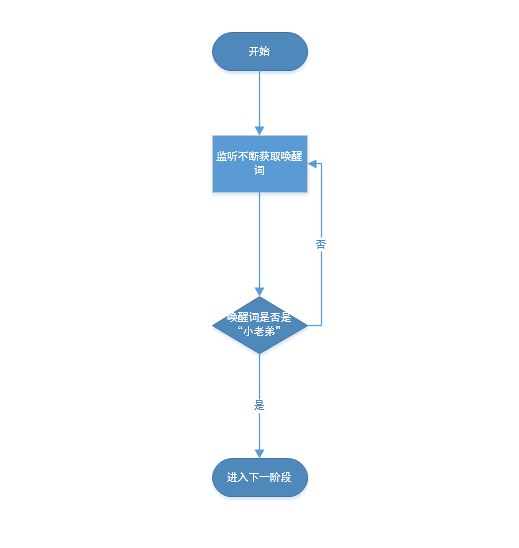
最终得到这个函数，变量是https://img-blog.csdn.net/20180312222049582，然后梯度下降一直迭代下去直到得到最优值。OK，核心原理讲完了，下面是具体工作部分，很简单。

1. 需求分析

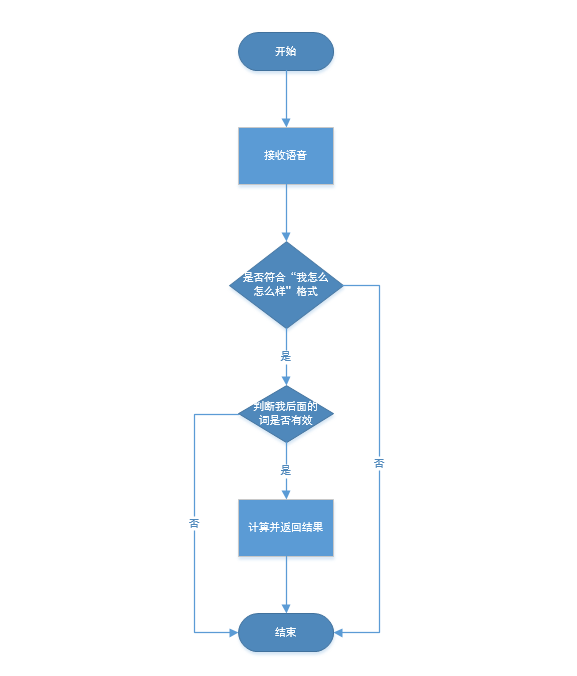
本系统首先做的事情有三个，如何设计用户层面的交互，还有选择系统本身深度学习的模式，还有获取庞大的数据给机器训练。

1. 详细设计
2. 用户层的交互

首先本次不是一个完整的智能，其不能完成正常的对话，不是处于一直交流的状态，所以选择以唤醒状态完成对话，中间对话和返回结果也是有特定的格式。语言唤醒snowboy，其工作方式是如图：

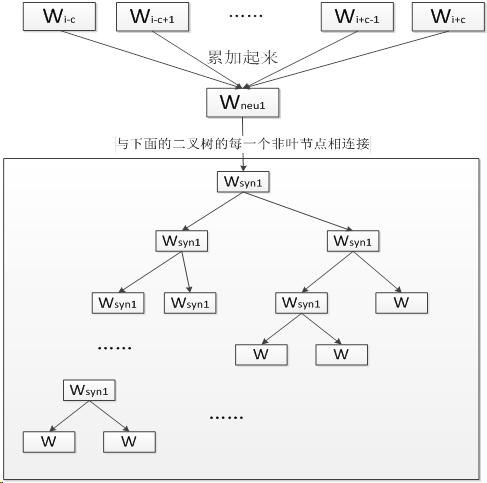


唤醒后对话流程，如下图：

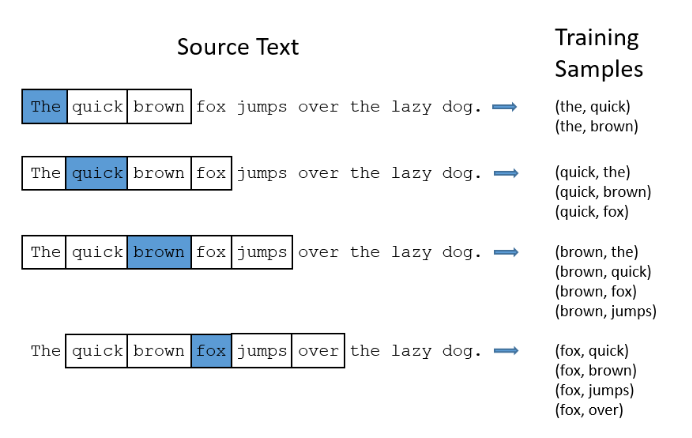


1. 学习模式选择

首先，已经确定过是以多个症状确定病名，又因为考虑到本人机器性能不是很好，所以选择了Word2vec无监督深度学习底下的cbow模式，训练速度快，因为其不需要特别大量的数据，而且支持在已经训练好的model下载新增数据。Word2vec的cbow的完整工作结构如下图（包括存储方式）：



获取输入层数据的流程如下图：



1. 数据获取

因为是医学方面的，所以要求比较权威，所以择了从百度百科上和百科名医上的爬取数据，词库是在搜狗输入法官网下载的，词库包括Flenci神经病学词汇、ICD-10疾病编码、病理学、妇产科病理学词库、胃肠科目、医学词汇大全。一般这俩个网站没有将数据放在restfull接口上，所以是以获取HTML元素拿到数据，这里有个点，如何确定数据在HTML元素哪里，其实数据的位置是有规律。拿到的数据以.txt文件存储，数据量最好一个G左右，毕竟是个概率模型，数据越大，数据越准确。

1. 最后成果
2. 用户唤醒，喊一声唤醒词“小老弟”，终端便会反馈 “在的，干什么”，演示视频链接：<ding.wav>
3. 接受病症，对其喊出以“我流鼻涕、发烧”，终端进入查询状态。演示视频链接：
4. 如果第二步成功完成，系统返回结果并进入等待唤醒状态，演示视频链接：

结语

本次论文经历颇为坎坷，首先第一个点，一开始我对人工智能是一无所知的（其实我现在对人工智能也只能说懂一点点，可能一点点都不到）所以首先第一件事，我找了个别人做好的model研究。第二个点，也就是人工智能不是我一开始所想的，用Python，直接把库拿过来，直接调就行了，其实如果要是不知道其原理，你都不知道怎么去调。第三点，很多比如CSDN、Stack Overflow或者博客园文章对我来说不是所有的都很准确，我看了大概有上百篇论文，有些论文中会有部分是错的，不过还好很大一部分我能看出来，不过还是有些我不确定是不是对的，所以蛮坑的。第四点，一些知识点，只有英文版的，中文没有，用百度翻译翻译出来的，我是看不太懂，比如里面的专有名词它也会给翻译，Python翻译成巨蟒，语句也不通顺，所以最后还是自己慢慢读。第五点，知识量是真的大，因为每个知识点可能或多或少都需要建立在其他知识点之上，所以看一篇论文的时候总会看到其他从来都没听的知识点，我这篇论文也是，也提到了很多其他知识点，因为我不可能在介绍我本篇论文的时候也会介绍其他和我论文没有多大关系的知识点，对我来说是浪费精力。但是这些基础知识点还是都要给弄懂。然后整个的结构在我看了这些文章之后就开始慢慢清晰了，至于做出来就很简单了。最后提一点，要用like \*nix的系统开发，不要用Windows，因为会有很多坑需要去补，还有我这里的snowboy压根就不支持Windows系统。附一张我整个开发的时间线，历时19天，这里没有加上我看论文的时间（反正蛮有成就感的）：

