Lecture 10

红色标注的语句，为重点。
蓝色下划线标注的语句，说明给出了参考阅读链接，可依兴趣阅读。
紫色加粗，表示参看附件。

写在Lecture10之前

本次分享的主题是用Python进行自然语言处理，按照计划，这也是我们分享的最后一个主题。不过经过收集大家的意见，在本次分享结束后，我还会在下周三前补充一个Python数据处理的实例，帮助大家更好地理解Numpy和Pandas的一般使用场景。

本节内容组织方式如下：1.简述自然语言处理（NLP）的理论支撑。2.介绍NLP作为工具，能够帮助我们解决哪些问题，并解释常见的误区。3.一个帮助大家使用NLP工具的教程，以调用api、调用预训练模型、自己训练模型这三种方式来覆盖大多数使用场景。

1.自然语言处理（NLP）的理论支撑

长话短说，自然语言处理的核心问题就是：如何让计算机理解人类语言。如果做进一步阐释的话，自然语言处理的任务就是将人类语言转换成计算机可以理解的形式。

按照历史的发展，人们陆续使用了三种方法表示人类语言：WordNet、One-hot编码、稠密向量。它们分别基于这样的思想：用同义词反义词表示一个词语、用互相正交的向量表示一个词语、用上下文表示一个词语。

——用*WordNet*表示人类语言

WordNet的核心思想是用同义词反义词表示一个词语。

在20世纪80年代中期，科学家们首次提出了WordNet，WordNet是一个包含了一系列同义词和反义词的庞大词库。就像一本大词典一样，每个单词都列出了它的同义词和反义词。人们相信，一个词的意思可以用这些同义词和反义词来表达。例如，在WordNet中，“good”的同义词可能包含“just”、“goodness”、“up right”等。实际上，这些词确实在某种程度上解释了“good”的含义，这也让WordNet真的具备了一点实用属性。

然而，WordNet的问题是显而易见的。首先，它忽略了词语在不同语境之间的细微差别，因为有些同义词/反义词只在某些上下文中是正确的。其次，创建一个WordNet需要巨量的人力投入和资金。最后，尽管它需要巨量的投入，但它仍然无法跟上时代，因为每天都有很多新词出现，而且随着时间的推移，庞大词典的维护也越来越困难。

Problems with resources like WordNet

- Great as a resource but missing nuance
  - e.g. “proficient” is listed as a synonym for “good”.
    This is only correct in some contexts.
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t compute accurate word similarity →

——用*One-hot*编码表示人类语言

WordNet最大的缺点是，即使真的做出了一个合格的词典，它的数据也并不符合计算机能够处理的形式，因为无论是同义词还是反义词，本质上都是难以处理的字符串。那么，有没有更加数字化的、方便计算机处理的词语表达形式呢?

带着这个问题，One-hot编码应运而生。人们通过编码，将每个单词都变成了一个独一无二的向量，这大大推动了整个NLP行业的进展，因为终于可以单词转化成易处理的形式，方便各种下游任务的展开了。

One-hot的编码逻辑就是针对一个固定的语料库（例如一本小说），将其中所有出现过的单词都进行编码，编码结果是一个仅由一堆0和一个1组成的向量，并且这些向量之间彼此正交。

例如在一个一共只有15个单词组成的语料库中，我们的One-Hot编码是15维的，motel和hotel作为两个不同的词汇，它们的1元素位置不同，因此它们的向量彼此正交。

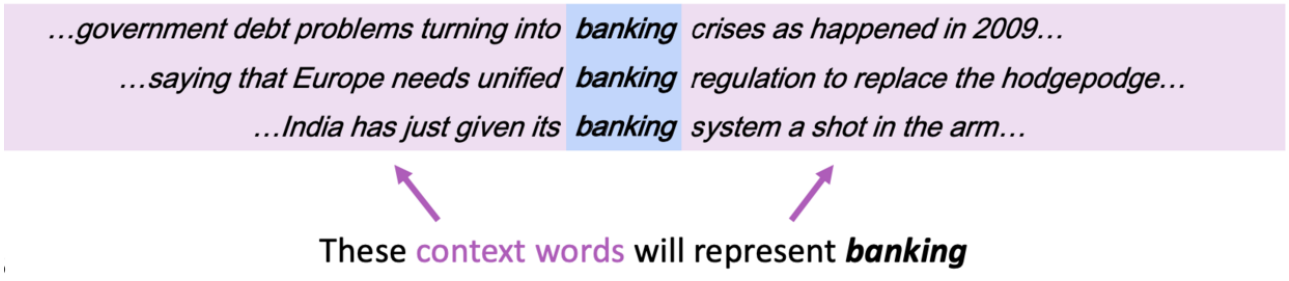
One-hot编码让我们能够执行一些相对简单的NLP任务，例如在其基础上演化出的词袋模型（BOW，Bag of Words），可以让我们把一个句子、一段文本向量化：对于一个句子，其向量中第i个元素表示编号为i的词语出现的次数。这样，我们就可以用向量的余弦相似度计算不同句子、文本之间的相似度，或者进行文本内容的自动分类、聚类等操作。

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

但是，One-hot编码的缺点也是显而易见的，首先，因为One-hot向量之间是正交的，因此我们无法计算两个词语之间的相似度，就无法进行更多细粒度的NLP任务。其次，对于一些比较大的语料库，One-hot向量的维度会达到百万级别，这就会带来“维度灾难”，使得计算机执行NLP任务的效率大大降低。

——用稠密向量表示人类语言

NLP领域的突飞猛进，开始于Google提出的Word2Vec方法，它通过大量语料资源的训练，将一个词汇转化成了一个稠密向量（如[1.123,3.443,1.456]），从而大大提高了一个向量能够容纳的信息。这种表示方法的核心思想是：一个词的意思可以被它上下文出现的其他词的概率所表示。



例如，在上图中，banking一词的意思被认为是由若干个上下文决定的。那么，对于任意一个出现在语料中的词汇w，我们定义一个固定窗口的上下文（如窗口为3表示前后各3个词为中心词的上下文），Word2Vec希望通过w的上下文，给出一个w的稠密向量，如：

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

这种稠密向量一般被称为“word embeddings”或者“word representation”。

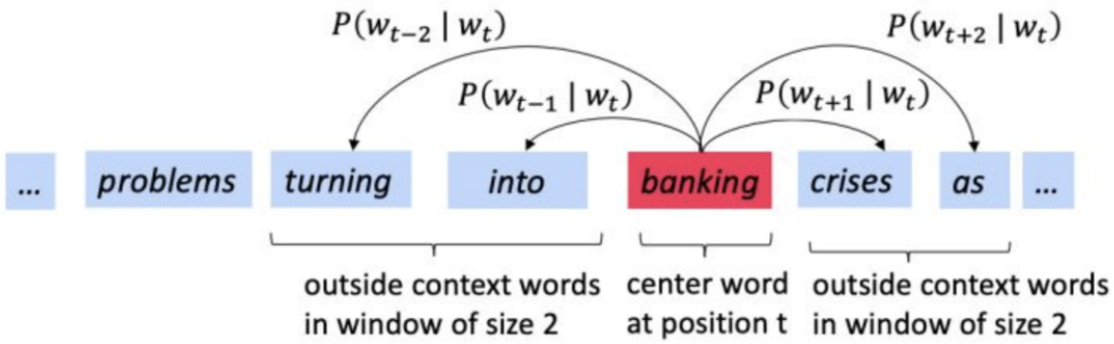
Word2Vec的流程分为如下几步：

——收集大批量的语料
——对于语料库中出现的词汇，均随机赋予一个初始向量
——依次扫描语料库的位置t，对于每个位置t，都有一个中心词c和上下文词汇o们
——使用中心词c于上下文词汇o们向量的余弦相似度来计算给定一个中心词c，上下文是o们的可能性（或者相反，给定一个中心词是o，上下文是c们的可能性）
——不断调整c和o的向量，使得如上所述可能性最大化（即最符合现有语料的情况）

具体可参考Google的论文：https://arxiv.org/pdf/1301.3781.pdf

我们用一个案例来加深理解：

在这个例子中，banking是中心词汇，我们的目标是使用它来预测上下文的词汇是啥，上下文的窗口设置是2，因此只用预测前两个和后两个词汇，如果预测结果不准确（不符合语料中的真实情况），我们就调整banking的词向量，最终得到比较好的预测结果。



调整逻辑如下：

首先，对于每个中心词c和上下文词汇o，我们计算softmax下的条件概率（其中u、v分别代表c、o的词向量）：

$$P(O=o \mid C=c)=\frac{\exp(\boldsymbol{u}_o^{\top}\boldsymbol{v}_c)}{\sum_{w\in \text{Vocab}}\exp(\boldsymbol{u}_w^{\top}\boldsymbol{v}_c)}$$

那么对于中心词c的所有上下文词汇o，我们可以得到极大似然函数（m为窗口大小）：

$$L(\theta)=\prod_{t=1}^{T}\prod_{\substack{-m\le j\le m\\ j\neq 0}}P\left(w_{t+j}\mid w_t;\theta\right)$$

在这个函数的基础上，我们就可以设定当前对应的损失函数（θ指中心词c对应的词向量）：

$$J(\theta)=-\frac{1}{T}\log L(\theta)=-\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}}\log P\left(w_{t+j}\mid w_t;\theta\right)$$

我们的目标就变成了优化θ，使得损失函数值最小。这时我们得到的θ就是中心词c的最佳词向量。

我们使用梯度下降法来调整θ

梯度下降具体可见：https://www.cnblogs.com/pinard/p/5970503.html

具体到实际操作中，词向量的生成和优化往往引入了深度学习，知名的CBOW、Skip-gram模型都可用神经网络作为词向量训练的承载体，准确的说，词向量是神经网络训练语言模型的“副产品”，由于和专业不相关，这里不做展开。

关于语言模型可以查看：https://www.cnblogs.com/chenyusheng0803/p/10978883.html

这样，在一次一次的优化中，我们得到了每个词在语料库上的最佳词向量，这些词向量通常包含了大量的多维度信息，从而表现出多种符合语言常识的规律。一个经典的例子是用king的词向量减去man加上women，大致等于queen的词向量。

包含了大量信息的词向量在各个指标上都远超前辈们，事实上，这也是至今为止NLP领域最常见最基础的词向量生成方法。

2.NLP能解决的问题

在了解了NLP的理论支撑后，我们以工具视角来了解它，看看它就能能为我们解决哪些问题。

首先要明确的是，NLP即自然语言处理，和我们一直在谈到文本分析并不完全是一回事。NLP更偏向概率学习、人工智能，解决的问题更偏向是让计算机有朝一日真的能理解人类语言，而文本分析的数学基础并不高深，解决的问题更多是关注如何让计算机帮我们提取文本的特征，进行挖掘。文本分析的经典应用场景包括使用词袋模型，根据以词频为基础的TF-IDF指标进行文本主题聚类、运用模式匹配来进行垃圾邮件过滤等。

NLP关注和解决的问题如下表所示：

问题	描述
中文分词	将/一句/中文/拆成/若干/个/词汇/方便/下一步/处理
词性标注	给分词完的词语标注词性
句法分析	分析一句话的语法，识别主谓宾等
自然语言生成	根据某种规则生成一大段文字（垃圾文章生成器）
文本分类	二分类如情感分析，多分类如新闻主题识别
信息检索	知识库管理，关键词检索
信息抽取	结构化提取文档信息，将“信息”转化存储成“知识”
问答系统	智能问答机器人
机器翻译	百度/谷歌翻译

如何将NLP能实现的功能和我们的研究与工作结合起来，这是大家需要思考的问题。但有必要指出的是，还是建议大家以工具的思维来看待NLP，上面列出的每一个细分领域，都足以作为一名博士生几年的研究方向。我们应当以使用NLP工具为主，而不是“欲与CS PhD试比高”。

~~最重要的原因还有，往太细分领域讲也有点超出我的知识范围了。~~

3.自己动手使用NLP工具

NLP领域的工具化倾向也非常严重，众多打包好的服务可以为我们铺平道路。

——使用百度*api*

对于大量常见任务，如分词、词性标注、句法分析、计算文本相似度、词向量查询、评论情感分析、文章主题概括、文章分类、新闻摘要、地址识别、文本纠错这些功能。百度都提供了免费的接口供大家调用。这也是我最为推荐的NLP工具使用方式，因为作为人工智能巨头，百度训练好的语言模型是极大概率优秀于我们自己训练出的模型的。

前面有提到过百度api，参见Lecture8-6。

最好的指引是百度自己的python-sdk手册：https://ai.baidu.com/ai-doc/NLP/tk6z52b9z

——使用预训练模型

当然，使用百度的api也有一些限制，最大的限制就是接口的访问不能够太频繁，会影响到某些大型任务的效率。所以我们有时会希望NLP工具能在本地运行。

这里推荐transformers这个Python模块，该模块被定义为方便我们下载、使用各大公司预训练好的模型，并且快速和PyTorch、TensorFlow等深度学习平台快速对接的工具。

关于transformers模块可以见：https://github.com/huggingface/transformers， 使用pip就可安装。

想要调用transformers的默认语言模型非常简单：

```
In [11]: from transformers import pipeline

         # Allocate a pipeline for sentiment-analysis
         classifier = pipeline('sentiment-analysis')
         classifier('在扬州搓澡真是太开心了！')
Out[11]: [{'label': 'POSITIVE', 'score': 0.612272322177887}]
```

可以看到，我们使用pipeline类来指定各种任务，例如在这里就指定其为情感分析，程序对中文给出了还可以的判定结果。

我们可以指定很多其他任务，例如文本生成、命名实体识别等。

不同任务的名称见：https://huggingface.co/transformers/task_summary.html

我们不仅可以指定模型的任务，针对我们的实际应用场景，我们还可以不仅仅局限于默认piepline方法背后的模型，自由地指定适合我们的预训练语言模型，如Facebook的BART、Google的GPT-2等，transformers模块提供了一系列训练好的模型开源供我们使用，基本包括了所有主流模型。

不同模型的名称可见：https://huggingface.co/transformers/pretrained_models.html

调用过程也非常简单：

```
In [35]: from transformers import AutoTokenizer, AutoModelForSequenceClassification

         model_name = "nlptown/bert-base-multilingual-uncased-sentiment"
         model = AutoModelForSequenceClassification.from_pretrained(model_name)
         tokenizer = AutoTokenizer.from_pretrained(model_name)
         classifier = pipeline('sentiment-analysis', model=model, tokenizer=tokenizer)

In [41]: classifier('去扬州搓澡很开心')
Out[41]: [{'label': '5 stars', 'score': 0.45882534980773926}]

In [42]: classifier('在学校澡堂洗澡不开心')
Out[42]: [{'label': '1 star', 'score': 0.3665970265865326}]
```

可以看到，我们通过from_pretrained( )方法换了一个语言模型，这个新的模型可以评估情感的强度，由1星到5星。

Transformers提供了详尽的文档供大家学习：

https://huggingface.co/transformers/quicktour.html

——自己训练模型

说实话，在我接触的项目中，还没有是需要自己训练语言模型的。

第一个原因是网上开源的模型实在是太过强大。它们往往是用巨型语料库，海量GPU训练多时才得到的，例如打破人们常规认知的OpenAI GPT-3，使用了45TB的文本数据集，模型参数高达1750亿个，在各项NLP指标上横扫千军。我们虽然没法使用GPT-3，但GPT-2的效果也已经非常可怕了，足以满足我们要求。

第二个原因是训练模型的成本过高，训练一个可用的语言模型需要海量文本（我没有）、强大且浩如烟海的GPU（我没有）、高额电费（是的），因此我也一直没有机会自己训练。

因此，我们在狭窄数据集上训练出来的，更多是Toy Model，只能满足极特殊的场景。

鉴于各位和实验室的电脑性能都不太适合语言模型训练，这里给出一个colab的样例，有丰富的注释供大家参考：

https://colab.research.google.com/github/huggingface/blog/blob/master/notebooks/01_how_to_train.ipynb#scrollTo=M1oqh0F6W3ad