> *https://www.cnblogs.com/guoyaohua/p/9240336.html*

2013年，Google开源了一款用于词向量计算的工具——word2vec，引起了工业界和学术界的关注。首先，word2vec可以在百万数量级的词典和上亿的数据集上进行高效地训练；其次，该工具得到的训练结果——词向量（word embedding），可以很好地度量词与词之间的相似性。随着深度学习（Deep Learning）在自然语言处理中应用的普及，很多人误以为word2vec是一种深度学习算法。其实word2vec算法的背后是一个浅层神经网络。另外需要强调的一点是，word2vec是一个计算word vector的开源工具。当我们在说word2vec算法或模型的时候，其实指的是其背后用于计算word vector的 CBoW 模型和 Skip-gram 模型。很多人以为word2vec指的是一个算法或模型，这也是一种谬误。接下来，本文将从统计语言模型出发，尽可能详细地介绍word2vec工具背后的算法模型的来龙去脉。

Statistical Language Model

在深入word2vec算法的细节之前，我们首先回顾一下自然语言处理中的一个基本问题：**如何计算一段文本序列在某种语言下出现的概率？之所为称其为一个基本问题，是因为它在很多NLP任务中都扮演着重要的角色。** 例如，在机器翻译的问题中，如果我们知道了目标语言中每句话的概率，就可以从候选集合中挑选出最合理的句子做为翻译结果返回。

统计语言模型给出了这一类问题的一个基本解决框架。对于一段文本序列

$$S = w_1, w_2, \ldots, w_T$$

它的概率可以表示为：

$$P(S) = P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t | w_1, w_2, \ldots, w_{t-1})$$

即将序列的联合概率转化为一系列条件概率的乘积。问题变成了如何去预测这些给定previous words下的条件概率：

$$p(w_t | w_1, w_2, \ldots, w_{t-1})$$

由于其巨大的参数空间，这样一个原始的模型在实际中并没有什么用。我们更多的是采用其简化版本——Ngram模型：

$$p(w_t | w_1, w_2, \ldots, w_{t-1}) \approx p(w_t | w_{t-n+1}, \ldots, w_{t-1})$$

常见的如bigram模型（N=2）和trigram模型（N=3）。事实上，由于模型复杂度和预测精度的限制，我们很少会考虑N>3的模型。

我们可以用最大似然法去求解Ngram模型的参数——等价于去统计每个Ngram的条件词频。

为了避免统计中出现的零概率问题（一段从未在训练集中出现过的Ngram片段会使得整个序列的概率为0），人们基于原始的Ngram模型进一步发展出了back-off trigram模型（用低阶的bigram和unigram代替零概率的trigram）和interpolated trigram模型（将条件概率表示为unigram、bigram、trigram三者的线性函数）。此处不再赘述。感兴趣者可进一步阅读相关的文献[3]。

Distributed Representation

不过，Ngram模型仍有其局限性。首先，由于参数空间的爆炸式增长，它无法处理更长程的context（N>3）。其次，它没有考虑词与词之间内在的联系性。例如，考虑"the cat is walking in the bedroom"这句话。如果我们在训练语料中看到了很多类似“the dog is walking in the bedroom”或是“the cat is running in the bedroom”这样的句子，那么，即使我们没有见过这句话，也可以从“cat”和“dog”（“walking”和“running”）之间的相似性，推测出这句话的概率[3]。然而， Ngram模型做不到。

这是因为，Ngram本质上是将词当做一个个孤立的原子单元（atomic unit）去处理的。这种处理方式对应到数学上的形式是一个个离散的one-hot向量（除了一个词典索引的下标对应的方向上是1 ，其余方向上都是0）。例如，对于一个大小为5的词典：{"I", "love", "nature", "luaguage", "processing"}，“nature”对应的one-hot向量为：[0,0,1,0,0] 。显然，one-hot向量的维度等于词典的大小。这在动辄上万甚至百万词典的实际应用中，面临着巨大的维度灾难问题（The Curse of Dimensionality）

于是，人们就自然而然地想到，能否用一个**连续的稠密向量去刻画一个word的特征**呢？这样，我们不仅可以直接刻画词与词之间的相似度，还可以建立一个从向量到概率的平滑函数模型，使得相似的词向量可以映射到相近的概率空间上。这个稠密连续向量也被称为word的 **distributed representation**[3]。

事实上，这个概念在**信息检索（Information Retrieval）**领域早就已经被广泛地使用了。只不过，在IR领域里，这个概念被称为**向量空间模型（Vector Space Model，以下简称VSM）**。

VSM是基于一种Statistical Semantics Hypothesis[4]：**语言的统计特征隐藏着语义的信息**（Statistical pattern of human word usage can be used to figure out what people mean）。例如，两篇具有相似词分布的文档可以被认为是有着相近的主题。这个Hypothesis有很多衍生版本。其中，比较广为人知的两个版本是**Bag of Words Hypothesis**和**Distributional Hypothesis**。前者是说，**一篇文档的词频（而不是词序）代表了文档的主题**；后者是说，**上下文环境相似的两个词有着相近的语义**。后面我们会看到，**word2vec算法也是基于Distributional的假设**。

那么，VSM是如何将稀疏离散的 one-hot 词向量映射为稠密连续的Distributional Representation的呢？

简单来说，基于**Bag of Words Hypothesis**，我们可以构造一个**term-document**矩阵A：矩阵的行$A_{i,:}$：对应着词典里的一个word；矩阵的列$A_{:,j}$对应着训练语料里的一篇文档；矩阵里的元素$A_{i,j}$代表着wordw_i在文档D_j中出现的次数（或频率）。那么，我们就可以提取行向量做为word的语义向量（不过，在实际应用中，我们更多的是用列向量做为文档的主题向量）。

类似地，我们可以基于**Distributional Hypothesis**构造一个**word-context的矩阵**。此时，**矩阵的列变成了context里的word，矩阵的元素也变成了一个context窗口里word的共现次数**。

注意，这两类矩阵的行向量所计算的相似度有着细微的差异：term-document矩阵会给经常出现在同一篇document里的两个word赋予更高的相似度；而word-context矩阵会给那些有着相同context的两个word赋予更高的相似度。后者相对于前者是一种更高阶的相似度，因此在传统的信息检索领域中得到了更加广泛的应用。

不过，这种co-occurrence矩阵仍然存在着数据稀疏性和维度灾难的问题。为此，人们提出了一系列对矩阵进行降维的方法（如LSI / LSA等）。这些方法大都是基于SVD的思想，将原始的稀疏矩阵分解为两个低秩矩阵乘积的形式。

关于VSM更多的介绍，可以进一步阅读文末的参考文献[4]。

Neural Network Language Model

接下来，让我们回到对统计语言模型的讨论。鉴于Ngram等模型的不足，2003年，Bengio等人发表了一篇开创性的文章：A neural probabilistic language model[3]。在这篇文章里，他们总结出了一套用神经网络建立统计语言模型的框架（Neural Network Language Model，以下简称NNLM），并首次提出了**word embedding**的概念（虽然没有叫这个名

字），从而奠定了包括word2vec在内后续研究word representation learning的基础。

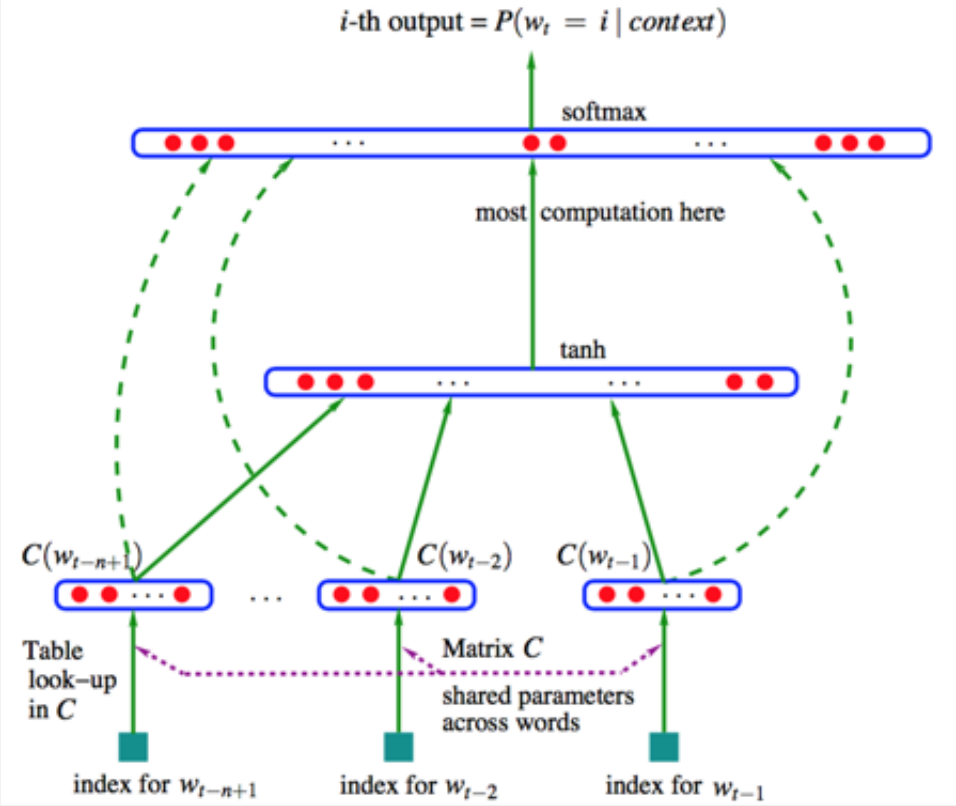
NNLM模型的基本思想可以概括如下：

1. 假定词表中的每一个word都对应着一个连续的特征向量；
2. 假定一个连续平滑的概率模型，输入一段词向量的序列，可以输出这段序列的联合概率；
3. **同时学习**词向量的权重和概率模型里的参数。

值得注意的一点是，这里的**词向量也是要学习的参数**。

在03年的论文里，Bengio等人采用了一个简单的前向反馈神经网络 $f(w_{t-n+1},\ldots,w_t)$ 来拟合一个词序列的条件概率$p(w_t|w_1,w_2,\ldots,w_{t-1})$。

整个模型的网络结构见下图：



我们可以将整个模型拆分成两部分加以理解：

1. 首先是一个线性的Embedding层。它将输入的N−1个one-hot词向量，通过一个共享的D×V的矩阵C，映射为N−1个分布式的词向量（distributed vector）。其中，V是词典的大小，D是Embedding向量的维度（一个先验参数）。**C矩阵里存储了要学习的word vector**。

2. 其次是一个简单的前向反馈神经网络g。它由一个tanh隐层和一个softmax输出层组成。通过将Embedding层输出的N−1个词向量映射为一个长度为V的概率分布向量，从而对词典中的word在输入context下的条件概率做出预估：

$$
\begin{aligned}
p(w_i|w_1,w_2,\ldots,w_{t-1}) \approx \\
f(w_i,w_{t-1},\ldots,w_{t-n+1}) = \\
g(w_i,C(w_{t-n+1}),\ldots,C(w_{t-1}))
\end{aligned}
$$

我们可以通过最小化一个cross-entropy的正则化损失函数来调整模型的参数θ：

$$
L(\theta) = \frac{1}{T}\sum_t \log f(w_t,w_{t-1},\ldots,w_{t-n+1}) + R(\theta)
$$

其中，模型的参数 θ 包括了Embedding层矩阵C的元素，和前向反馈神经网络模型g里的权重。这是一个巨大的参数空间。不过，在用SGD学习更新模型的参数时，**并不是所有的参数都需要调整（例如未在输入的context中出现的词对应的词向量）。计算的瓶颈主要是在softmax层的归一化函数**上（需要对词典中所有的word计算一遍条件概率）。

然而，抛却复杂的参数空间，我们不禁要问，为什么这样一个简单的模型会取得巨大的成功呢？

>>> Important

仔细观察这个模型就会发现，它其实在同时解决两个问题：**一个是统计语言模型里关注的条件概率$p(w_t|context)$的计算；一个是向量空间模型里关注的词向量的表达**。而这两个问题本质上并不独立。通过引入连续的词向量和平滑的概率模型，我们就可以在一个连续空间里对序列概率进行建模，从而从根本上**缓解数据稀疏性和维度灾难的问题**。另一方面，以条件概率 $p(w_t|context)$ 为学习目标去更新词向量的权重，具有更强的导向性，同时也与VSM里的Distributional Hypothesis不谋而合。

<<< Important

CBoW & Skip-gram Model

铺垫了这么多，终于要轮到主角出场了。

不过在主角正式登场前，我们先看一下NNLM存在的几个问题。

一个问题是，同Ngram模型一样，NNLM模型只能处理定长的序列。在03年的论文里，Bengio等人将模型能够一次处理的序列长度N提高到了5，虽然相比bigram和trigram已经是很大的提升，但依然缺少灵活性。

因此，Mikolov等人在2010年提出了一种RNNLM模型[7]，用递归神经网络代替原始模型里的前向反馈神经网络，并将Embedding层与RNN里的隐藏层合并，从而解决了变长序列的问题。

另一个问题就比较严重了。NNLM的训练太慢了。即便是在百万量级的数据集上，即便是借助了40个CPU进行训练，NNLM也需要耗时数周才能给出一个稍微靠谱的解来。显然，对于现在动辄上千万甚至上亿的真实语料库，训练一个NNLM模型几乎是一个impossible mission。

这时候，还是那个Mikolov站了出来。他注意到，原始的NNLM模型的训练其实可以拆分成两个步骤：

1. 用一个简单模型训练出连续的词向量；
2. 基于词向量的表达，训练一个连续的Ngram神经网络模型。而NNLM模型的计算瓶颈主要是在第二步。

如果我们只是想得到word的连续特征向量，是不是可以对第二步里的神经网络模型进行简化呢？

Mikolov是这么想的，也是这么做的。他在2013年一口气推出了两篇paper，并开源了一款计算词向量的工具——至此，word2vec横空出世，主角闪亮登场。

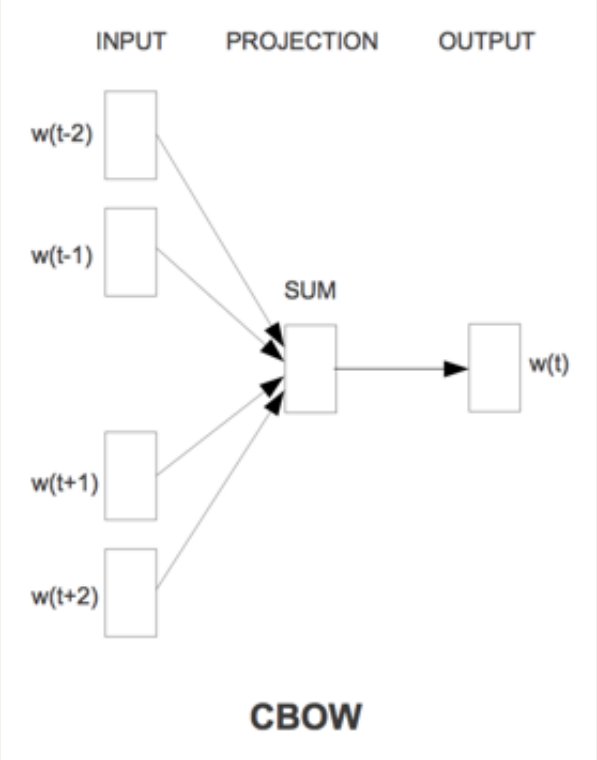
下面，我将带领大家简单剖析下word2vec算法的原理。有了前文的基础，理解word2vec算法就变得很简单了。

首先，我们对原始的NNLM模型做如下改造：

1. 移除前向反馈神经网络中非线性的hidden layer，直接将中间层的Embedding layer与输出层的softmax layer连接；

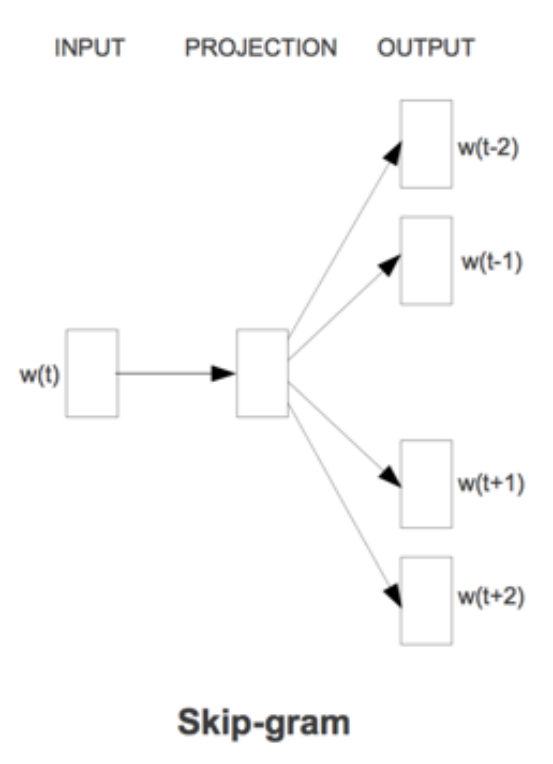
2. 忽略上下文环境的序列信息：输入的所有词向量均汇总到同一个Embedding layer；
3. 将Future words纳入上下文环境

得到的模型称之为**CBoW模型（Continuous Bag-of-Words Model）**，也是**word2vec算法的第一个模型**：



从数学上看，CBoW模型等价于一个词袋模型的向量乘以一个Embedding矩阵，从而得到一个连续的embedding向量。这也是CBoW模型名称的由来。

CBoW模型依然是从context对target word的预测中学习到词向量的表达。反过来，我们能否从**target word对context的预测中学习到word vector呢**？答案显然是可以的：



这个模型被称为**Skip-gram模型**（名称源于该模型在训练时会对上下文环境里的word进行采样）。

如果将Skip-gram模型的前向计算过程写成数学形式，我们得到：

$$p(w_o|w_i) = \frac{e^{U_o \cdot V_i}}{\sum_j e^{U_j \cdot V_i}}$$

其中，V_i 是Embedding层矩阵里的列向量，也被称为 w_i 的 input vector。U_j 是softmax层矩阵里的行向量，也被称为 w_i 的output vector.

$$p(w_o|w_i) = \frac{e^{U_o \cdot V_i}}{\sum_j e^{U_j \cdot V_i}}$$

因此，Skip-gram模型的本质是 **计算输入word的input vector与目标word的output vector之间的余弦相似度，并进行softmax归一化**。我们要学习的模型参数正是这两类词向量。

然而，直接对词典里的 V 个词计算相似度并归一化，显然是一件极其耗时的impossible mission。为此，Mikolov引入了两种优化算法：层次**Softmax（Hierarchical Softmax）**和负采样（**Negative Sampling**）。

Hierarchical Softmax[5]

层次Softmax的方法最早由Bengio在05年引入到语言模型中。**它的基本思想是将复杂的归一化概率分解为一系列条件概率乘积的形式**：

$$p(v|context) = \prod_{i=1}^{m} p(b_i(v)|b_1(v), \ldots, b_{i-1}(v), context)$$

其中，每一层条件概率对应一个二分类问题，可以通过一个简单的逻辑回归函数去拟合。这样，我们将对 V 个词的概率归一化问题，转化成了对$logV$个词的概率拟合问题。

我们可以通过构造一颗分类二叉树来直观地理解这个过程。首先，我们将原始字典D划分为两个子集D1、D2，并假设在给定context下，target word属于子集D1的概率
$p(w_t \in D_1|context)$服从logistical function的形式：

$$p(w_t \in D_1|context) = \frac{1}{1 + e^{-U_{D_{root}} \cdot V_{w_t}}}$$

其中， $U_{D_{root}}$ 和 V_{wt} 都是模型的参数。

接下来，我们可以对子集D1和D2进一步划分。**重复这一过程，直到集合里只剩下一个word**。这样，我们就将原始大小为V的字典D转换成了一颗深度为logV的二叉树。**树的叶子节点与原始字典里的word一一对应**；非叶节点则对应着某一类word的集合。**显然，从根节点出发到任意一个叶子节点都只有一条唯一路径——这条路径也编码了这个叶子节点所属的类别**。

同时，从根节点出发到叶子节点也是一个随机游走的过程。因此，我们可以基于这颗二叉树对叶子节点出现的似然概率进行计算。例如，对于训练样本里的一个target word w_t,假设其对应的二叉树编码为$\{1, 0, 1, \ldots, 1\}$，则我们构造的似然函数为：

$$p(w_t|context) = p(D_1 = 1|context)p(D_2 = 0|D_1 = 1) \ldots p(w_t|D_k = 1)$$

乘积中的每一项都是一个逻辑回归的函数。

我们可以通过最大化这个似然函数来求解二叉树上的参数——非叶节点上的向量，用来计算游走到某一个子节点的概率。

>>> Important

层次Softmax是一个很巧妙的模型。它通过构造一颗二叉树，将目标概率的计算复杂度从最初的V降低到了logV 的量级。不过付出的代价是人为增强了词与词之间的耦合性。例如，一个word出现的条件概率的变化，会影响到其路径上所有非叶节点的概率变化，间接地对其他word出现的条件概率带来不同程度的影响。 因此，构造一颗有意义的二叉树就显得十分重要。实践证明，在实际的应用中，基于Huffman编码的二叉树可以满足大部分应用场景的需求。

<<< Important

Negative Sampling[6]

负采样的思想最初来源于一种叫做Noise-Contrastive Estimation的算法[6]，原本是为了解决那些无法归一化的概率模型的参数预估问题。与改造模型输出概率的Hierarchical Softmax算法不同，NCE算法改造的是模型的似然函数。

以Skip-gram模型为例，其原始的似然函数对应着一个Multinomial的分布。在用最大似然法求解这个似然函数时，我们得到一个cross-entropy的损失函数：

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

式中的$p(w_{t+j}|w_t)$是一个在整个字典上归一化了的概率

而在NCE算法中，我们构造了这样一个问题：对于一组训练样本，我们想知道，target word的出现，是来自于context的驱动，还是一个事先假定的背景噪声的驱动？显然，我们可以用一个逻辑回归的函数来回答这个问题：

$$p(D=1|w, context) = \frac{p(w|context)}{p(w|context) + kp_n(w)} = \sigma(\log p(w|context) - \log kp_n(w))$$

这个式子给出了一个 target word w 来自于 context 驱动的概率。其中，k 是一个先验参数，表明噪声的采样频率。$p(w|context)$ 是一个非归一化的概率分布，这里采用 softmax 归一化函数中的分子部分。$p_n(w)$则是背景噪声的词分布。通常采用word的unigram分布。

通过对噪声分布的 k 采样，我们得到一个新的数据集。其中，label标记了数据的来源（真实数据分布还是背景噪声分布？）。在这个新的数据集上，我们就可以用最大化上式中逻辑回归的似然函数来求解模型的参数。

而Mikolov在2013年的论文里提出的**负采样算法，是NCE的一个简化版本。在这个算法里，Mikolov抛弃了NCE似然函数中对噪声分布的依赖，直接用原始softmax函数里的分子定义了逻辑回归的函数，进一步简化了计算**：

$$p(D=1|w_o,w_i)=\sigma(U_o\cdot V_i)$$

此时，模型相应的目标函数变为：

$$J(\theta)=\log\sigma(U_o\cdot V_i)+\sum_{j=1}^{k}E_{w_j\sim p_n(w)}[\log\sigma(-U_j\cdot V_i)]$$

除了这里介绍的层次Softmax和负采样的优化算法，Mikolov在13年的论文里还介绍了另一个trick：**下采样（subsampling）。其基本思想是在训练时依概率随机丢弃掉那些高频的词**：

$$p_{discard}(w)=1-\sqrt{\frac{t}{f(w)}}$$

其中，t 是一个先验参数，一般取为10^{-5}。$f(w)$ 是w在语料中出现的频率。

实验证明，这种下采样技术可以显著提高低频词的词向量的准确度。

Beyond the Word Vector

介绍完word2vec模型的算法和原理，我们来讨论一些轻松点的话题——模型的应用。

13年word2vec模型横空出世后，人们最津津乐道的是它学到的向量在语义和语法相似性上的应用——尤其是这种相似性居然对数学上的加减操作有意义[8]！最经典的一个例子是，$v(\text{”}\ King\ \text{”})-v(\text{”}\ Man\ \text{”})+v(\text{”}\ Woman\ \text{”})=v(\text{”}\ Queen\ \text{”})$。然而，这种例子似乎并没有太多实际的用途。

除此之外，word2vec模型还被应用于**机器翻译**和**推荐系统**领域。

Machine Translation[9]

与后来提出的在sentence level上进行机器翻译的RNN模型不同，word2vec模型主要是用于词粒度上的机器翻译。

具体来说，我们首先从大量的单语种语料中学习到每种语言的word2vec表达，再借助一个小的双语语料库学习到两种语言word2vec表达的线性映射关系W。构造的损失函数为：

$$J(W) = \sum_{i=1}^{n} ||Wx_i - z_i||^2$$

在翻译的过程中，我们**首先将源语言的word2vec向量通过矩阵 W 映射到目标语言的向量空间上；再在目标语言的向量空间中找出与投影向量距离最近的word做为翻译的结果返回**。

其原理是，不同语言学习到的word2vec向量空间在几何上具有一定的同构性。映射矩阵 W 本质上是一种空间对齐的线性变换。

Item2Vec[11]

本质上，word2vec模型是在word-context的co-occurrence矩阵基础上建立起来的。因此，任何基于co-occurrence矩阵的算法模型，都可以套用word2vec算法的思路加以改进。

比如，推荐系统领域的协同过滤算法。

协同过滤算法是建立在一个user-item的co-occurrence矩阵的基础上，通过行向量或列向量的相似性进行推荐。如果我们将同一个user购买的item视为一个context，就可以建立一个item-context的矩阵。进一步的，可以在这个矩阵上借鉴CBoW模型或Skip-gram模型计算出item的向量表达，在更高阶上计算item间的相似度。

关于word2vec更多应用的介绍，可以进一步参考这篇文献[10]。

Word Embedding

最后，我想简单阐述下我对Word Embedding的几点思考。不一定正确，也欢迎大家提出不同的意见。

Word embedding最早出现于Bengio在03年发表的开创性文章中[3]。通过嵌入一个线性的投影矩阵（projection matrix），将原始的one-hot向量映射为一个稠密的连续向量，并通过一个语言模型的任务去学习这个向量的权重。这一思想后来被广泛应用于包括word2vec在内的各种NLP模型中。

>>> Important

Word Embedding的训练方法大致可以分为两类：一类是无监督或弱监督的预训练；一类是端对端（end to end）的有监督训练。

无监督或弱监督的预训练以word2vec和auto-encoder为代表。这一类模型的特点是，不需要大量的人工标记样本就可以得到质量还不错的Embedding向量。不过因为缺少了任务导向，可能和我们要解决的问题还有一定的距离。因此，我们往往会在得到预训练的Embedding向量后，用少量人工标注的样本去fine-tune整个模型。

相比之下，端对端的有监督模型在最近几年里越来越受到人们的关注。与无监督模型相比，端对端的模型在结构上往往更加复杂。同时，也因为有着明确的任务导向，端对端模型学习到的Embedding向量也往往更加准确。例如，通过一个Embedding层和若干个卷积层连接而成的深度神经网络以实现对句子的情感分类，可以学习到语义更丰富的词向量表达。

>>> Important

Word Embedding的另一个研究方向是在更高层次上对Sentence的Embedding向量进行建模。

我们知道，word是sentence的基本组成单位。一个最简单也是最直接得到sentence embedding的方法是将组成sentence的所有word的embedding向量全部加起来——类似于CBoW模型。

显然，这种简单粗暴的方法会丢失很多信息。

另一种方法借鉴了word2vec的思想——将sentence或是paragraph视为一个特殊的word，然后用CBoW模型或是Skip-gram进行训练[12]。这种方法的问题在于，对于一篇新文章，总是需要重新训练一个新的sentence2vec。此外，同word2vec一样，这个模型缺少有监督的训练导向。

个人感觉比较靠谱的是第三种方法——基于word embedding的端对端的训练。Sentence本质上是word的序列。因此，在word embedding的基础上，我们可以连接多个RNN模型或是卷积神经网络，对word embedding序列进行编码，从而得到sentence embedding。

这方面的工作已有很多。有机会，我会再写一篇关于sentence embedding的综述。

References

[1]: Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, January 17). Efficient Estimation of Word Representations in Vector Space. arXiv.org.

[2]: Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013, October 17). Distributed Representations of Words and Phrases and their Compositionality. arXiv.org.

[3]: Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. The Journal of Machine Learning Research, 3, 1137–1155.

[4]: Turney, P. D., & Pantel, P. (2010). From frequency to meaning: vector space models of semantics. Journal of Artificial Intelligence Research, 37(1).

[5]: Morin, F., & Bengio, Y. (2005). Hierarchical Probabilistic Neural Network Language Model. Aistats.

[6]: Mnih, A., & Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation, 2265–2273.

[7]: Mikolov, T., Karafiát, M., Burget, L., & Cernocký, J. (2010). Recurrent neural network based language model. Interspeech.

[8]: Mikolov, T., Yih, W., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. Hlt-Naacl.

[9]: Mikolov, T., Le, Q. V., & Sutskever, I. (2013, September 17). Exploiting Similarities among Languages for Machine Translation. arXiv.org.

[10]: Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research, 12(Aug), 2493–2537.

[11]: Barkan, O., & Koenigstein, N. (2016, March 14). Item2Vec: Neural Item Embedding for Collaborative Filtering. arXiv.org.

[12]: Le, Q. V., & Mikolov, T. (2014, May 16). Distributed Representations of Sentences and Documents. arXiv.org.