这个系列介绍一下常见预训练模型的发展历程，按时间顺序主要分为三个阶段。这篇文章介绍一下Transformer出现之前的一些古早的算法（虽然实际应该不会再用了，但是面试还是会问的）；第二篇会介绍Transformer时代的GPT、BERT；第三篇介绍后BERT时代的XLNet、Roberta、ALBERT、Transformer-XL等。由于笔者不打算从事纯NLP工作，所以第三部分不会再介绍更加前沿的算法了。

---

1. **Latent Semantic Analysis, LSA： 草履虫时代**

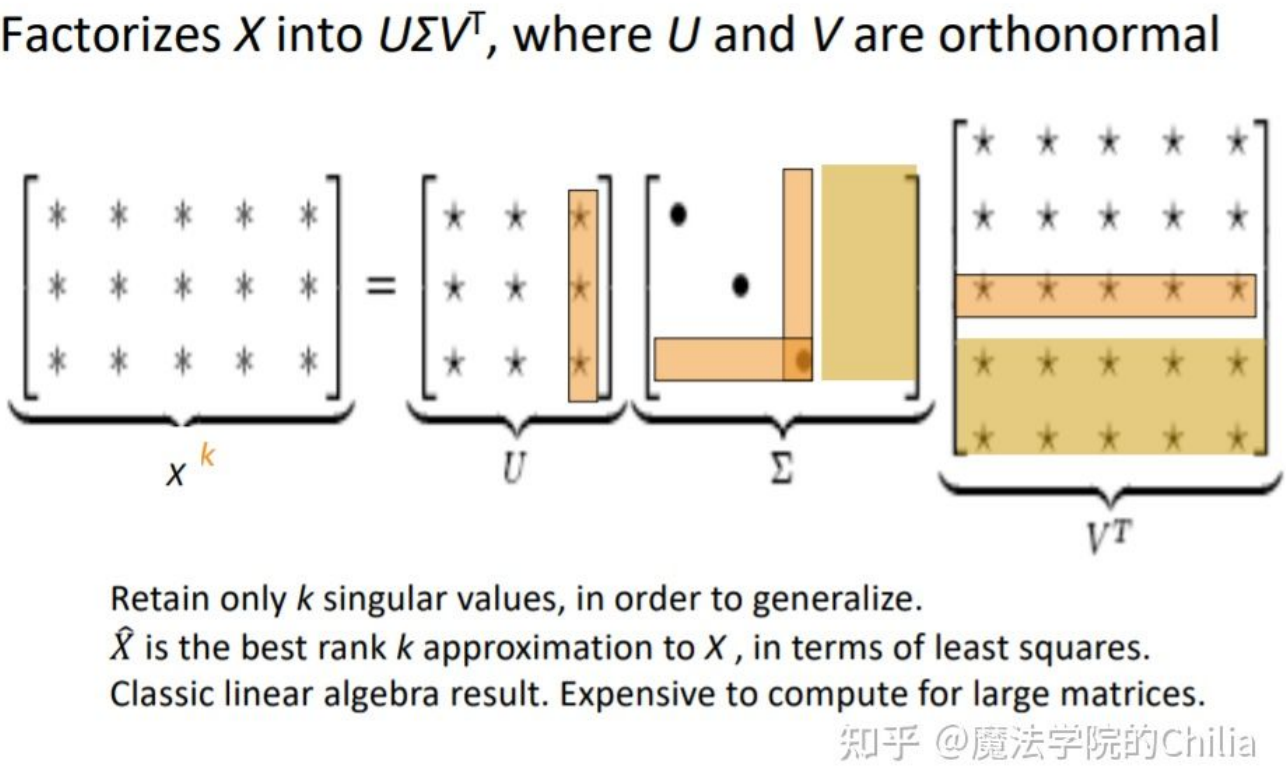
简单来说，就是使用滑窗来构建每个词与词之间的**共现矩阵**，之后再对矩阵进行SVD分解，找到最大的k个特征值对应的特征向量进行降维，并以此作为词语的稠密表示。这样就可以把握住词与词的相似性(a word's meaning is given by the words that frequently appear close-by).

例如：

- Example corpus:
  - 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

将这个稀疏矩阵进行SVD分解降维，得到**稠密**向量表示：



2. Word2vec[2013年]和GloVe[2015年]： 三叶虫时代

之前已经讲过了Word2vec，即是利用Skip-gram/CBOW两种任务，使用大规模语料集来最大化似然函数（最小化损失函数），最后得到每个词的预训练向量表示。

潜在语义分析（Latent Semantic Analysis）使用的这种**count based**模型与Word2Vec 这种**direct prediction**模型，它们各有优缺点。

Count based模型:

- 优点：有效的利用了统计信息
- 缺点：仅能表示词语的相似性，有的时候产生的word vector对于解释词的含义如word analogy等任务效果不好。(primarily used to capture word similarity)

Direct Prediction模型：

- 优点: 可以概括比相似性更为复杂的信息，进行word analogy等任务时效果较好. (capture complex patterns beyond word similarity)
- 缺点: 对**统计**信息利用的不够充分。

所以，GloVe就是要综合这两种算法的优点。"GloVe"是"**Glo**bal **Ve**ctor"的缩写，表示它可以有效的利用**全局统计信息**。这里, "全局统计信息"就是指在整个语料库中，一个词语在另一个词语周围出现的概率 (即共现概率)。

(1) 共现概率(co-occurrence probability):

如何有效的利用word-word co-occurrence count并能学习到词语背后的含义呢?

首先定义一些符号：对于矩阵 X，X_{ij} 代表了单词 j 出现在单词 i 上下文中的次数，则 $X_i = \sum_k X_{ik}$ 即代表所有出现在单词 i 的上下文中的单词次数。我们用 $P_{ij} = P(j|i) = X_{ij}/X_i$ 来代表单词 j 出现在单词 i 上下文中的概率，即co-occurrence probability。

用一个例子来解释如何用co-occurrence probability来表示词语含义：

	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9 x 10^{-4}	6.6 x 10^{-5}	3.0 x 10^{-3}	1.7 x 10^{-5}
$P(x\|\text{steam})$	2.2 x 10^{-5}	7.8 x 10^{-4}	2.2 x 10^{-3}	1.8 x 10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5 x 10^{-2}	1.36	0.96

例如我们想区分"ice"与"steam"，它们之间的关系可通过它们与不同单词 x 的co-occurrence probability的**比值**来描述。例如对于"solid"，"solid"出现在"ice"周围的概率为 1.9×10^{-4}；"solid"出现在"steam"周围的概率为 $2.2 \times 10{-}5$. 这两个概率*本身*没有什么意义，有意义的是它们的**比值** $\frac{P(solid|ice)}{P(solid|steam)}$.这个比值为8.9，是一个较大的值。这是因为solid更常用来描述ice的状态而不是steam的状态，所以在ice的上下文中出现几率较大；对于gas则恰恰相反；而对于"water"这种描述ice与steam均可、或者"fashion"这种与两者都没什么联系的单词，则比值接近于1。

(2) GloVe 模型

文章直接给出了GloVe模型的损失函数：

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \hat{w}_j + b_i + \hat{b}_j - logX_{ij})^2$$

了理解这个损失函数，我们先看最后一项

$$w_i^T \hat{w}_j + b_i + \hat{b}_j - logX_{ij}$$

。其中：

- w_i 是词语 i 作为中心词(如上例中的ice, steam)的向量表示，是我们要学习的参数。
- $\hat{w}_j$ 是词语 j 作为周围词（如上例中的solid, gas, water, fashion）的向量表示，是我们要学习的参数。
- b_i , $\hat{b_j}$ 是我们要学习的偏置参数。
- X_{ij} 是周围词 j 出现在中心词 i 周围的次数。

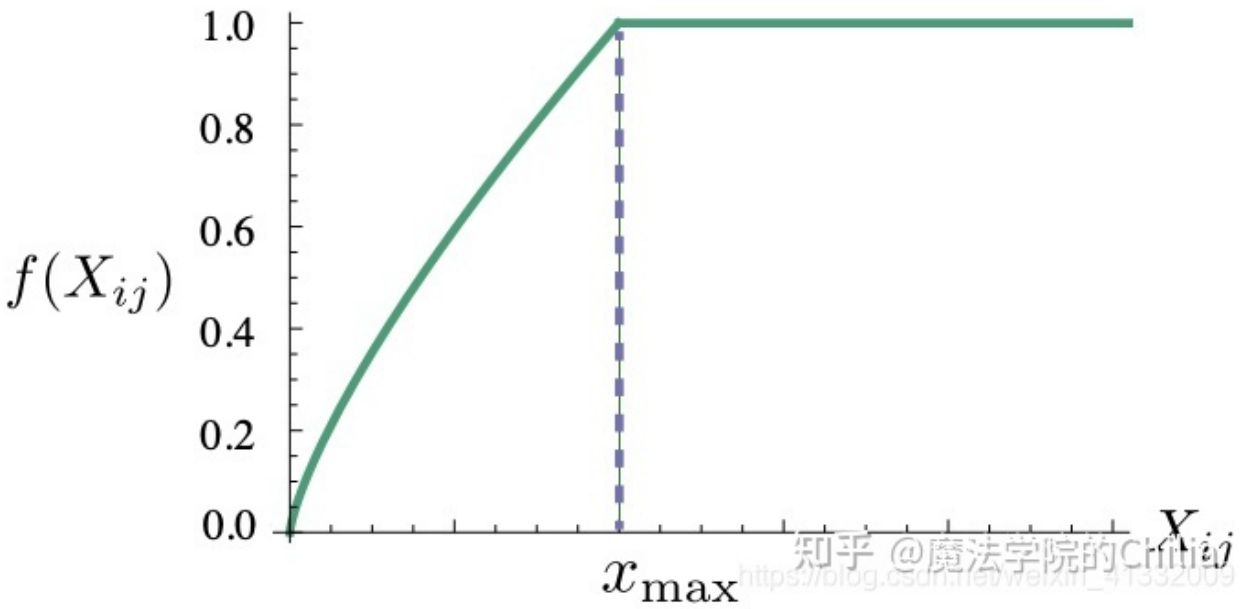
以这个作为损失是因为我们希望

$$w_i^T \hat{w}_j + b_i + \hat{b}_j = logX_{ij}$$

.

$f(X_{ij})$是个权值，加上这个权值是因为需要给那些较少发生的co-occurrence较小的权重。文中取的函数形式为：

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$



最后再对词汇表中的词两两求一次损失，将所有的损失加起来得到最终的损失即可。

（3）GloVe与word2vec比较

word2vec是**局部**语料库训练的，其特征提取是基于滑窗的；而glove的滑窗是为了构建co-occurance matrix，是基于**全局**语料的，可见glove需要事先统计共现概率；因此，word2vec可以进行**在线**学习，glove则需要**统计固定语料信息。**

word2vec损失函数是交叉熵；glove的损失函数是最小平方损失函数。

虽然GloVe的作者在原论文中说GloVe结合了SVD与Word2Vec的优势，训练速度快并且在各项任务中性能优于Word2Vec，但是我们应该持有怀疑的态度看待这一结果，可能作者在比较结果时对于GloVe模型参数选择较为精细而Word2Vec参数较为粗糙导致GloVe性能较好，或者换另一个数据集，改换样本数量，两者的性能又会有不同。实际上，在另一篇论文*Evaluation methods for unsupervised word embeddings* 中基于各种intrinsic和extrinsic任务的性能比较中，Word2Vec结果要优于或不亚于GloVe。

（4）GloVe与SVD比较

LSA（Latent Semantic Analysis）可以基于co-occurance matrix构建词向量，实质上是基于全局语料采用SVD进行矩阵分解，然而SVD计算复杂度高；glove可看作是对LSA一种**优化的高效矩阵分解算法**，采用Adagrad对最小平方损失进行优化：

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \hat{w}_j + b_i + \hat{b}_j - logX_{ij})^2$$

3. Fasttext[2016年]：甲胄鱼类时代

论文：Enriching Word Vectors with Subword Information

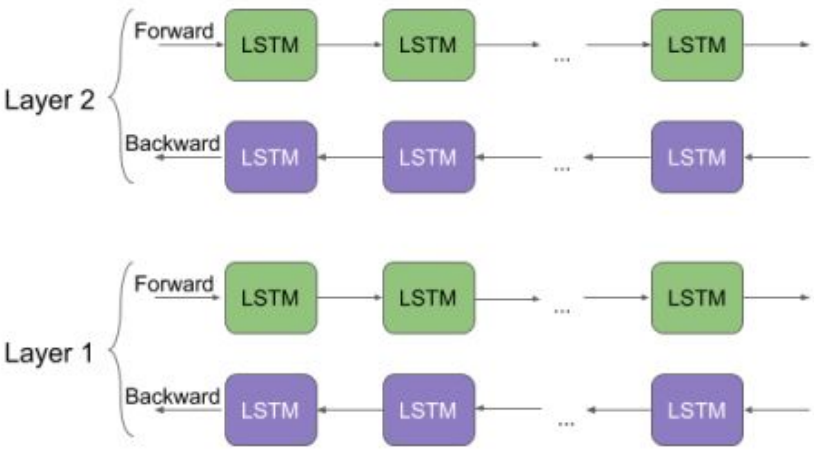
第一次提出用subword(char n-gram)来进行skip-gram训练，并将一个word的embedding用subword embedding求和来表示。使用subword而不是用空格分隔的word也成为之后预训练模型的趋势。

4. Elmo[2018年]：小巧聪明的伤齿龙

论文：Deep contextualized word representations

"contextualized"指的就是Elmo能够根据不同的**语境**给相同的词以不同的embedding，这就解决了之前这些方法都不能解决的"一词多义"问题。Elmo在SQuAD, NER, and SST都取得了比较好的效果。

具体来说，Elmo使用了双层、双向LSTM，用language model作为预训练任务：



- 首先，使用character-level CNN来把输入的每个词打成向量，输入到第一层双向LSTM中
- 在forward pass和backward pass中使用language model作为预训练的task，其损失函数如下：

$$\sum_{k=1}^{N} \Big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \Big)$$

前文 token embedding softmax 后文

- forward pass和backward pass的输出结果再输入到下一层LSTM中。用多层LSTM的原因是，用原文的话来讲，"...higher level LSTM states capture context-dependent aspects of word *meaning*...while lower-level states model aspects of *syntax*(e.g., they can be used to do POS)"
- 最后Elmo的输出就是raw word vectors与两层LSTM的前向、后向输出vector的求和。

对于下游任务，Elmo的做法是冻结LSTM的参数，只把输出的词语向量表示输入到下游任务的模型中。