

深度长文：NLP的巨大肩膀（上）

原创：许维 PaperWeekly 前天

我们都知道，牛顿说过一句名言 "*If I have seen further, it is by standing on the shoulders of giants*". 无可否认，牛顿取得了无与匹敌的成就，人类历史上最伟大的科学家之一，但同样无可否认的是，牛顿确实吸收了大量前人的研究成果，诸如哥白尼、伽利略和开普勒等人，正因如此，联合国为了纪念伽利略首次将望远镜用作天文观测四百周年，2009年的时候，通过了“国际天文年”的决议，并选中《巨人的肩膀》(Shoulders Of Giants) 作为主题曲。我想这大概是告诉后人，“吃水不忘挖井人”，牛顿的成就固然伟大，他脚下的“巨人肩膀”伽利略也同样不能忘了。

也许，“巨人肩膀”无处不在，有些人善于发现，有些人选择性失明，而牛顿的伟大之处在于他能够发现，这是伟大其一，更为重要的是，他还能自己造了梯子爬上“巨人的肩膀”，并成为一个新的“巨人肩膀”，这是伟大其二。

而回到这篇文章的主题，作为平凡人的我们，暂且先把如何发现并造了梯子云云撇开不谈，让我们先来捋一捋现在 NLP 当中可能的“巨人肩膀”在哪里，以及有哪些已经造好的“梯子”可供攀登，余下的，就留给那些立志成为“巨人肩膀”的人文志士们吧。

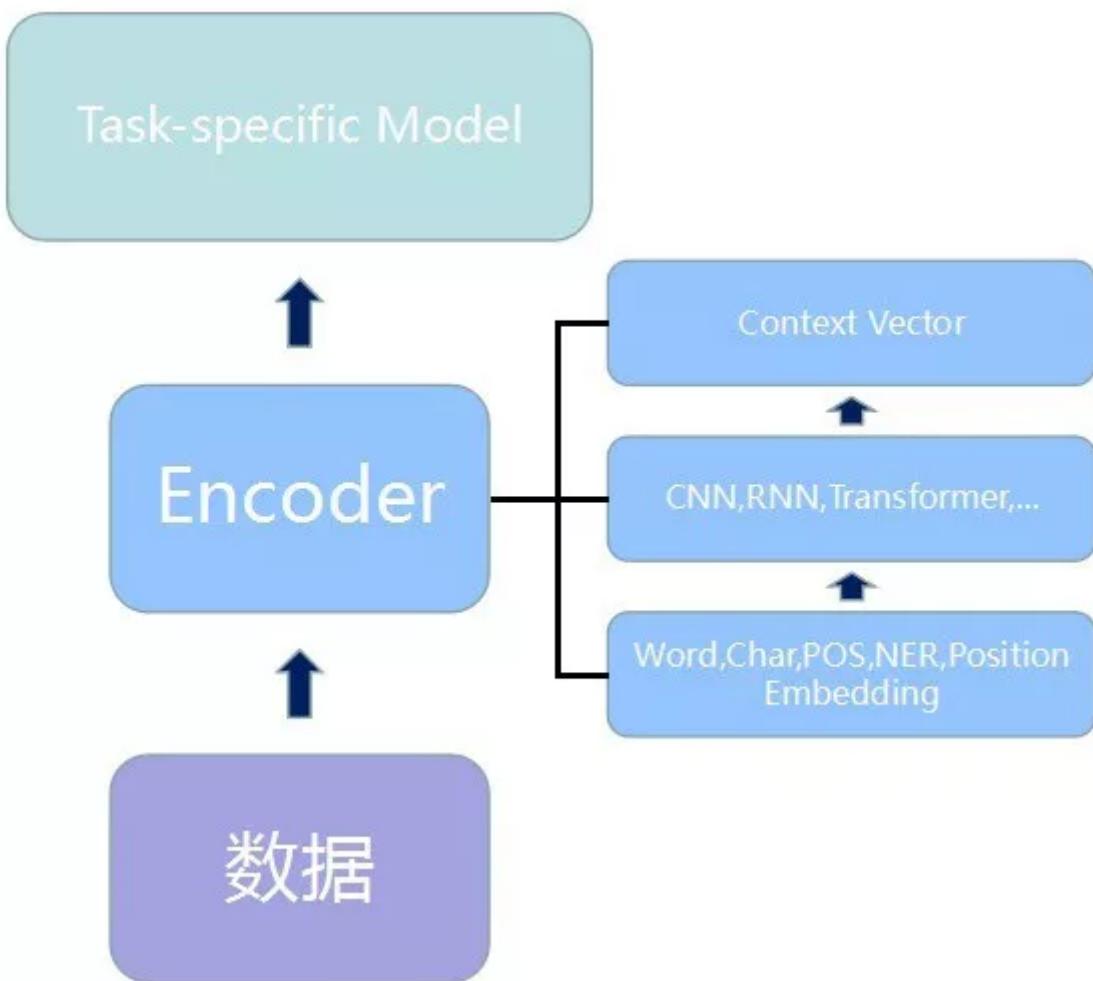
作者 | 许维

单位 | 腾讯知文

研究方向 | 智能客服、问答系统

通常来说，NLP 中监督任务的基本套路都可以用三个积木来进行归纳：

- 文本数据搜集和预处理
- 将文本进行编码和表征
- 设计模型解决具体任务



其中数据处理阶段自不用说，各个任务按照各自的逻辑去处理和得到相应的输入。而其中的第二阶段 Encoder 模块与第三阶段的 Task-specific Model 模块，通常来说，界限并不是特别清晰，二者之间互有渗透。

回顾过去基于深度学习的 NLP 任务可以发现，几乎绝大多数都比较符合这三层概念。比如很多生成任务的 Seq2Seq 框架中不外乎都有一个 Encoder 和一个 Decoder。对应到这里，Decoder 更像是一个 Task-specific Model，然后相应的将 Encoder 做一些细微调整，比如引入 Attention 机制等等。

对于一些文本分类任务的结构，则 Encoder 模块与 Task-specific Model 模块的区别更为明显和清晰，Encoder 负责提取文本特征，最后接上一些全连接层和 Softmax 层便可以当做 Task-specific Model 模块，如此便完成了一个文本分类任务。

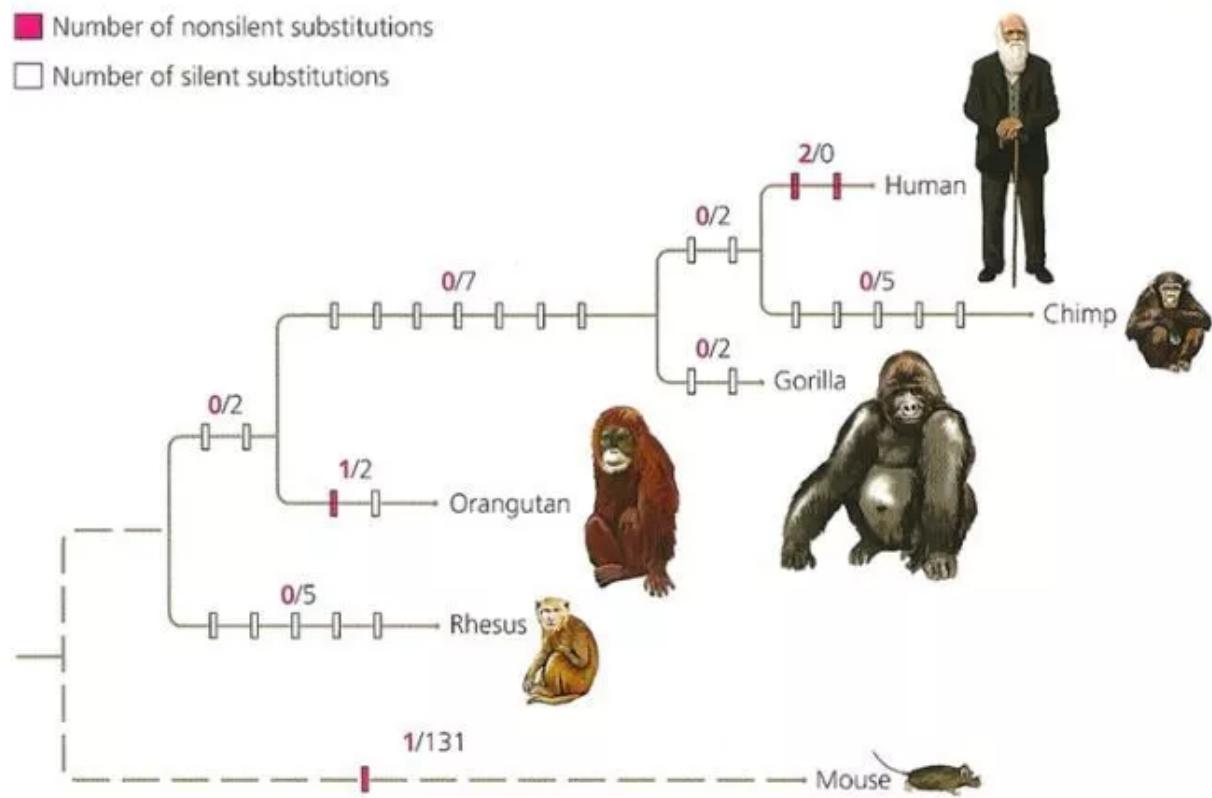
既然很多的 NLP 任务都可以用这三个模块来进行归纳，并且其中数据层和 Task-specific Model 模块层可能根据具体任务的不同需要做一些相应设计，而 Encoder 层便可以作为一个相对比较通用的模块来使用。那么自然会有一个想法，能不能类似于图像领域中的 ImageNet 上预训练的各种模型，来做一个 NLP 中预训练好的 Encoder 模块，然后我拿来直接利用就好了？

应该说，这个想法并不难想到，NLP 人也花了一些时间去思考究竟该如何设计一些更通用的可以迁移利用的东西，而不是所有的任务都要“from scratch”。因为如何尽量利用已有的知识、经验和工具，避免重复造轮子，想尽一切办法“站在巨人的肩膀上”快速发展，我想大概也是最朴素的“发展是硬道理”的体现。

为了更好地厘清这个思路，我们先来看看人类和 CV (Computer Vision, 计算机视觉) 曾经都是如何“站在巨人的肩膀上”的。

人类的巨大肩膀

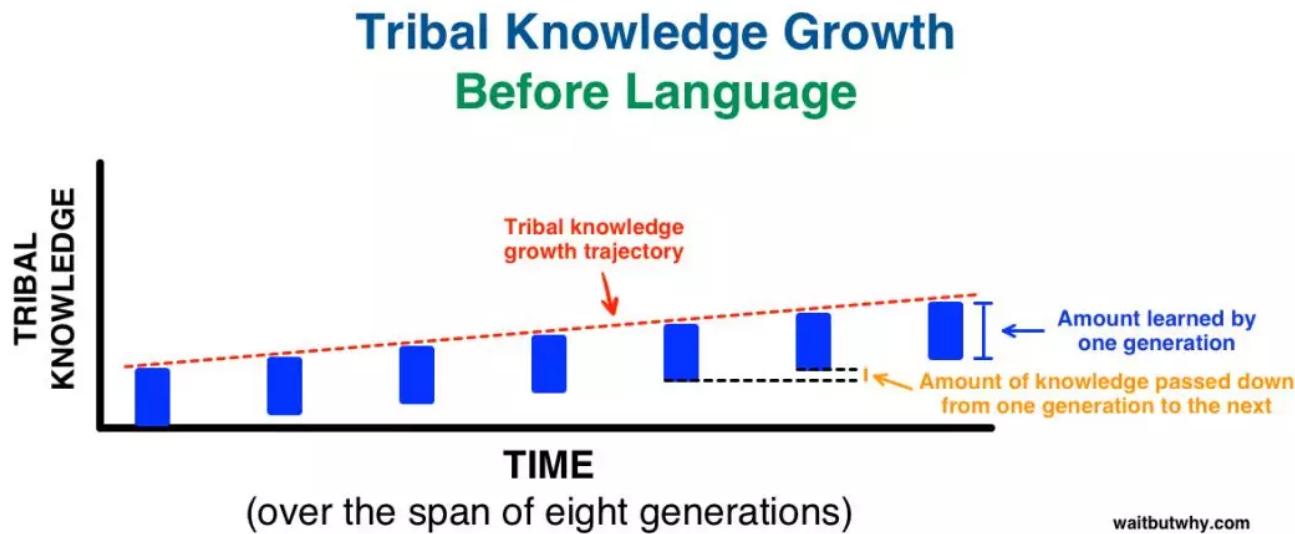
大约在 20 万年前，人类祖先 FOXP2 基因的 2 处（相对于其他原始猿类，如下图）极其微小却又至为关键的突变，让人类祖先逐渐拥有了语言能力，从此人类逐渐走上了一条不同于其他所有动物的文明演化之路。



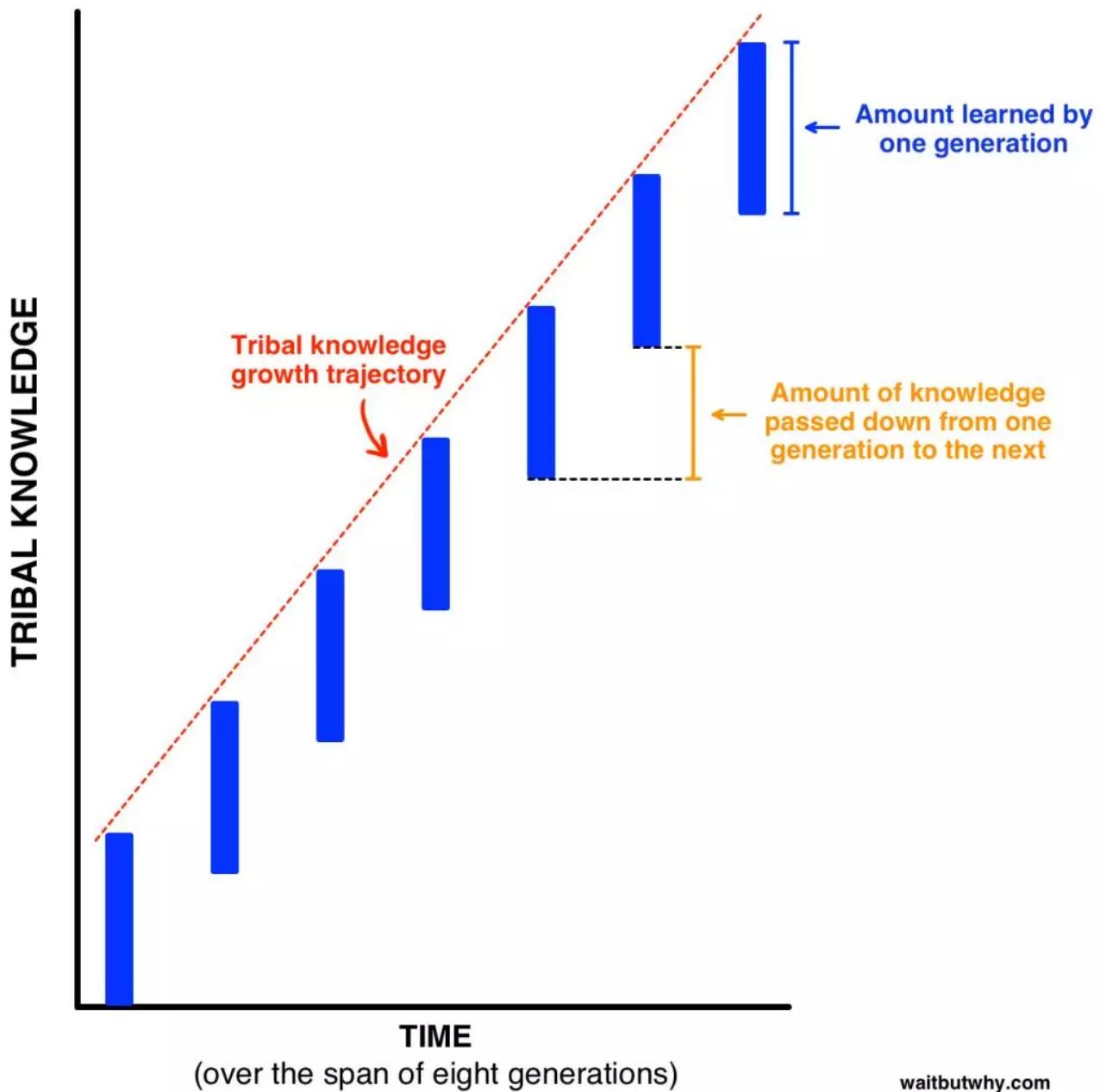
而人类语言以及随后产生的文字也是人类区别于其他动物的一个至关重要的特征，它使得人类协同合作的能力变得极为强大，且让代际间经验与文化的传承效率大大提升。知名博主 Tim Urban——连大佬 Elon Musk 都是他的铁杆粉丝——在 2017 年 4 月发表的巨长博文中（其实也是为了商业

互吹 Musk 的 Neuralink），Tim 分析了为什么语言能够加速人类文明的发展，并最终从几十万年前智能生命竞赛跑道上所有一切潜在对手中大比分强势胜出。

在这个过程中，语言起了非常关键的作用。在语言产生前，人类社会发展非常缓慢，表现为代际间的传承效率非常低下，而自从语言诞生以后，人类社会的发展速度得到极大的提升，这主要体现在父辈的生存技能和经验能够通过快速有效的方式传承给子辈，不仅如此，在同辈之间通过语言的沟通和交流，宝贵的经验也能够通过语言这种高效的媒介迅速传播到原始部落的每一个角落。于是，人类的每一代都能够得以在他们的父辈打下的江山上，一日千里，终成这颗蓝色星球上的无二霸主。



Tribal Knowledge Growth After Language



waitbutwhy.com

不过，我觉得 Urban 想说却并没有说出来的是，即便在人类语言诞生之前，人类祖先也可以通过可能已经先于语言而诞生的学习与认知能力，做到以“代”为单位来进行传承与进化，只不过不同于基因进化，这是一种地球生命全新的进化方式，在效率上已经比其他生物的进化效率高的多得多。

地球上自生命诞生以来一直被奉为圭臬的基因进化法则，往往都是以一个物种为单位，上一代花了生命代价学习到的生存技能需要不断的通过非常低效的“优胜劣汰，适者生存”的丛林法则，写进到该物种生物的基因中才算完事，而这往往需要几万年乃至几百万年才能完成。而在这个过程中，比其他物种强得多的学习能力是人类制胜的关键。

上面两个事实，前者说明了语言是加速文明进化的润滑剂，而后者说明了强大的学习能力是人类走出一条有人类特色的发展之路，从而脱离基因进化窠臼的最为重要的因素。

也就是说，对于人类而言，他们的父辈，同辈，以及一切同类，乃至大自然万事万物都是他们的“巨人肩膀”；而语言和学习能力则是人类能够站上“巨人肩膀”的“梯子”。

回到本文的主题，对于人类的钢铁“儿子”AI来说，CV 和 NLP 是当前 AI 最火爆的两个领域之二，一个要解决钢铁“儿子”的视觉功能，一个要解决钢铁“儿子”的语言或认知能力，那么什么又是这个钢铁“儿子”的“巨人肩膀”和“梯子”呢？我们先来看看 CV 中的情况。

CV的巨人肩膀

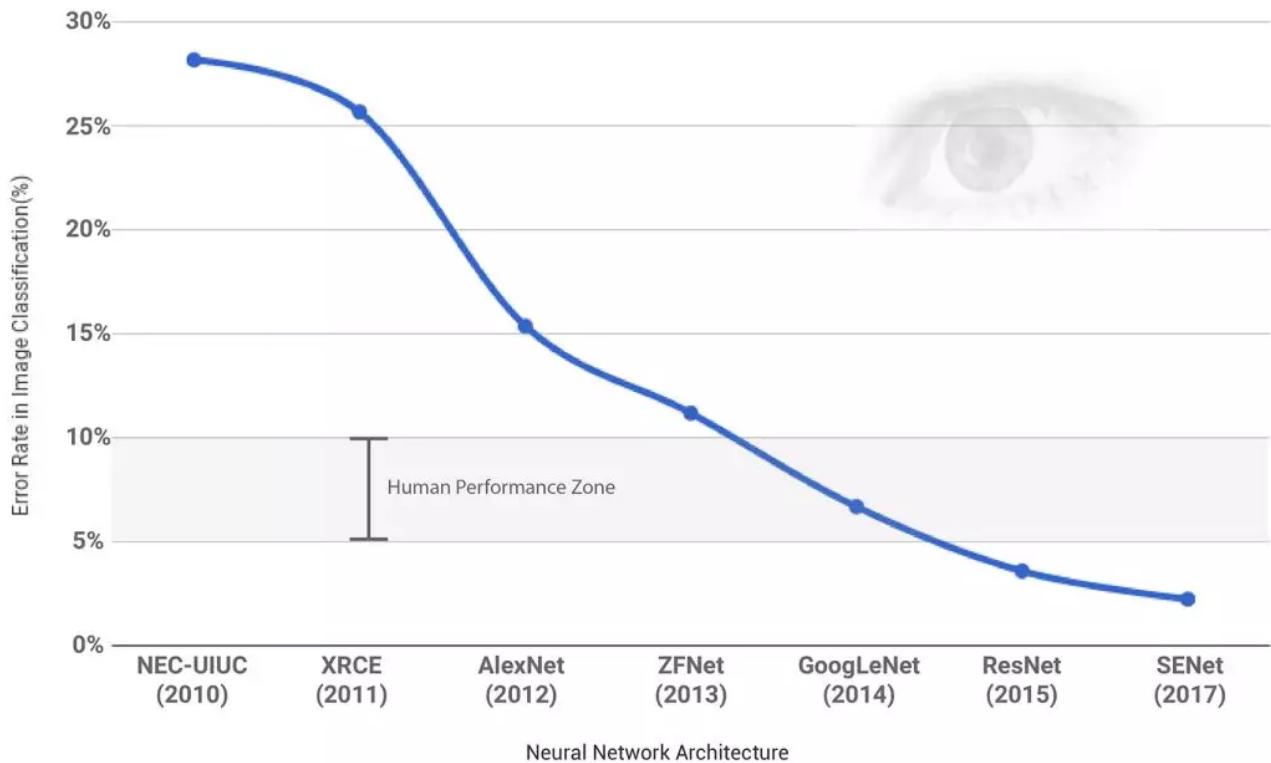
ImageNet 是 2009 年由李飞飞团队邓嘉等人提出，并迅速发展成为 CV 领域最知名的竞赛 ILSVRC。从 2010 年举办第一届，到 2017 年李飞飞宣布最后一届，前后总共举办 8 年，这八年间先后在这个比赛中涌现了一大批推动 AI 领域尤其是 CV 领域大发展的算法和模型。特别值得一提的是 2012 年 Hinton 团队提出了 AlexNet，超过当时第二名效果 41%，一下子引爆了 AI 领域，因此 2012 年也被称为“深度学习元年”。

ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
 - 1.2 M train
 - 100k test.

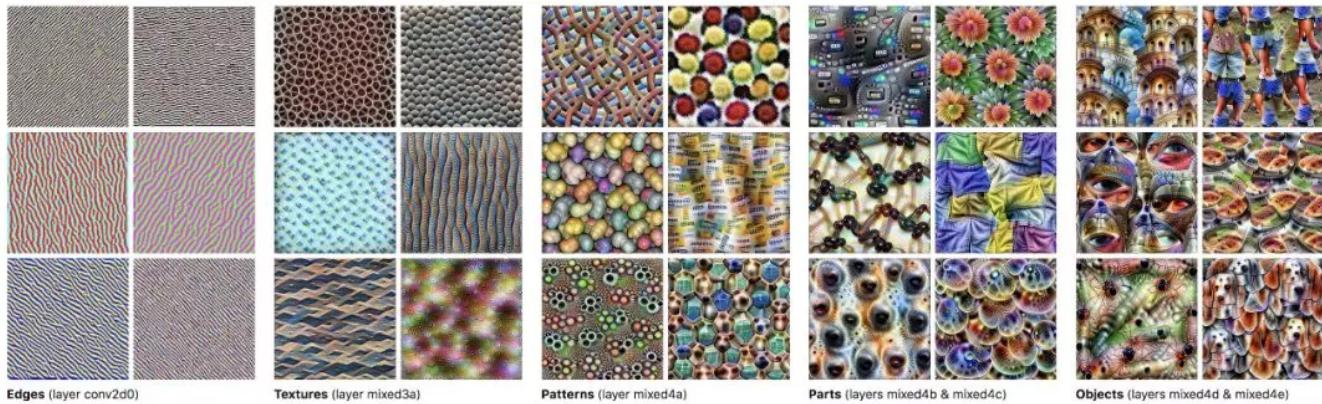




随之而来，大家发现如果用已经在 ImageNet 中训练好的模型，并用这些模型中的参数来初始化新任务中的模型，可以显著的提升新任务下的效果。这种站在“巨人肩膀”上的方法已经被成功运用到很多 CV 任务中，诸如物体检测和语义分割等。不仅如此，更重要的是，这种充分使用预训练模型的方法可以非常便利地迁移到一些获取标注数据较为困难的新场景中，从而极大的改善模型对标注数据数量的要求，并降低标注数据的成本。

因此，利用大规模数据集预训练模型进行迁移学习的方法被认为是 CV 中的标配。以至于 2018 年的早些时候，大神何凯明所在的 FAIR 团队利用 Instgram 中数十亿张带有用户标签的图片进行预训练，而后将其在 ImageNet 的比赛任务中进行 fine-tune，取得了最好的成绩 (arXiv:1805.00932)。只不过，由于预训练的数据过于庞大，该工作动用了 336 块 GPU 预训练了 22 天，不得不说实在都是土豪们才玩得动的游戏，这一点和下文要介绍的 NLP 中的预训练步骤何其相似。

不过为何这种预训练的模式能够有效？这背后有什么更深刻的内涵吗？为此，Google Brain 团队将 2014 年的 ImageNet 冠军 GoogleNet 的中间层进行了可视化，可以发现模型的较低层学到的主要是物体的边缘，往高层后逐步就变成了成型的物体了。一般来说，物体的边缘和纹路都是一些比较通用的视觉特征，因此将这一部分对应的模型参数用来初始化 task-specific 模型中的参数，意味着模型就不需要再从头开始学习这些特征，从而大大提升了训练效率和性能。



总结起来就是，CV 中的“巨人肩膀”是 ImageNet 以及由之而来 Google 等公司或团队在大规模数据集上预训练得到的模型，而“梯子”便是 transfer learning 之下的 fine-tuning。

寻找NLP的巨人肩膀

和 CV 领域中深度学习的惊艳表现相比，对于 NLP 任务来讲，深度学习的应用一直没有带来让人眼前特别一亮的表现。ImageNet 中的图片分类任务，深度学习早已超越人类的分类准确率，而这一目标对于 NLP 中的深度学习来说，似乎成了不太可能完成的任务，尤其是在那些需要深层语义理解的任务当中，更是如此。

但即便如此，困难从来未曾阻止人类想要教给他“儿子”理解“长辈”的话并开口说“人话”的雄心，忧心忡忡的人类家长恨不得也给 AI 来一次 FOXP2 基因突变——正像 20 万年前上帝的一次神来之笔给人类带来了语言能力一样。

2018 年 9 月，DeepMind 主办的 Deep Learning Indaba 2018 大会在南非举行，**ULMFit** 的作者之一 Sebastian Ruder 在大会上做了一个很精彩的名为 **Frontiers of Natural Language Processing** 的报告，前后分为两个部分：第一部分梳理近些年 NLP 的发展；第二部分探讨了当前 NLP 遇到的一些困难。

在参考这个报告的同时，回到本文最开头，这里将主要着重于 NLP 中最为重要的 Encoder 模块，并抛去具体的模型之争（诸如 CNN、RNN 和 Transformer 等），想要努力梳理出一条 NLP 任务中如何更有效站上“巨人肩膀”的一些模式出来。

本质上，自然语言理解 NLU 的核心问题其实是如何从语言文字的表象符号中抽取出来蕴含在文字背后的真实意义，并将其用计算机能够读懂的方式表征出来。当然这通常对应的是数学语言，表征是如此重要，以至于 2012 年的时候 Yoshua Bengio 作为第一作者发表了一篇表征学习的综述 **Representation Learning: A Review and New Perspectives**，并随后在 2013 年和深度

学习三大巨头的另一位巨头 Yann LeCun 牵头创办 ICLR，这一会议至今才过去 5 年时间，如今已是 AI 领域最负盛名的顶级会议之一。可以说，探究 NLP 或 NLU 的历史，同样也是探究文本如何更有效表征的历史。

梯子出现之前

犹如生命的诞生之初，混沌而原始。在 word2vec 诞生之前，NLP 中并没有一个统一的方法去表示一段文本，各位前辈和大师们发明了许多的方法：从 one-hot 表示一个词到用 bag-of-words 来表示一段文本，从 k-shingles 把一段文本切分成一些文字片段，到汉语中用各种序列标注方法将文本按语义进行分割，从 tf-idf 中用频率的手段来表征词语的重要性，到 text-rank 中借鉴 page-rank 的方法来表征词语的权重，从基于 SVD 纯数学分解词文档矩阵的 LSA，到 pLSA 中用概率手段来表征文档形成过程并将词文档矩阵的求解结果赋予概率含义，再到 LDA 中引入两个共轭分布从而完美引入先验……

语言模型

而以上这些都是相对比较传统的方法，在介绍基于深度学习的方法之前，先来看看语言模型。实际上，语言模型的本质是对一段自然语言的文本进行预测概率的大小，即如果文本用 S_i 来表示，那么语言模型就是要求 $P(S_i)$ 的大小，如果按照大数定律中频率对于概率无限逼近的思想，求这个概率大小，自然要用这个文本在所有人类历史上产生过的所有文本集合中，先求这个文本的频率 $P(S_i)$ ，而后便可以通过如下公式来求得：

$$P(S_i) = \frac{c(S_i)}{\sum_{j=0}^{\infty} c(S_j)}$$

这个公式足够简单，但问题是全人类所有历史的语料这种统计显然无法实现，因此为了将这个不可能的统计任务变得可能，有人将文本不做一个整体，而是把它拆散成一个个的词，通过每个词之间的概率关系，从而求得整个文本的概率大小。假定句子长度为 T ，词用 x 表示，即：

$$P(S_i) = P(x_0, x_1, \dots, x_T) = P(x_0)P(x_1|x_0)P(x_2|x_0, x_1)\dots P(x_T|x_0, x_1, x_2, \dots, x_{T-1})$$

然而，这个式子的计算依然过于复杂，我们一般都会引入马尔科夫假设：假定一个句子中的词只与它前面的 n 个词相关，特别地，当 $n=1$ 的时候，句子的概率计算公式最为简洁：

$$P(S_i) = P(x_0, x_1, \dots, x_T) = P(x_0)P(x_1|x_0)P(x_2|x_1)\dots P(x_T|x_{T-1}) = P(x_0)\prod_{i=0}^{T-1} P(x_{i+1}|x_i)$$

并且把词频的统计用来估计这个语言模型中的条件概率，如下：

$$P(x_{i+1}|x_i) = \frac{c(x_{i+1}, x_i)}{c(x_i)}$$

这样一来，语言模型的计算终于变得可行。然而，这种基于统计的语言模型却存在很多问题：

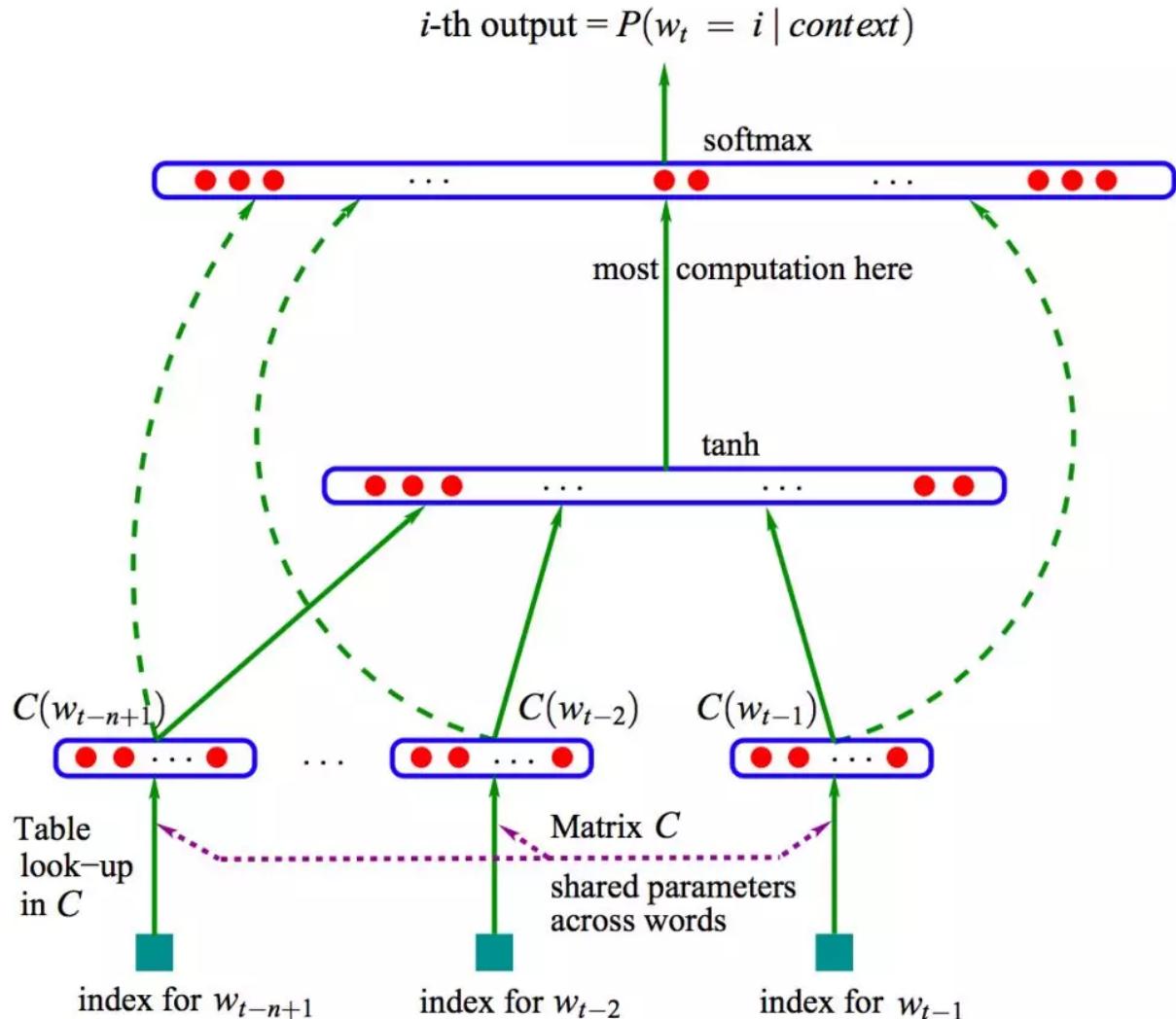
第一，很多情况下 $c(x_{i+1}, x_i)$ 的计算会遇到特别多零值，尤其在 n 取值较大时，这种数据稀疏导致的计算为 0 的现象变得特别严重。所以统计语言模型中一个很重要的方向便是设计各种平滑方法来处理这种情况。

第二，另一个更为严重的问题是，基于统计的语言模型无法把 n 取得很大，一般来说在 3-gram 比较常见，再大的话，计算复杂度会指数上升。这个问题的存在导致统计语言模型无法建模语言中上下文较长的依赖关系。

第三，统计语言模型无法表征词语之间的相似性。

NNLM

这些缺点的存在，迫使 2003 年 Bengio 在他的经典论文 ***A Neural Probabilistic Language Model*** 中，首次将深度学习的思想融入到语言模型中，并发现将训练得到的 NNLM (Neural Net Language Model, 神经网络语言模型) 模型的第一层参数当做词的分布式表征时，能够很好地获取词语之间的相似度。



撇去正则化项，NNLM 的极大目标函数对数似然函数，其本质上是个 N-Gram 的语言模型，如下所示：

$$L = \frac{1}{T} \sum_t \log P(w_t | w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta)$$

$$P(w_t | w_{t-1}, \dots, w_{t-n+1}; \theta) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

其中，归一化之前的概率大小（也就是 logits）为：

$$y = b + Wx + Utanh(d + Hx)$$

$$x = (C(w_{t-1}), C(w_{t-2}), \dots, C(w_{t-n+1}))$$

x 实际上就是将每个词映射为 m 维的向量，然后将这 $n-1$ 个词的向量 concat 起来，组合成一个 $(n-1)*m$ 维的向量。

这里可以将 **NNLM** 的网络结构拆分为三个部分：第一部分，从词到词向量的映射，通过 C 矩阵完成映射，参数个数为 $|V| * m$ ；第二部分，从 x 到隐藏层的映射，通过矩阵 H ，这里的参数个数为 $|H| * m * (n-1)$ ；第三部分，从隐藏层到输出层的映射，通过矩阵 U ，参数个数为 $|V| * |H|$ ；第四部分，从 x 到输出层的映射，通过矩阵 W ，参数个数为 $|V| * m * (n-1)$ 。

因此，如果算上偏置项的参数个数（其中输出层为 $|V|$ ，输入层到隐藏层为 $|H|$ ）的话，NNLM 的参数个数为：

$$|V|(1 + |H| + mn) + |H|(1 + mn - m)$$

可见 NNLM 的参数个数是所取窗口大小 n 的线性函数，这便可以让 NNLM 能对更长的依赖关系进行建模。不过 NNLM 的最主要贡献是非常有创见性地将模型的第一层特征映射矩阵当做词的分布式表示，从而可以将一个词表征为一个向量形式，这直接启发了后来的 word2vec 的工作。

这许多的方法和模型正如动荡的春秋战国，诸子百家群星闪耀，然而却也谁都未能有绝对的实力统一当时的学术界。即便到了秦始皇，虽武力空前强大，踏平六国，也有车同轨书同文的壮举，却依然未能把春秋以降的五彩缤纷的思想学术界统一起来。直到历史再历一百年，汉武帝终于完成了这个空前绝后的大手笔，“罢黜百家独尊儒术”，这也直接成为了华夏文化的核心基础。不禁要问，如果把 NNLM 当做文化统一前的“书同文”苗头，那 NLP 中的“独尊儒术”在哪里？

历史突破——梯子来了

自 NNLM 于 2003 年被提出后，后面又出现了很多类似和改进的工作，诸如 LBL，C&W 和 RNNLM 模型等等，这些方法主要从两个方面去优化 NNLM 的思想：其一是 NNLM 只用了左边的 $n-1$ 个词，如何利用更多的上下文信息便成为了一个很重要的优化思路（如 Mikolov 等人提出的

RNNLM)；其二是 NNLM 的一个非常大的缺点是输出层计算量太大，如何减小计算量使得大规模语料上的训练变得可行，这也是工程应用上至关重要的优化方向（如 Mnih 和 Hinton 提出的 LBL 以及后续的一系列模型）。

为了更好理解 NNLM 之后以及 word2vec 诞生前的情况，我们先来看看现有的几个主要模型都有哪些优缺点。

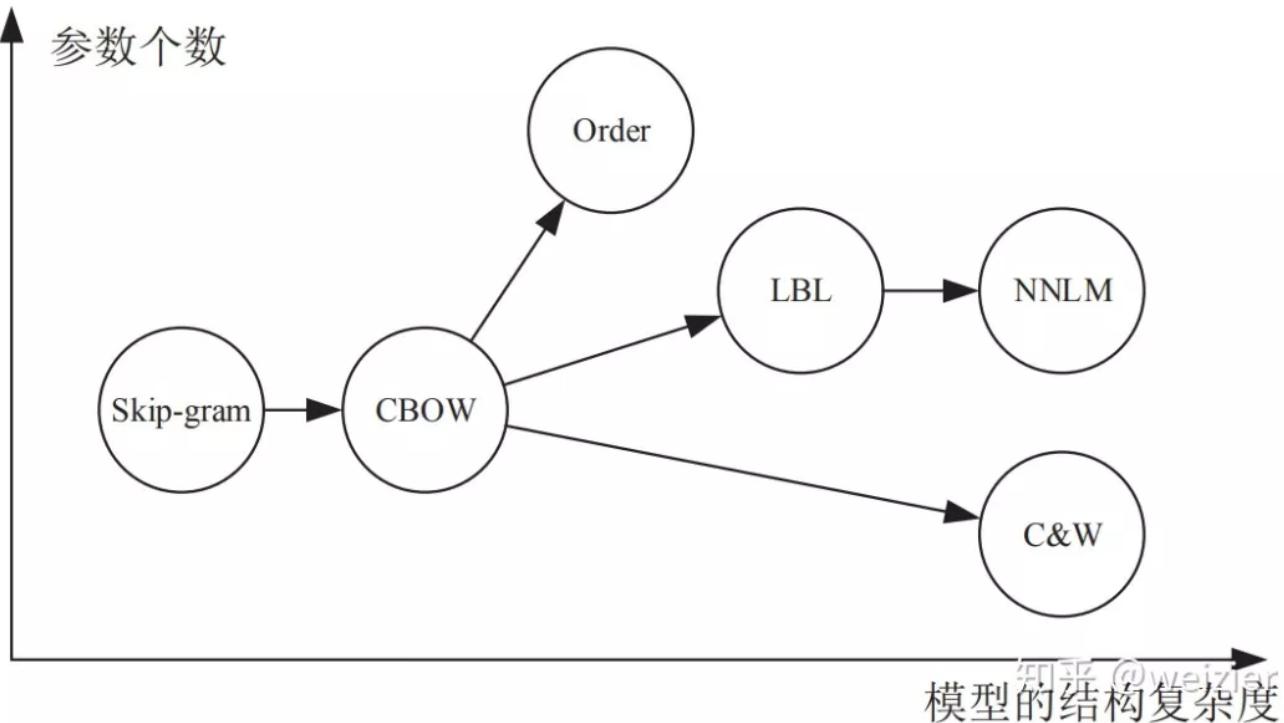
NNLM 虽然将 N-Gram 的阶 n 提高到了 5，相比原来的统计语言模型是一个很大的进步，但是为了获取更好的长程依赖关系，5 显然是不够的。再者，因为 NNLM 只对词的左侧文本进行建模，所以得到的词向量并不是语境的充分表征。

还有一个问题就更严重了，NNLM 的训练依然还是太慢，在论文中，Bengio 说他们用了 40 块 CPU，在含有 1400 万个词，只保留词频相对较高的词之后词典大小为 17964 个词，只训练了 5 个 epoch，但是耗时超过 3 周。按这么来算，如果只用一块 CPU，可能需要 2 年多，这还是在仅有 1400 万个词的语料上。如此耗时的训练过程，显然严重限制了 NNLM 的应用。

2007 年 Mnih 和 Hinton 提出的 LBL 以及后续的一系列相关模型，省去了 NNLM 中的激活函数，直接把模型变成了一个线性变换，尤其是后来将 Hierarchical Softmax 引入到 LBL 后，训练效率进一步增强，但是表达能力不如 NNLM 这种神经网络的结构。

2008 年 Collobert 和 Weston 提出的 C&W 模型不再利用语言模型的结构，而是将目标文本片段整体当做输入，然后预测这个片段是真实文本的概率，所以它的工作主要是改变了目标输出。由于输出只是一个概率大小，不再是词典大小，因此训练效率大大提升，但由于使用了这种比较“别致”的目标输出，使得它的词向量表征能力有限。

2010 年 Mikolov（对，还是同一个 Mikolov）提出的 RNNLM 主要是为了解决长程依赖关系，时间复杂度问题依然存在。



而这一切，似乎都在预示着一个新时代的来临。

CBOW 和 Skip-gram

2013 年，Tomas Mikolov 连放几篇划时代的论文，其中最为重要的是两篇，***Efficient estimation of word representations in vector space*** 首次提出了 CBOW 和 Skip-gram 模型，进一步地在 ***Distributed Representations of Words and Phrases and their Compositionality*** 中，又介绍了几种优化训练的方法，包括 Hierarchical Softmax（当然，这个方法早在 2003 年，Bengio 就在他提出 NNLM 论文中的 Future Work 部分提到了这种方法，并于 2005 年把它系统化发表了一篇论文），Negative Sampling 和 Subsampling 技术。

放出两篇论文后，当时仍在谷歌工作的 Mikolov 又马不停蹄放出了大杀器——word2vec 工具，并在其中开源了他的方法。顺便提一下的是，很多人以为 word2vec 是一种模型和方法，**其实 word2vec 只是一个工具**，背后的模型是 CBOW 或者 Skip-gram，并且使用了 Hierarchical Softmax 或 Negative Sampling 这些训练的优化方法。

所以准确说来，word2vec 并不是一个模型或算法，只不过 Mikolov 恰好在当时把他开源的工具包起名叫做 word2vec 而已。不过为了叙述简单，在下文我将用 word2vec 来指代上面提到 Mikolov 两篇论文中的一整个相关的优化思想。

word2vec 对于前人的优化，主要是两方面的工作：模型的简化和训练技巧的优化。我们先来看看模型的简化方面，也就是耳熟能详的 CBOW 和 Skip-gram。

对于 CBOW 而言，我们可以从它的名字上一窥究竟，它的全称是 Continuous Bag-of-Words，也就是连续的词袋模型，为什么取这个名字，先来看看它的目标函数。

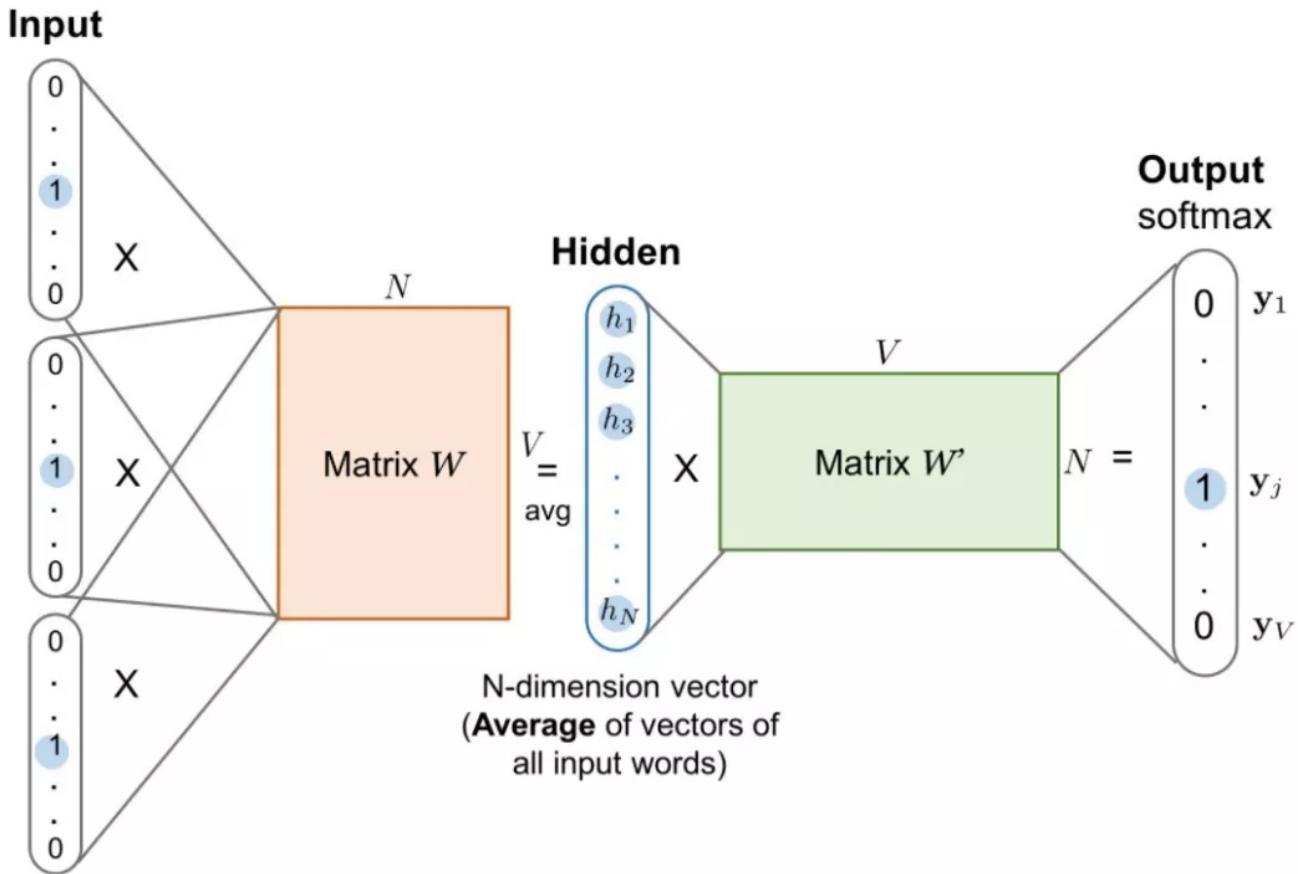
$$\frac{1}{T} \sum_{t=1}^T \log P(w_t | c_t)$$

$$P(w_t | c_t) = \frac{\exp(e'(w_t)^T x)}{\sum_{i=1}^{|V|} \exp(e'(w_i)^T x)}, x = \sum_{i \in c} e(w_i)$$

首先，CBOW 没有隐藏层，本质上只有两层结构，输入层将目标词语境 c 中的每一个词向量简单求和（当然，也可以求平均）后得到语境向量，然后直接与目标词的输出向量求点积，目标函数也就是要让这个与目标词向量的点积取得最大值，对应的与非目标词的点积尽量取得最小值。

从这可以看出，CBOW 的第一个特点是取消了 NNLM 中的隐藏层，直接将输入层和输出层相连；第二个特点便是在求语境 context 向量时候，语境内的词序已经丢弃（这个是名字中 Continuous 的来源）；第三，因为最终的目标函数仍然是语言模型的目标函数，所以需要顺序遍历语料中的每一个词（这个是名字中 Bag-of-Words 的来源）。

因此有了这些特点（尤其是第二点和第三点），Mikolov 才把这个简单的模型取名叫做 CBOW，简单却有效的典范。



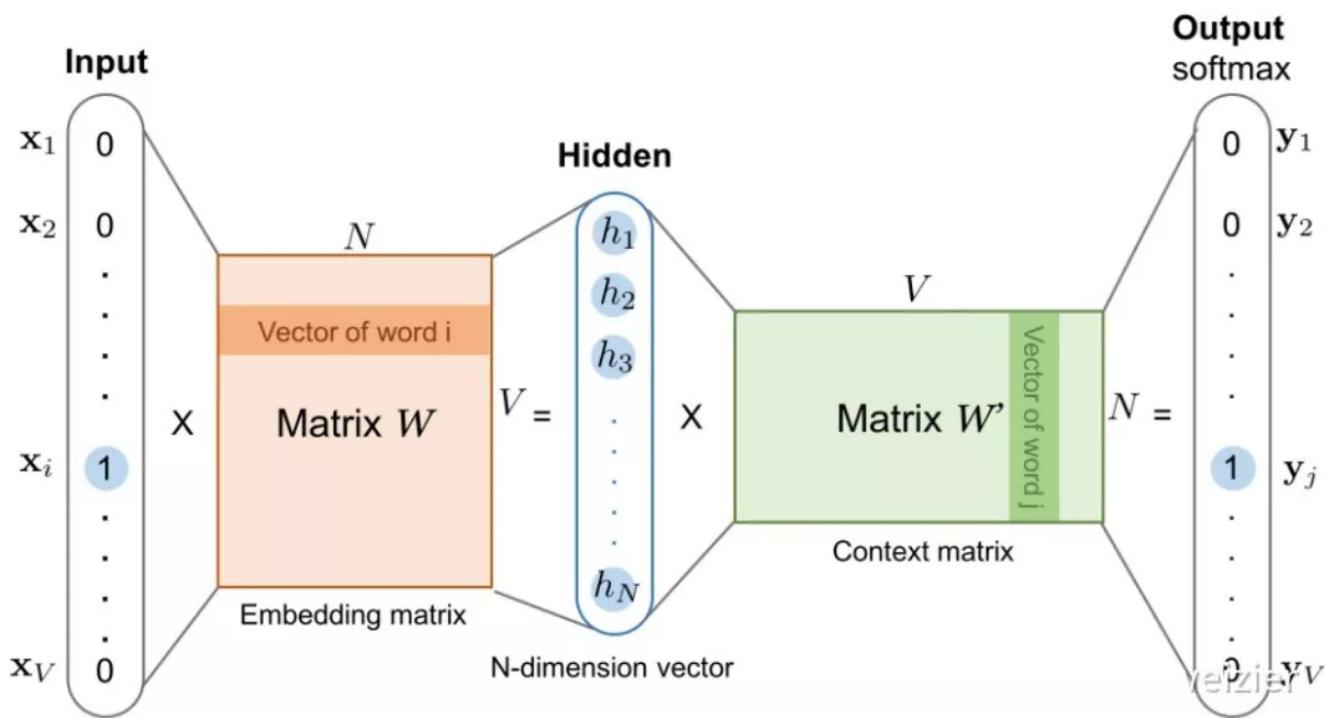
需要注意的是这里每个词对应到两个词向量，在上面的公式中都有体现，其中 $e(wt)$ 是词的输入向量，而 $e'(wt)$ 则是词的输出向量，或者更准确的来讲，前者是 CBOW 输入层中跟词 wt 所在位置相连的所有边的权值（其实这就是词向量）组合成的向量，而是输出层中与词 wt 所在位置相连的所有边的权值组合成的向量，所以把这一向量叫做输出向量。

同样地，和 CBOW 对应，Skip-gram 的模型基本思想和 CBOW 非常类似，只是换了一个方向：CBOW 是让目标词的输出向量 $e'(wt)$ 拟合语境的向量 x ；而 Skip-gram 则是让语境中每个词的输出向量尽量拟合当前输入词的向量 $e(wt)$ ，和 CBOW 的方向相反，因此它的目标函数如下：

$$\frac{1}{T} \sum_{t=1}^T \sum_{j \in c} \log P(w_j | w_t)$$

$$P(w_j|w_t) = \frac{\exp(e'(w_j)^T e(w_t))}{\sum_{i=1}^{|V|} \exp(e'(w_i)^T e(w_t))}$$

可以看出目标函数中有两个求和符号，最里面的求和符号的意义便是让当前的输入词分别和该词对应语境中的每一个词都尽量接近，从而便可以表现为该词与其上下文尽量接近。



和 CBOW 类似，Skip-gram 本质上也只有两层：输入层和输出层，输入层负责将输入词映射为一个词向量，输出层负责将其经过线性映射计算得到每个词的概率大小。

再细心一点的话，其实无论 CBOW 还是 Skip-gram，本质上都是两个全连接层的相连，中间没有任何其他的层。因此，这两个模型的参数个数都是 $2 \times |e| \times |V|$ ，其中 $|e|$ 和 $|V|$ 分别是词向量的维度和词典的大小，相比上文中我们计算得到 NNLM 的参数个数 $|V|(1+|H|+|e|n) + |H|(1+|e|n-|e|)$ 已经大大减小，且与上下文所取词的个数无关了，也就是终于避免了 N-gram 中随着阶数 N 增大而使得计算复杂度急剧上升的问题。

然而，Mikolov 大神说了，这些公式是“impractical”的，他的言下之意是计算复杂度依然和词典大小有关，而这通常都意味着非常非常大，以下是他原话：

..., and W is the number of words in the vocabulary. This formulation is impractical because the cost of computing $\nabla \log p(w_O/w_I)$ is proportional to W , which is often large (10^5 - 10^7 terms).

不得不说，大神就是大神，将模型已经简化到了只剩两个全连接层（再脱就没了），依然不满足，还“得寸进尺”地打起了词典的“小算盘”，那么 Mikolov 的“小算盘”是什么呢？

他在论文中首先提到了 Hierarchical Softmax，认为这是对 full softmax 的一种优化手段，而 Hierarchical Softmax 的基本思想就是首先将词典中的每个词按照词频大小构建出一棵 Huffman 树，保证词频较大的词处于相对比较浅的层，词频较低的词相应的处于 Huffman 树较深层的叶子节点，每一个词都处于这棵 Huffman 树上的某个叶子节点。

第二，将原本的一个 $|V|$ 分类问题变成了 $\log |V|$ 次的二分类问题，做法简单说来就是，原先要计算 $P(w_t|c_t)$ 的时候，因为使用的是普通的 softmax，势必要求词典中的每一个词的概率大小。

为了减少这一步的计算量，在 Hierarchical Softmax 中，同样是计算当前词 w_t 在其上下文中的概率大小，只需要把它变成在 Huffman 树中的路径预测问题就可以了，因为当前词 w_t 在 Huffman 树中对应到一条路径，这条路径由这棵二叉树中从根节点开始，经过一系列中间的父节点，最终到达当前这个词的叶子节点而组成，那么在每一个父节点上，都对应的是一个二分类问题（本质上就是一个 LR 分类器），而 Huffman 树的构造过程保证了树的深度为 $\log |V|$ ，所以也就只需要做 $\log |V|$ 次二分类便可以求得 $P(w_t|c_t)$ 的大小，这相比原来 $|V|$ 次的计算量，已经大大减小了。

接着，Mikolov 又提出了负采样的思想，而这一思想也是受了 C&W 模型中构造负样本方法启发，同时参考了 Noise Contrastive Estimation (NCE) 的思想，用 CBOW 的框架简单来讲就是，负采样每遍历到一个目标词，为了使得目标词的概率 $P(w_t|c_t)$ 最大，根据 softmax 函数的概率公式，也就是让分子中的 $e'(w_t)^T x$ 最大，而分母中其他非目标词的 $e'(w_i)^T x$ 最小。

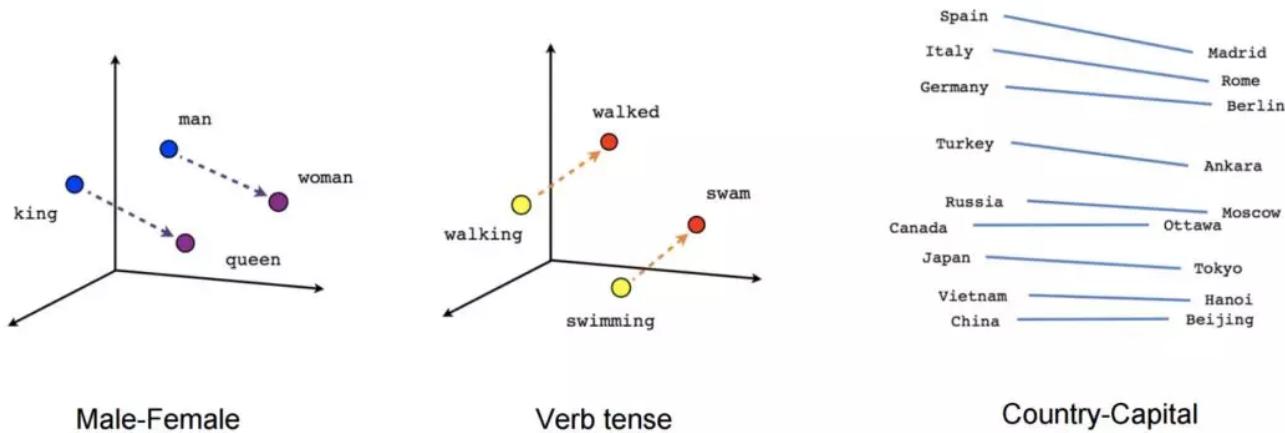
普通 softmax 的计算量太大就是因为它把词典中所有其他非目标词都当做负例了，而负采样的思想特别简单，就是每次按照一定概率随机采样一些词当做负例，从而就只需要计算这些负采样出来的负例了，那么概率公式便相应变为：

$$P(w_t|c_t) = \frac{\exp(e'(w_t)^T x)}{\sum_{i=1}^K \exp(e'(w_i)^T x)}, x = \sum_{i \in c} e(w_i)$$

仔细和普通 softmax 进行比较便会发现，将原来的 $|V|$ 分类问题变成了 K 分类问题，这便把词典大小对时间复杂度的影响变成了一个常数项，而改动又非常的微小，不可谓不巧妙。

除此之外，Mikolov 还提到了一些其他技巧，比如对于那些超高频率的词，尤其是停用词，可以使用 Subsampling 的方法进行处理，不过这已经不是 word2vec 最主要的内容了。

自此，经过模型和训练技巧的双重优化，终于使得大规模语料上的训练成为了现实，更重要的是，得到的这些词向量能够在语义上有非常好的表现，能将语义关系通过向量空间关系表征出来。



word2vec 的出现，极大促进了 NLP 的发展，尤其是促进了深度学习在 NLP 中的应用（不过有意思的是，word2vec 算法本身其实并不是一个深度模型，它只有两层全连接），利用预训练好的词向量来初始化网络结构的第一层几乎已经成了标配，尤其是在只有少量监督数据的情况下，如果不拿预训练的 embedding 初始化第一层，几乎可以被认为是在蛮干。

在此之后，一大批 word embedding 方法大量涌现，比较知名的有 GloVe 和 fastText 等等，它们各自侧重不同的角度，并且从不同的方向都得到了还不错的 embedding 表征。

GloVe

先来看看 GloVe 的损失函数：

$$J = \sum_{i=1}^V \sum_{j=1}^V f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

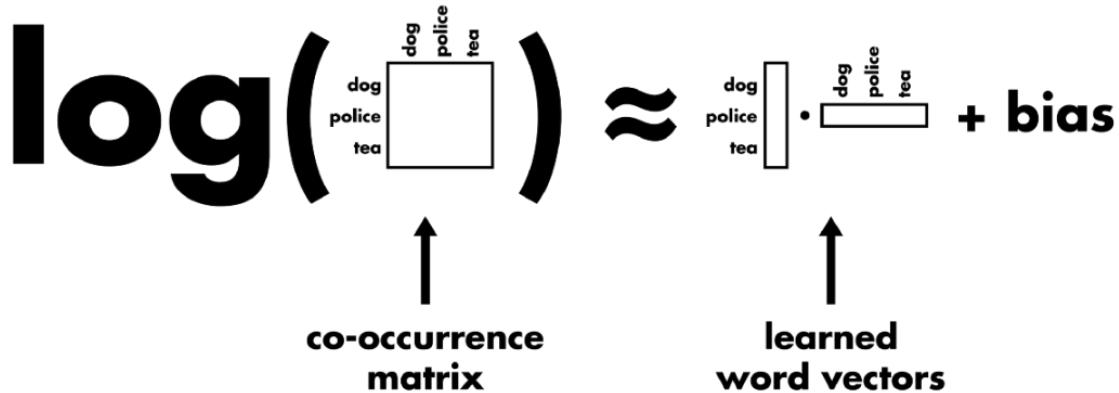
其中 X_{ij} 是两个词 i 和 j 在某个窗口大小中的共现频率（不过 GloVe 对其做了一些改进，共现频率相应有一个衰减系数，使得距离越远的词对共现频率越小一些）， $f(X_{ij})$ 是一个权重系数，主要目的是共现越多的 pair 对于目标函数贡献应该越大，但是又不能无限制增大，所以对共现频率过大的 pair 限定最大值，以防训练的时候被这些频率过大的 pair 主导了整个目标函数。

剩下的就是算法的核心部分了，两个b值是两个偏置项，撇去不谈，那么剩下的 $(w_i^T w_j - \log X_{ij})^2$ 其实就是一个普通的均方误差函数， w_i 是当前词的向量， w_j 对应的是与其在同一个窗口中出现的共现词的词向量，两者的向量点乘要去尽量拟合它们共现频率的对数值。

从直观上理解，如果两个词共现频率越高，那么其对数值当然也越高，因而算法要求二者词向量的点乘也越大，而两个词向量的点乘越大，其实包含了两层含义：

第一，要求各自词向量的模越大，通常来说，除去频率非常高的词（比如停用词），对于有明确语义的词来说，它们的词向量模长会随着词频增大而增大，因此两个词共现频率越大，要求各自词向量模长越大是有直觉意义的，比如“魑魅魍魉”假如能被拆分成两个词，那么“魑魅”和“魍魉”这两个词的共现频率相比“‘魑魅’和其他词的共现频率要大得多，对应到“魑魅”的词向量，便会倾向于在某个词向量维度上持续更新，进而使得它的模长也会比较偏大。

第二，要求这两个词向量的夹角越小，这也是符合直觉的，因为出现在同一个语境下频率越大，说明这两个词的语义越接近，因而词向量的夹角也偏向于越小。



此外，可以从 GloVe 使用的损失函数中发现，它的训练主要是两个步骤：统计共现矩阵和训练获取词向量，这个过程其实是没有我们通常理解当中的模型的。

更遑论神经网络，它整个的算法框架都是基于矩阵分解的做法来获取词向量的，本质上和诸如 LSA 这种基于 SVD 的矩阵分解方法没有什么不同，只不过 SVD 分解太过于耗时，运算量巨大，相同点是 LSA 也是输入共现矩阵，不过一般主要以词-文档共现矩阵为主，另外，LSA 中的共现矩阵没有做特殊处理，而 GloVe 考虑到了对距离较远的词对做相应的惩罚等等。

然而，相比 word2vec，GloVe 却更加充分的利用了词的共现信息，word2vec 中则是直接粗暴的让两个向量的点乘相比其他词的点乘最大，至少在表面上看来似乎是没有用到词的共现信息，不像

GloVe 这里明确的就是拟合词对的共现频率。

不过更有意思的还是，GloVe 和 word2vec 似乎有种更为内在的联系，再来看看他们的目标函数有什么不一样，这是 Skip-gram 的目标函数（这里在原来的基础上加上了对于语料的遍历 N）：

$$\frac{1}{NT} \sum_{n=1}^N \sum_{t=1}^T \sum_{j \in c} \log P(w_{nj} | w_{nt})$$

$$P(w_j | w_t) = \frac{\exp(e'(w_j)^T e(w_t))}{\sum_{i=1}^{|V|} \exp(e'(w_i)^T e(w_t))}$$

而这个目标函数是按照语料的顺序去遍历，如果先把语料当中的相关词进行合并，然后按照词典序进行遍历，便可以证明 Skip-gram 实际上和 GloVe 的优化目标一致，有兴趣的可以参考证明细节 (arXiv:1611.05962)，这里不再赘述。

fastText

word2vec 和 GloVe 都不需要人工标记的监督数据，只需要语言内部存在的监督信号即可以完成训练。而与此相对应的，**fastText** 则是利用带有监督标记的文本分类数据完成训练。

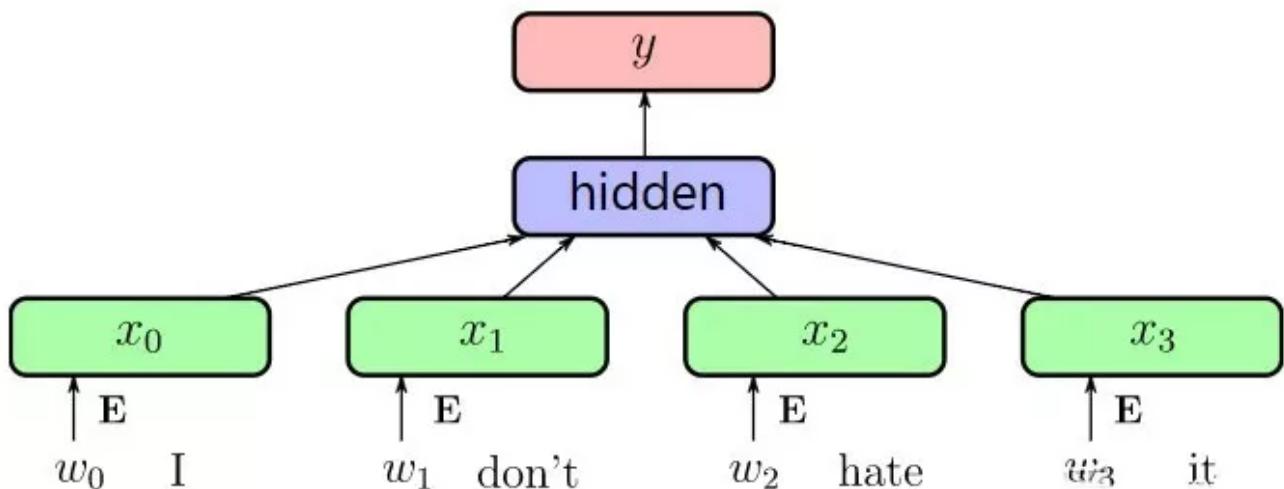
本质上没有什么特殊的，模型框架就是 CBOW，只不过与普通的 CBOW 有两点不一样，分别是输入数据和预测目标的不同，在输入数据上，CBOW 输入的是一段区间中除去目标词之外的所有其他词的向量加和或平均，而 **fastText** 为了利用更多的语序信息，将 **bag-of-words** 变成了 **bag-of-features**，也就是下图中的输入 x 不再仅仅是一个词，还可以加上 bigram 或者是 trigram 的信息等等。

第二个不同在于，**CBOW** 预测目标是语境中的一个词，而 **fastText** 预测目标是当前这段输入文本的类别，正因为需要这个文本类别，因此才说 fastText 是一个监督模型。

而相同点在于，fastText 的网络结构和 CBOW 基本一致，同时在输出层的分类上也使用了 Hierarchical Softmax 技巧来加速训练。

$$-\frac{1}{N} \sum_{n=1}^N y_n \log f(BAx_n), x_n = \sum_{i=1}^{l_n} x_{n,i}$$

这里的 $x_{n,i}$ 便是语料当中第 n 篇文档的第 i 个词以及加上 N-gram 的特征信息。从这个损失函数便可以知道 fastText 同样只有两个全连接层，分别是 A 和 B，其中 A 便是最终可以获取的词向量信息。



fastText 最大的特点在于快，论文中对这一点也做了详细的实验验证，在一些分类数据集上，fastText 通常都可以把 CNN 结构的模型要耗时几小时甚至几天的时间，急剧减少到只需要消耗几秒钟，不可谓不“fast”。

不过说个八卦，为什么 fastText 结构和 CBOW 如此相似。感兴趣的读者想要继续深挖的话，还可以看看 2015 年 ACL 的一篇论文 ***Deep Unordered Composition Rivals Syntactic Methods for Text Classification***，结构又是何其相似，并且比 fastText 的论文探讨的更为深入一些，但是 fastText 是 2016 年的文章，剩下的大家自己去想好了。

这里面大概一个特别重要的原因就是 fastText 的作者之一便是 3 年前 CBOW 的提出者 Mikolov 本人，只不过昔日的 Mikolov 还在谷歌，如今 3 年时间一晃而过，早已是 Facebook 的人了。

爬上第一级梯子的革命意义

为什么说 word2vec 的出现极大促进了 NLP 领域的发展？

通常以为，word2vec一类的方法会给每一个词赋予一个向量的表征，不过似乎从来没有人想过这个办法为什么行之有效？难道是皇帝的新衣？按理来说，包括NNLM在内，这些方法的主要出发点都是将一个词表示成词向量，并将其语义通过上下文来表征。

其实，这背后是有理论基础的，1954年Harris提出分布假说（distributional hypothesis），这一假说认为：上下文相似的词，其语义也相似，1957年Firth对分布假说进行了进一步阐述和明确：词的语义由其上下文决定（a word is characterized by the company it keeps），30年后，深度学习的祖师爷Hinton也于1986年尝试过词的分布式表示。

而word2vec的贡献远不止是给每一个词赋予一个分布式的表征，私以为，**它带来了一种全新的NLP模型建立方法**。在这之前，大多数NLP任务都要在如何挖掘更多文本语义特征上花费大量时间，甚至这一部分工作占去了整个任务工作量的绝大部分。

而以word2vec为代表的distributed representation方法大量涌现后（尤其是因为大规模语料上的预训练词向量成为现实，并且被证明确实行之有效之后），算法人员发现利用word2vec在预训练上学习到的词向量，初始化他们自己模型的第一层，会带来极大效果的提升，以至于这五年以来，几乎业内的默认做法便是要用无论word2vec还是GloVe预训练的词向量，作为模型的第一层，**如果不这么做，大约只有两个原因：1) 你很土豪，有钱标注大量监督数据；2) 你在蛮干。**

而这一思想，绝不是如它表象所显示的一样，似乎和过去做文本特征没什么太大区别，是的，表象确实是这样，无非是把一个词用了一堆数字来表征而已，**这和离散化的特征有什么本质区别吗？**

有，因为它开启了一种全新的NLP模型训练方式——迁移学习。基本思想便是利用一切可以利用的现成知识，达到快速学习的目的，这和人类的进化历程何其相似。

虽然咿咿呀呀囫囵吞枣似的刚开始能够说得三两个词，然而这是“NLP的一小步，人类AI的一大步”。正如人类语言产生之初，一旦某个原始人类的喉部发出的某个音节，经无比智慧和刨根问底考证的史学家研究证实了它具有某个明确的指代意义（无论它指代什么，即便是人类的排泄物），这便无比庄严的宣示着一个全新物种的诞生，我想迁移学习在NLP中的这一小步，大概与此同担当。

梯子的一级半

除了在 word 级别的 embedding 方法上有大量模型和算法的涌现，同样地，在 char 级别、句子级别和段落级别同样有大量模型提出。

word2vec 开源随后的第一年，也就是在 2014 年，还是 Mikolov，在他和另一位作者合作的一篇论文 **Distributed Representations of Sentences and Documents** 中，提出了可以借鉴 word2vec 思想的两种结构：PV-DM 和 PV-DBOW，分别对应 word2vec 中的 CBOW 和 Skip-gram。

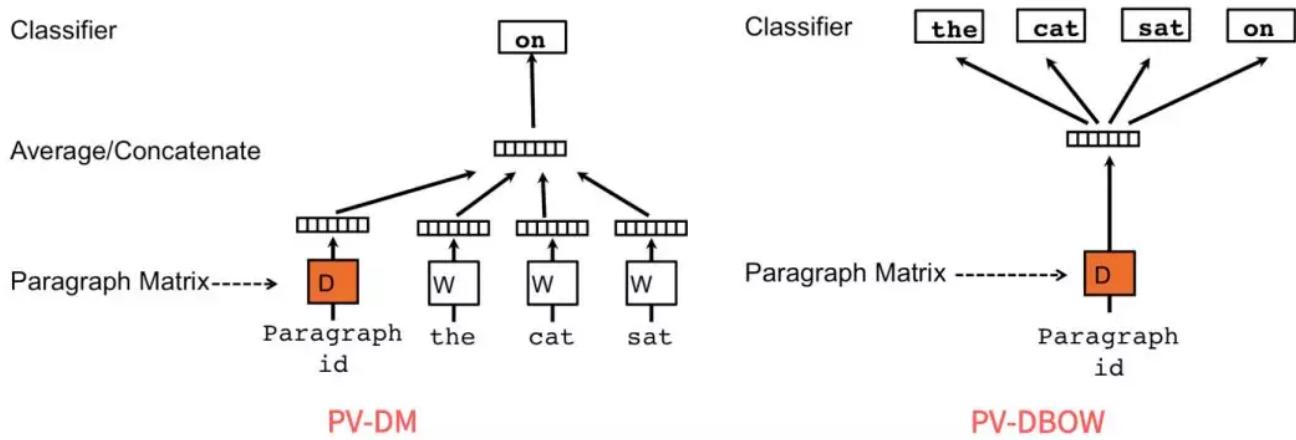
PV-DM 和 PV-DBOW

PV-DM 的全称是 Distributed Memory Model of Paragraph Vectors，和 CBOW 类似，也是通过上下文预测下一个词，不过在输入层的时候，同时也维护了一个文档 ID 映射到一个向量的 look-up table，模型的目的便是将当前文档的向量以及上下文向量联合输入模型，并让模型预测下一个词。

训练结束后，对于现有的文档，便可以直接通过查表的方式快速得到该文档的向量，而对于新的一页文档，那么则需要将已有的 look-up table 添加相应的列，然后重新走一遍训练流程，只不过此时固定好其他的参数，只调整 look-up table，收敛后便可以得到新文档对应的向量了。

PV-DBOW 的全称则是 Distributed Bag of Words version of Paragraph Vector，和 Skip-gram 类似，通过文档来预测文档内的词，训练的时候，随机采样一些文本片段，然后再从这个片段中采样一个词，让 PV-DBOW 模型来预测这个词。

以此分类任务作为训练方法，说白了，本质上和 Skip-gram 是一样的。这个方法有个致命的弱点，就是为了获取新文档的向量，还得继续走一遍训练流程，并且由于模型主要是针对文档向量预测词向量的过程进行建模，其实很难去表征词语之间的更丰富的语义结构，所以这两种获取文档向量的方法都未能大规模应用开来。

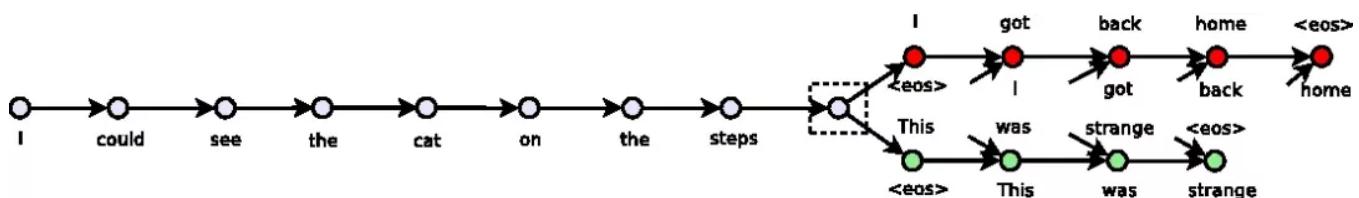


Skip-thoughts

2015 年，多伦多大学的 Kiros 等人提出了一个很有意思的方法叫 Skip-thoughts。同样也是借鉴了 Skip-gram 的思想，但是和 PV-DBOW 中利用文档来预测词的做法不一样的是，**Skip-thoughts** 直接在句子间进行预测，也就是将 Skip-gram 中以词为基本单位，替换了以句子为基本单位，具体做法就是选定一个窗口，遍历其中的句子，然后分别利用当前句子去预测和输出它的上一句和下一句。

对于句子的建模利用的 RNN 的 sequence 结构，预测上一个和下一个句子时候，也是利用的一个 sequence 的 RNN 来生成句子中的每一个词，所以这个结构本质上就是一个 Encoder-Decoder 框架，只不过和普通框架不一样的是，Skip-thoughts 有两个 Decoder。

在今天看来，这个框架还有很多不完善或者可以改进的地方（作者也在论文中分别提到了这些 future works），比如输入的 Encoder 可以引入 attention 机制，从而让 Decoder 的输入不再只是依赖 Encoder 最后一个时刻的输出；Encoder 和 Decoder 可以利用更深层的结构；Decoder 也可以继续扩大，可以预测上下文中更多的句子；RNN 也不是唯一的选择，诸如 CNN 以及 2017 年谷歌提出的 Transformer 结构也可以利用进来，后来果不其然谷歌的 BERT 便借鉴了这一思路，当然这是后话了，留下暂且不表。



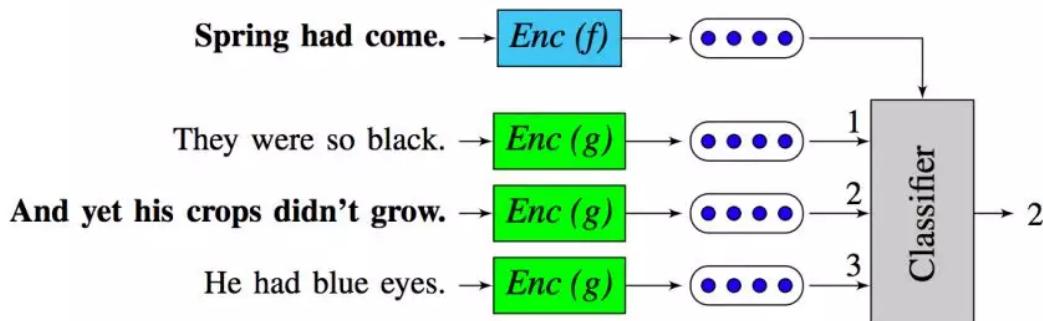
Quick-thoughts

2018 年的时候，在 Skip-thoughts 的基础上，Google Brain 的 Logeswaran 等人将这一思想做了进一步改进，他们认为 Skip-thoughts 的 Decoder 效率太低，且无法在大规模语料上很好的训练（这是 RNN 结构的通病）。

所以他们把 **Skip-thoughts** 的生成任务改进成为了一个分类任务，具体说来就是把同一个上下文窗口中的句子对标记为正例，把不是出现在同一个上下文窗口中的句子对标记为负例，并将这些句子对输入模型，让模型判断这些句子对是否是同一个上下文窗口中，很明显，这是一个分类任务。可以说，仅仅几个月之后的 **BERT** 正是利用的这种思路。而这些方法都和 **Skip-thoughts** 一脉相承。

Spring had come. → **Enc** → (•••) → **Dec** → **And yet his crops didn't grow.**

(a) Conventional approach

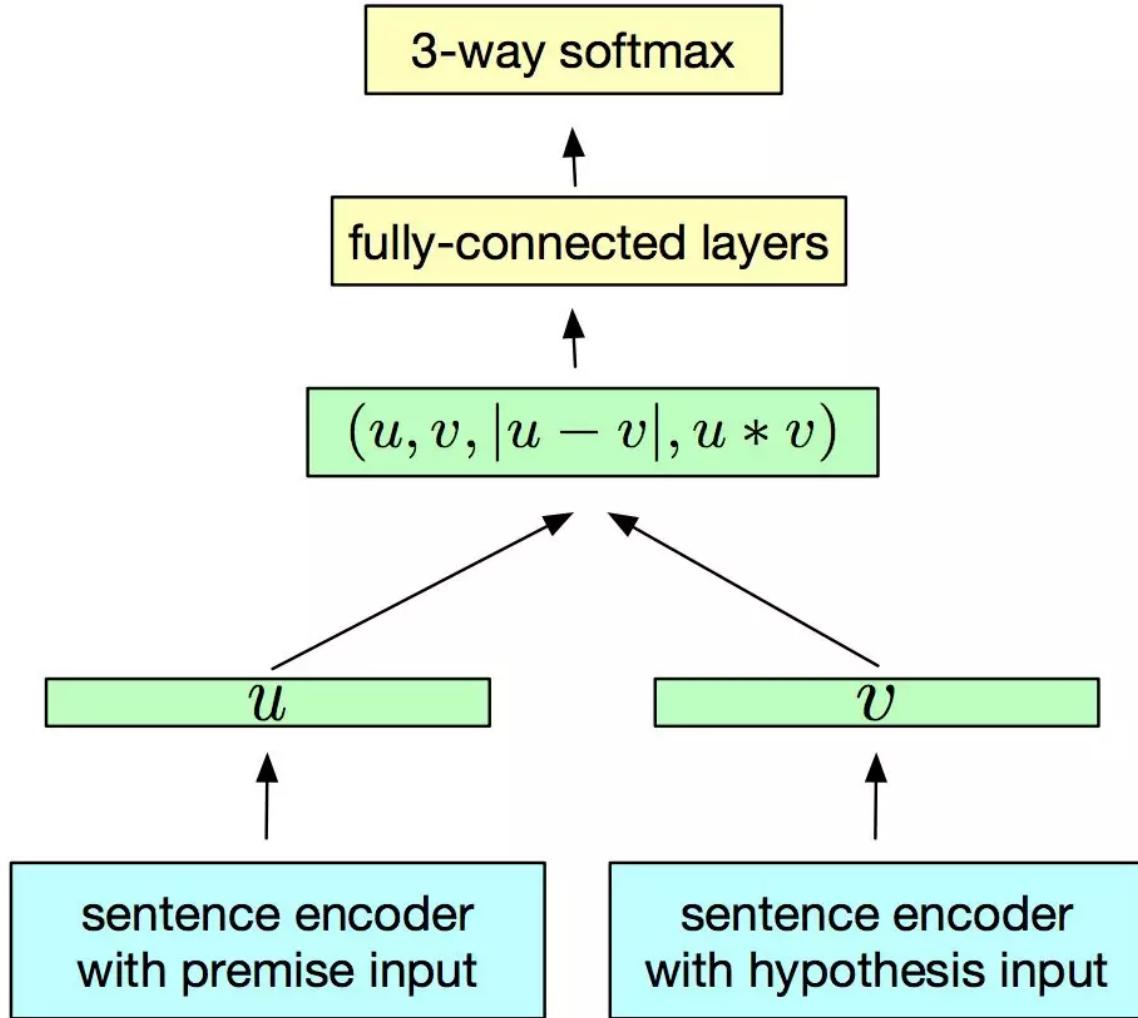


(b) Proposed approach

InferSent

除了 Skip-thoughts 和 Quick-thoughts 这两种不需要人工标记数据的模型之外，还有一些从监督数据中学习句子表示的方法。

比如 2017 年 Facebook 的研究人员 Conneau 等人提出的 InferSent 框架，它的思想特别简单，先设计一个模型在斯坦福的 SNLI (Stanford Natural Language Inference) 数据集上训练，而后将训练好的模型当做特征提取器，以此来获得一个句子的向量表示，再将这个句子的表示应用在新的分类任务上，来评估句子向量的优劣。框架结构如下图所示：



这个框架最底层是一个 Encoder，也就是最终要获取的句子向量提取器，然后将得到的句子向量通过一些向量操作后得到句子对的混合语义特征，最后接上全连接层并做 SNLI 上的三分类任务。

做过句子匹配任务的一定知道，这个框架是一个最基本（甚至也是最简陋）的句子匹配框架。对于底层的 Encoder 来说，论文作者分别尝试了 7 种模型，然后分别以这些模型作为底层的 Encoder 结构在 SNLI 上进行监督训练。

训练完成后，在新的分类任务上进行评估，最后发现当 Encoder 使用 BiLSTM with max pooling 结构时，对于句子的表征性能最好。对具体细节感兴趣的可以参考他们的论文 ***Supervised Learning of Universal Sentence Representations from Natural Language Inference Data.***

General Purpose Sentence Representation

此外，除了 InferSent 这种单个任务的监督学习外，最新的工作逐渐将多任务的联合学习应用到获取句子的表征中。

例如 Subramanian 等人发表在 ICLR 2018 上的 ***Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning***，就提出了利用四种不同的监督任务来联合学习句子的表征，这四种任务分别是：Natural Language Inference，Skip-thoughts，Neural Machine Translation 以及 Constituency Parsing 等。

作者的出发点也特别简单，通用的句子表征应该通过侧重点不同的任务来联合学习到，而不是只有一个特定任务来学习句子表征，后来作者在论文中的实验也确实证明了这点。

实验的具体做法是，先用联合学习的方法在上述四个任务上进行训练，训练结束后，将模型的输出作为句子的表征（或者把这个联合学习的模型作为特征提取器），然后直接在这个表征上接上非常简单的全连接层做分类器，并且同时保证最底层的特征提取器中参数不动（也就是只把它当做特征提取器），再在新的分类任务上做训练（只训练最后接上的全连接层分类器），最后根据训练出来的简单分类器在各自分类任务的测试集上做评估。

最后作者惊喜的发现很多任务上他们的简单分类器都要超过当时的最好结果，并且他们还发现联合训练中不同的任务对于句子表征中的不同方面有不同的贡献。

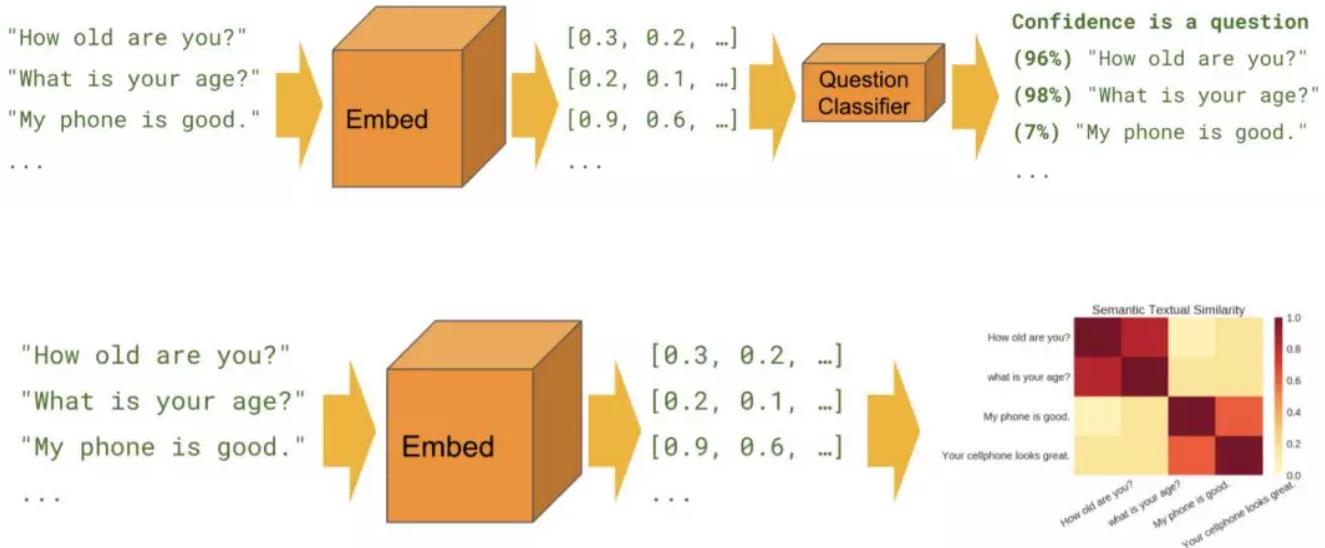
Universal Sentence Encoder

同样在 2018 年，谷歌的 Daniel Cer 等人在论文 ***Universal Sentence Encoder*** 中提出的思路基本和 ***General Purpose Sentence Representation*** 的工作一样，只不过作者提出了利用 Transformer 和 DAN (上文提到过的和 CBOW 与 fastText 都神似的 ***Deep Unordered Composition Rivals Syntactic Methods for Text Classification***) 两种框架作为句子的 Encoder。

Transformer 结构更为复杂，参数更多，训练也相对比较耗时，但是一般来说效果会更好一些。对应的，DAN 结构简单，只有两个隐藏层（甚至可以减小为只需要一个隐藏层），参数比较少，训练相对比较省时省资源，但是一般来说效果会差一些（并不是绝对，论文中也发现某些场景下 DAN 的效果甚至更好）。然后作者既在无标记数据上训练，也在监督数据上训练，最后在十个分类任务上进行迁移学习的评估。此外，作者还放出了他们预训练好的 Encoder，可以供迁移学习的句子特征提取器使用。

预训练 Encoder：

<https://tfhub.dev/google/universal-sentence-encoder/2>



• END •

点击以下标题查看更多往期内容：

- [自动机器学习（AutoML）最新综述](#)
- [自然语言处理中的语言模型预训练方法](#)
- [从傅里叶分析角度解读深度学习的泛化能力](#)
- [深度解读DeepMind新作：史上最强GAN图像生成器](#)
- [两行代码玩转Google BERT句向量词向量](#)
- [本周有哪些值得读的AI论文？进来告诉你答案](#)
- [TensorSpace：超酷炫3D神经网络可视化框架](#)
- [NIPS 2018：基于条件对抗网络的领域自适应方法](#)



#投 稿 通 道#

让你的论文被更多人看到

如何才能让更多的优质内容以更短路径到达读者群体，缩短读者寻找优质内容的成本呢？答案就是：你不认识的人。

总有一些你不认识的人，知道你想知道的东西。PaperWeekly 或许可以成为一座桥梁，促使不同背景、不同方向的学者和学术灵感相互碰撞，迸发出更多的可能性。

PaperWeekly 鼓励高校实验室或个人，在我们的平台上分享各类优质内容，可以是最新论文解读，也可以是学习心得或技术干货。我们的目的只有一个，让知识真正流动起来。

来稿标准：

- 稿件确系个人**原创作品**，来稿需注明作者个人信息（姓名+学校/工作单位+学历/职位+研究方向）
- 如果文章并非首发，请在投稿时提醒并附上所有已发布链接
- PaperWeekly 默认每篇文章都是首发，均会添加“原创”标志

投稿邮箱：

- 投稿邮箱：hr@paperweekly.site
- 所有文章配图，请单独在附件中发送
- 请留下即时联系方式（微信或手机），以便我们在编辑发布时和作者沟通



现在，在「知乎」也能找到我们了
进入知乎首页搜索「**PaperWeekly**」
点击「**关注**」订阅我们的专栏吧

关于PaperWeekly

PaperWeekly 是一个推荐、解读、讨论、报道人工智能前沿论文成果的学术平台。如果你研究或从事 AI 领域，欢迎在公众号后台点击「交流群」，小助手将把你带入 PaperWeekly 的交流群里。



▽ 点击 | 阅读原文 | 获取最新论文推荐

阅读原文