

# NLP 的巨人肩膀（下）：从 CoVe 到 BERT

原创：许维 PaperWeekly 今天

---

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

## 戈多会来吗？

在前文深度长文：NLP 的巨人肩膀（上）中，我们介绍了好几种获取句子表征的方法，然而值得注意的是，我们并不是只对如何获取更好的句子表征感兴趣。

其实更有趣的是，这些方法在评估他们各自模型性能的时候所采取的方法，回过头去进行梳理，我们发现，无论是稍早些的 *InferSent*，还是 2018 年提出的 *Quick-thoughts* 和 *Multi-task*

**Learning** 获取通用句子表征的方法，他们无一例外都使用了同一种思路：将得到的句子表征，在新的分类任务上进行训练，而此时的模型一般都只用一个全连接层，然后接上softmax进行分类。

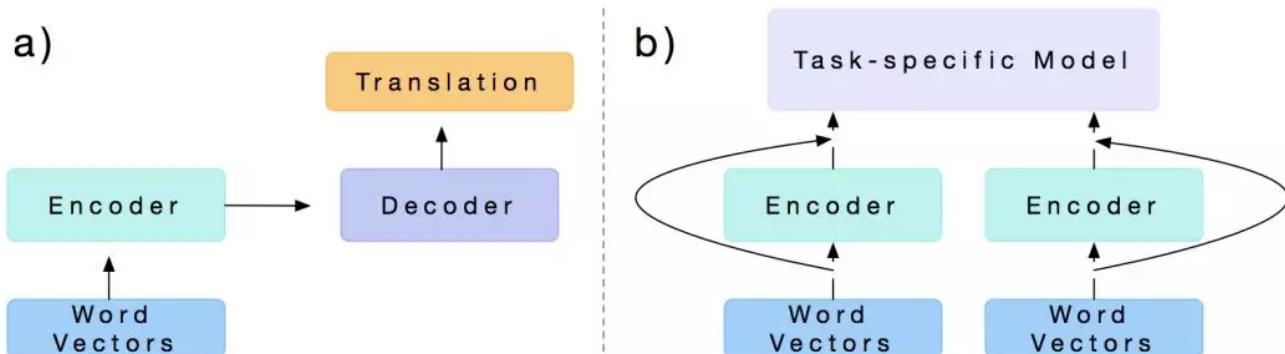
分类器足够简单，足够浅层，相比那些在这些分类任务上设计的足够复杂的模型来说简直不值一提。然而令人大跌眼镜的是，这些简单的分类器都能够比肩甚至超越他们各自时代的最好结果，这不能不说是个惊喜。而创造这些惊喜的背后功臣，就是迁移学习。更进一步地，迁移学习的本质，就是给爬上“巨人的肩膀”提供了一架结实的梯子。

具体的，在这些句子级别的任务中，属于 **InferSent** 和 **Quick-thoughts** 这些模型的“巨人肩膀”便是他们各自使用的训练数据，迁移学习最后给他们搭了一个梯子，然而这个梯子并没有很好上，磕磕绊绊，人类 AI 算是站在第一级梯子上，试探性的伸出了一只腿，另一只腿即将跨出，只可惜并不知道是否有他们苦苦等待了五年之久的戈多？

## 一丝曙光

2017 年，Salesforce 的 Bryan McCann 等人发表了一篇文章 **Learned in Translation: Contextualized Word Vectors**，论文首先用一个 Encoder-Decoder 框架在机器翻译的训练语料上进行预训练，而后用训练好的模型，只取其中的 Embedding 层和 Encoder 层，同时在一个新的任务上设计一个 task-specific 模型，再将原先预训练好的 Embedding 层和 Encoder 层的输出作为这个 task-specific 模型的输入，最终在新的任务场景下进行训练。

他们尝试了很多不同的任务，包括文本分类，Question Answering，Natural Language Inference 和 SQuAD 等，并在这些任务中，与 GloVe 作为模型的输入时候的效果进行比较。实验结果表明他们提出的 Context Vectors 在不同任务中都带来了不同程度效果的提升。



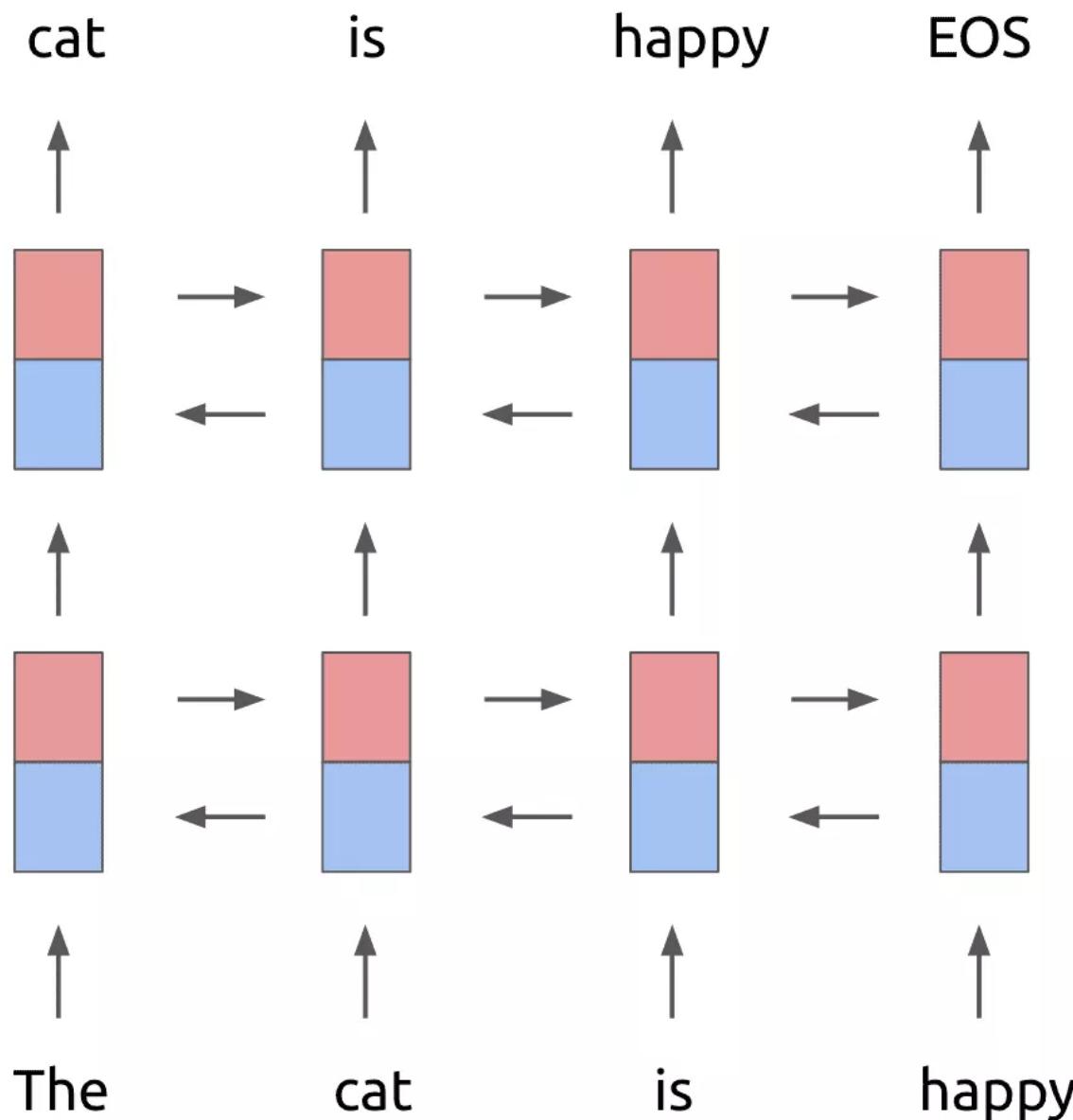
和上文中提到的诸如 Skip-thoughts 方法有所不同的是，CoVe 更侧重于如何将现有数据上预训练得到的表征迁移到新任务场景中，而之前的句子级任务中大多数都只把迁移过程当做一个评估他们

表征效果的手段，因此观念上有所不同。

那么，CoVe 似乎通过监督数据上的预训练，取得了让人眼前一亮的结果，是否可以进一步地，撇去监督数据的依赖，直接在无标记数据上预训练呢？

## ELMo

2018 年的早些时候，AllenNLP 的 Matthew E. Peters 等人在论文 *Deep contextualized word representations*（该论文同时被 ICLR 和 NAACL 接受，并且获得了 NAACL 最佳论文奖，可见其含金量）中首次提出了 *ELMo*，它的全称是 Embeddings from Language Models，从名称上可以看出，ELMo 为了利用无标记数据，使用了语言模型，我们先来看看它是如何利用语言模型的。

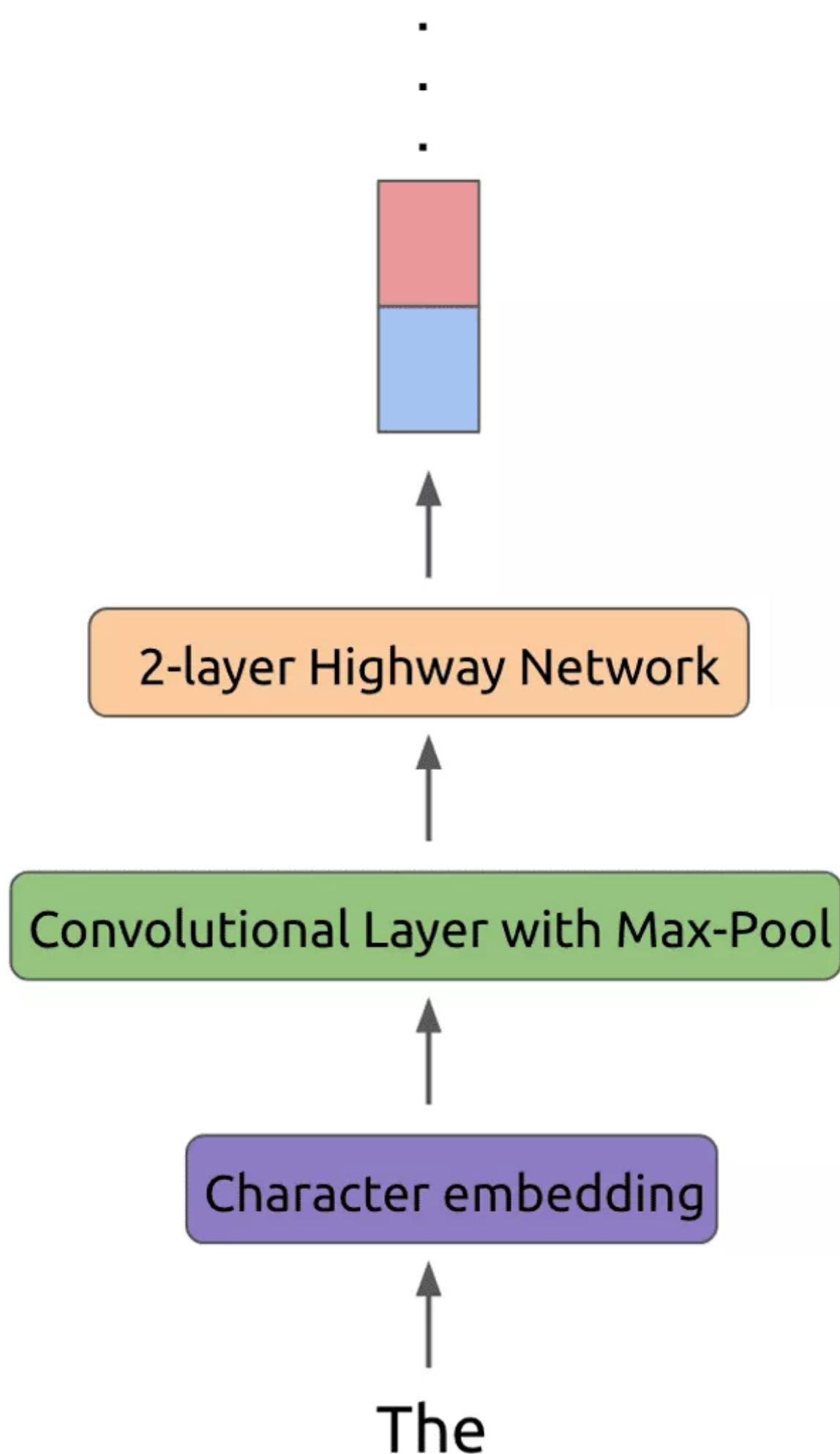


基本框架是一个双层的 Bi-LSTM，不过在第一层和第二层之间加入了一个残差结构（一般来说，残差结构能让训练过程更稳定）。做预训练的时候，ELMo 的训练目标函数为：

$$\sum_{k=1}^N \log p(t_k | t_1, \dots, t_{k-1}) + \log p(t_k | t_{k+1}, \dots, t_N)$$

这个式子很清晰，前后有两个概率，第一个概率是来自于正向的由左到右的 RNN 结构，在每一个时刻上的 RNN 输出（也就是这里的第二层 LSTM 输出），然后再接一个 Softmax 层将其变为概率含义，就自然得到了  $p(t_k | t_1, \dots, t_{k-1})$ ；与此类似，第二个概率来自反向的由右到左的 RNN 结构，每一个时刻 RNN 的输出经过 Softmax 层后也能得到一个概率大小，表示从某个词的下文推断该词的概率大小。

ELMo 的基本框架便是 2-stacked biLSTM + Residual 的结构，不过和普通 RNN 结构的不同之处在于，ELMo 借鉴了 2016 年 Google Brain 的 Rafal Jozefowicz 等人发表的 ***Exploring the Limits of Language Modeling***，其主要改进在于输入层和输出层不再是 word，而是变为了一个 char-based CNN 结构，ELMo 在输入层和输出层考虑了使用同样的这种结构，该结构如下图示：



这样做有什么好处呢？因为输入层和输出层都使用了这种 CNN 结构，我们先来看看输出层使用这种结构怎么用，以及有什么优势。我们都知道，在 CBOW 中的普通 Softmax 方法中，为了计算每个词的概率大小，使用的如下公式的计算方法：

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

说白了，也就是先通过向量点乘的形式计算得到 logits，然后再通过 softmax 变成概率意义，这本质上和普通分类没什么区别，只不过是一个较大的  $|V|$  分类问题。

现在我们假定 char-based CNN 模型是现成已有的，对于任意一个目标词都可以得到一个向量表示  $\text{CNN}(t_k)$ ，当前时刻的 LSTM 的输出向量为  $h$ ，那么便可以通过同样的方法得到目标词的概率大小：

$$p(t_k|t_1, \dots, t_{k-1}) = \frac{\exp(\text{CNN}(t_k)^T h)}{\sum_{i=1}^{|V|} \exp(\text{CNN}(t_i)^T h)}, h = \text{LSTM}(t_k|t_1, \dots, t_{k-1})$$

在原论文中，把这种先经过 CNN 得到词向量，然后再计算 Softmax 的方法叫做 CNN Softmax。

**利用 CNN 解决有三点优势值得注意：**

1. CNN 能减少普通做 Softmax 时全连接层中的必须要有的  $|V|h$  的参数规模，只需保持 CNN 内部的参数大小即可。一般来说，CNN 中的参数规模都要比  $|V|h$  的参数规模小得多；
2. CNN 可以解决 OOV (Out-of-Vocabulary) 问题，这个在翻译问题中尤其头疼；
3. 在预测阶段，CNN 对于每一个词向量的计算可以预先做好，更能够减轻 inference 阶段的计算压力。

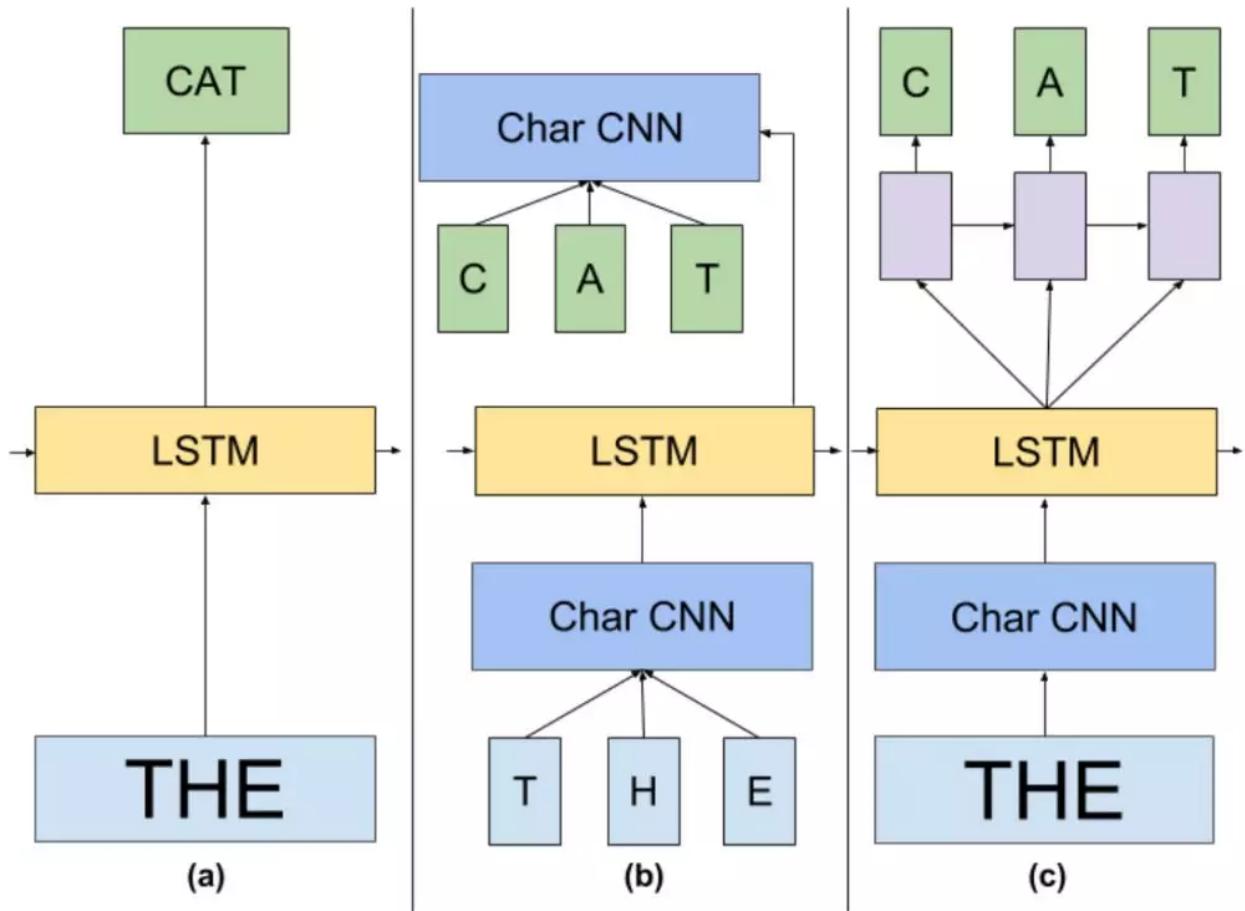
补充一句，普通 Softmax 在大词典上的计算压力，都是因为来自于这种方法需要把一个神经网络的输出通过全连接层映射为单个值（而每个类别需要一个映射一次）。

一次  $h$  大小的计算规模， $|V|$  次映射便需要总共  $|V|*h$  这么多次的映射规模），对于每个类别的映射参数都不同，而 CNN Softmax 的好处就在于能够做到对于不同的词，映射参数都是共享的，这

个共享便体现在使用的 CNN 中的参数都是同一套，从而大大减少参数的规模。

同样的，对于输入层，ELMo 也是用了一样的 CNN 结构，只不过参数不一样而已。和输出层中的分析类似，输入层中 CNN 的引入同样可以减少参数规模。不过 *Exploring the Limits of Language Modeling* 文中也指出了训练时间会略微增加，因为原来的 look-up 操作可以做到更快一些，对 OOV 问题也能够比较好的应对，从而把词典大小不再限定在一个固定的词典大小上。

最终 ELMo 的主要结构便如下图 (b) 所示，可见输入层和输出层都是一个 CNN，中间使用 Bi-LSTM 框架，至于具体细节便如上两张图中所示。



最后，在大规模语料上训练完成的这种 CNN-BIG-LSTM 模型，怎么用呢？其实，如果把每一层的输出结果拿出来，这里大概有三层的词向量可以利用：输入层 CNN 的输出，即是 LSTM 的输入向量，第一层 LSTM 的输出和第二层的输出向量。

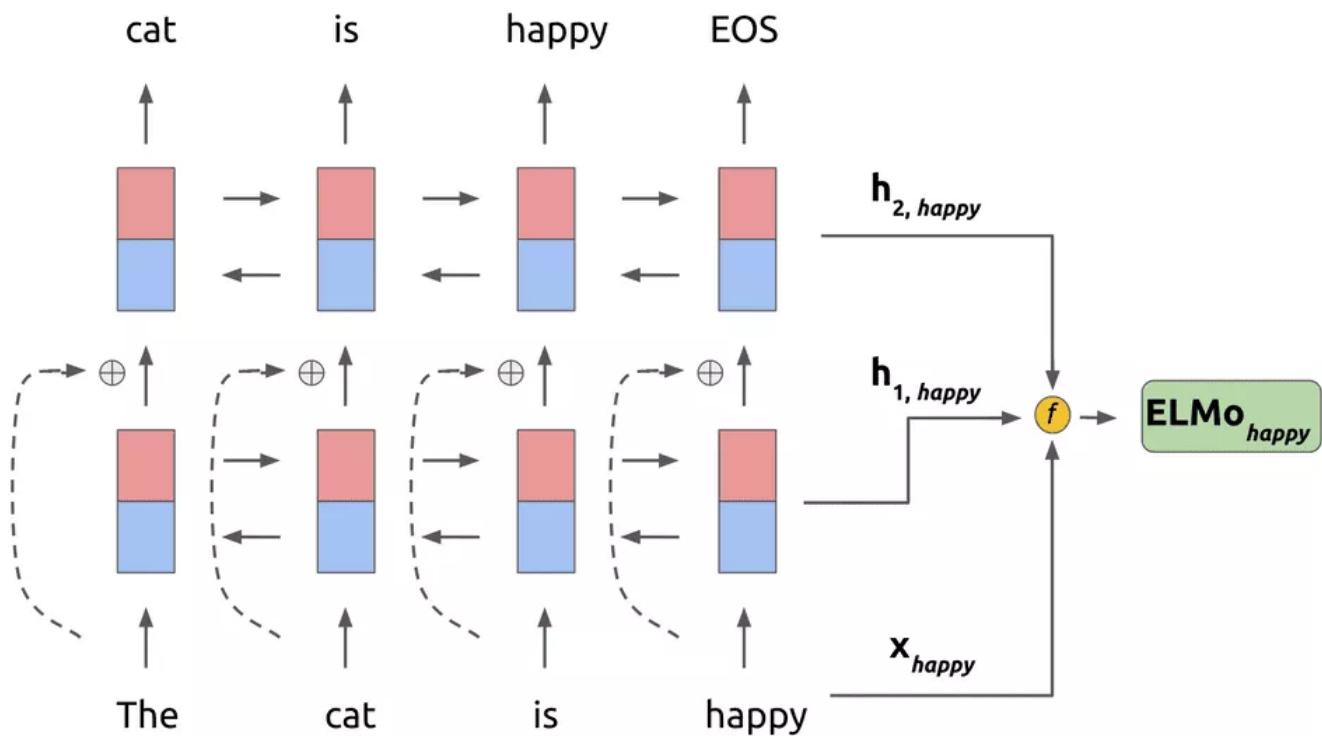
又因为 LSTM 是双向的，因此对于任意一个词，如果 LSTM 的层数为 L 的话，总共可获得的向量个数为  $2L+1$ ，表示如下：

$$R_k = x_k, \overrightarrow{\mathbf{h}}_{k,j}, \overleftarrow{\mathbf{h}}_{k,j}, j = [1, 2, \dots, L]$$

到这里还只是把 ELMo 的向量给抽取出来了。对于每一个词，可以根据下面的式子得到它的向量，其中  $\gamma$  是一个 scale 因子，加入这个因子主要是想将 ELMo 的向量与具体任务的向量分布拉平到同一个分布水平，这时候便需要这么一个缩放因子了。

另外， $s_j$  便是针对每一层的输出向量，利用一个 softmax 的参数来学习不同层的权值参数，因为不同任务需要的词语意义粒度也不一致，一般认为浅层的表征比较倾向于句法，而高层输出的向量比较倾向于语义信息。因此通过一个 softmax 的结构让任务自动去学习各层之间的权重，自然也是比较合理的做法。

$$\mathbf{ELMo}_{k,task} = \gamma^{task} \sum_j j = 0^L s_j^{task} \mathbf{h}_{k,j}$$



前面我们说过，无论是基于传统统计的 N-gram 还是普通神经网络的 NNLM 结构，都会有一个很严重的问题，那就是计算复杂度随着上下文窗口 N 大小的增大急剧上升。其中 N-gram 是指数上升，NNLM 是以  $|d| \times N$  的形式增加， $|d|$  是词向量的维度，虽然 NNLM 已经改观了很多，但依然是一个斜率很大的线性增加关系。

后来 CBOW 和 Skip-gram 以及再后来的 GloVe 终于做到了计算复杂度与所选窗口大小无关，只与词典大小和词向量维度相关。

不过需要指出的是，这里讨论的计算复杂度只是预测单个词的计算时间复杂度，如果是求整个输入序列的话，还是避免不了要与序列长度相关，在这一点上和下面要分析的 RNN 在横向的时间序列上有一个时间复杂度，其原因是一致的。

并且近些年得益于硬件持续的摩尔定律发挥威力，机器的计算能力也有长足的进步，因此在这两方面因素的作用下，以 word2vec 为代表的方法大放光彩，引领了一波 NLP 的发展浪潮。

然而，在今天看来，无论 word2vec 中的模型、还是 GloVe 的模型，都过于简单，它们都受限于所使用的模型表征能力，某种意义上都只能得到比较偏上下文共现意义上的词向量，并且也很少考虑过词序对于词的意义的影响（比如 CBOW 从其名称来看就是一个 bag-of-words，在模型的输入中没有词序的概念）。

理论上，RNN 结构的计算复杂度，跟两个方向上都有关系，一方面是纵向上，另一方面是横向，**纵向上主要是 RNN 结构本身的时间复杂度**，这个复杂度只与 RNN 结构内部的 hidden state 维度以及模型结构的复杂度，在 ELMo 中的话还跟词典大小相关（因为最后一层还是一个词典大小上的分类问题，以及输入也需要维护一个词典大小的 loop up 操作）。

**在横向上的计算复杂度，就主要是受制于输入序列的长度**，而 RNN 结构本身因为在时间序列上共享参数，其自身计算复杂度这一部分不变，因而总的 ELMo 结构计算复杂度主要有词典大小、隐藏层输出维度大小、模型的结构复杂度以及最后的输入序列长度。

前三者可以认为和之前的模型保持一致，最后的输入序列长度，也只是与其保持线性关系，虽然系数是单个 RNN 单元的计算复杂度，斜率依然很大（通常 RNN 结构的训练都比较费时），但是在机器性能提升的情况下，这一部分至少不是阻碍词向量技术发展的最关键的因素了。

因此，在新的时代下，机器性能得到更进一步提升的背景下，算法人员都急需一种能够揭示无论词还是句子更深层语义的方法出现，我想 ELMo 正是顺应了这种时代的需要而华丽诞生。

**ELMo 的思想足够简单，相比它的前辈们，可以说 ELMo 并没有本质上的创新，连模型也基本是引用和拼接别人的工作，它的思想在很多年前就已经有人在用，并没有特别新奇的地方。**

这似乎从反面证明了真正漂亮的工作从来不是突出各自的模型有多么绚丽，只有无其他亮点的论文，才需要依靠描摹了高清足够喜人眼球的图片去吸引评审人的注意力。因此从这个角度去看，似

乎可以得出一个啼笑皆非的结论：论文的漂亮程度与论文图的漂亮程度呈反比。

但同时它的效果又足够惊艳，它的出现，在 2018 年初，这个也许在 NLP 历史上并没有多么显眼的年头，掀起了一阵不小的波澜，至少在 6 项 NLP 任务上横扫当时最好的结果，包括 SQuAD, SNLI, SRL, NER, Coref 和 SST-5。

而后来的故事以及可预见的将来里，这或许仅仅只是一个开始，就如山洪海啸前的一朵清秀的涟漪。

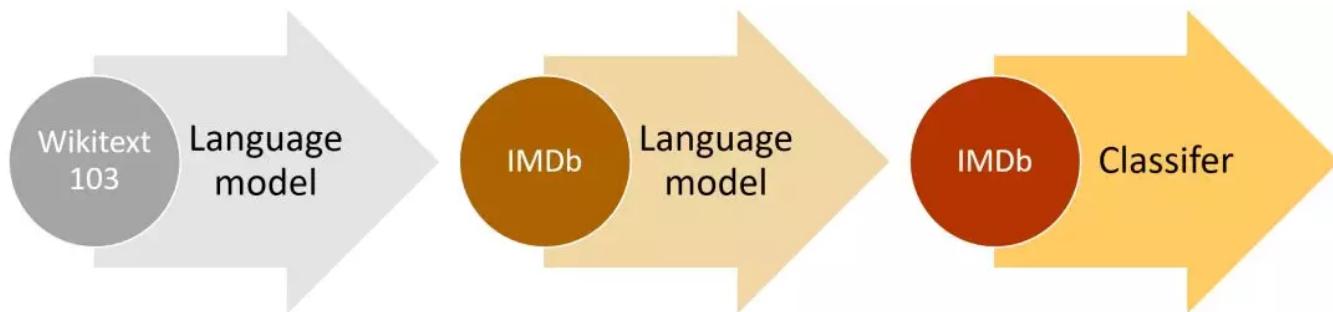
## ULMFit

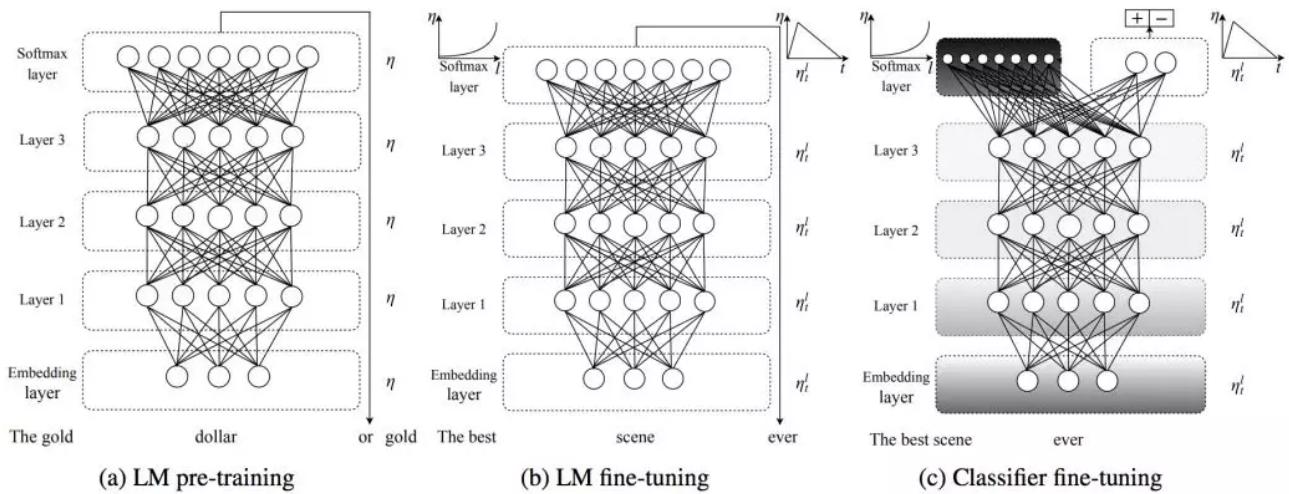
差不多和 ELMo 同期，另一个同样非常惊艳的工作也被提出来，这个团队是致力于将深度学习普及和易用的 Fast AI，而论文两位共同作者之一的 Jeremy Howard，其实就是 Fast AI 的创始人，是 Kaggle 之前的 president 和首席科学家，并且亲自参与过 Kaggle 上很多比赛，长期排在排行榜的第一。

在论文 *Universal Language Model Fine-tuning for Text Classification* 中，他们提出了 **ULMFit** 结构，其实这本质上是他们提出的一个方法，而不是具体的某种结构或模型。只不过正如论文标题所言，他们主要把它应用于文本分类问题中。

和 ELMo 相同的地方在于，ULMFit 同样使用了语言模型，并且预训练的模型主要也是LSTM，基本思路也是预训练完成后在具体任务上进行 finetune，但不同之处也有很多。

首先，**ULMFit** 的预训练和 **finetune** 过程主要分为三个阶段，分别是在大规模语料集上（比如 Wikitext 103，有 103 million 个词）先预训练，然后再将预训练好的模型在具体任务的数据上利用语言模型来 finetune（第一次 finetune，叫做 LM finetune），再根据具体任务设计的模型上，将预训练好的模型作为这个任务模型的多层，再一次 finetune（第二次 finetune，如果是分类问题的话可以叫做 Classifier finetune），整个过程如下所示：





其次，所使用的模型来自于 2017 年 Salesforce 的论文 ***Regularizing and Optimizing LSTM Language Models***，在这篇文章中，他们提出了 ***AWD-LSTM***，正如名字中所揭示的，这个框架更多的是一种训练方法，主要思想分为两大块，其中 Averaged SGD 是指先将模型训练到一定 epoch，然后再将其后的每一轮权值进行平均后，得到最终的权值，用公式表示就是，普通的 SGD 方法权值更新过程为：

$$w_{k+1} = w_k - \gamma_k \nabla f(w_k)$$

其中  $k$  代表迭代次数，而  $f$  则是 loss function，这就是普通的一个 SGD 权值更新迭代式子，那么 ASGD 则把它变成了：

$$w = \frac{1}{K - T + 1} \sum_{i=T}^K w_i$$

其中  $T$  是一个阈值，而  $K$  是总共的迭代次数，这个式子的意思就是把迭代到第  $T$  次之后，对该参数在其后的第  $T$  轮到最后一轮之间的所有值求平均，从而得到最后模型的该参数值。而相应的，普通的 SGD 则是直接取  $w = w_k$  作为最后模型的参数值。

除了使用 ASGD 方法训练模型之外，在普通 LSTM 上一个时刻和下一个时刻之间的隐藏层之间是有连接的，并且这个连接通过一个全连接的矩阵相连，而这个模型则用了 DropConnect 的方法随

机 drop 掉一些连接，从而减少了一些过拟合的风险，当然在输入层到隐藏层之间也有正常的 dropout 操作。

第三，微调方法设计非常精妙。作者提出了几种微调的技巧，它们是：**discriminative fine-tuning, slanted triangular learning rates** 以及 **gradual unfreezing**，分别来看一下。

discriminative fine-tune 的基本思想是针对不同层在训练更新参数的时候，赋予不同的学习率。这里的出发点是，对于 NLP 的深度学习模型来说，不同层的表征有不同的物理含义，比如浅层偏句法信息，高层偏语义信息，因此对于不同层的学习率不同，自然就是比较合理的了。具体公式如下：

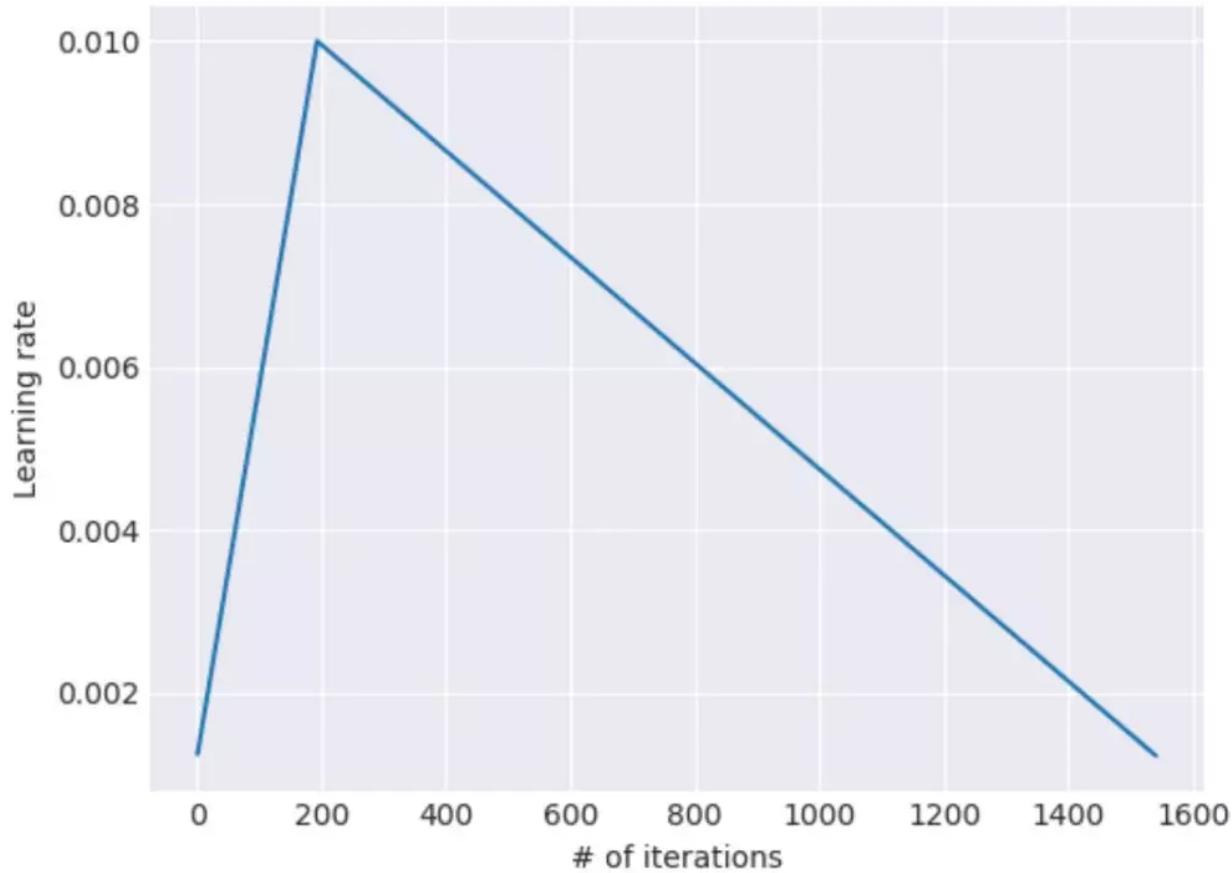
$$\theta_t^l = \theta_{t-1}^l + \eta^l \nabla_{\theta^l} J(\theta)$$

这里的  $\eta^l$  便是不同层  $l$  有不同的学习率，原文也给出了具体的选择：先指定最后一层的学习率，然后根据下式得到前面层的学习率，基本思想是让浅层的学习率要更小一些。

$$\eta^{l-1} = \frac{\eta^l}{2.6}$$

而对于 slanted triangular learning rates 来说，主要思想便是在 finetune 的第一阶段，希望能够先稳定住原来已经在大规模语料集上预训练好的参数，所以选择一个比较小的 finetune 学习率。而后希望能够逐步加大学习率，使得学习过程能够尽量快速。当训练接近尾声时，逐步减小学习率，这样让模型逐渐平稳收敛。

这个思想，个人觉得大概借鉴了 2017 年谷歌提出 Transformer 时用到的 warm up 的学习率调节方法，这个方法也是在训练的时候先将学习率逐步增大，然后再逐步减小。因此，这样一个三段论式的学习过程，用图表示如下：



另一个 finetune 的技巧是 gradual unfreezing，主要思想是把预训练模型在新任务上 finetune 时，逐层解冻模型，也就是先 finetune 最后一层，然后再解冻倒数第二层，把倒数第二层和最后一层一起 finetune，然后再解冻第三层。以此类推，逐层往浅层推进，最终 finetune 整个模型或者终止到某个中间层。这样做的目的也是为了 finetune 过程能够更平稳。

当然，值得提出的是，因为 ULMFiT 中包含了两次 finetune，即在新任务上用语言模型 finetune 和在新任务上 finetune 训练一个最终的 task-specifi-model（比如分类器）。

而论文中主要把 discriminative fine-tuning 和 slanted triangular learning rates 这两个技巧用在了语言模型的 finetune 阶段，把最后一个 gradual unfreezing 的技巧应用在最终 task-specifi-model 的 finetune 阶段。

通过上面的这些方法，ULMFiT 最终在分类任务上表现惊艳，尤其是只需要 100 个标记数据，就能够学习到一个表现非常 comparable 的分类器。不得不说，这个过程中预训练的语言模型，对最终表现起到了至关重要的作用。

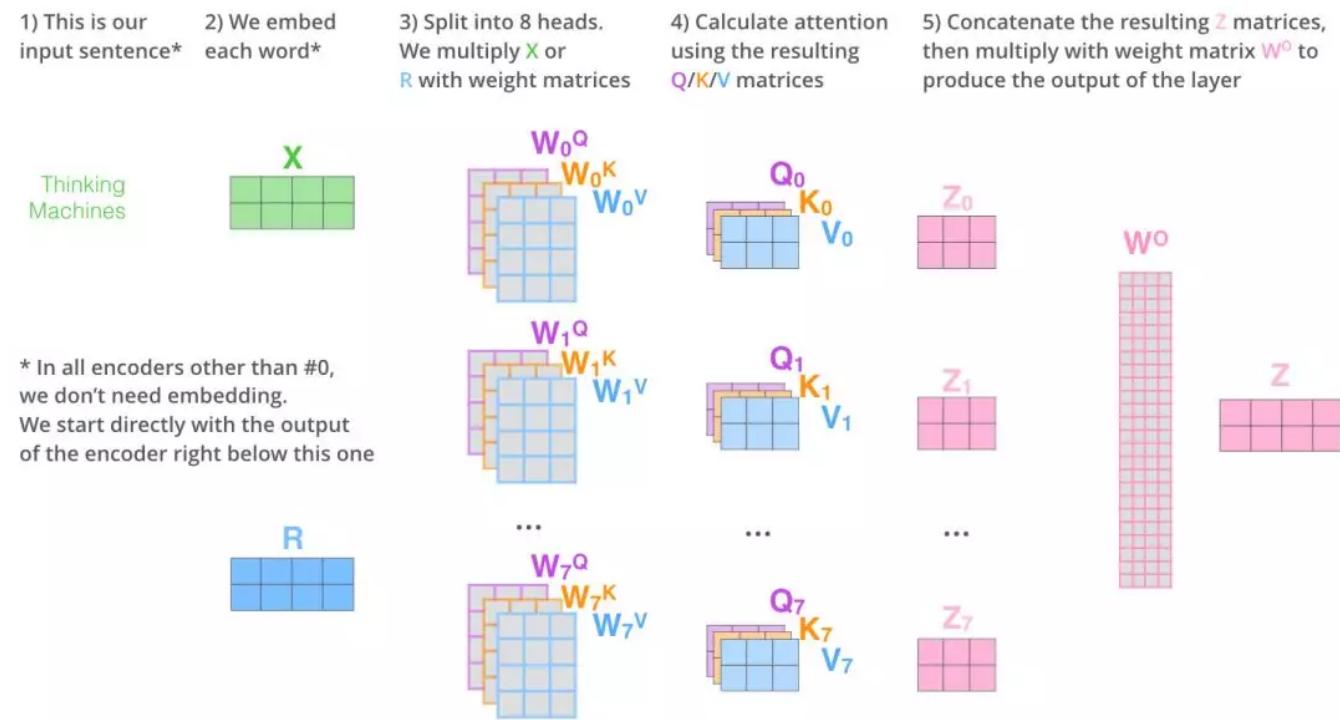
## GPT

大规模语料集上的预训练语言模型这把火被点燃后，整个业界都在惊呼，原来预训练的语言模型远不止十年前 Bengio 和五年前 Mikolov 只为得到一个词向量的威力。

然而很快在 2018 年 6 月，不再属于“钢铁侠”马斯克的 OpenAI，发了一个大新闻，相关论文是 ***Improving Language Understanding by Generative Pre-Training***，往这把火势正猛的烈焰上加了一剂猛料，从而将这把火推向了一个新的高潮。

OpenAI 的猛料配方里，第一剂主料便是谷歌于 2017 年中提出的 **Transformer 框架** (*Attention Is All You Need*)。因此，让我们先来弄明白 Transformer 里面都有什么东西。

私以为，Transformer 里最为核心的机制是 Self-attention，正因为 Self-attention 的存在，才使得 Transformer 在做类似翻译问题的时候，可以让其 Encoder 不用做序列输入，而是将整个序列一次全输入，并且超长序列的输入也变得可能。而具体到 Self-attention 中，可以用下图表示：



简单说来，输入为句子的矩阵，先分别通过三个全连接矩阵将输入矩阵变化为三个矩阵，分别为  $Q$ ,  $K$  和  $V$ ，然后通过  $Q$  和  $K$  计算得到一些权值，将这些权值加权求和到  $V$  矩阵上，便可以得到一个新的矩阵表示。

而 Self-attention 中的多头机制便是将这样的操作分别进行多次，让句子的表征充分学习到不同的侧重点，最终将这些多头学习出来的表征 concat 到一起，然后再同一个全连接网络，便可以得到这个句子最终 Self-attention 下新的表示。

将其中的每一个头的操作过程用公式表示如下，需要注意的是 softmax 是针对矩阵的 row 方向进行操作得到的。所以，说白了，这个公式表示的意思就是针对  $V$  进行加权求和，加权权值通过  $Q$  和  $K$  的点乘得到。

$$\text{softmax} \left( \frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}$$

$$= \mathbf{Z}$$

不过其实 Self-attention 和普通 attention 机制在形式上几乎完全等价。主要区别在于，对于普通的 attention 机制，输入可能有多个，并且下式在求得  $e_{ij}$  中的  $v_a^T$  实际上是一个全连接网络，将式子右边的部分（也就是 attention 的输入）映射为一个值，从而可以根据这个值计算 attention 的权值大小。

除此之外，普通 attention 和 self-attention 并没有本质不同，最大的区别还是在于在自我输入上计算 attention 权值大小。

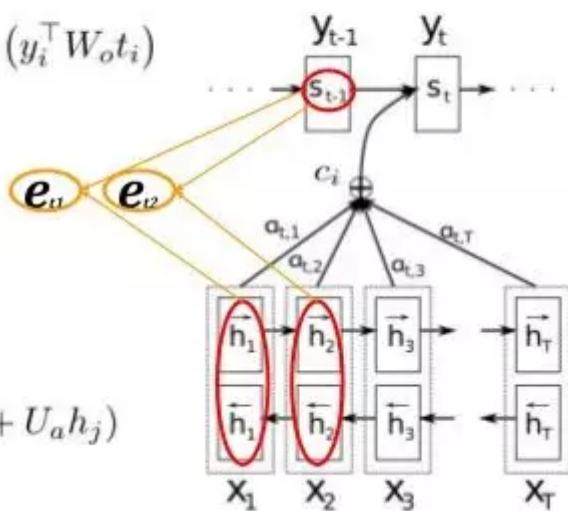
## Attention Mechanism

$$\text{Softmax} \quad p(y_i | s_i, y_{i-1}, c_i) \propto \exp (y_i^\top W_o t_i)$$

$$\text{Context} \quad c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

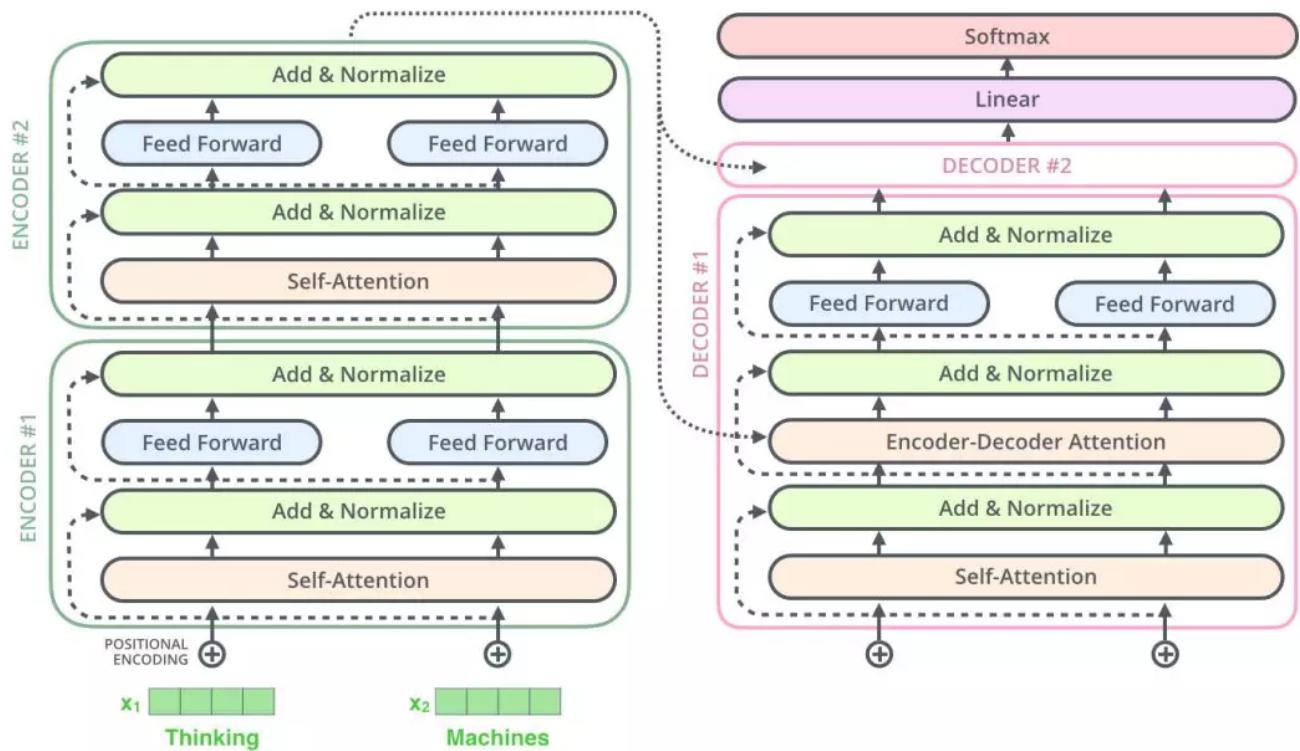
$$\alpha_{ij} = \frac{\exp (e_{ij})}{\sum_{k=1}^{T_x} \exp (e_{ik})}$$

$$e_{ij} = v_a^\top \tanh (W_a s_{i-1} + U_a h_j)$$



在 Transformer 的 Encoder 中，还有一些其他设计，比如加入 position embedding（因为 Transformer 的 Encoder 中不是时序输入词序列，因此 position embedding 也是主要位置信息）；Residual 结构，使得模型的训练过程更为平稳；此外还有 normalization 层，接着便是 feed forward 层（本质上是一个两层的全连接网络，中间加一个 ReLu 的激活函数）。

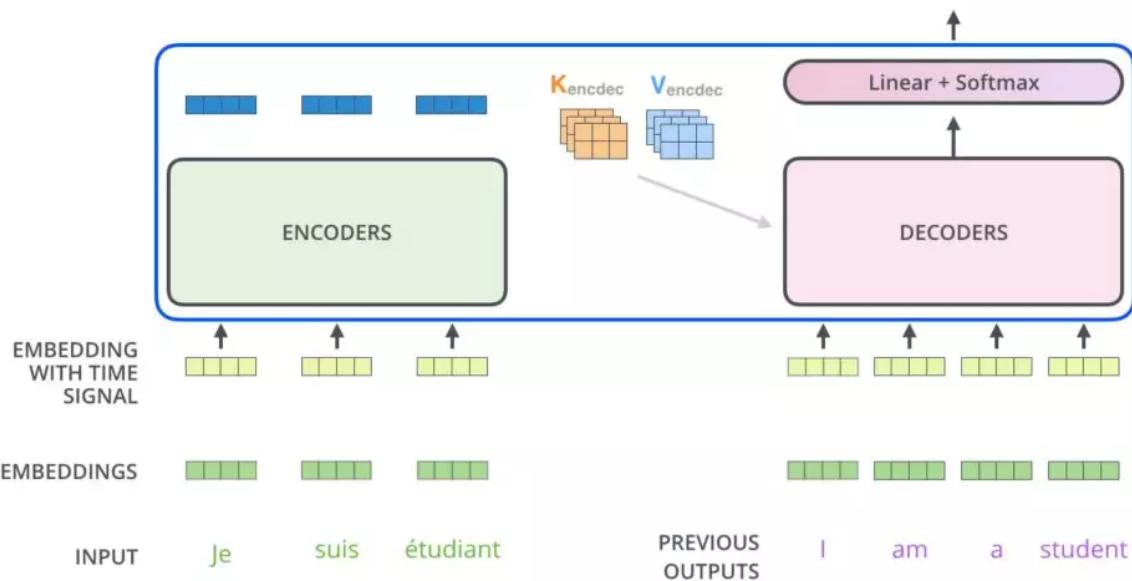
Decoder 的结构与此类似，只不过在进行 decode 的时候，会将 Encoder 这边的输出作为 Decoder 中 Self-attention 时的 K 和 V。



对于 decode 过程，具体来看，大致过程如下。

Decoding time step: 1 2 3 4 5 6

OUTPUT I am a student &lt;end of sentence&gt;



Decoder 实际上还有很多细节，一般来说，训练时 Decoder 中的输入可以用矩阵形式一次完成当前整个序列的 decode 过程，因为 ground truth 已经提前知道，只需做好每个词的 mask 就好（为了避免待预测的词影响到当前的输入）。

然而在做 inference 的时候，Decoder 必须按照序列输入，因为在生成每一个词的时候，必须先生成它的前一个词，无法一次将整个序列全部生成（当然理论上也可以，但是效果并不好）。

在矩阵运算过程中，Decoder 中有许多 mask 操作，参与运算的三个矩阵 Q, K 和 V 都要做许多 mask 操作，主要有两方面作用：一方面是消除输入句子本身长度之外的 padding 影响，另一方面是 decode r 必须要求不能提前看到待生成的词。

除了 mask 操作，另外值得注意的是，和 Encoder 中只有一种类型 Self-attention 不同的是，Decoder 的 attention 实际包含两部分：

**第一部分是带有 mask 的 Self-attention**，通过 mask 将 decode 阶段的 attention 限定只会 attention 到已经生成过的词上，因此叫做 Mask Self-attention。

**第二部分是普通的 Self-attention 操作**，不过这时的 K 和 V 矩阵已经替换为 Encoder 的输出结果，所以本质上并非一个 Self-attention。

下面的动图很好地表现了 decoding 过程，生成每一个词的时候，既和 Encoder 的输出信息有关，也和已经生成过的词相关。

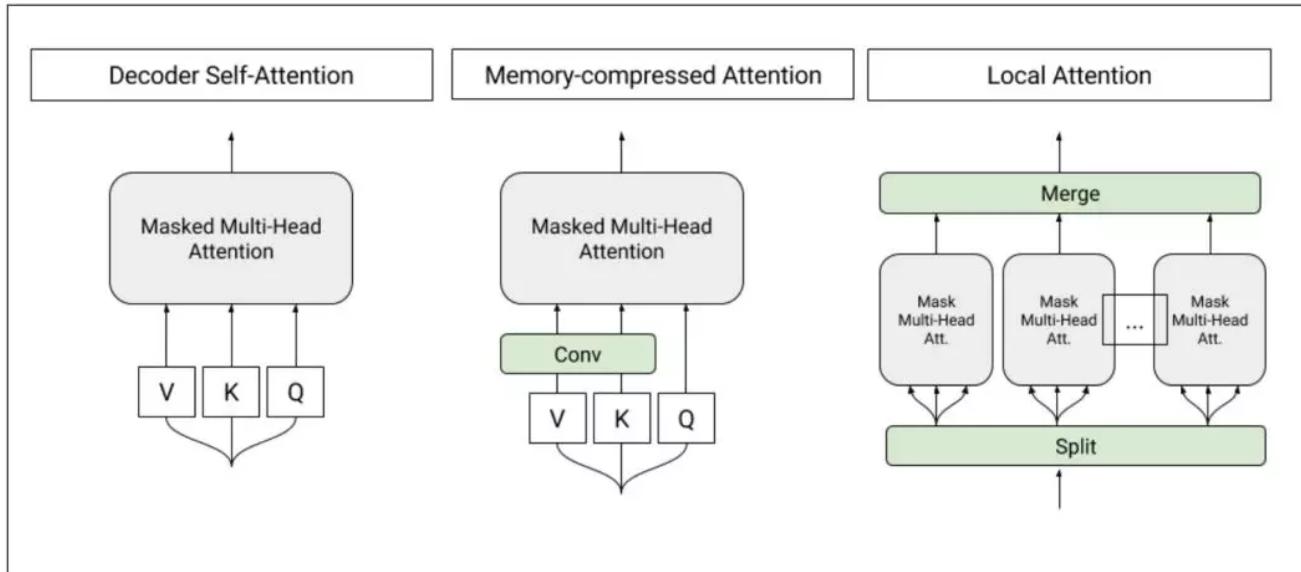
大体介绍完 Transformer 后，再看看 GPT 中是怎么用 Transformer 的。按照论文的说法，GPT 使用的 Transformer 是只用了 Decoder，因为对于语言模型来讲，确实不需要 Encoder 的存在。

而具体模型，他们参考了 2018 年早些时候谷歌的 ***Generating Wikipedia by Summarizing Long Sequences***，GPT 名称中的 Generative 便是源自这篇文章，二者都有用到生成式方法来训练模型，也就是生成式 Decoder。

关于这篇论文中提到的 **T-DMCA**，实际上就是一个 Decoder，只不过这篇文章中要做超长的序列输入（可以长达 11000 个词），为了能够高效节省时间和内存的处理如此长的序列，做了一些 Memory-Compressed 工作，主要是两方面：

一方面是把一个 batch 内部的序列按长度进行分组，然后分别在每个组内部进行 self-attention 操作，避免将一些很短的句子也 padding 到整个语料的最大长度；另一方面，通过 CNN 操作，把 K

和 V 压缩到序列长度更小的一个矩阵，同时保持 Q 不变，这样也能在相当程度上减少计算量。



除了这些具体的模型细节外，GPT 本质上就是用了语言模型的目标函数来优化和训练 Transformer-Decoder，这和上文提到过的语言模型保持一致。

利用语言模型的目标函数预训练完成后，便可以在具体任务上进行 finetune，和 ULMFiT 中的 finetune 分为两个阶段的方法不一样的是，GPT 直接把这两个过程糅合到一个目标函数中，如：

$$L_3(C) = L_2(C) + \lambda L_1(C)$$

其中  $L_2$  是 task-specific 的目标函数， $L_1$  则是语言模型的目标函数。论文中说这种联合学习方式能够让训练效果更好。而在具体如何做迁移学习的方面，GPT 大概也同样借鉴了上面提到的 **Generating Wikipedia by Summarizing Long Sequences** 中的做法，非常巧妙地将整个迁移学习的框架做到非常精简和通用。

分类问题中，直接在原序列的开始和末尾添加表示开始和末尾的符号，在 Text Entailment 问题中，将 Premise 和 Hypothesis 通过一个中间分隔符“\$”连接起来成为一个序列，然后同样在开头和末尾添加标记符号。

文本相似问题中，因为序列 1 和序列 2 没有先后关系，因此将先后关系相反的两个序列作为输入。在 Question Answering 中，将 query 和每个候选的 answer 都分别连接成一个序列作为输入，最

后按各自的打分进行排序。

因此，这套输入的表示方法，基本可以使用同一个输入框架来表征许多文本问题（以至于后来的 **BERT** 直接借用了这套做法）。

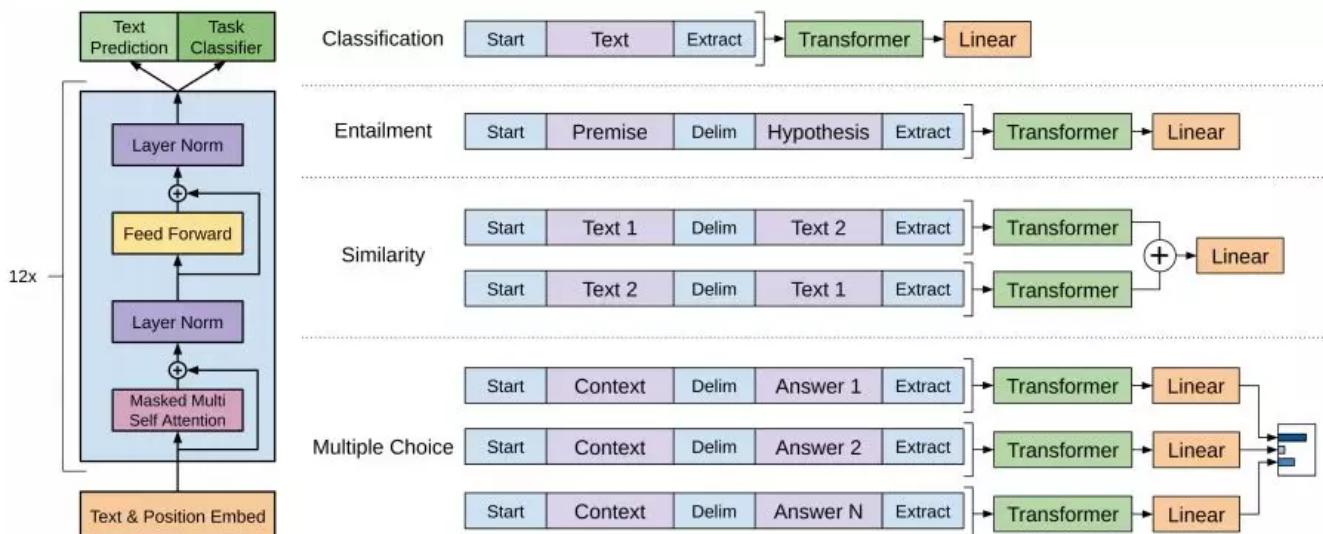
除此之外，在输出层，只需要接入一个很简单的全连接层或 MLP，根本不需要非常复杂的模型设计。而整个 finetune 阶段，新加入的参数极少，只有输出层以及输入层中添加的一些特殊标记（比如分隔符）。

正是因为有了输入层和输出层的这种通用化设计考虑，一旦中间的 Transformer（当然，正如前文所说，这里的 Transformer 在使用语言模型进行预训练时只有 Decoder 部分。在将其当做文本特征提取器的时候，相应的也可以很便利的将其变成 Encoder）表征能力足够强大，迁移学习在 NLP 任务中的威力也会变得更为强大。

果不其然，GPT 在公布的结果中，一举刷新了 12 项 NLP 任务中的 9 项榜单，效果不可谓不惊艳。然而对于 OpenAI 来讲，GPT 底层模型使用的是谷歌提出的 Tranformer，正是依靠了 Transformer 的强大表征能力，使得最终的效果有了一个坚实的基础。

然而仅仅过了四个月之后的 **BERT** 横空出世，同样也是用了 Transformer，同样是谷歌，甚至很多思想也是直接借鉴 GPT。GPT 作为与 BERT 气质最为接近的工作，同时也是 BERT 的前辈，得到的待遇差别如此之大，不知道 GPT 是否有些可惜和遗憾。

相比 BERT，GPT 并没有带来特别巨大的反响，他的惊艳亮相，迅速变为水里的一声闷响，掀起了一阵涟漪后迅速消散，将整个舞台让位于正值青春光艳照人的 BERT，颇有点“成也萧何败也萧何”的味道。



## BERT

如果要用一句时下正流行的话来形容 BERT 的出现，这句话大概再恰当不过：一切过往，皆为序章。

2018 年 10 月 11 日，这似乎是个绝对平凡的日子。说句题外话，无独有偶，OpenAI 在其博客放出 GPT 的时间恰好不多不少是 4 个整月前，即 2018 年 6 月 11 日），然而 Google AI 的 Jacob Devlin 和他的合作者们悄悄地在 arXiv 上放上了他们的最新文章，名为 ***BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.***

随后先是在 Twitter 上引发了一波浪潮，同样是 Google AI 团队的 Thang Luong 在其 Twitter 上直言这项工作是一个划时代的工作。然后在中文社区也迅速发酵，那几天 NLP 从（搬）业（砖）人员的朋友圈基本被 BERT 刷屏，即便时至一个多月后的今日，这一现象也基本不见消退的迹象，几乎大家见面闲聊开口必谈 BERT，似乎做 NLP 的大可不必使用华夏光辉灿烂文明之一的“吃过了吗？”，而替换为今日的“BERT 跑起来了吗？”，可见一斑。

不得不说，Jacob 和他的小伙伴们真是大手笔，和 4 个月前他的前辈 GPT 至少一样，乃至野心更大。除了 GPT 刷过的那些榜之外，BERT 还添加了一项任务 SQuAD（这大概是当前 NLP 最为火热的任务之一），也许这里的每一项任务都可能是一个研究小组的活命本钱，甚至还有可能是某个评教授职称的救命稻草。

然而，BERT 丝毫不放在眼里，直接将它的铁蹄踏入 11 项 NLP 任务，将途中目见耳闻所遭遇的一切荡平无余，留给在这之前仍在苦苦挣扎于某个排行榜中的人们无尽的错愕和唏嘘，而 BERT 似乎不曾心软和迟疑片刻。

很快，大概过了不到一个月，Google AI 把他们已经预训练好的 BERT 模型公布出来，包括英语的 base 和 large 模型，另外针对其他语言也放出了中文（仅有的一个非英语的单个模型）和一个 102 种语言的混合语言预训练模型，这再次触发和引爆了 NLP 界的集体高潮。

不过，为何 BERT 能如此引人注目，不妨来一探究竟。私以为，BERT 最主要的几个特征分别是：

- 利用了真双向的 Transformer；

- 为了利用双向信息，改进了普通语言模型成为完形填空式的 Mask-LM (Mask-Language Model)；
- 利用 Next Sentence Prediction 任务学习句子级别信息；
- 进一步完善和扩展了 GPT 中设计的通用任务框架，使得 BERT 能够支持包括：句子对分类任务、单句子分类任务、阅读理解任务和序列标注任务。

为了更深入理解 BERT，我们分别来看看他的这些特征。

首先看看 BERT 如何使用双向 Transformer。其实很好理解，用一句话来回答为什么 BERT 使用双向 Transformer：BERT 用了 Transformer 的 Encoder 框架。

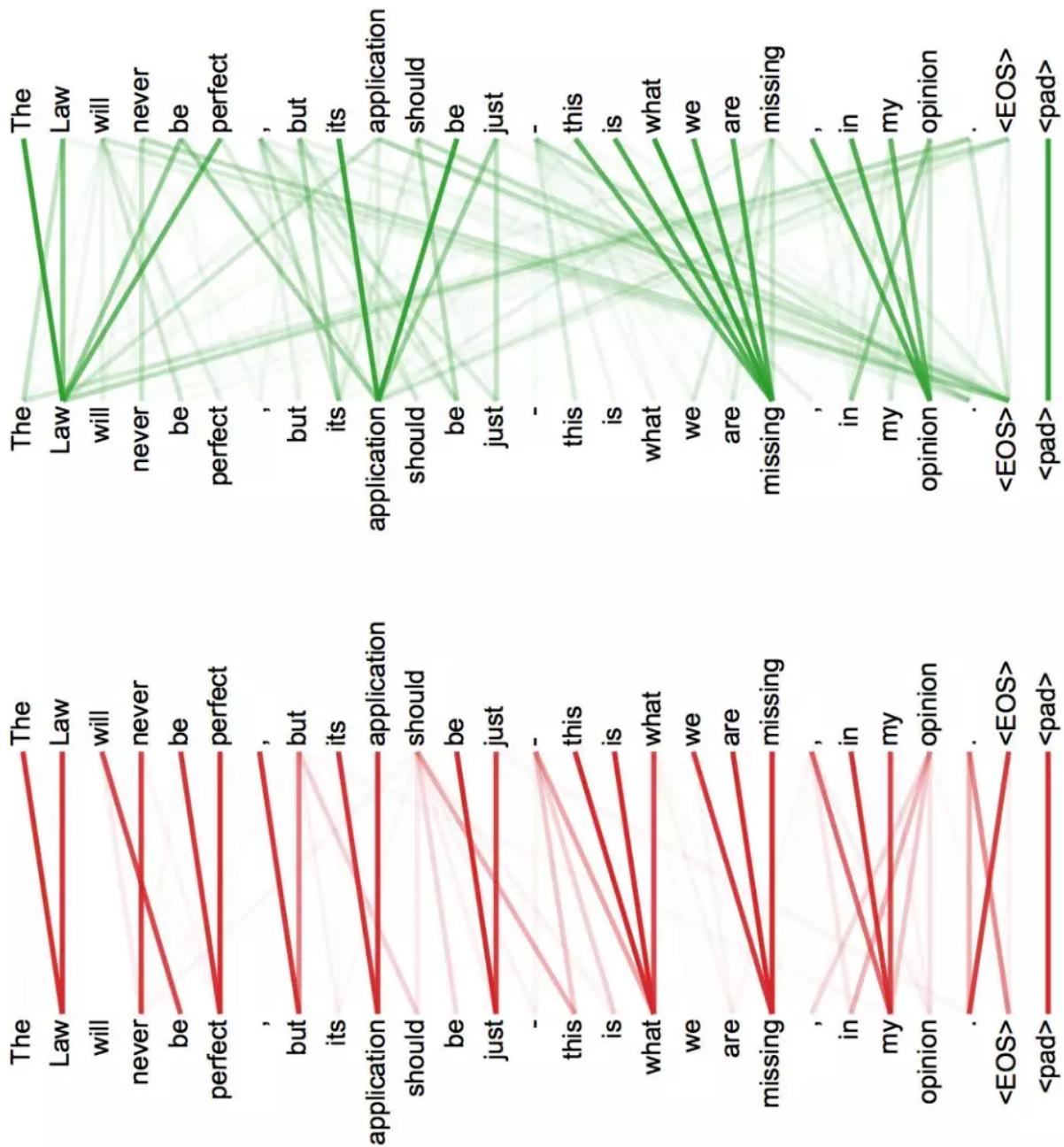
但是，我想这样的答案自然是要扣分的，更“求生欲”一点的答案是：因为 Encoder 中用了 Self-attention 机制，而这个机制会将每一个词在整个输入序列中进行加权求和得到新的表征。

更通俗的说法是每一个词在经过 Self-attention 之后，其新的表征将会是整个输入序列中所有词（当然也包括它本身）的加权求和，每一个词都达到了“我中有你，你中有我”的境界。

如果经过更多的 transformer 的 block (意味着经过更多 Self-attention)，那么互相交融的程度将会更高，类似于 BERT 这样深的结构 (Base 模型是 12 层，Large 模型是 24 层)，经过所有 block 后，彼时，遑论“我中有你，你中有我”了，一切早已是“我在哪里”和“我是谁”这样的哲学式命题了，也就是连“你我”的界限都早已混沌的状态了。

因此，说 BERT 是双向的语言模型结构，不仅丝毫不过分，它实质上是一种将序列中每一个词之间无比交融在一块的模型。更可贵的是，交融的姿势还可以多种多样，这可从 Large 版本 BERT 的多头机制中 Head 个数多达 16 个，体会到这种交融的程度，可谓丝丝入扣。

这里仅给出一例，下图只是两个 head 学习到的交融模式，如果多达 16 个 head，这样的交融模式还要重复 16 次，可见一斑。



而相应的在 ELMo 与 GPT 中，它们并没有用上这种交融模式，也就是它们本质上还是一个单向的模型，ELMo 稍微好一点，将两个单向模型的信息 concat起来。GPT 则只用了单向模型，这是因为它没有用上 Transformer Encoder、只用了 Decoder 的天生基因决定的。

其实，很多人就把这种 **left-to-right** 的 Transformer 框架叫做 **Decoder**，因为事实上 **Decoder** 就是如此（具体做的时候需要提前把未来待生成的词做好 mask，细节上通过上三角矩阵来实现），这也是 OpenAI 把他们的模型叫做"Generative"的原因所在。

然而，使用双向 Transformer 会有一个问题。正如上面的分析，即便对于 Base 版 BERT 来说，经过 12 个 block，每个 block 内部都有 12 个多头注意力机制，到最后一层的输出，序列中每个位置上对应的词向量信息，早已融合了输入序列中所有词的信息。

而普通的语言模型中，是通过某个词的上下文语境预测当前词的概率。如果直接把这个套用到 Transformer 的 Encoder 中，会发现待预测的输出和序列输入已经糅合在一块了，说白了就是 Encoder 的输入已经包含了正确的监督信息了，相当于给模型泄题了，如此说来普通语言模型的目标函数无法直接套用。

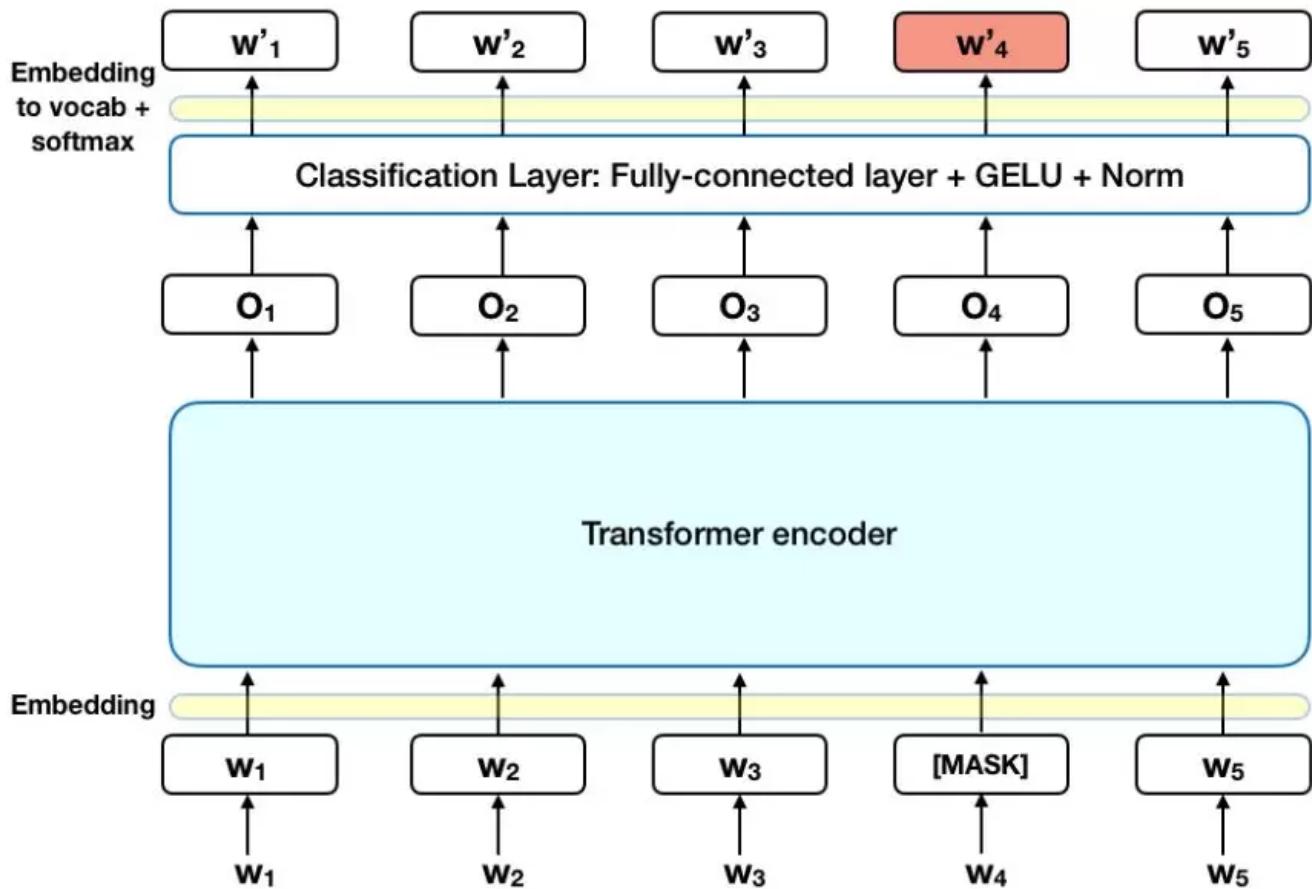
那么，如何解决 Self-attention 中带来了表征性能卓越的双向机制，却又同时带来了信息泄露的这一问题？

BERT 的作者很快联想到了，如果我把原来要预测整个句子的输出，改为只预测这个句子中的某个词，并且把输入中这个词所在位置挖空，这样一来，岂不就不会存在泄露真题的问题了？

Jacob 是这样想的（实际上是参考了很早之前提出的 Cloze 问题），这位以单人徒手搭建大工程著称的牛人行动力超强，立马就把这一方法吸收和实现到 BERT 中，**他们的想法也特别简单：**

- 输入序列依然和普通Transformer保持一致，只不过把挖掉的一个词用"[MASK]"替换；
- Transformer 的 Encoder 部分按正常进行；
- 输出层在被挖掉的词位置，接一个分类层做词典大小上的分类问题，得到被 mask 掉的词概率大小。

正是因为加了 mask，因此 BERT 才把这种方法叫做 Masked-LM，整个过程如下所示：



这就直接把普通语言模型中的生成问题（正如 GPT 中把它当做一个生成问题一样，虽然其本质上也是一个序列生成问题），变为一个简单的分类问题，并且也直接解决了 Encoder 中多层 Self-attention 的双向机制带来的泄密问题（单层 Self-attention 是真双向，但不会带来泄密问题，只有多层累加的 Self-attention 才会带来泄密问题），使得语言模型中的真双向机制变为现实。

不过，BERT 针对如何做“[MASK]”，做了一些更深入的研究，它做了如下处理：

- 选取语料中所有词的 15% 进行随机 mask；
- 选中的词在 80% 的概率下被真实 mask；
- 选中的词在 10% 的概率下不做 mask，而被随机替换成其他一个词；
- 选中的词在 10% 的概率下不做 mask，仍然保留原来真实的词。

至于为什么要这么做，BERT 也给出了足够感性的解释。对于要做 mask，这个原因上面已经分析了，就是为了解决双向机制的泄密问题而设计的；而为什么还要有一部分概率不做真正的 mask，而是输入一个实际的词，这样做好处是尽量让训练和 finetune 时输入保持一致，因为 finetune 输入中是没有“[MASK]”标记的。

对于保留为原来的真实词，也就是真的有 10% 的情况下是泄密的（占所有词的比例为  $15\% * 10\% = 1.5\%$ ），作者说这样能够给模型一定的 bias，相当于是额外的奖励，将模型对于词的表征能拉向词的真实表征。

笔者个人理解是：因为输入层是待预测词的真实 embedding，在输出层中的该词位置得到的 embedding，是经过层层 Self-attention 后得到的，这部分 embedding 里肯定多少保留有部分输入 embedding 的信息，而这部分的多少就是通过输入一定比例真实词所带来的额外奖励，最终会使得模型的输出向量朝输入层的真实 embedding 有一个偏移。如果全用 mask 的话，模型只需保证输出层的分类准确，对于输出层的向量表征并不关心，因此可能会导致最终的向量输出效果并不好。

最后，BERT 对选中的词在 10% 概率下不做 mask，而是被随机替换成为一个其他词，这样做的目的，BERT 也给出了他们的解释：因为模型不知道哪些词是被 mask 的，哪些词是 mask 了之后又被替换成了一个其他的词，这会迫使模型尽量在每一个词上都学习到一个全局语境下的表征，因而也能让 BERT 获得更好的语境相关的词向量（这正是解决一词多义的最重要特性）。

其实更感性的解释是，因为模型不知道哪里有坑，所以随时都要提心吊胆，保持高度的警惕。正如一马平川的大西北高速公路，通常认为都是路线直，路面状况好，但如果掉以轻心，一旦有了突发情况，往往也最容易出事故，鲁棒性不高；而反倒是山间小路，明确告诉了每一位司机路面随时都有坑，并且没法老远就提前知道，所以即便老司机也只能小心翼翼的把稳方向盘慢慢开，这样做反倒鲁棒性更高。

正所谓《左传》中所言“居安思危，思则有备，有备无患，敢以此规”，BERT 的这种设计又何尝不是“居安思危”的典范，Jacob 给出的原文解释如下：

*The Transformer encoder does not know which words it will be asked to predict or which have been replaced by random words, so it is forced to keep a distributional contextual representation of every input token.*

然而可惜的是，Jacob 并没有在论文中做更细致的实验，来证明这些言论的正确性，因此可能会存在其他的更优的比例组合。

除了用上 Mask-LM 的方法使得双向 Transformer 下的语言模型成为现实，BERT 还利用和借鉴了 Skip-thoughts 方法中的句子预测问题，来学习句子级别的语义关系。具体做法则是将两个句子组合成一个序列，组合方式会按照下面将要介绍的方式，然后让模型预测这两个句子是否为先后近邻的两个句子，也就是会把“Next Sentence Prediction”问题建模成为一个二分类问题。

训练的时候，数据中有 50% 的情况这两个句子是先后关系，而另外 50% 的情况下，这两个句子是随机从语料中凑到一起的，也就是不具备先后关系，以此来构造训练数据。句子级别的预测思路和之前介绍的 Skip-thoughts 基本一致，当然更本质的思想来源还是来自于 word2vec 中的 skip-gram 模型。

在预训练阶段，因为有两个任务需要训练：Mask-LM 和 Next Sentence Prediction，因此 BERT 的预训练过程实质上是一个 Multi-task Learning。

BERT 的损失函数由两部分组成，第一部分是来自 Mask-LM 的单词级别分类任务，另一部分是句子级别的分类任务。通过这两个任务的联合学习，可以使得 BERT 学习到的表征既有 token 级别信息，同时也包含了句子级别的语义信息。具体损失函数如下：

$$L(\theta, \theta_1, \theta_2) = L_1(\theta, \theta_1) + L_2(\theta, \theta_2)$$

其中  $\theta$  是 BERT 中 Encoder 部分的参数， $\theta_1$  是 Mask-LM 任务中在 Encoder 上所接的输出层中的参数， $\theta_2$  则是句子预测任务中在 Encoder 接上的分类器参数。因此，在第一部分的损失函数中，如果被 mask 的词集合为  $M$ ，因为它是一个词典大小  $|V|$  上的多分类问题，那么具体说来有：

$$L_1(\theta, \theta_1) = - \sum_{i=1}^M \log p(m = m_i | \theta, \theta_1), m_i \in [1, 2, \dots, |V|]$$

在句子预测任务中，也是一个分类问题的损失函数：

$$L_2(\theta, \theta_2) = - \sum_{j=1}^N \log p(n = n_j | \theta, \theta_2), n_j \in [IsNext, NotNext]$$

因此，两个任务联合学习的损失函数是：

$$L(\theta, \theta_1, \theta_2) = - \sum_{i=1}^M \log p(m = m_i | \theta, \theta_1) - \sum_{j=1}^N \log p(n = n_j | \theta, \theta_2)$$

具体的预训练工程实现细节方面，BERT 还利用了一系列策略，使得模型更易于训练，比如对于学习率的 warm-up 策略（和上文提到的 ULMFiT 以及 Transformer 中用到的技巧类似），使用的激活函数不再是普通的 ReLu，而是 GeLu，也是用了 dropout 等常见的训练技巧。

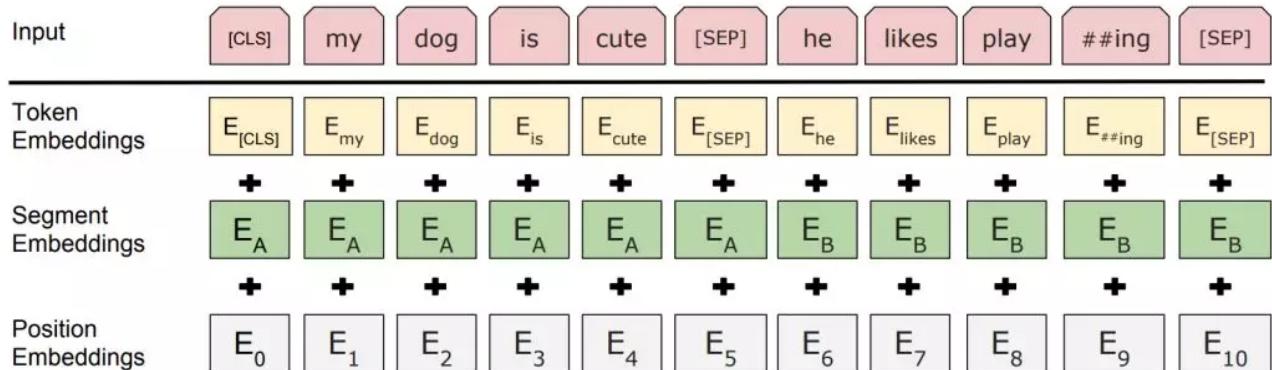
此外，BERT 使用的语料比 GPT 也要大得多（GPT 用的是 BooksCorpus，800 million 个词，BERT 除了 BooksCorpus 之外，还加入了 Wikipedia 数据集，2500 million 个词，总共有 3.3 billion 个词）。

最后，当 BERT 预训练完成后，针对如何利用预训练好的模型，迁移到特定任务背景下，BERT 在 GPT 的基础上，将这个非常重要的工作进行了更深入的设计。由于中间的 Encoder 对于所有任务几乎都可以直接利用，因此这部分的设计主要分为两方面：输入层和输出层。

在输入层方面，思路和 GPT 基本类似，如果输入只有一个句子，则直接在句子前后添加句子的起始标记位和句子的结束符号。在 BERT 中，起始标记都用“[CLS]”来表示，结束标记符用“[SEP]”表示，对于两个句子的输入情况，除了起始标记和结束标记之外，两个句子间通过“[SEP]”来进行区分。

除了这些之外，BERT 还用两个表示当前是句子 A 或句子 B 的向量来进行表示。对于句子 A 来说，每一词都会添加一个同样的表示当前句子为句子 A 的向量，如果有句子 B 的话，句子 B 中的每个词也会添加一个表示当前句子为句子 B 的向量。

当然，和 Transformer 中一样，为了引入序列中词的位置信息，也用了 position embedding，因此，最终的输入层大概是：



除了输入层要尽量做到通用之外，根据不同任务设计不同的输出层也变得尤为重要，BERT 主要针对四类任务考虑和设计了一些非常易于移植的输出层，这四类任务分别是：单个序列文本分类任

务、两个序列文本分类任务、阅读理解任务和序列标注任务，句子或答案选择任务。

对于单序列文本分类任务和序列对的文本分类任务使用框架基本一致，只要输入层按照上面的方法做好表示即可。

这两个分类任务都是利用 Encoder 最后一层的第一个时刻“[CLS]”对应的输出作为分类器的输入，这个时刻可以得到输入序列的全局表征，并且因为监督信号从这个位置反馈给模型，因而实际上在 finetune 阶段也可以使得这一表征尽量倾向于全局的表征。

当然，即便如此，同样也是可以通过一些简单易行的办法将其他时刻甚至其他层内的表征拿来用的，个人认为并不需要局限在这一个时刻上的表征。另外， finetune 阶段，在 BERT Encoder 基础上，这些分类任务因为只需要接一个全连接层，因此增加的参数只有  $H \times K$ ，其中  $H$  是 Encoder 输出层中隐状态的维度， $K$  是分类类别个数。

对于 SQuAD 1.1 任务来说，需要在给定段落中找到正确答案所在区间，这段区间通过一个起始符与终止符来进行标记，因此只需预测输入序列中哪个 token 所在位置是起始符或终止符即可。

这个过程只需维护两个向量，分别是起始符和终止符的向量，不过与其说是向量，还不如说是两个对应的全连接层，只不过这两个全连接有些特殊，在这两个全连接层中，输入层的节点个数为 BERT Encoder 输出的节点个数，也就是  $H$ ，而这两个全连接层的输出层都只有一个节点。

这两个全连接层的作用就是要把序列输出的向量通过这两个全连接层映射为一个实数值，可以等同于打分，然后再根据这个打分在序列的方向上选取最大值作为起始符和终止符。因此这个过程实际上只增加了  $H \times 2$  的参数量，可谓比分类任务的参数增加还要少（至少和 2 分类任务的参数一致）。

这里有两个需要注意的地方：其一，无论对于起始符和终止符也好，它们的判断和普通分类任务不一样，不是直接针对每个 token 进行是否是起始符的分类（如果是这样的话，参数量要增加  $H \times 4$ ），而是类似于在全体输入序列上进行排序，因此针对起始符的判断来讲只需要一个  $H \times 1$  的全连接就可以了。

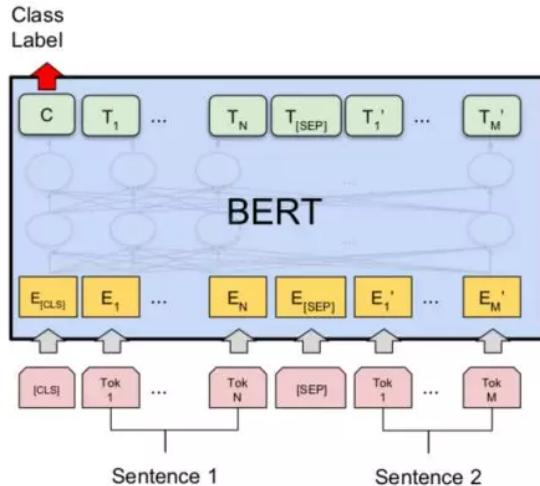
第二个需要注意的是，训练时无须担心终止符是否在起始符后面，因为即便模型训练时预测是终止符在起始符前面，这一错误现象会通过损失函数反馈给模型，然而在做 inference 时，就必须保证终止符一定要在起始符后面。

BERT 也考虑了如何在序列标注任务上进行 finetune，对于序列中的每个 token 而言，实际上就是一个分类任务。和前面提到的普通分类任务不一样的是，这里的分类需要针对序列中的每个词做分类，参数增加在  $H \times K$ ，这里的  $K$  是序列标注中标注的种类个数。

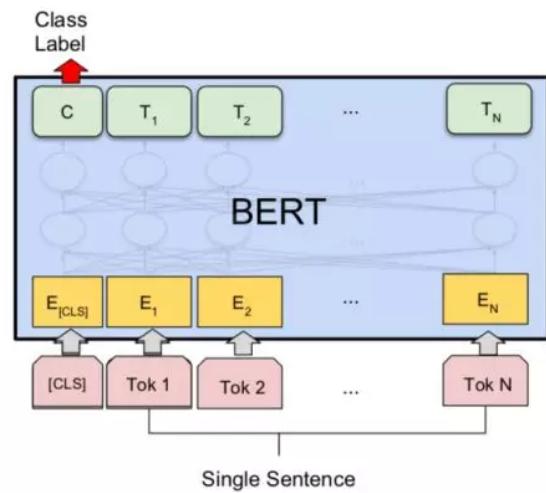
最后对于 SWAG 任务来讲，因为需要在给定一个句子后，从四个候选句子中选择一个最有可能是该句子的下一个句子，这里面通常包含了一些常识推理。

BERT 的做法是将前置句子和四个候选句子分别进行组合成一个句子对，并按照之前讨论过的方法输入到 Encoder，这样便得到四个 pair 的句子表征，然后只需要维护一个向量（或者说是一个多到 1 的全连接层），便可以给每一个候选句子进行打分，从而得到四个候选句子中最有可能是下一个的句子。

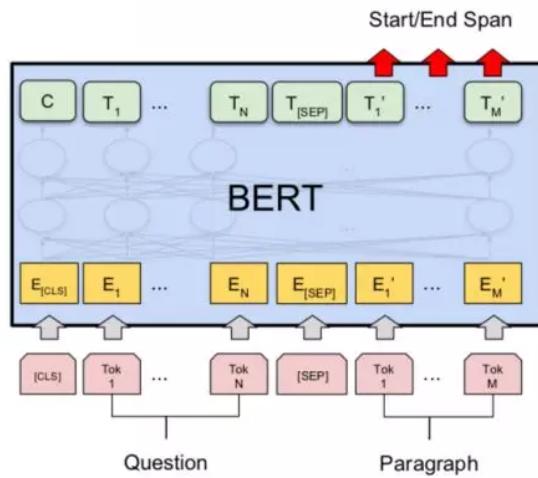
这个全连接只需要增加  $H \times 1$  的参数量，这一点和 SQuAD 1.1 任务的情况类似。此外，这种做法和 GPT 对于 Question Selection 的做法也比较类似，理论上应该也可以应用到 WikiQA 这种数据集上的，finetune 阶段同样只需要增加  $H \times 1$  的参数量。



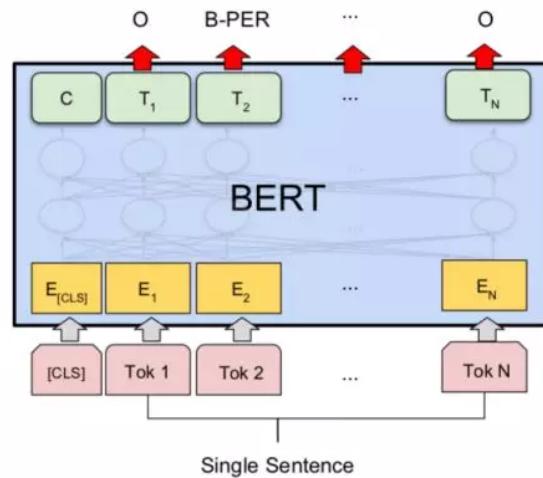
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



(c) Question Answering Tasks:  
SQuAD v1.1



(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

最后，我们再次总结 BERT 的几个主要特点：

- Transformer Encoder 因为有 Self-attention 机制，BERT 自带双向功能；
- 因为双向功能以及多层 Self-attention 机制的影响，使得 BERT 必须使用 Cloze 版的语言模型 Masked-LM 来完成 token 级别的预训练；
- 为了获取比词更高级别的句子级别语义表征，BERT 加入了 Next Sentence Prediction 来和 Masked-LM 一起做联合训练；
- 为了适配多任务下的迁移学习，BERT 设计了更通用的输入层和输出层。

然后，我们再来看看 BERT 都站在了哪些“巨人肩膀”上：

- 针对第一点，双向功能是 Transformer Encoder 自带的，因此这个“巨人肩膀”是 Transformer；
- 第二点中 Masked-LM 的巨人肩膀是语言模型、CBOW 以及 Cloze 问题；
- 第三点中 Next Sentence Prediction 的“巨人肩膀”是 Skip-gram、Skip-thoughts 和 Quick-thoughts 等工作；
- 第四点中，对输入层和输出层的改造，借鉴了 T-DMCA 以及 GPT 的做法。

再来看看预训练使用的数据方面，ELMo 使用的是 1B Word Benchmark 数据集，词数为 10 亿，然而句子间的顺序是随机打乱的，无法建模句子级别的依赖关系，所以 GPT 和 BERT 都弃用这个数据集；GPT 使用的是 BooksCorpus，词数为 8 亿；BERT 除了使用 BooksCorpus 之外，还加入了 25 亿级别的 Wikipedia 数据集，因此整体使用的数据集为 33 亿个词。可见 BERT 使用的数据集是最大的。

模型方面，ELMo 使用的是一个 Bi-LSTM 结构，输入层和输出层使用 CNN 来进行建模。而 GPT 使用了 12 个 block，隐藏层节点数为 768，多头为 12，这也是 BERT 的 base 版本（base 版本就是为了跟 GPT 做对比），参数量按照 BERT base 版本估计为 110M。而 BERT 再次把模型变得更大了，使用 24 个 block，并且隐藏层节点数与多头都相应扩大，作者似乎想要表明他们设计的模型是有史以来最大的模型之一。

在计算资源比拼方面，GPT 在 8 块 GPU 上预训练了一个月。参数规模和 GPT 差不多的 BERT base 版本，在 16 块 TPU 上只花了 4 天，BERT 在论文中提到此处时，还不失时机给它们家的 TPU 放了个链接，打了一波广告。对于参数规模为 GPT 三倍之多的 BERT large 版本，在 64 块 TPU 上训练耗时也依然达到 4 天之久。

	数据集	模型	参数量	训练时间	训练配置
<b>ELMo</b>	1B Word Benchmark 10亿词	CNN-BiLSTM + Residual Connection	90M	-	-
<b>GPT</b>	BooksCorpus 8亿词	Transformer Decoder, L=12, H=768, A=12	110M	1个月	8 GPU
<b>BERT</b>	BooksCorpus+Wikipedia 33亿词	Transformer Encoder, L=24, H=1024, A=16	340M	4天	64 TPU

知乎 @weizier

因此，总结来看，BERT 的出现既踩在了一众前辈们的“巨人肩膀”上，通过精心设计和一些训练技巧造出了一个庞大模型，最后又舍得花钱砸下巨大资源，给众人上演了一场炫目无比的烟花秀。与其说 BERT 带来了一次翻天覆地的革新，不如说 BERT 团队糅合能力超群，以及还有一点也非常重要的：有钱。

然而，和普通的烟花秀不一样，普通烟花易逝，而 BERT 说不定可真是将一朵烟花怒放在每一位 NLP 算法人员心中，它先是让我们看了一场横扫 11 项任务的表演秀（也是实力秀），宣告世人这场烟花不一般。

更为可贵的是，当众人惊魂未定之时，很快，BERT 又把他们排练烟花秀的一切设备（模型）全都公之于众，甚至把排练得到的效果最好的布景（预训练参数）和点火器（finetune 超参数）也全都奉献了出来，似乎愿景也足够伟大：让每个人都可以上演一场自己的烟花秀，将预训练好的模型在复杂任务上简单进行一次 finetune，便立马能够得到一个非常高的 baseline。

沉寂了五年之久的 NLP 界，似乎迎来了一个新的时代。

## 如何爬上梯子的第二级

预训练语言模型的优势在于：

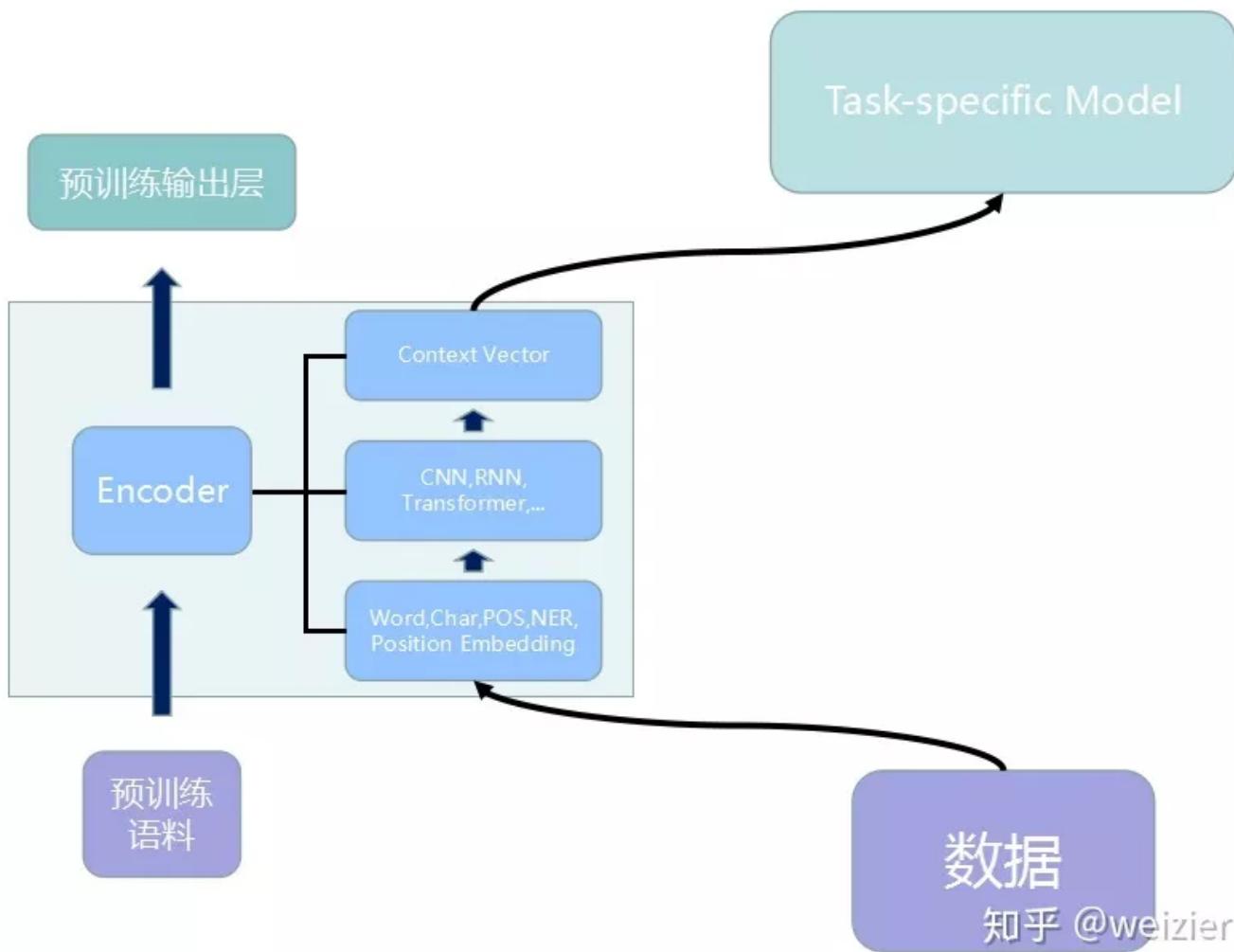
- 近乎无限量的优质数据

- 无需人工标注
- 一次学习多次复用
- 学习到的表征可在多个任务中进行快速迁移

问题是如何利用这些预训练好的模型，Google 们已经给我们提供巨人的肩膀了，那“梯子”在哪呢？我们怎么爬上去这些“巨人肩膀”？也就是如何使用这些预训练好的模型。一般来说，可以有三种方式来使用。它们分别是：

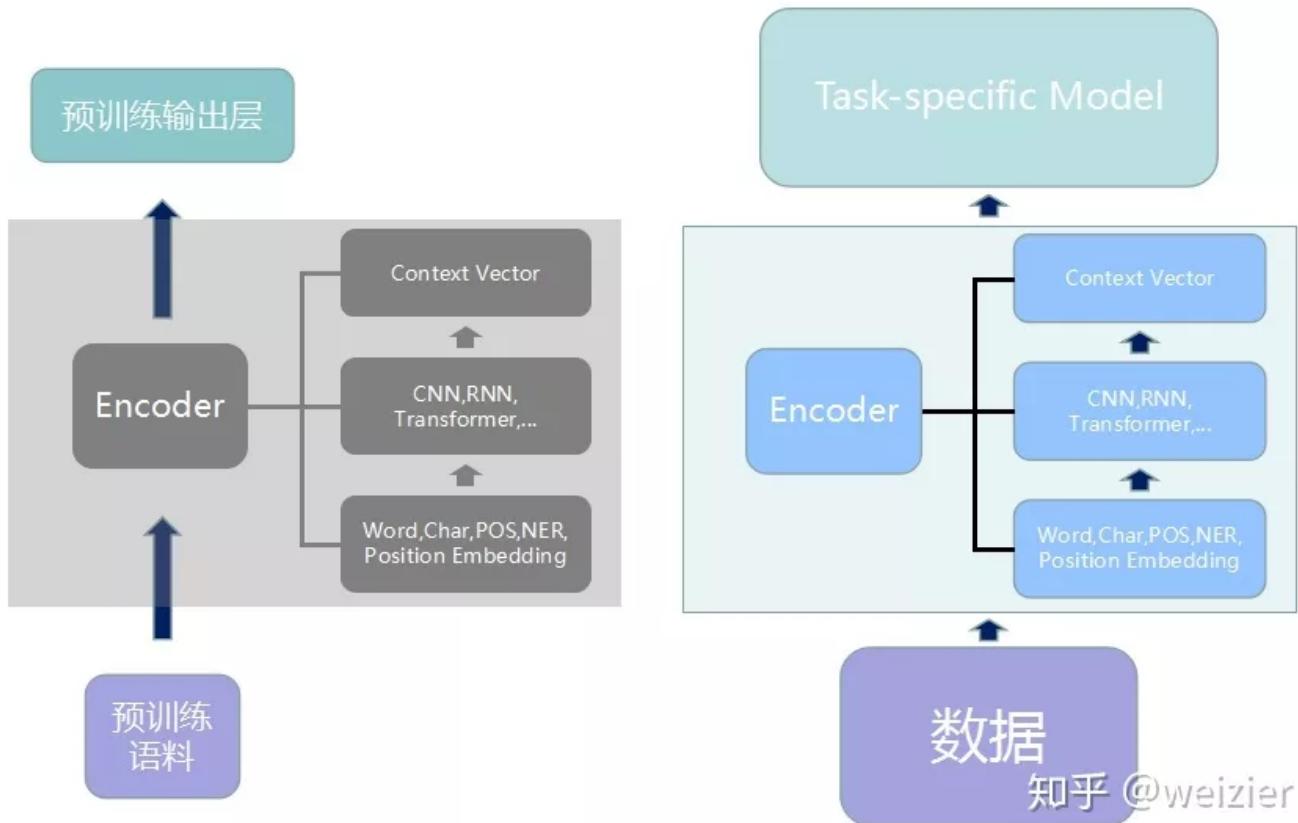
1. 将预训练模型当做一个特征提取器，直接将预训练模型的输出层去掉，然后使用去掉输出层之后的最后一层输出作为特征，输入到我们自己精心设计好的 Task-specific 模型中去。在训练过程中，作为特征提取器的部分（比如 BERT Encoder）的参数是不变的。

另外，特征提取器并不一定只能用最后一层的输出向量，也可以使用中间的某些层，甚至也可以借鉴 ELMo 的做法，在各层之间利用一个 softmax 来学习各自的权值，或者也可以尝试一些更为复杂的各层组合方式。



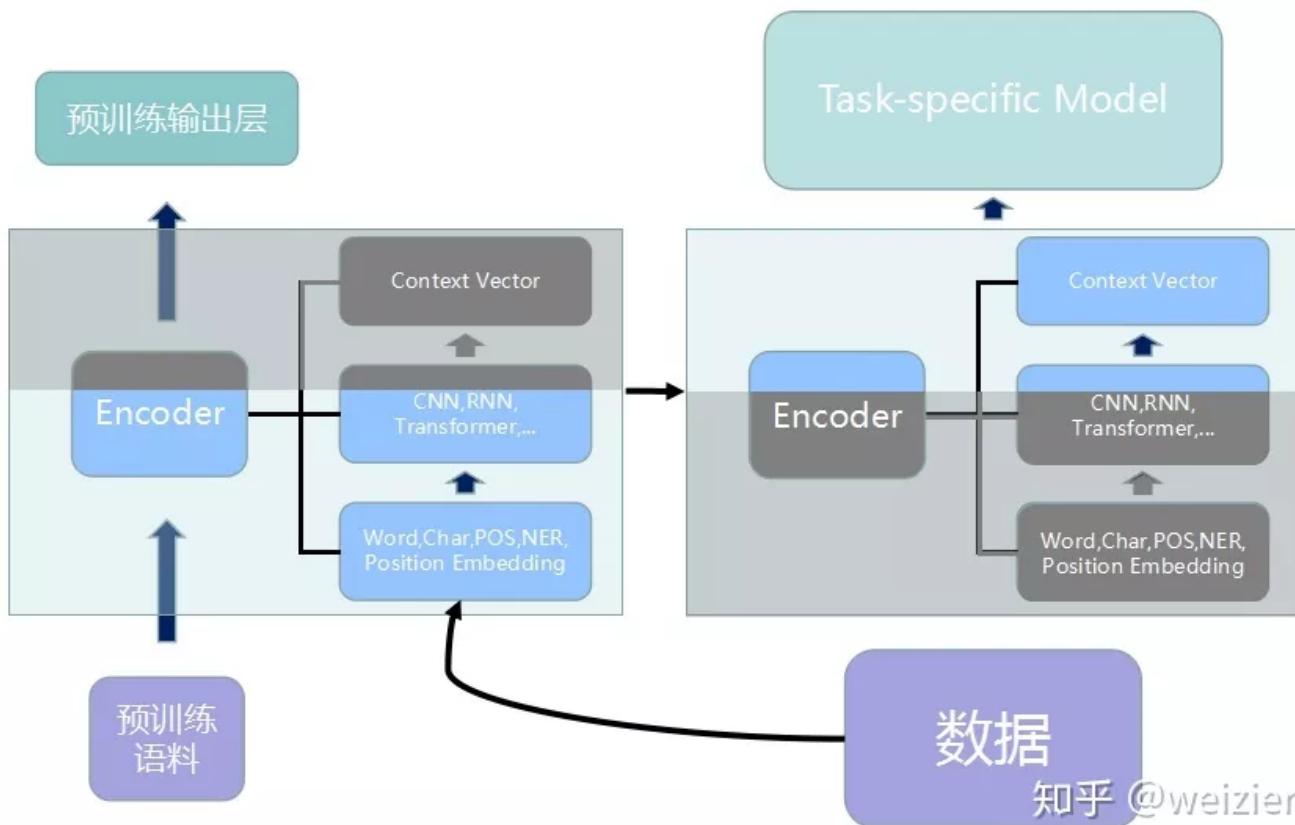
2. 将预训练模型整体接入 Task-specific 模型，继而重新在新的数据集上整体重新训练。当然训练技巧可以有很多种，比如 ULMFiT 的三角学习率和逐层解冻或者是 Transformer 的 warmup 策略（上文都有提到），这些训练技巧非常重要，需要好好把控，否则很容易学崩了，甚至让原有预训练语言模型的优势都被新的 finetune 抹去了，因此需要实验设计一个比较好的 finetune 策略。

此外，和特征提取器的接入方式类似，预训练模型的接入不一定只能接最后层的输出，可以尝试更复杂的接入方式，比如 DenseNet；



3. 和上面两种极端情况相反，或者说综合了上面两种方式的方案，即保留预训练模型的一部分，另外一部分则和 Task-specific 模型一起 finetune。

在某些情况下，这可能是个比较合理的选择，比如预训练模型比较深（NLP 模型通常来讲比 CV 模型都要浅很多），以及训练数据不算太多的情况，这个时候一方面要保证预训练模型在大规模语料上曾经学习到的表征，另一方面因为又要做新数据下的迁移，但是数据量比较少，重新 finetune 整个模型可能不太合适，容易导致模型的鲁棒性不高，那么似乎选择最后的一些层进行选择性的 finetune 会是比较好的方案。



上面这三种方案，既包括了 BERT 所言的 feature-based 使用方法，也包括了 BERT 的 finetune 方法。另外 GPT 和 BERT 也给我们提供了很好的输入层与输出层通用包，再加上一些训练技巧，一般的 NLP 任务应该是足够应付了。

然而，GPT 和 BERT 即便刷新了十多项NLP任务，但似乎各自默契的都不曾尝试过在生成问题中大展身手。比如二者的主要征战沙场 GLUE 连名字中都直指这个“胶水”排行榜主要就是语言理解问题。

除了 GLUE 外，GPT 额外刷过的数据集有 4 个：SNLI, SciTail, RACE 和 Stroy Cloze，其中 SNLI 和 SciTail 都是自然语言推断问题，本质上是一个句子级别的分类问题，RACE 和 Stroy Cloze 是 QA 或阅读理解问题，本质上可以认为分别是一个 token 级别的分类以及句子级别的打分排序问题。

而 BERT 额外刷过的三个数据集中 SQuAD 本质上是一个 token 级别的分类问题，NER 也是如此，SWAG 则是从候选句子中选择正确的推断，是一个句子级别的打分排序问题。

GPT 刷了 12 项数据集（打破其中 9 项纪录），BERT 刷了 11 项数据集（打破其中 11 项），然而无一例外这些数据集全是自然语言理解领域的问题，而对于 NLP 中另一个庞大领域自然语言生成，不约而同地选择了放弃，是未曾注意，还是选择视而不见，背后的原因我无从猜测。

不过，无论 GPT 和 BERT 是什么原因没有选择在生成问题中去尝试，私以为，想要把 GPT 和 BERT 的预训练模型迁移到生成问题中，大约应该也不是一件非常困难的事。

首先 GPT 本质上就是一个生成模型，而 BERT 虽使用的是 Encoder，但把它改造成一个 Decoder 应该不是什么特别困难的事，因为本质上 Encoder 和 Decoder 基本框架相同，只不过 Decoder 需要做好两件事情：

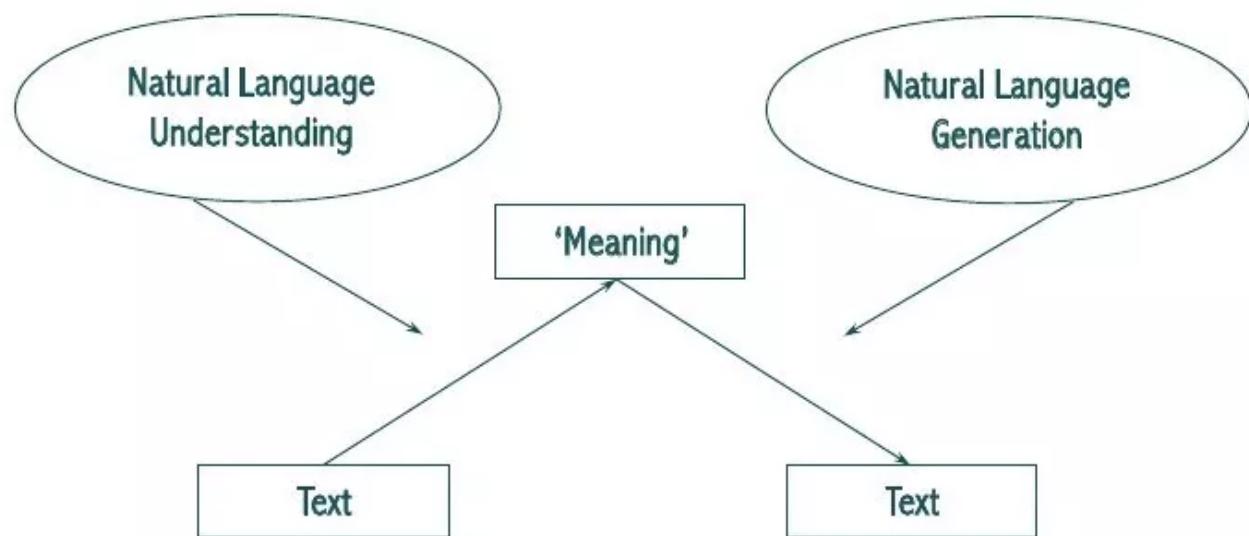
第一件事情是 Decoder 中也有 Self-attention 层，和 Encoder 中的 Self-attention 不同的是，Decoder 需要做好 mask，以避免 attention 到未来待预测的词上去。第二件事情是 Decoder 中有一个交叉 attention，用于获取来自于 Encoder 侧的信息。

但是，这里还涉及到 Encoder-Decoder 整体这种 Seq2Seq 框架是否要保留的问题。如果使用 GPT 用的 T-DMCA 这种仅用 Decoder 做生成问题的，那么改造相对会更易行，如果要保留 Encoder-Decoder 这种双结构，那么改造可能会相对复杂一些。

乐观一点，如果这种改造非常成功（似乎已经具备有充分的理由来相信这一点），那么 NLP 似乎会迎来一个阶段性的大一统时代：也就是一个基本模型做到一个闭环。其流程是：从文本到语义表征，再从语义表征重新生成文本。这样看来，至少在 NLP 领域（CV 等其他 AI 领域，留给各位看官自行脑洞好了），未来大约还是很值得期待的一件事情。

## NLP = NLU + NLG

---



那么以谷歌为代表的这些工作，会不会像五年以前一样，也即将开启一个新的时代呢？能不能让 AI 成功爬上第二级梯子呢？让我们拭目以待。

## 后现代启示录？

回过头来看此文，通篇大体都是反观 AI 儿子理解他人说的话写的书，无非都是些人类母亲的育儿笔记，徒当以史为鉴可以知兴替罢了。

实事求是的说，人类母亲的教学成果目前看来，至少还不算太赖，寄养在各大公司（以 Google, Facebook 和 OpenAI 等为代表）的 AI 儿子们也开始理解母亲的一些稍微简单些的指令，甚至也能咿咿呀呀的开口说话了，不能不说这十多年的 AI 成长大家有目共睹。

但成果毕竟属于过去，望子成龙的人类母亲对于儿子的今后该如何安排？毕竟儿子以后还要上清华北大，毕竟七大姑八大姨的饭桌上，人类母亲还指望着儿子来长脸的不是？

不过非要问人类母亲的育儿计划，首先那些有权有势的托儿所所长们（以 Google, Facebook 和 OpenAI 等为代表）如果真雄心万丈也一定不愿透露半点风声，其次可能人类母亲自己也弄的不太清楚，无非是走一步瞧一步罢了。然而没什么能够阻挡人类对于未来的好奇心，追本溯源按图索骥之下，说不定也能窥的一些蛛丝马迹。

因而，不妨抬头看看星空，也看看未来，以期 AI 的今后也能刷刷朋友圈，感叹一下年华易逝，甚而至于能体会得到人类母亲养儿育女的艰难，哪怕只有丝毫，大概人类母亲也就不枉费了一片可能和人类历史一样悠久的望子成龙之心。

闲话少叙，让我们开开脑洞，大言不惭的看看今后 NLP 的一些可能性。只不过我人微言轻，暂且幻想一下皇帝的金锄头吧。

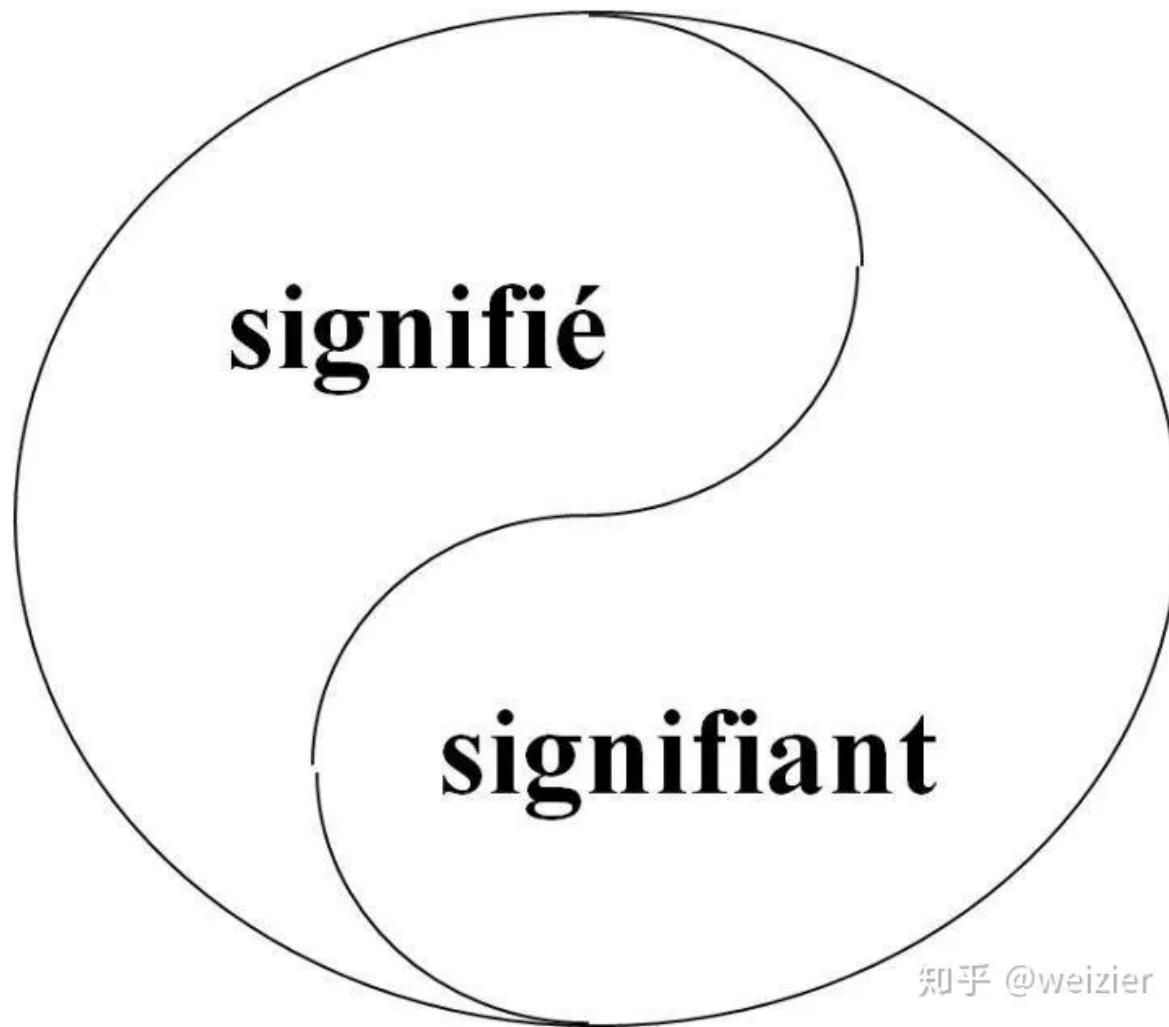
一般来说，语言学流派大约可以分为五个流派：历史比较语言学、结构主义语言学、转换－生成语言学、系统功能语言学和认知语言学这五个大的学派，其中历史比较语言学年代比较久远了，注重搜集多种语言历史资料，通过对比归纳总结等等方法，进而理解语言的演化和发展历程，历史比较语言学的主要贡献在于它把语言学从其他学科里抽离出来成为一门独立的学科。

而结构主义语言学（大约兴起于20世纪20年代左右）的开山鼻祖索绪尔（同时他也是现代语言学之父），对于语言的理解，他有一个非常经典的类比，他认为语言就像是一盘象棋，棋盘上的每一

个棋子都有特定的游戏规则，棋子只不过是这些特定游戏规则的一个实物标记而已，如果某天弄丢了某一个棋子，我们立马可以拿任何一个小物件来代替这个棋子的功能，只要游戏规则不变，那么棋局依然能够照常进行，棋子替换前后基本不会有任何差别。

索绪尔认为语言里的基本单位诸如词语习语等等都只不过是棋盘上的棋子（**signifiant**, 能指），重要的不是棋子本身，而是棋子背后对应的一整套规则，也就是语言里的各种结构或约定俗称的指代或用法（**signifié**, 所指）。象棋的整个规则便是我们所使用的一整套语言系统，比如象棋的规则便是我们的汉语，而国际象棋的规则便是英语，两种象棋各自使用的棋子各有不同，规则自然也有所区别。

索绪尔之后，结构主义语言学又进一步发展成为三个独立子学派，但它们的共同点都是和索绪尔一脉相承的，也就是都认为语言本质上是一个符号系统，符号只是表面，更重要的是挖掘和研究符号系统内在的规则和联系。结构主义语言学兴起后，标志着现代语言学的诞生，并且在一段时期内不仅显著的影响了后来的语言学流派，还影响了其他的一些人文学科。



到了 1957 年，乔姆斯基出版《句法结构》一书，标志着转换生成语言学的诞生。乔姆斯基的主要观点则认为人类天生有一套普遍语法，我们日常的语言系统只不过是这套普遍语法的形式化体现而

已。

因此他认为语言中存在着深层结构和表层结构，深层结构与意义和概念相关，而表层结构则与语音和符号相关，这两层结构由一套转换语法连接起来，也就是由深层结构生成表层结构。他为此还特别举了一些例子，为了说明两个句子表层结构类似乃至完全一样，但是深层语义完全不相关的现象，比如：*Flying planes can be dangerous.*

这个句子可以因为同一份表层结构对应到两套不同的深层结构，而表达出完全不一样的意义，第一种是 *flying* 作为形容词，那么此时句子想表达的意义是在天上正在飞行的飞机很危险；而第二种是把 "*flying planes*" 解构成一种动宾结构，那么这个句子表达的意思就成了开飞机这个动作很危险。

乔姆斯基举这个例子，就是为了说明传统的结构主义语言学对于这种句子根本无能为力，因为它们的表层结构是完全一致的，从而得出语言有一种更为本质的深层结构的结论。

但是个人理解，与其说是语言学的深层结构，还不如说是人类认知体系或者客观世界的结构化表示，因为深层结构必然对应到人类的认知体系。并且乔姆斯基过于强调了普遍语法的天生性，而忽略了后天语言的塑造和习得。

后来乔姆斯基的众多弟子（乔治莱考夫等人）也公开宣判与自己老师乔姆斯基的语言学观点决裂，并各自独立发展出了一些新的语言学理论，比如乔治莱考夫的认知语言学流派。



# *Flying planes can be dangerous.*

- **a) the action of flying planes can be dangerous**
- **Planes can be dangerous**
- **X (people) fly planes**
- **b) the planes that fly can be dangerous.**
- **Planes can be dangerous**
- **Planes fly**

我们可以发现从历史比较语言学，到结构主义语言学，再到转换生成语言学，这三个流派基本可以认为是一脉相承的，只不过所想要揭示的东西越来越深层，从历史比较语言学着重于研究语言学的一些表现形式和演化形态，到结构主义语言学着重于研究符号系统背后的结构，再到转换生成语言学认为结构主义语言学揭示的只是语言的表层结构，因而自己提出了深层结构的概念。

而到了 20 世纪八九十年代，以韩礼德为代表的一些语言学家，逐渐注意到无论是结构主义还是转换生成语言学，它们都片面的强调语言学的理论形式，而忽视了语言的社会的功能属性，韩礼德认为这语言的结构和语法等等理论形式，以及社会功能属性是语言的两个方面，这二者都很重要，他想要把二者统一起来。

因此发展了他自己的语言学观点，他提出了系统语法和功能语法的概念，前者强调语言的内在结构和形式理论，这点和索绪尔的结构主义一脉相承，但他同时还提出了功能语法，着重语言的社会属性，说明语言是社会交流和沟通的媒介，不过由于在系统语法上的影响力远不如乔姆斯基等流派的影响力，功能语法的建树倒是别的流派所缺乏的，因此把韩礼德的学说就叫做系统功能语言学了。

而上文提到的乔姆斯基的弟子之一乔治莱考夫，认为他的老师对于人类天生具有普遍语法的观点是大错特错，尤其是转换生成语法中认为深层结构由一些有限的普遍语法组成，着重语言规则的形式

化。

乔治莱考夫严厉抨击了他老师的观点，与转换生成语法着重形式和规则的探讨相反，认知语言学认为语言不是人类的一个独立系统，人类的一切语言行为都只不过是人类对于客观世界和精神世界亲身体验的外化，也就是说，语言无论把它叫做符号也好，叫做语法结构也好，叫做系统也好，都逃离不了语言是人类认知系统的一部分，语言并不独立存在。

人类在长期的自然演化中，形成了一套自己独有的认知系统，而通过这套认知系统，人类可以获得对于身边客观环境以及自己内部精神世界的一套独特体验，这套体验如果映射到外在的语音或文上，便形成了我们人类的语言系统。因此人类的语言与人类其他认知能力没有本质上的区别。

不过，认知语言学并没有一味地架空自己的语言学观念，它也适时地道出了语法规则和人类的认知系统有一定的对应性，言下之意便是说语法规则是人类认知系统抽象化的外在体现。

可以看出，在这一点上，认知语言学比转换生成语言学又更进了一步，直接从研究人类的认知角度来研究人类的语言学。乔治莱考夫在语言学中引入了隐喻的概念，认为隐喻是人类最重要的认知活动之一，也是语言中最普遍的现象。



然而回到当下的自然语言处理，似乎现在谈认知，有点为时尚早。在自然语言处理的发展历程中，除了乔姆斯基的转换生成语言学曾经在上世纪六七十年代大放异彩，一度成为当时自然语言处理的

主流方法，除此之外，似乎理论语言学界一直和自然语言处理脱钩比较严重，这一方面揭示了人类语言系统的复杂，另一方面，也预示着现在的自然语言处理缺乏严密的理论支持。

说白了，现在的 NLP 界，尤其以 BERT 为代表的做法，实际上是简单粗暴的代名词，它们把人类语言习得过程中的轻量、泛化和低功耗，用大数据、大模型和大计算量炮轰的荡然无存。

站在眼力所及之处的满目疮痍上，我们依然无法让机器真正明白“喵星人”这个词对于人类而言，其背后究竟意味着一个怎样的复杂情绪和体验（更有趣的是，即便对于每个人类个体而言，其背后的体验也是千差万别的，可见语言笼阔了符号性、认知性、社会性和个体性之后的复杂程度）。

然而在身后留下一片废墟和尸骨的 BERT，正准备重建一个伟大新帝国的 BERT，它究竟是要选择开心呢还是难过？对于人类母亲而言，这大概会是个问题。

不过无论如何，帝国的建立总是艰难的。让人类母亲足可欣慰的是，即便步履维艰，托儿所里的 AI 儿子们依然一步一步坚定的踩上了简陋的梯子。

站在巨人的肩膀上回首，我们发现，和人类母亲的语言习得方式依靠某一个机缘巧合的基因突变不同，AI 踏出了一条只属于它自己的路，未来的巨人肩膀和梯子在哪里，我们是否需要选用新的巨人肩膀和梯子，这都是成长的必经之痛，一切都留给未来去证实或证伪吧。

• 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](mailto: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