

# Transformer一统江湖：自然语言处理三大特征抽取器比较



新智元  
01-14 14:20



## 新智元推荐

作者：知乎专栏：深度学习前沿笔记

作者：张俊林

**【新智元导读】**自然语言处理中的三大特征处理器：RNN、CNN、Transformer，它们目前谁各方面占据优势？未来谁又更有前途呢？这篇文章用目前的各种实验数据给出了说明，结论是：放弃幻想，全面拥抱Transformer。

在辞旧迎新的时刻，大家都在忙着回顾过去一年的成绩（或者在灶台前含泪数锅），并对2019做着规划，当然也有不少朋友执行力和工作效率比较高，直接把2018年初制定的计划拷贝一下，就能在3秒钟内完成2019年计划的制定，在此表示祝贺。2018年从经济角度讲，对于所有人可能都是比较难过的一年，而**对于自然语言处理领域来说，2018年无疑是个收获颇丰的年头**，而诸多技术进展如果只能选择一项来讲的话，那么当之无愧的应该就是**Bert模型**了。

在上一篇介绍Bert的文章“从Word Embedding到Bert模型—自然语言处理中的预训练技术发展史”[1]里，我曾大言不惭地宣称如下两个个人判断：一个是**Bert这种两阶段的模式（预训练 + Finetuning）必将成为NLP领域研究和工业应用的流行方法**；第二个是从**NLP领域的特征抽取器角度来说，Transformer会逐步取代RNN成为最主流的特征抽取器**。关于特征抽取器方面的判断，上面文章限于篇幅，只是给了一个结论，并未给出具备诱惑力的说明，看过我文章的人都知道我不是一个随便下结论的人（那位正在补充下一句：“你随便起来不是……”的同学请住口，请不要泄露国家机密，你可以继续睡觉，吵到其它同学也没有关系，哈哈），但是为什么当时我会下这个结论呢？本文可以看做是上文的一个外传，会给出比较详实的证据来支撑之前给出的结论。



新智元

最近更新：01-14 14:20

简介：人工智能垂直媒体。

## 作者最新文章

定了！教育部公布35所新增AI本科高校名单

无人车驶出寒冬？

快速上手深度强化学习？学会TensorForce就够了

## 相关文章



新手司机会有哪些特征？

问答 04-03



1岁以内宝宝发育动作的标准，快来看看你家...

妮喃 01-14



北京市和天津市两所语言类大学

走过路过自然飘过 01-12



上幼儿园的孩子如果有这4种特征，说明以后...

妮喃 01-10



我们也在给孩子施“语言暴力”

建筑与生活 01-14

- **RNN** 人老珠黄，已经基本完成它的历史使命，将来会逐步退出历史舞台；
- **CNN** 如果改造得当，将来还是有希望有自己在 NLP 领域的一席之地，如果改造成功程度超出期望，那么还有一丝可能作为割据一方的军阀，继续生存壮大，当然我认为这个希望不大，可能跟宋小宝打篮球把姚明打哭的概率相当；
- 而新欢 **Transformer** 明显会很快成为 NLP 里担当大任的最主流的特征抽取器。

至于将来是否会出现新的特征抽取器，一枪将 Transformer 挑落马下，继而取而代之成为新的特征抽取山大王？这种担忧其实是挺有必要的，毕竟李商隐在一千年前就告诫过我们说：“君恩如水向东流，得宠忧移失宠愁。莫向樽前奏花落，凉风只在殿西头。”当然这首诗看样子目前送给 RNN 是比较贴切的，至于**未来 Transformer 是否会失宠**？这个问题的答案基本可以是肯定的，无非这个时刻的来临是 3 年之后，还是 1 年之后出现而已。当然，我希望如果是在读这篇文章的你，或者是我，在未来的某一天，从街头拉来一位长相普通的淑女，送到韩国整容，一不小心偏离流水线整容工业的美女模板，整出一位天香国色的绝色，来把 Transformer 打入冷宫，那是最好不过。但是在目前的状态下，即使是打着望远镜，貌似还没有看到有这种资质的候选人出现在我们的视野之内。

我知道如果是一位严谨的研发人员，不应该在目前局势还没那么明朗的时候做出如上看似有些武断的明确结论，所以这种说法可能会引起争议。但是这确实就是我目前的真实想法，至于根据什么得出的上述判断？这种判断是否有依据？依据是否充分？相信你在看完这篇文章可以有个属于自己的结论。

可能谈到这里，有些平常吃亏吃的少所以喜欢挑刺的同学会质疑说：你凭什么说 NLP 的典型特征抽取器就这三种呢？你置其它知名的特征抽取器比如 Recursive NN 于何地？嗯，是，很多介绍 NLP 重要进展的文章里甚至把 Recursive NN 当做一项 NLP 里的重大进展，除了它，还有其它的比如 Memory Network 也享受这种部局级尊贵待遇。但是我一直都不太看好这两个技术，而且不看好很多年了，目前情形更坚定了这个看法。而且我免费奉劝你一句，没必要在这两个技术上浪费时间，至于为什么，因为跟本文主题无关，以后有机会再详细说。

上面是结论，下面，我们正式进入举证阶段。

战场侦查：NLP 任务的特点及任务类型





NLP 任务的特点和图像有极大的不同，上图展示了一个例子，NLP 的输入往往是一句话或者一篇文章，所以它有几个特点：首先，**输入是个一维线性序列**，这个好理解；其次，**输入是不定长的**，有的长有的短，而这点其实对于模型处理起来也会增加一些小麻烦；再次，**单词或者子句的相对位置关系很重要**，两个单词位置互换可能导致完全不同的意思。如果你听到我对你说：“你欠我那一千万不用还了”和“我欠你那一千万不用还了”，你听到后分别是什么心情？两者区别了解一下；另外，句子中的长距离特征对于理解语义也非常关键，例子参考上图标红的单词，特征抽取器能否具备长距离特征捕获能力这一点对于解决 NLP 任务来说也是很关键的。

上面这几个特点请记清，**一个特征抽取器是否适配问题领域的特点，有时候决定了它的成败，而很多模型改进的方向，其实就是改造得使它更匹配领域问题的特性**。这也是为何我在介绍 RNN、CNN、Transformer 等特征抽取器之前，先说明这些内容的原因。

NLP 是个很宽泛的领域，包含了几十个子领域，理论上只要跟语言处理相关，都可以纳入这个范围。但是如果我们对大量 NLP 任务进行抽象的话，会发现绝大多数 NLP 任务可以归结为几大类任务。两个看似差异很大的任务，在解决任务的模型角度，可能完全是一样的。

## NLP的四大类任务

- **序列标注**：分词/POS Tag/NER/语义标注...
- **分类任务**：文本分类 / 情感计算...
- **句子关系判断**：Entailment/QA/自然语言推理...
- **生成式任务**：机器翻译 / 文本摘要...

通常而言，绝大部分 NLP 问题可以归入上图所示的**四类任务**中：

- 一类是**序列标注**，这是最典型的 NLP 任务，比如中文分词，词性标注，命名实体识别，语义角色标注等都可以归入这一类问

- 第二类是**分类任务**，比如我们常见的文本分类，情感计算等都可以归入这一类。它的特点是不管文章有多长，总体给出一个分类类别即可。
- 第三类任务是**句子关系判断**，比如 Entailment，QA，语义改写，自然语言推理等任务都是这个模式，它的特点是给定两个句子，模型判断出两个句子是否具备某种语义关系；
- 第四类是**生成式任务**，比如机器翻译，文本摘要，写诗造句，看图说话等都属于这一类。它的特点是输入文本内容后，需要自主生成另外一段文字。

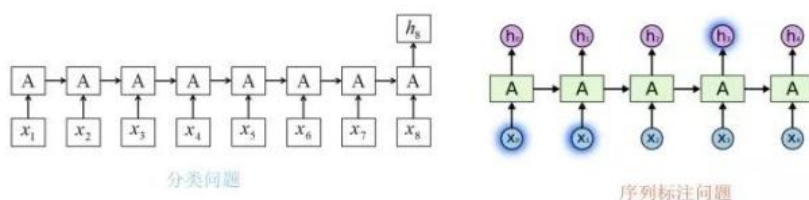
解决这些不同的任务，从模型角度来讲什么最重要？是**特征抽取器的能力**。尤其是深度学习流行开来后，这一点更凸显出来。因为深度学习最大的优点是“端到端（end to end）”，当然这里不是指的从客户端到云端，意思是以前研发人员得考虑设计抽取哪些特征，而端到端时代后，这些你完全不用管，把原始输入扔给好的特征抽取器，它自己会把有用的特征抽取出来。

身为资深 Bug 制造者和算法工程师，你现在需要做的事情就是：选择一个好的特征抽取器，选择一个好的特征抽取器，选择一个好的特征抽取器，喂给它大量的训练数据，设定好优化目标（loss function），告诉你你想让它干嘛……然后你觉得你啥也不用干等结果就行了是吧？那你是见过的整个宇宙中最乐观的人……你大量时间其实是用在调参上……。从这个过程可以看出，如果我们有个强大的特征抽取器，那么中初级算法工程师沦为调参侠也就是个必然了，在 AutoML（自动那啥）流行的年代，也许以后你想当调参侠而不得，李斯说的“吾欲与若复牵黄犬，俱出上蔡东门逐狡兔，岂可得乎！”请了解一下。所以请珍惜你半夜两点还在调整超参的日子吧，因为对于你来说有一个好消息一个坏消息，好消息是：对于你来说可能这样**辛苦**的日子不多了！坏消息是：对于你来说可能这样辛苦的日子**不多了**！！！那么怎么才能成为算法高手？你去设计一个更强大的特征抽取器呀。

下面开始分叙三大特征抽取器。

沙场老将 RNN：廉颇老矣，尚能饭否

## RNN



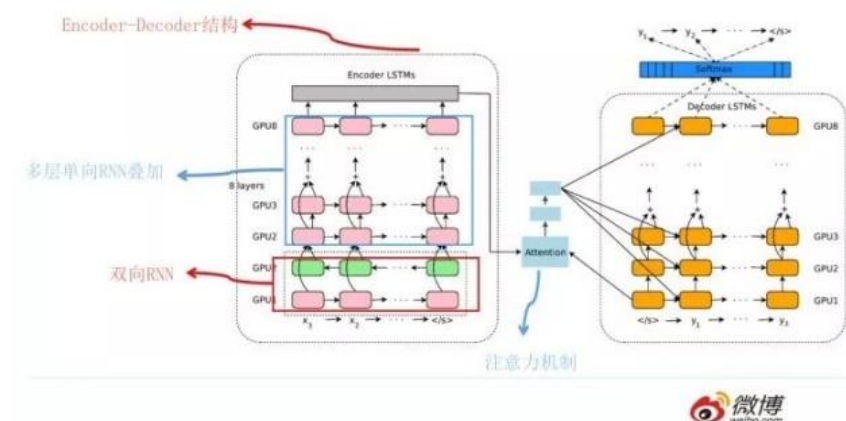
RNN 模型我估计大家都熟悉，就不详细介绍了，模型结构参考上图，核心是每个输入对应隐层节点，而隐层节点之间形成了线性序列，信息由前向后在



## 为何 RNN 能够成为解决 NLP 问题的主流特征抽取器

我们知道，RNN 自从引入 NLP 界后，很快就成为吸引眼球的明星模型，在 NLP 各种任务中被广泛使用。但是原始的 RNN 也存在问题，它采取线性序列结构不断从前往后收集输入信息，但这种线性序列结构在反向传播的时候存在优化困难问题，因为反向传播路径太长，容易导致严重的梯度消失或梯度爆炸问题。为了解决这个问题，后来引入了 LSTM 和 GRU 模型，通过增加中间状态信息直接向后传播，以此缓解梯度消失问题，获得了很好的效果，于是很快 LSTM 和 GRU 成为 RNN 的标准模型。其实图像领域最早由 HighwayNet/Resnet 等导致模型革命的 skip connection 的原始思路就是从 LSTM 的隐层传递机制借鉴来的。经过不断优化，后来 NLP 又从图像领域借鉴并引入了 attention 机制（从这两个过程可以看到不同领域的相互技术借鉴与促进作用），叠加网络把层深作深，以及引入 Encoder-Decoder 框架，这些技术进展极大拓展了 RNN 的能力以及应用效果。下图展示的就是非常典型的使用 RNN 来解决 NLP 任务的通用框架技术大礼包，在更新的技术出现前，你可以在 NLP 各种领域见到这个技术大礼包的身影。

典型的RNN应用技术大礼包



上述内容简单介绍了 RNN 在 NLP 领域的大致技术演进过程。那么为什么 RNN 能够这么快在 NLP 流行并且占据了主导地位呢？主要原因还是因为 RNN 的结构天然适配解决 NLP 的问题，NLP 的输入往往是个不定长的线性序列句子，而 RNN 本身结构就是个可以接纳不定长输入的由前向后进行信息线性传导的网络结构，而在 LSTM 引入三个门后，对于捕获长距离特征也是非常有效的。所以 RNN 特别适合 NLP 这种线形序列应用场景，这是 RNN 为何在 NLP 界如此流行的根本原因。

## RNN 在新时代面临的两个严重问题

RNN 在 NLP 界一直红了很多年（2014-2018？），在 2018 年之前，大部分各个子领域的 State of Art 的结果都是 RNN 获得的。但是最近一年来，眼看着 RNN 的领袖群伦的地位正在被动摇，所谓各领风骚 3-5 年，看来网红模型也不例外。

那这又是因为什么呢？主要有两个原因。

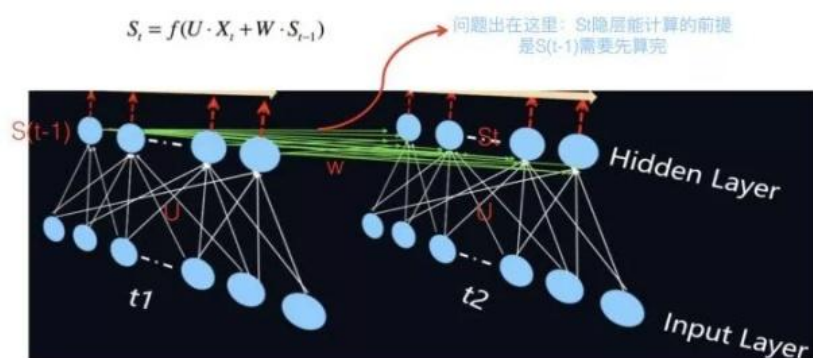
第一个原因在于**一些后起之秀新模型的崛起**，比如经过特殊改造的 CNN 模型，以及最近特别流行的 Transformer，这些后起之秀尤其是 Transformer 的应用效果相比 RNN 来说，目前看具有明显的优势。这是个主要原因，老人如果干不过新人，又没有脱胎换骨自我革命的能力，自然要自觉或不自愿地退出历史舞台，这是自然规律。至于 RNN 能力偏弱的具体证据，本文后

的，所以也想或者正在想一些改进方法，试图给 RNN 延年益寿。至于这些方法是什么，有没有作用，后面也陆续会谈。

另外一个严重阻碍 RNN 将来继续走红的问题是：**RNN 本身的序列依赖结构对于大规模并行计算来说相当之不友好**。通俗点说，就是 RNN 很难具备高效的并行计算能力，这个乍一看好像不是太大的问题，其实问题很严重。如果你仅仅满足于通过改 RNN 发一篇论文，那么这确实不是大问题，但是如果工业界进行技术选型的时候，在有快得多的模型可用的前提下，是不太可能选择那么慢的模型的。一个没有实际落地应用支撑其存在价值的模型，其前景如何这个问题，估计用小脑思考也能得出答案。

那问题来了：为什么 RNN 并行计算能力比较差？是什么原因造成的？

## RNN为什么并行计算能力差？



我们知道，RNN 之所以是 RNN，能将其和其它模型区分开的最典型标志是：T 时刻隐层状态的计算，依赖两个输入，一个是 T 时刻的句子输入单词  $X_t$ ，这个不算特点，所有模型都要接收这个原始输入；关键的是另外一个输入，T 时刻的隐层状态  $S_t$  还依赖 T-1 时刻的隐层状态  $S_{t-1}$  的输出，这是最能体现 RNN 本质特征的一点，RNN 的历史信息是通过这个信息传输渠道往后传输的，示意参考上图。那么为什么 RNN 的并行计算能力不行呢？问题就出在这里。因为 T 时刻的计算依赖 T-1 时刻的隐层计算结果，而 T-1 时刻的计算依赖 T-2 时刻的隐层计算结果……这样就形成了所谓的序列依赖关系。就是说只能先把第 1 时间步的算完，才能算第 2 时间步的结果，这就造成了 RNN 在这个角度上是无法并行计算的，只能老实地按着时间步一个单词一个单词往后走。

而 CNN 和 Transformer 就不存在这种序列依赖问题，所以对于这两者来说并行计算能力就不是问题，每个时间步的操作可以并行一起计算。

那么能否针对性地对 RNN 改造一下，提升它的并行计算能力呢？如果可以的话，效果如何呢？下面我们讨论一下这个问题。

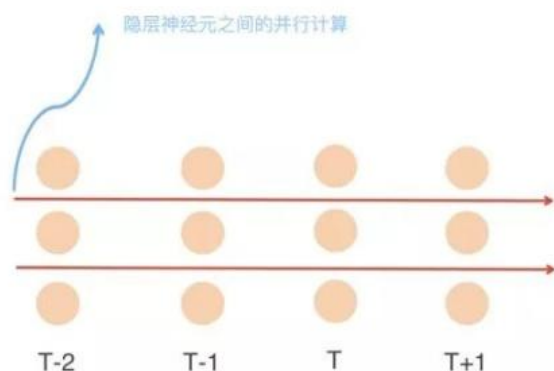
### 如何改造 RNN 使其具备并行计算能力？

上面说过，RNN 不能并行计算的症结所在，在于 T 时刻对 T-1 时刻计算结果的依赖，而这体现在隐层之间的全连接网络上。既然症结在这里，那么要想解决问题，也得在这个环节下手才行。在这个环节多做点什么事情能够增加 RNN 的并行计算能力呢？你可以想一想。

断连续时间步（ $T-1$  到  $T$  时刻）之间的隐层连接。

我们先来看第一种方法，现在我们的问题转化成了：我们仍然要保留任意连续时间步（ $T-1$  到  $T$  时刻）之间的隐层连接，但是在这个前提下，我们还要能够做到并行计算，这怎么处理呢？因为只要保留连续两个时间步的隐层连接，则意味着要计算  $T$  时刻的隐层结果，就需要  $T-1$  时刻隐层结果先算完，这不又落入了序列依赖的陷阱里了吗？嗯，确实是这样，但是为什么一定要在不同时间步的输入之间并行呢？没有人说 RNN 的并行计算一定发生在不同时间步上啊，你想想，隐层是不是也是包含很多神经元？那么在隐层神经元之间并行计算行吗？如果你要是还没理解这是什么意思，那请看下图。

## RNN的一种并行计算思路



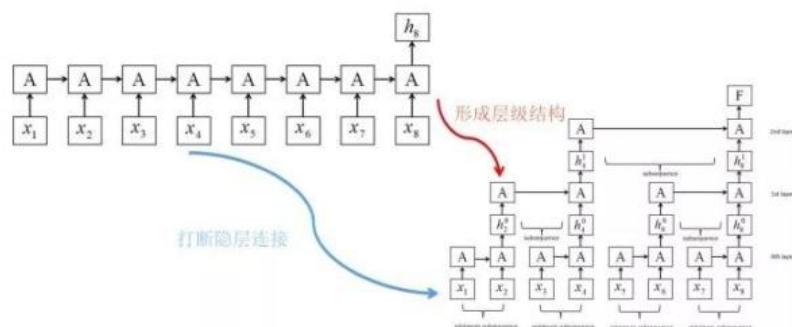
上面的图只显示了各个时间步的隐层节点，每个时间步的隐层包含 3 个神经元，这是个俯视图，是从上往下看 RNN 的隐层节点的。另外，连续两个时间步的隐层神经元之间仍然有连接，上图没有画出来是为了看着简洁一些。这下应该明白了吧，假设隐层神经元有 3 个，那么我们可以形成 3 路并行计算（红色箭头分隔开成了三路），而每一路因为仍然存在序列依赖问题，所以每一路内仍然是串行的。大思路应该明白了是吧？但是了解 RNN 结构的同学会发现这样还遗留一个问题：隐层神经元之间的连接是全连接，就是说  $T$  时刻某个隐层神经元与  $T-1$  时刻所有隐层神经元都有连接，如果是这样，是无法做到在神经元之间并行计算的，你可以想想为什么，这个简单，我假设你有能力想明白。那么怎么办呢？很简单， $T$  时刻和  $T-1$  时刻的隐层神经元之间的连接关系需要改造，从之前的全连接，改造成对应位置的神经元（就是上图被红箭头分隔到同一行的神经元之间）有连接，和其它神经元没有连接。这样就可以解决这个问题，在不同路的隐层神经元之间可以并行计算了。

第一种改造 RNN 并行计算能力的方法思路大致如上所述，这种方法的代表就是论文“Simple Recurrent Units for Highly Parallelizable Recurrence”中提出的 **SRU 方法**，它最本质的改进是把隐层之间的神经元依赖由全连接改成了哈达马乘积，这样  $T$  时刻隐层单元本来对  $T-1$  时刻所有隐层单元的依赖，改成了只是对  $T-1$  时刻对应单元的依赖，于是可以在隐层单元之间进行并行计算，但是收集信息仍然是按照时间序列来进行的。所以其并行性是在隐层单元之间发生的，而不是在不同时间步之间发生的。

这其实是比较巧妙的一种方法，但是它的问题在于其并行程度上限是有限的，并行程度取决于隐层神经元个数，而一般这个数值往往不会太大，再增加并行性已经不太可能。另外每一路并行线路仍然需要序列计算，这也会拖

阅读理解及 MT 任务上只做了效果评估，没有和 CNN 进行速度比较，我估计这是有原因的，因为复杂任务往往需要深层网络，其它的就不妄作猜测了。

## Sliced Recurrent Neural Networks



第二种改进典型的思路是：为了能够在不同时间步输入之间进行并行计算，那么只有一种做法，那就是**打断隐层之间的连接，但是又不能全打断**，因为这样基本就无法捕获组合特征了，所以唯一能选的策略就是部分打断，比如每隔 2 个时间步打断一次，但是距离稍微远点的特征如何捕获呢？只能加深层深，通过层深来建立远距离特征之间的联系。代表性模型比如上图展示的 Sliced RNN。我当初看到这个模型的时候，心里忍不住发出杠铃般的笑声，情不自禁地走上前跟他打了个招呼：你好呀，CNN 模型，想不到你这个糙汉子有一天也会穿上粉色裙装，装扮成 RNN 的样子出现在我面前啊，哈哈。了解 CNN 模型的同学看到我上面这句话估计会莞尔会心一笑：这不就是简化版本的 CNN 吗？不了解 CNN 的同学建议看完后面 CNN 部分再回头来看看是不是这个意思。

那经过这种改造的 RNN 速度改进如何呢？论文给出了速度对比实验，归纳起来，SRNN 速度比 GRU 模型快 5 到 15 倍，嗯，效果不错，但是跟对比模型 DC-CNN 模型速度比较起来，比 CNN 模型仍然平均慢了大约 3 倍。这很正常但是又有点说不过去，说正常是因为本来这就是把 RNN 改头换面成类似 CNN 的结构，而片段里仍然采取 RNN 序列模型，所以必然会拉慢速度，比 CNN 慢再正常不过了。说“说不过去”是指的是：既然本质上是 CNN，速度又比 CNN 慢，那么这么改的意义在哪里？为什么不直接用 CNN 呢？是不是？前面那位因为吃亏吃的少所以爱抬杠的同学又会说了：也许人家效果特别好吧。嗯，从这个结构的作用机制上看，可能性不太大。你说论文实验部分证明了这一点呀，我认为实验部分对比试验做的不充分，需要补充除了 DC-CNN 外的其他 CNN 模型进行对比。当然这点纯属个人意见，别当真，因为我讲起话来经常摇头晃脑，此时一般会有人惊奇地跟我反馈说：为什么你一讲话我就听到了水声？

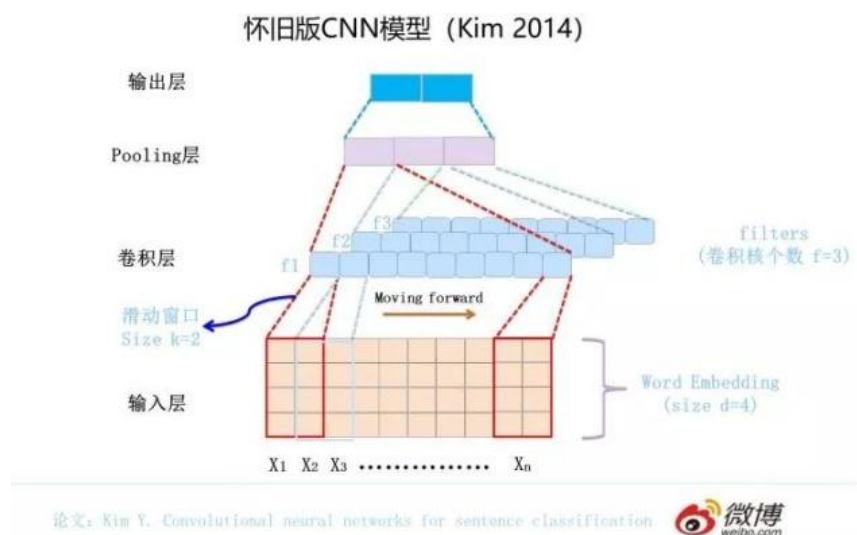
上面列举了两种大的改进 RNN 并行计算能力的思路，我个人对于 RNN 的并行计算能力持悲观态度，主要因为 RNN 本质特性决定了我们能做的选择太少。无非就是选择打断还是不打断隐层连接的问题。如果选择打断，就会面临上面的问题，你会发现它可能已经不是 RNN 模型了，为了让它看上去还像是 RNN，所以在打断片段里仍然采取 RNN 结构，这样无疑会拉慢速度，所以这是个两难的选择，与其这样不如直接换成其它模型；如果我们选择不打断，貌似只能在隐层神经元之间进行并行，而这样做的缺点是：一方面并



偏师之将 CNN：刺激战场绝地求生

在一年多前，CNN 是自然语言处理中除了 RNN 外最常见的深度学习模型，这里介绍下 CNN 特征抽取器，会比 RNN 说得详细些，主要考虑到大家对它的熟悉程度可能没有 RNN 那么高。

## NLP 中早期的怀旧版 CNN 模型



最早将 CNN 引入 NLP 的是 Kim 在 2014 年做的工作，论文和网络结构参考上图。一般而言，输入的字或者词用 **Word Embedding** 的方式表达，这样本来一维的文本信息输入就转换成了二维的输入结构，假设输入  $X$  包含  $n$  个字符，而每个字符的 Word Embedding 的长度为  $d$ ，那么输入就是  $d \times n$  的二维向量。

卷积层本质上是个特征抽取层，可以设定超参数  $F$  来指定卷积层包含多少个卷积核 (Filter)。对于某个 Filter 来说，可以想象有一个  $d \times k$  大小的移动窗口从输入矩阵的第一个字开始不断往后移动，其中  $k$  是 Filter 指定的窗口大小， $d$  是 Word Embedding 长度。对于某个时刻的窗口，通过神经网络的非线性变换，将这个窗口内的输入值转换为某个特征值，随着窗口不断往后移动，这个 Filter 对应的特征值不断产生，形成这个 Filter 的特征向量。这就是卷积核抽取特征的过程。卷积层内每个 Filter 都如此操作，就形成了不同的特征序列。Pooling 层则对 Filter 的特征进行降维操作，形成最终的特征。一般在 Pooling 层之后连接全联接层神经网络，形成最后的分类过程。

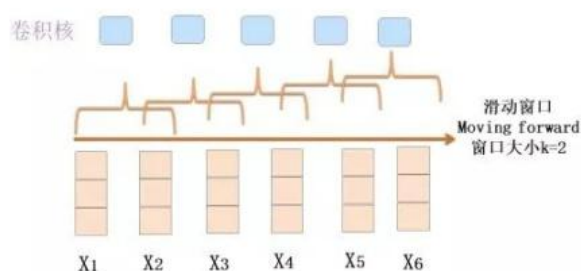
这就是最早应用在 NLP 领域 CNN 模型的工作机制，用来解决 NLP 中的**句子分类任务**，看起来还是很简洁的，之后陆续出现了在此基础上的改进模型。这些怀旧版 CNN 模型在一些任务上也能和当时怀旧版本的 RNN 模型效果相当，所以在 NLP 若干领域也能野蛮生长，但是在更多的 NLP 领域，还是处于被 RNN 模型压制到抑郁症早期的尴尬局面。那为什么在图像领域打遍天下无敌手的 CNN，一旦跑到 NLP 的地盘，就被 RNN 这个地头蛇压制得无颜见图像领域江东父老呢？这说明这个版本的 CNN 还是有很多问题的，其实最根本的症结所在还是老革命遇到了新问题，主要是到了新环境没有针对新环境的特性做出针对性的改变，所以面临水土不服的问题。

CNN 能在 RNN 纵横的各种 NLP 任务环境下生存下来吗？谜底即将揭晓。

下面我们先看看怀旧版 CNN 存在哪些问题，然后看看我们的 NLP 专家们是如何改造 CNN，一直改到目前看上去还算效果不错的现代版本 CNN 的。

首先，我们先要明确一点：CNN 捕获到的是什么特征呢？从上述怀旧版本 CNN 卷积层的运作机制你大概看出来了，关键在于卷积核覆盖的那个滑动窗口，CNN 能捕获到的特征基本都体现在这个滑动窗口里了。大小为  $k$  的滑动窗口轻轻的穿过句子的一个个单词，荡起阵阵涟漪，那么它捕获了什么？其实它捕获到的是单词的  $k$ -gram 片段信息，这些  $k$ -gram 片段就是 CNN 捕获到的特征， $k$  的大小决定了能捕获多远距离的特征。

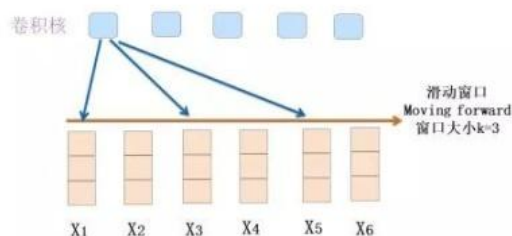
## 单卷积层无法捕获远距离特征



远距离特征: X1=“虽然” X5=“但是”

说完这个，我们来看 Kim 版 CNN 的第一个问题：它只有一个卷积层。表面上看上去好像是深度不够的问题是吧？我会反问你：为什么要把 CNN 作深呢？其实把深度做起来是手段，不是目的。只有一个卷积层带来的问题是：对于远距离特征，单层 CNN 是无法捕获到的，如果滑动窗口  $k$  最大为 2，而如果有个远距离特征距离是 5，那么无论上多少个卷积核，都无法覆盖到长度为 5 的距离的输入，所以它是**无法捕获长距离特征**的。

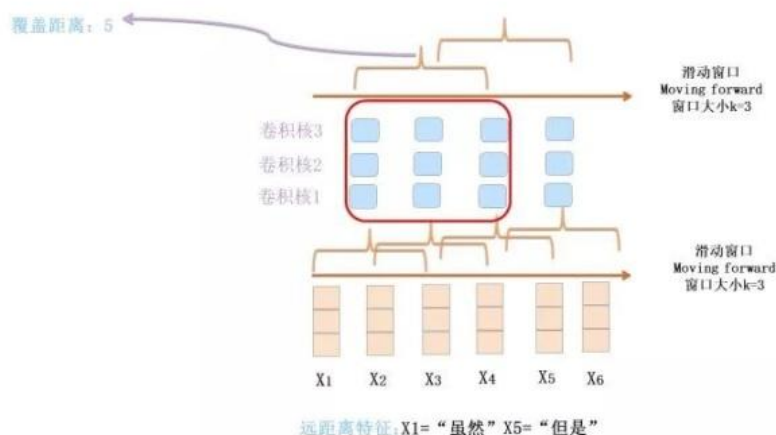
## Dilated CNN 捕获远距离特征



远距离特征: X1=“虽然” X5=“但是”

那么怎样才能**捕获到长距离的特征**呢？有两种典型的改进方法：一种是假设我们仍然用单个卷积层，滑动窗口大小  $k$  假设为 3，就是只接收三个输入单词，但是我们想捕获距离为 5 的特征，怎么做才行？显然，如果卷积核窗口仍然覆盖连续区域，这肯定是完不成任务的。提示一下：你玩过跳一跳是

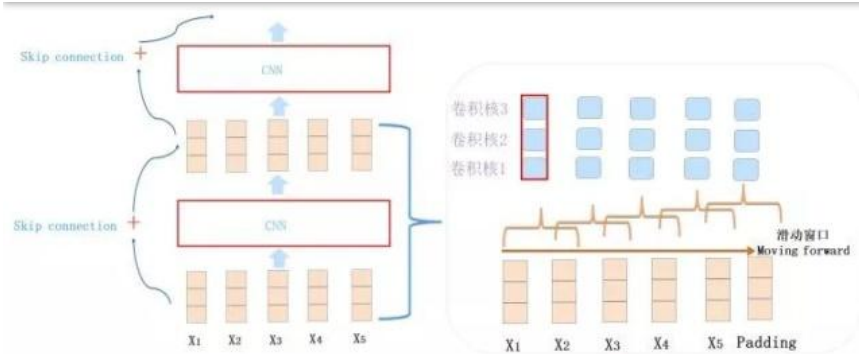
## 加深CNN网络来捕获远距离特征



第二种方法是把深度做起来。第一层卷积层，假设滑动窗口大小  $k$  是 3，如果再往上叠一层卷积层，假设滑动窗口大小也是 3，但是第二层窗口覆盖的是第一层窗口的输出特征，所以它其实能覆盖输入的距离达到了 5。如果继续往上叠加卷积层，可以继续增大卷积核覆盖输入的长度。

上面是两种典型的解决 CNN 远距离特征捕获能力的方案，Dilated CNN 偏技巧一些，而且叠加卷积层时超参如何设置有些学问，因为连续跳接可能会错过一些特征组合，所以需要精心调节参数搭配，保证所有可能组合都被覆盖到。相对而言，把 CNN 作深是主流发展方向。上面这个道理好理解，其实自从 CNN 一出现，人们就想各种办法试图把 CNN 的深度做起来，但是现实往往是无情的，发现怎么折腾，CNN 做 NLP 问题就是做不深，做到 2 到 3 层卷积层就做不上去了，网络更深对任务效果没什么帮助（请不要拿 CharCNN 来做反例，后来研究表明使用单词的 2 层 CNN 效果超过 CharCNN）。目前看来，还是深层网络参数优化手段不足导致的这个问题，而不是层深没有用。后来 Resnet 等图像领域的新技术出现后，很自然地，人们会考虑把 Skip Connection 及各种 Norm 等参数优化技术引入，这才能慢慢把 CNN 的网络深度做起来。

上面说的是 Kim 版本 CNN 的第一个问题，无法捕获远距离特征的问题，以及后面科研人员提出的主要解决方案。回头看 Kim 版本 CNN 还有一个问题，就是那个 Max Pooling 层，这块其实与 CNN 能否保持输入句子中单词的位置信息有关系。首先我想问个问题：RNN 因为是线性序列结构，所以很自然它天然就会把位置信息编码进去；那么，CNN 是否能够保留原始输入的相对位置信息呢？我们前面说过对于 NLP 问题来说，位置信息是很有用的。其实 CNN 的卷积核是能保留特征之间的相对位置的，道理很简单，滑动窗口从左到右滑动，捕获到的特征也是如此顺序排列，所以它在结构上已经记录了相对位置信息了。但是如果卷积层后面立即接上 Pooling 层的话，Max Pooling 的操作逻辑是：从一个卷积核获得的特征向量里只选中并保留最强的那一个特征，所以到了 Pooling 层，位置信息就被扔掉了，这在 NLP 里其实是有信息损失的。所以在 NLP 领域里，目前 CNN 的一个发展趋势是抛弃 Pooling 层，靠全卷积层来叠加网络深度，这背后是有原因的（当然图像领域也是这个趋势）。



上图展示了在 NLP 领域能够施展身手的摩登 CNN 的主体结构，通常由 1-D 卷积层来叠加深度，使用 Skip Connection 来辅助优化，也可以引入 Dilated CNN 等手段。比如 ConvS2S 主体就是上图所示结构，Encoder 包含 15 个卷积层，卷积核 kernel size=3，覆盖输入长度为 25。当然对于 ConvS2S 来说，卷积核里引入 GLU 门控非线性函数也有重要帮助，限于篇幅，这里不展开说了，GLU 貌似是 NLP 里 CNN 模型必备的构件，值得掌握。再比如 TCN（论文：An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling），集成了几项技术：利用 Dilated CNN 拓展单层卷积层的输入覆盖长度，利用全卷积层堆叠层深，使用 Skip Connection 辅助优化，引入 Casual CNN 让网络结构看不到 T 时间步后的数据。不过 TCN 的实验做得有两个明显问题：一个问题是任务除了语言模型外都不是典型的 NLP 任务，而是合成数据任务，所以论文结论很难直接说就适合 NLP 领域；另外一点，它用来进行效果比较的对比方法，没有用当时效果很好的模型来对比，比较基准低。所以 TCN 的模型效果说服力不太够。其实它该引入的元素也基本引入了，实验说服力不够，我觉得可能是它命中缺 GLU 吧。

除此外，简单谈一下 CNN 的位置编码问题和并行计算能力问题。上面说了，CNN 的卷积层其实是保留了相对位置信息的，只要你在设计模型的时候别手贱，中间层不要随手瞎插入 Pooling 层，问题就不大，不专门在输入部分对 position 进行编码也行。但是也可以类似 ConvS2S 那样，专门在输入部分给每个单词增加一个 position embedding，将单词的 position embedding 和词向量 embedding 叠加起来形成单词输入，这样也可以，也是常规做法。

至于 CNN 的并行计算能力，那是非常强的，这其实很好理解。我们考虑单层卷积层，首先对于某个卷积核来说，每个滑动窗口位置之间没有依赖关系，所以完全可以并行计算；另外，不同的卷积核之间也没什么相互影响，所以也可以并行计算。CNN 的并行度是非常自由也非常高的，这是 CNN 的一个非常好的优点。

以上内容介绍了怀旧版 CNN 是如何在 NLP 修罗场一步步通过自我进化生存到今天的。CNN 的进化方向，如果千言万语一句话归纳的话，那就是：想方设法把 CNN 的深度做起来，随着深度的增加，很多看似无关的问题就随之解决了。就跟我们国家最近 40 年的主旋律是发展经济一样，经济发展好了，很多问题就不是问题了。最近几年之所以大家感到各方面很困难，症结就在于经济不行了，所以很多问题无法通过经济带动来解决，于是看似各种花样的困难就冒出来，这是一个道理。

那么介绍了这么多，摩登版 CNN 效果如何呢？与 RNN 及 Transforme 比起来怎样？别着急，后面会专门谈这个问题。



## Hi,我是Transformer



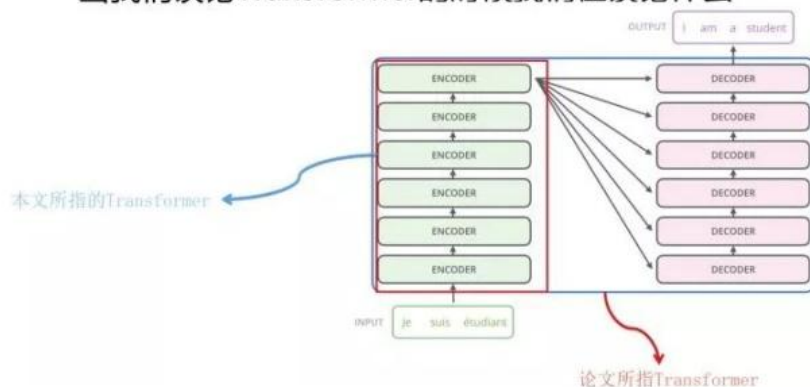
“如今走过这世间  
万般流连  
翻过岁月不同侧脸  
措不及防闯入你的笑颜”



Transformer 是谷歌在 17 年做机器翻译任务的 “**Attention is all you need**” 的论文中提出的，引起了相当大的反响。每一位从事 NLP 研发的同仁都应该透彻搞明白 Transformer，它的重要性毫无疑问，尤其是在你看完我这篇文章之后，我相信你的紧迫感会更迫切，我就是这么一位善于制造焦虑的能手。不过这里没打算重点介绍它，想要入门 Transformer 的可以参考以下三篇文章：一个是 Jay Alammar 可视化地介绍 Transformer 的博客文章 [The Illustrated Transformer](#)，非常容易理解整个机制，建议先从这篇看起，这是中文翻译版本；第二篇是 Calvo 的博客：[Dissecting BERT Part 1: The Encoder](#)，尽管说是解析 Bert，但是因为 Bert 的 Encoder 就是 Transformer，所以其实它是在解析 Transformer，里面举的例子很好；再然后可以进阶一下，参考哈佛大学 NLP 研究组写的 “[The Annotated Transformer](#).”，代码原理双管齐下，讲得也很清楚。

下面只说跟本文主题有关的内容。

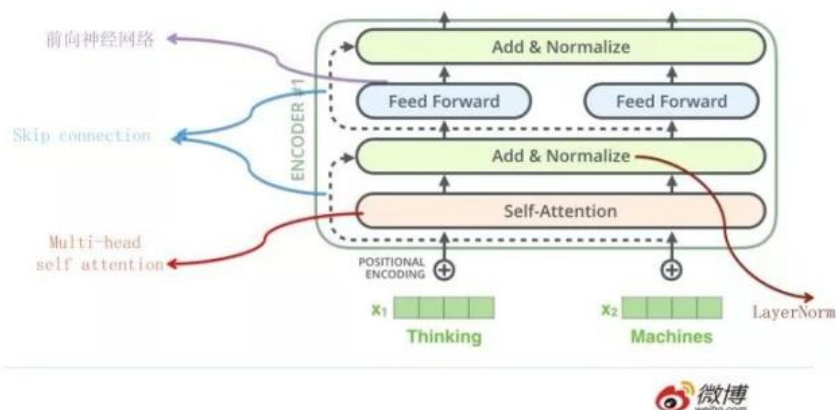
### 当我们谈论Transformer的时候我们在谈论什么



这里要澄清一下，本文所说的 Transformer 特征抽取器并非原始论文所指。我们知道，“Attention is all you need” 论文中说到的 Transformer 指的是完整的 Encoder-Decoder 框架，而我这里是从特征抽取器角度来说的，你可以简单理解为**论文中的 Encoder 部分**。因为 Encoder 部分目的比较单纯，就是从原始句子中提取特征，而 Decoder 部分则功能相对比较多，除了特征提取功能外，还包含语言模型功能，以及用 attention 机制表达的翻译模型功能。所以这里请注意，避免后续理解概念产生混淆。

Alammar 把每个 Block 称为 Encoder 不太符合常规叫法) 是由若干个相同的 Transformer Block 堆叠成的。这个 Transformer Block 其实才是 Transformer 最关键的地方，核心配方就在这里。那么它长什么样子呢？

## Transformer Block长什么样子？



它的照片见上图，看上去是不是很可爱，有点像安卓机器人是吧？这里需要强调一下，尽管 Transformer 原始论文一直重点在说 Self Attention，但是目前来看，能让 Transformer 效果好的，不仅仅是 Self attention，这个 Block 里所有元素，包括 Multi-head self attention，Skip connection，LayerNorm，FF 一起在发挥作用。为什么这么说？你看到后面会体会到这一点。

我们针对 NLP 任务的特点来说下 Transformer 的对应解决方案。首先，自然语言一般是个不定长的句子，那么这个不定长问题怎么解决呢？Transformer 做法跟 CNN 是类似的，一般设定输入的最大长度，如果句子没那么长，则用 Padding 填充，这样整个模型输入起码看起来是定长的了。另外，NLP 句子中单词之间的相对位置是包含很多信息的，上面提过，RNN 因为结构就是线性序列的，所以天然会将位置信息编码进模型；而 CNN 的卷积层其实也是保留了位置相对信息的，所以什么也不做问题也不大。但是对于 Transformer 来说，为了能够保留输入句子单词之间的相对位置信息，必须得做点什么。为啥它必须得做点什么呢？因为输入的第一层网络是 Multi-head self attention 层，我们知道，Self attention 会让当前输入单词和句子中任意单词发生关系，然后集成到一个 embedding 向量里，但是当所有信息到了 embedding 后，位置信息并没有被编码进去。所以，Transformer 不像 RNN 或 CNN，必须明确的在输入端将 Position 信息编码，Transformer 是用位置函数来进行位置编码的，而 Bert 等模型则给每个单词一个 Position embedding，将单词 embedding 和单词对应的 position embedding 加起来形成单词的输入 embedding，类似上文讲的 ConvS2S 的做法。而关于 NLP 句子中长距离依赖特征的问题，Self attention 天然就能解决这个问题，因为在集成信息的时候，当前单词和句子中任意单词都发生了联系，所以一步到位就把这个事情做掉了。不像 RNN 需要通过隐层节点序列往后传，也不像 CNN 需要通过增加网络深度来捕获远距离特征，Transformer 在这点上明显方案是相对简单直观的。说这些是为了单独介绍下 Transformer 是怎样解决 NLP 任务几个关键点的。

Transformer 有两个版本：**Transformer base**和**Transformer Big**。两者结构其实是一样的，主要区别是包含的 Transformer Block 数量不同，Transformer base 包含 12 个 Block 叠加，而 Transformer Big 则扩张一倍，

## 华山论剑：三大特征抽取器比较

结合 NLP 领域自身的特点，上面几个部分分别介绍了 RNN / CNN / Transformer 各自的特性。从上面的介绍，看上去好像三大特征抽取器在 NLP 领域里各有所长，推想起来要是把它们拉到 NLP 任务竞技场角斗，一定是互有胜负，各擅胜场吧？

事实究竟如何呢？是三个特征抽取器三花齐放还是某一个一枝独秀呢？我们通过一些实验来说明这个问题。

为了更细致和公平地做对三者进行比较，我准备从几个不同的角度来分别进行对比，我原先打算从以下几个维度来进行分析判断：句法特征提取能力；语义特征提取能力；长距离特征捕获能力；任务综合特征抽取能力。上面四个角度是从 NLP 的特征抽取器能力强弱角度来评判的，另外再加入并行计算能力及运行效率，这是从是否方便大规模实用化的角度来看的。


因为目前关于特征抽取器句法特征抽取能力方面进行比较的文献很少，好像只看到一篇文章，结论是 CNN 在句法特征提取能力要强于 RNN，但是因为是比较早的文章，而且没有对比 transformer 在句法特征抽取方面的能力，所以这块很难单独比较，于是我就简化为对以下几项能力的对比：

1. 语义特征提取能力；
2. 长距离特征捕获能力；
3. 任务综合特征抽取能力；
4. 并行计算能力及运行效率

三者在这些维度各自表现如何呢？下面我们分头进行说明。

### 语义特征提取能力

#### RNN / CNN / Transformer 语义特征提取能力对比



Model	DE→EN				DE→FR			
	PPL	2014	2017	Acc(%)	PPL	2012	Acc(%)	
RNNS2S	5.7	29.1	30.1	84.0	7.06	16.4	72.2	
ComS2S	6.3	29.1	30.4	82.3	7.93	16.8	72.7	
Transformer	<b>4.3</b>	<b>32.7</b>	33.7	<b>90.3</b>	<b>4.9</b>	<b>18.7</b>	<b>76.7</b>	
uedin-wmt17	-	-	<b>35.1</b>	87.9	-	-	-	
TransRNN	5.2	30.5	31.9	86.1	6.3	17.6	74.2	

Table 5: The results of different architectures on *newstest* sets and *ContraWSD*. PPL is the perplexity on the validation set. Acc means accuracy on the test set.

Transformer

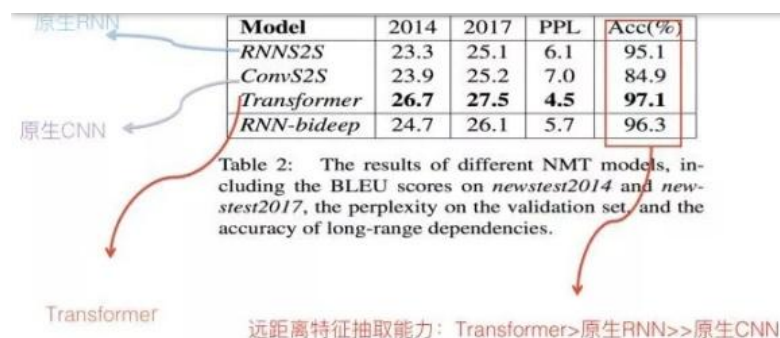
语义特征抽取能力：Transformer>>原生CNN==原生RNN

论文：Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures



从语义特征提取能力来说，目前实验支持如下结论：**Transformer 在这方面的能力非常显著地超过 RNN 和 CNN**（在考察语义类能力的任务 WSD 中，Transformer 超过 RNN 和 CNN 大约 4-8 个绝对百分点），RNN 和 CNN 两者能力差不太多。

### 长距离特征捕获能力



Model	2014	2017	PPL	Acc(%)
<i>RNNS2S</i>	23.3	25.1	6.1	95.1
<i>ConvS2S</i>	23.9	25.2	7.0	84.9
<i>Transformer</i>	<b>26.7</b>	<b>27.5</b>	<b>4.5</b>	<b>97.1</b>
<i>RNN-bideep</i>	24.7	26.1	5.7	96.3

Table 2: The results of different NMT models, including the BLEU scores on *newstest2014* and *newstest2017*, the perplexity on the validation set, and the accuracy of long-range dependencies.

Transformer 远距离特征抽取能力: Transformer > 原生RNN >> 原生CNN

论文: Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures 

在长距离特征捕获能力方面，目前在特定的长距离特征捕获能力测试任务中（主语 - 谓语一致性检测，比如 we.....are...），实验支持如下结论：原生 CNN 特征抽取器在这方面极为显著地弱于 RNN 和 Transformer，Transformer 微弱优于 RNN 模型（尤其在主语谓语距离小于 13 时），能力由强到弱排序为 Transformer > RNN >> CNN；但在比较远的距离上（主语谓语距离大于 13），RNN 微弱优于 Transformer，所以综合看，可以认为 **Transformer 和 RNN 在这方面能力差不太多，而 CNN 则显著弱于前两者。**

那么为什么 CNN 在捕获长距离特征方面这么弱呢？这个我们在前文讲述 CNN 的时候就说过，CNN 解决这个问题是靠堆积深度来获得覆盖更长的输入长度的，所以 CNN 在这方面的表现与卷积核能够覆盖的输入距离最大长度有关系。如果通过增大卷积核的 kernel size，同时加深网络深度，以此来增加输入的长度覆盖。实验证明这能够明显提升 CNN 的 long-range 特征捕获能力。但是尽管如此，CNN 在这方面仍然显著弱于 RNN 和 Transformer。这个问题背后的原因是什么呢（因为上述主语 - 谓语一致性任务中，CNN 的深度肯定可以覆盖 13-25 这个长度了，但是表现还是很弱）？其实这是一个很好的值得探索的点。

对于 Transformer 来说，Multi-head attention 的 head 数量严重影响 NLP 任务中 Long-range 特征捕获能力：结论是 head 越多越有利于捕获 long-range 特征。在上页 PPT 里写明的论文出来之前，有个工作（论文：Tran. The Importance of Being Recurrent for Modeling Hierarchical Structure）的结论和上述结论不一致：它的结论是在“主语 - 谓语一致性”任务上，Transformer 表现是弱于 LSTM 的。如果综合这两篇论文，我们看似得到了相互矛盾的结论，那么到底谁是正确的呢？Why Self-attention 的论文对此进行了探索，它的结论是：这个差异是由于两个论文中的实验中 Transformer 的超参设置不同导致的，其中尤其是 multi-head 的数量，对结果影响严重，而如果正确设置一些超参，那么之前 Trans 的论文结论是不成立的。也就是说，我们目前仍然可以维持下面结论：**在远距离特征捕获能力方面，Transformer 和 RNN 能力相近，而 CNN 在这方面则显著弱于前两者。**

## 任务综合特征抽取能力

上面两项对比是从特征抽取的两个比较重要的单项能力角度来评估的，其实更重要的是在具体任务中引入不同特征抽取器，然后比较效果差异，以此来综合评定三者的综合能力。那么这样就引出一个问题：NLP 中的任务很多，哪些任务是最具有代表性的呢？答案是机器翻译。你会看到很多 NLP 的重要的创新模型都是在机器翻译任务上提出来的，这背后是有道理的，因为机器翻译基本上是对 NLP 各项处理能力综合要求最高的任务之一，要想获得高质



在机器翻译上作出的，这里给个背后原因的解釋，以避免被质疑任务单一，没有说服力的问题。当然，我预料到那位“因为吃亏少.... 爱挑刺”的同学会这么质问我，没关系，即使你对此提出质疑，我依然能够拿出证据，为什么这么讲，请往后看。

那么在以机器翻译为代表的综合特征抽取能力方面，三个特征抽取器哪个更好些呢？

## RNN / CNN / Transformer综合特征提取能力对比

原生RNN

原生CNN

Model	DE→EN				DE→FR		
	PPL	2014	2017	Acc(%)	PPL	2012	Acc(%)
RNNS2S	5.7	29.1	30.1	84.0	7.06	16.4	72.2
ConvS2S	6.3	29.1	30.4	82.3	7.93	16.8	72.7
Transformer	<b>4.3</b>	<b>32.7</b>	33.7	<b>90.3</b>	<b>4.9</b>	<b>18.7</b>	<b>76.7</b>
uedin-wmt17	-	-	<b>35.1</b>	87.9	-	-	-
TransRNN	5.2	30.5	31.9	86.1	6.3	17.6	74.2

Table 5: The results of different architectures on newstest sets and ContraWSD. PPL is the perplexity on the validation set. Acc means accuracy on the test set.

Transformer

综合特征抽取能力：Transformer>原生CNN==原生RNN

论文：Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures. 微博 weibo.com

先给出一个机器翻译任务方面的证据，仍然是 why Self attention 论文的结论，对比实验结果数据参考上图。在两个机器翻译任务中，可以看到，翻译质量指标 BLEU 证明了如下结论：**Transformer 综合能力要明显强于 RNN 和 CNN**（你要知道，技术发展到现在阶段，BLEU 绝对值提升 1 个点是很难的事情），而 RNN 和 CNN 看上去表现基本相当，貌似 CNN 表现略好一些。

你可能觉得一个论文的结论不太能说明问题，那么我再给出一个证据，不过这个证据只对比了 Transformer 和 RNN，没带 CNN 玩，不过关于说服力我相信你不会质疑，实验对比数据如下：

## RNN / CNN / Transformer综合特征提取能力对比

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (mc= Matthews correlation, acc=Accuracy, pc=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STS-B (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	<b>70.3</b>	<b>81.8</b>	<b>88.1</b>	<b>56.0</b>
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	<b>75.0</b>	<b>47.9</b>	<b>92.0</b>	<b>84.9</b>	<b>83.2</b>	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

Transformer特征抽取能力远强于LSTM

论文：Improving Language Understanding by Generative Pre-Training

微博 weibo.com

上面是 GPT 论文的实验结论，在 8 个不同的 NLP 任务上，在其它条件相同的情况下，只是把特征抽取器从 Transformer 换成 LSTM，平均下来 8 个任务得分掉了 5 个点上。这具备足够说服力吗？

来你服气。如果归纳一下的话，现在能得出的结论是这样的：从综合特征抽取能力角度衡量，Transformer 显著强于 RNN 和 CNN，而 RNN 和 CNN 的表现差不多，如果一定要在这两者之间比较的话，通常 CNN 的表现要稍微好于 RNN 的效果。

当然，需要强调一点，本部分所说的 RNN 和 CNN 指的是原生的 RNN 和 CNN 模型，就是说你可以在经典的结构上增加 attention，堆叠层次等各种改进，但是不包含对本身结构特别大的变动，就是说支持整容，但是不支持变性。这里说的原生版本指的是整容版本，我知道你肯定很关心有没有变性版本的 RNN 和 CNN，我负责任地跟你说，有。你想知道它变性之后是啥样子？等会你就看到了，有它们的照片给你。

## 并行计算能力及运算效率

关于三个特征抽取器的并行计算能力，其实我们在前文分述三个模型的时候都大致提过，在此仅做个归纳，结论如下：

RNN 在并行计算方面有严重缺陷，这是它本身的序列依赖特性导致的，所谓成也萧何败也萧何，它的这个线形序列依赖性非常符合解决 NLP 任务，这也是为何 RNN 一引入到 NLP 就很快流行起来的原因，但是也正是这个线形序列依赖特性，导致它在并行计算方面要想获得质的飞跃，看起来困难重重，近乎是不太可能完成的任务。

而对于 CNN 和 Transformer 来说，因为它们不存在网络中间状态不同时间步输入的依赖关系，所以可以非常方便及自由地做并行计算改造，这个也好理解。

所以归纳一下的话，可以认为并行计算能力由高到低排序如下：  
**Transformer 和 CNN 差不多，都远远远远强于 RNN。**

我们从另外一个角度来看，先抛开并行计算能力的问题，单纯地比较一下三个模型的计算效率。可能大家的直观印象是 Transformer 比较重，比较复杂，计算效率比较低，事实是这样的吗？

### RNN / CNN / Transformer 计算效率对比

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types.  $n$  is the sequence length,  $d$  is the representation dimension,  $k$  is the kernel size of convolutions and  $r$  the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

$n$ : 句子长度  
 $d$ : embedding size

单层计算效率：三者都包含平方项，取决于  $d$  和  $n$  谁的平均值大

原文: Tensor2Tensor for Neural Machine Translation



上图列出了单层的 Self attention / RNN / CNN 的计算效率，首先要注意：上面列的是 Self attention，不是 Transformer 的 Block，因为 Transformer Block 里其实包含了好几层，而不是单层。我们先说 self attention，等会说 Transformer Block 的计算量。

为每一个单词都需要和任意一个单词发生关系来计算 attention，所以包含一个  $n$  的平方项。而 RNN 和 CNN 的平方项则是 embedding size。那么既然都包含平方项，怎么比较三个模型单层的计算量呢？首先容易看出 CNN 计算量是大于 RNN 的，那么 self attention 如何与其它两者比较呢。可以这么考虑：如果句子平均长度  $n$  大于 embedding size，那么意味着 Self attention 的计算量要大于 RNN 和 CNN；而如果反过来，就是说如果 embedding size 大于句子平均长度，那么明显 RNN 和 CNN 的计算量要大于 self attention 操作。而事实上是怎样？我们可以想一想，一般正常的句子长度，平均起来也就几十个单词吧。而当前常用的 embedding size 从 128 到 512 都常见，所以在大多数任务里面其实 self attention 计算效率是要高于 RNN 和 CNN 的。

但是，那位因为吃亏吃的少所以喜欢挑刺的同学会继续质问我：“哥，我想知道的是 Transformer 和 RNN 及 CNN 的计算效率对比，不是 self attention。另外，你能降低你脑袋里发出的水声音量吗？”。嗯，这个质问很合理，我来粗略估算一下，因为 Transformer 包含多层，其中的 skip connection 后的 Add 操作及 LayerNorm 操作不太耗费计算量，我先把它忽略掉，后面的 FFN 操作相对比较耗时，它的时间复杂度应该是  $n$  乘以  $d$  的平方。所以如果把 Transformer Block 多层当作一个整体和 RNN 及 CNN 单层对比的话，Transformer Block 计算量肯定是要多于 RNN 和 CNN 的，因为它本身也包含一个  $n$  乘以  $d$  的平方，上面列出的 self attention 的时间复杂度就是多出来的计算量。这么说起来，单个 Transformer Block 计算量大于单层 RNN 和 CNN，没毛病。

上面考虑的是三者单层的计算量，可以看出结论是：**Transformer Block > CNN > RNN**。如果是考虑不同的具体模型，会与模型的网络层深有很大关系，另外还有常见的 attention 操作，所以问题会比较复杂，这里不具体讨论了。

说完非并行情形的三者单层计算量，再说回并行计算的问题。很明显，对于 Transformer 和 CNN 来说，那个句子长度  $n$  是可以通过并行计算消掉的，而 RNN 因为序列依赖的问题，那个  $n$  就消不掉，所以很明显，把并行计算能力考虑进来，RNN 消不掉的那个  $n$  就很要命。这只是理论分析，实际中三者计算效率到底如何呢？我们给出一些三者计算效率对比的实验结论。

论文 “Convolutional Sequence to Sequence Learning” 比较了 ConvS2S 与 RNN 的计算效率，证明了跟 RNN 相比，CNN 明显速度具有优势，在训练和在线推理方面，CNN 比 RNN 快 9.3 倍到 21 倍。论文 “Dissecting Contextual Word Embeddings: Architecture and Representation” 提到了 Transformer 和 CNN 训练速度比双向 LSTM 快 3 到 5 倍。论文 “The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation” 给出了 RNN / CNN / Transformer 速度对比实验，结论是：Transformer Base 速度最快；CNN 速度次之，但是比 Transformer Base 慢了将近一倍；Transformer Big 速度再次，主要因为它的参数量最大，而吊在车尾最慢的是 RNN 结构。

总而言之，关于三者速度对比方面，目前的主流经验结论基本如上所述：**Transformer Base 最快，CNN 次之，再次 Transformer Big，最慢的是 RNN**。RNN 比前两者慢了 3 倍到几十倍之间。

## 综合排名情况

效果方面来说，Transformer 明显优于 CNN，CNN 略微优于 RNN。速度方面 Transformer 和 CNN 明显占优，RNN 在这方面劣势非常明显。这两者再综合起来，如果我给的排序结果是 Transformer>CNN>RNN，估计没有什么问题吧？那位吃亏..... 爱挑刺的同学，你说呢？

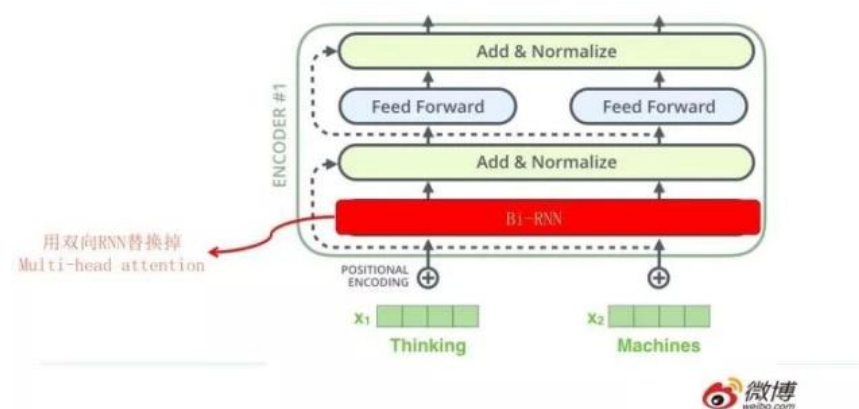
从速度和效果折衷的角度看，对于工业界实用化应用，我的感觉在特征抽取器选择方面配置 Transformer base 是个较好的选择。

三者的合流：向 Transformer 靠拢

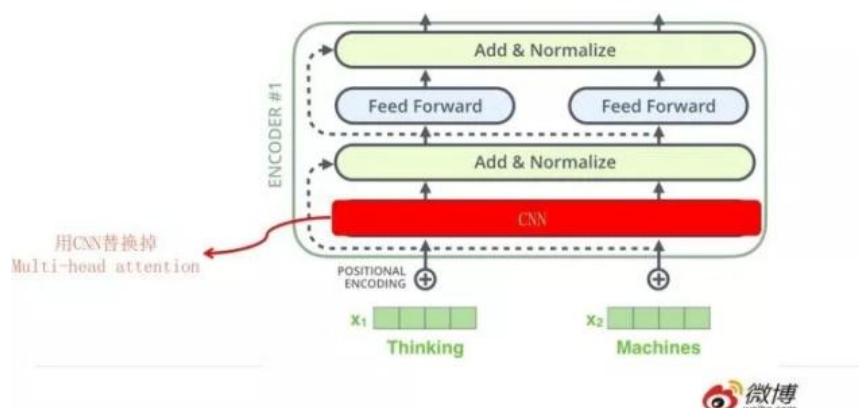
上文提到了，Transformer 的效果相对原生 RNN 和 CNN 来说有比较明显的优势，那么是否意味着我们可以放弃 RNN 和 CNN 了呢？事实倒也并未如此。我们聪明的科研人员想到了一个巧妙的改造方法，我把它叫做“寄居蟹”策略（就是上文说的“变性”的一种带有海洋文明气息的文雅说法）。什么意思呢？我们知道 Transformer Block 其实不是只有一个构件，而是由 multi-head attention/skip connection/Layer Norm/Feed forward network 等几个构件组成的小系统，如果我们把 RNN 或者 CNN 塞到 Transformer Block 里会发生什么事情呢？这就是寄居蟹策略的基本思路。

那么怎么把 RNN 和 CNN 塞到 Transformer Block 的肚子里，让它们背上重重的壳，从而能够实现寄居策略呢？

### 如何把Bi-RNN塞到Transformer Block肚子里？



### 如何把CNN塞到Transformer Block肚子里？






在。当然这只是说明一个大方向，具体的策略可能有些差异，但是基本思想八九不离十。

那么如果 RNN 和 CNN 采取这种寄居策略，效果如何呢？他们还爬的动吗？其实这种改造方法有奇效，能够极大提升 RNN 和 CNN 的效果。而且目前来看，RNN 或者 CNN 想要赶上 Transformer 的效果，可能还真只有这个办法了。

## RNN向Transformer整容过程及整容效果



Model	IWSLT EN→DE BLEU	WMT'17 EN→DE BLEU	WMT'17 EN→DE METEOR	WMT'17 LV→EN BLEU	WMT'17 LV→EN METEOR
Transformer	25.4 ± 0.1	27.6 ± 0.0	47.2 ± 0.1	18.5 ± 0.0	51.3 ± 0.1
RNMT	23.2 ± 0.2	25.5 ± 0.2	45.1 ± 0.1	-	-
- input feeding	23.1 ± 0.2	24.6 ± 0.1	43.8 ± 0.2	-	-
RNN	22.8 ± 0.2	23.8 ± 0.1	43.3 ± 0.1	15.2 ± 0.1	45.9 ± 0.1
+ mh	23.7 ± 0.4	24.4 ± 0.1	43.9 ± 0.1	16.0 ± 0.1	47.1 ± 0.1
+ pos	23.9 ± 0.2	24.1 ± 0.1	43.5 ± 0.2	-	-
+ norm	23.7 ± 0.1	24.0 ± 0.2	43.2 ± 0.1	15.2 ± 0.1	46.3 ± 0.2
+ multi-att-1h	24.5 ± 0.0	25.2 ± 0.1	44.9 ± 0.1	16.6 ± 0.2	49.1 ± 0.2
/ multi-att	24.4 ± 0.3	25.5 ± 0.0	45.3 ± 0.0	17.0 ± 0.2	49.4 ± 0.1
+ ff	25.1 ± 0.1	26.7 ± 0.1	46.4 ± 0.2	17.8 ± 0.1	50.5 ± 0.1


Table 3: Transforming an RNN into a Transformer style architecture. + shows the incrementally added variation. / denotes an alternative variation to which the subsequent + is relative to.

论文: How Much Attention Do You Need? A Granular Analysis of Neural Machine Translation Architectures



我们看看 RNN 寄居到 Transformer 后，效果是如何的。上图展示了对原生 RNN 不断进行整容手术，逐步加入 Transformer 的各个构件后的效果。我们从上面的逐步变身过程可以看到，原生 RNN 的效果在不断稳定提升。但是与土生土长的 Transformer 相比，性能仍然有差距。

## CNN向Transformer整容过程及整容效果



Model	IWSLT EN-DE BLEU	WMT'17 EN→DE BLEU	WMT'17 EN→DE METEOR	WMT'17 LV→EN BLEU	WMT'17 LV→EN METEOR
Transformer	25.4 ± 0.1	27.6 ± 0.0	47.2 ± 0.1	18.5 ± 0.0	51.3 ± 0.1
CNN GLU	24.3 ± 0.4	25.0 ± 0.3	44.4 ± 0.2	16.0 ± 0.5	47.4 ± 0.4
+ norm	24.1 ± 0.1	-	-	-	-
+ mh	24.2 ± 0.2	25.4 ± 0.1	44.8 ± 0.1	16.1 ± 0.1	47.6 ± 0.2
+ ff	25.3 ± 0.1	26.8 ± 0.1	46.0 ± 0.1	16.4 ± 0.2	47.9 ± 0.2
CNN ReLU	23.6 ± 0.3	23.9 ± 0.1	43.4 ± 0.1	15.4 ± 0.1	46.4 ± 0.3
+ norm	24.3 ± 0.1	24.3 ± 0.2	43.6 ± 0.1	16.0 ± 0.2	47.1 ± 0.5
+ mh	24.2 ± 0.2	24.9 ± 0.1	44.4 ± 0.1	16.1 ± 0.1	47.5 ± 0.2
+ ff	25.3 ± 0.3	26.9 ± 0.1	46.1 ± 0.0	16.4 ± 0.2	47.9 ± 0.1

Table 4: Transforming a CNN based model into a Transformer style architecture.

论文: How Much Attention Do You Need? A Granular Analysis of Neural Machine Translation Architectures



类似的，上图展示了对 CNN 进行不断改造的过程以及其对应效果。同样的，性能也有不同幅度的提升。但是也与土家 Transformer 性能存在一些差距。

这说明什么？我个人意见是：**这说明 Transformer 之所以能够效果这么好，不仅仅 multi-head attention 在发生作用，而是几乎所有构件都在共同发挥作用，是一个小小的系统工程。**

但是从上面结果看，变性版本 CNN 好像距离 Transformer 真身性能还是比不上，有些数据集差距甚至还很大，那么是否意味着这条路也未必走的通呢？Lightweight convolution 和 Dynamic convolutions 给人们带来一丝曙

身相当。那它做了什么能够达成这一点呢？也是寄居策略。就是用 Lightweight convolution 和 Dynamic convolutions 替换掉 Transformer 中的 Multi-head attention 模块，其它构件复用了 Transformer 的东西。和原生 CNN 的最主要区别是采用了 Depth-wise separable CNN 以及 softmax-normalization 等优化的 CNN 模型。

而这又说明了什么呢？我觉得这说明了一点：**RNN 和 CNN 的大的出路在于寄生到 Transformer Block 里**，这个原则没问题，看起来也是他俩的唯一出路。但是，要想效果足够好，在塞进去的 RNN 和 CNN 上值得花些功夫，需要一些新型的 RNN 和 CNN 模型，以此来配合 Transformer 的其它构件，共同发挥作用。如果走这条路，那么 RNN 和 CNN 翻身的一天也许还会到来。

尽管如此，我觉得 RNN 这条路仍然不好走，为什么呢，你要记得 RNN 并行计算能力差这个天生缺陷，即使把它塞到 Transformer Block 里，别说现在效果还不行，就算哪天真改出了一个效果好的，但是因为它的并行能力，会整体拖慢 Transformer 的运行效率。所以我综合判断 RNN 这条路将来也走不太通。

2019 来自未来的消息：总结

很多年前的小学语文课本上有句话，是这么说的：“张华考上了北京大学；李萍进了中等技术学校；我在百货公司当售货员：我们都有光明的前途”。我们小的时候看到这句话，对此深信不疑，但是走到 2019 的今天，估计已经没有父母愿意跟他们的孩子说这句话了，毕竟欺骗孩子是个挺不好的事情。如果套用这句话来说明 NLP 的三大特征抽取器的前途的话，应该是这样的：“Transformer 考上了北京大学；CNN 进了中等技术学校，希望有一天能够考研考进北京大学；RNN 在百货公司当售货员：我们都有**看似光明**的前途。”

我们把上文的所有证据都收集起来进行逻辑推理，可以模仿曹雪芹老师，分别给三位 NLP 界佳丽未来命运写一句判词。当然，再次声明，这是我个人判断。

## 进退维谷的 RNN

为什么说 RNN 进退维谷呢？有几个原因。

首先，如果靠原生的 RNN（包括 LSTM，GRU 以及引入 Attention 以及堆叠层次等各种你能想到的改进方法，可以一起上），目前很多实验已经证明效果比起 Transformer 有较大差距，现在看基本没有迎头赶上的可能，所以原生的 RNN 从效果来讲是处于明显劣势的。

其次，原生的 RNN 还有一个致命的问题：并行计算能力受限制太严重。想要大规模实用化应用？目前看希望渺茫。我们前面说过，决定了 RNN 本身的根本特质是：T 时刻隐层节点对前向输入及中间计算结果的序列依赖，因为它要线形序列收集前面的信息，这是 RNN 之所以是 RNN 的最主要特点。正是它的这个根本特质，使得 RNN 的并行计算能力想要获得根本解决基本陷入了一个两难的境地：要么仍然保持 RNN 序列依赖的根本特性，这样不论怎么改造，因为这个根本还在，所以 RNN 依旧是 RNN，所谓“我就是我，是不一样的烟火”，但是如果这样，那么其并行能力基本无法有力发挥，天花板很低；当然除此外，还有另外一条路可走，就是把这种序列依赖关系

你记忆中的 RNN 了。就是说，对 RNN 来说，要么就认命接受慢的事实，躲进小楼成一统，管他春夏与秋冬，仅仅是学术界用来发表论文的一种载体，不考虑大规模实用化的问题。要么就彻底改头换面变成另外一个人，如果真走到这一步，我想问的是：你被别人称为高效版本的 RNN，你自己好意思答应吗？这就是 RNN 面临的两难境地。

再次，假设我们再乐观一点，把对 RNN 的改造方向定位为将 RNN 改造成类似 Transformer 的结构这种思路算进来：无非就是在 Transformer 的 Block 里，把某些部件，当然最可行的是把 Multi-head self attention 部件换成 RNN。我们就算退一步讲，且将这种大幅结构改造的模型也算做是 RNN 模型吧。即使这样，已经把自己整形成长得很像 Transformer 了，RNN 依然面临上述原生 RNN 所面临的同样两个困境：一方面即使这种连变性削骨都上的大幅度整容版本的 RNN，效果虽然有明显提升，但是仍然比不过 Transformer；另外，一旦引入 RNN 构件，同样会触发 Transformer 结构的并行计算能力问题。所以，目前 Transformer 发动机看上去有点带不动 RNN 这个队友。

综合以上几个因素，我们可以看出，RNN 目前处于进退两难的地步，我觉得它被其它模型替换掉只是时间问题，而且好像留给它的时间不多了。当然，这是我个人意见。我说这番话的时候，你是不是又听到了水声？

我看到网上很多人还在推 RNN 说：其实还是 RNN 好用。我觉得这其实是一种错觉。之所以会产生这个错觉，原因来自两个方面：一方面是因为 RNN 发展历史长，所以有大量经过优化的 RNN 框架可用，这对技术选型选择困难症患者来说是个福音，因为你随手选一个知名度还可以的估计效果就不错，包括对一些数据集的前人摸索出的超参数或者调参经验；而 Transformer 因为历史太短，所以各种高效的语言版本的优秀框架还少，选择不多。另外，其实我们对 Transformer 为何有效目前还不是特别清楚，包括相关的各种数据集合上的调参经验公开的也少，所以会觉得调起来比较费劲。随着框架越来越多，以及经验分享越来越充分，这个不再会是问题。这是一方面。另外一方面，很多人反馈对于小数据集 RNN 更好用，这固然跟 Transformer 的参数数量比较多有关系，但是也不是没有解决办法，一种方式是把 Block 数目降低，减少参数量；第二种办法是引入 Bert 两阶段训练模型，那么对于小数据集来说会极大缓解效果问题。所以综合这两方面看，RNN 貌似在某些场合还有优势，但是这些所谓的优势是很脆弱的，这其实反映的是我们对 Transformer 整体经验不足的事实，随着经验越来越丰富，RNN 被 Transformer 取代基本不会有什么疑问。

## 一息尚存的 CNN

CNN 在 14 年左右在 NLP 界刚出道的时候，貌似跟 RNN 比起来表现并不算太好，算是落后生，但是用发展的眼光看，未来的处境反而看上去比 RNN 的状态还要占优一些。之所以造成这个奇怪现象，最主要的原因有两个：一个是因为 CNN 的天生自带的高并行计算能力，这对于延长它的生命力发挥了很大作用。这就决定了与 Transformer 比起来，它并不存在无法克服的困难，所以仍然有希望；第二，早期的 CNN 做不好 NLP 的一个很大原因是网络深度做不起来，随着不断借鉴图像处理的新型 CNN 模型的构造经验，以及一些深度网络的优化 trick，CNN 在 NLP 领域里的深度逐步能做起来了。而既然深度能做起来，那么本来 CNN 做 NLP 天然的一个缺陷：无法有效捕获长距离特征的问题，就得到了极大缓解。目前看可以靠堆深度或者结合 dilated CNN 来一定程度上解决这个问题，虽然还不够好，但是仍然是那句话，希望还在。

Transformer 的，典型的还是长距离特征捕获能力方面，原生的 CNN 版本模型仍然极为显著地弱于 RNN 和 Transformer，而这点在 NLP 界算是比较严重的缺陷。好，你可以说：那我们把 CNN 引到 Transformer 结构里，比如代替掉 Self attention，这样和 Transformer 还有一战吧？嗯，是的，目前看貌似只有这条路是能走的通的，引入 depth separate CNN 可以达到和 Transformer 接近的效果。但是，我想问的是：你确认长成这样的 CNN，就是把 CNN 塞到 Transformer Block 的肚子里，你确认它的亲朋好友还能认出它吗？

当然，我之所以写 CNN 一息尚存，是因为我觉得把 CNN 塞到 Transformer 肚子里这种方案，对于篇章级别的 NLP 任务来说，跟采取 self attention 作为发动机的 Transformer 方案对比起来，是具有极大优势的领域，也是适合它的战场，后面我估计会出现一些这方面的论文。为什么这么讲？原因下面会说。

## 稳操胜券的 transformer

我们在分析未来 NLP 的三大特征抽取器哪个会胜出，我认为，起码根据目前的信息来看，其实 Transformer 在很多战场已经赢了，在这些场地，它未来还会继续赢。为什么呢？上面不是说了吗，原生的 RNN 和 CNN，总有一些方面显著弱于 Transformer（并行计算能力或者效果，或者两者同时都比 Transformer 弱）。那么他们未来的希望，目前大家都寄托在把 RNN 和 CNN 寄生在 Transformer Block 里。RNN 不用说了，上面说过它的进退维艰的现状。单说 CNN 吧，还是上一部分的那句话，我想问的是：你确认长成这样的 CNN，就是把 CNN 塞到 Transformer Block 的肚子里，你确认它的亲朋还能认出它吗？

目前能够和 Transformer 一战的 CNN 模型，基本都已经长成 Transformer 的模样了。而这又说明了什么呢？难道这是 CNN 要能战胜 Transformer 的迹象吗？这是一道留给您的思考题和辩论题。当然，我不参加辩论。

Transformer 作为一个新模型，并不是完美无缺的。它也有明显的缺点：首先，对于长输入的任务，典型的比如篇章级别的任务（例如文本摘要），因为任务的输入太长，Transformer 会有巨大的计算复杂度，导致速度会急剧变慢。所以估计短期内这些领地还能是 RNN 或者长成 Transformer 模样的 CNN 的天下（其实目前他俩这块做得也不好），也是目前看两者的希望所在，尤其是 CNN 模型，希望更大一些。但是是否 Transformer 针对长输入就束手无策，没有解决办法呢？我觉得其实并不是，比如拍脑袋一想，就能想到一些方法，虽然看上去有点丑陋。比如可以把长输入切断分成 K 份，强制把长输入切短，再套上 Transformer 作为特征抽取器，高层可以用 RNN 或者另外一层 Transformer 来接力，形成 Transformer 的层级结构，这样可以把  $n^2$  的计算量极大减少。当然，这个方案不优雅，这个我承认。但是我提示你一下：这个方向是个值得投入精力的好方向，你留意一下我这句话，也许有意想不到的收获。（注：上面这段话是我之前早已写好的，结果今天（1 月 12 日）看见媒体号在炒作：“Transforme-XL，速度提升 1800 倍”云云。看了新闻，我找来 Transformer-XL 论文看了一下，发现**它解决的就是输入特别长的问题**，方法呢其实大思路和上面说的内容差不太多。说这么多的意思是：我本不想删除上面内容，为避免发出来后，那位“爱挑刺”同学说我拷贝别人思路没引用。我决定还是不改上面的说法，因为这个点子实在是太容易想到的点子，我相信你也能想到。）除了这个缺点，Transformer 整体结构确实显得复杂了一些，如何更深刻认识它的作用机理，然后进一步简化它，这也是一个好的探索方向，这句话也请留意。还



为 Transformer 通过 Self attention 使得远距离特征直接发生关系，按理说距离不应该成为它的问题，但是效果竟然不如 RNN，这背后的原因是什么呢？这也是很有价值的一个探索点。

我预感到我可能又讲多了，能看到最后不容易，上面几段话算是送给有耐心的同学的礼物，其它不多讲了，就此别过，请忽略你听到的哗哗的水声。

以上。

### 作者简介：

**张俊林**，中国中文信息学会理事，目前在新浪微博 AI Lab 担任资深算法专家。在此之前，张俊林曾经在阿里巴巴任资深技术专家，以及在百度和用友担任技术经理及技术总监等职务。同时他是技术书籍《这就是搜索引擎：核心技术详解》（该书荣获全国第十二届出版优秀图书奖）、《大数据日知录：架构与算法》的作者。

参考链接：

[1]从 Word Embedding 到 Bert 模型——自然语言处理中的预训练技术发展史<https://zhuanlan.zhihu.com/p/49271699>

# 请手动点个赞吧

