「课代表来了」跟李沐读论文之——BERT

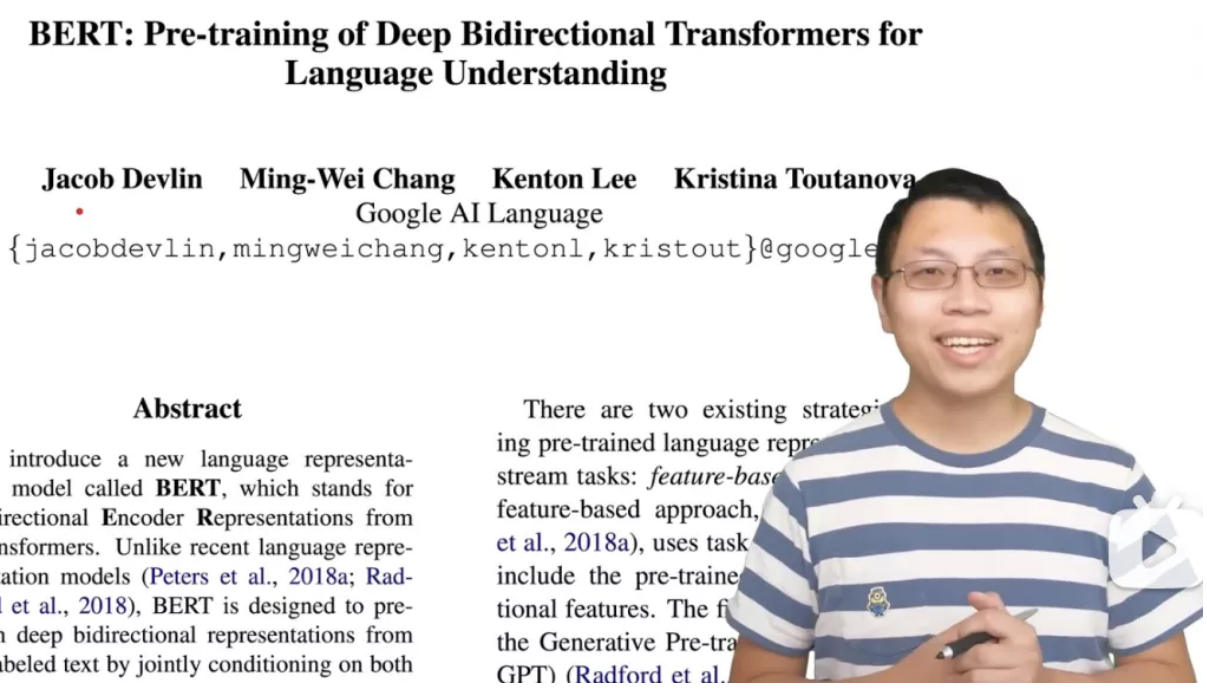
原创 郭必扬 SimpleAI 2021-12-25 16:39

收录于话题
#Nice NLP Notes 20 #不一样的论文解读 17

「课代表来了」跟李沐读论文之——BERT

> 墙裂推荐沐神在B站上开的一系列公开课，尤其是最近出的“论文精读”系列，真的讲得太好了。可能很多经典论文咱们之前也读过，但是听李沐老师再讲一遍，我们又可以收获很多新知识，尤其是李沐老师结合自己丰富的经验做出的很多精彩评论。
>
> 上次我分享了沐神讲解Transformer的相关笔记：
> 「课代表来了」跟李沐读论文之——Transformer
> 本期我们继续读基于Transformer的大作——BERT.
>
> - B站视频地址：https://b23.tv/XbDJENb

标题/作者



BERT：Pre-trainingof Deep Bidirectional Transformers for Language Understanding

> Comments by Li Mu: 作者是Google AI Language团队的几位小哥，据说是一作当时突然有了一个idea，然后花了几星期去写代码、跑实验，发现效果特别好，最后就成了这篇文章。可能从idea到成型就几个月。

摘要

BERT这个名字是从 Bidirectional Encoder Representations from Transformers得来的，猜测是为了凑出Bert这个词，因为前面的著名工作ELMo就是美国家喻户晓的动画片芝麻街中的主角之一。在BERT出来之后，后面的研究者就开始想方设法地把芝麻街中的重要人物都用了个遍。

主要对比对象是ELMo和GPT。最大的作用就是我们可以只是使用预训练好的BERT模型，添加一个任务相关的输出层，就可以在下游任务上达到SOTA水平，极大地降低了NLP任务的门槛。而前面的ELMo则需要对模型进行修改。

最后讲了BERT的效果非常好，即列出了在benchmark上的绝对精度，还列出了相对精度，在11个NLP任务上都达到了SOTA。

> Comments by Li Mu: 在摘要中直接进行跟前人工作的对比，这种写法是很有意思的（在你的模型很大程度上基于或者对比前人工作的话，是可以且应该直接在最开始进行介绍的）。在说明模型效果的时候，绝对精度和相对精度都是需要的，前者让我们知道在公共数据集上的绝对实力（尤其对于小同行），后者则给读者（尤其是更广泛的读者甚至外行）一个关于模型效果的直观的感受。

Intro

> BERT不是第一个做NLP预训练的，而是第一次让这个方法出圈了。

从intro部分我们可以知道，language model pre-training其实之前多年前就有了。

使用预训练模型来帮助下游任务的时候，现有的做法有两种：

- feature-based方式，例如ELMo，就是把预训练的表示作为额外的特征，加入到特定任务的模型中；
- fine-tuning方式，例如GPT，尽可能少的引入任务相关的参数，而主要是在预训练好的参数上面进行微调；

前面的ELMo和GPT的方法，都是使用单向的语言模型来学习通用的语言表示。例如在GPT中，作者设计了一种从左到右的架构，在Transformer的self-attention中每个token只能attend到前面的token。在更早的ELMo中，由于使用的是RNN的架构，更加是单向的语言模型。这一点严重限制了作为预训练使用的语言表示能力。比如在做NER的时候，我们都是可以看到上下文的。

BERT主要就是为了解决这种单向的限制，设计了一种"mask language modeling"(MLM)的方式，来进行双向的语言模型预训练。这一点是借鉴了完形填空（cloze）任务。另外，作者还设计了一个叫"next sentence prediction"(NSP)的任务来预训练，即判断两个句子是否是相邻的，还是随机的，这样可以学习句子层面的信息。

下图展示了BERT跟前面工作的结构上的对比（在最新版的论文中，这个图是在附录部分，在最初的版本中这则是文章第一个图）：

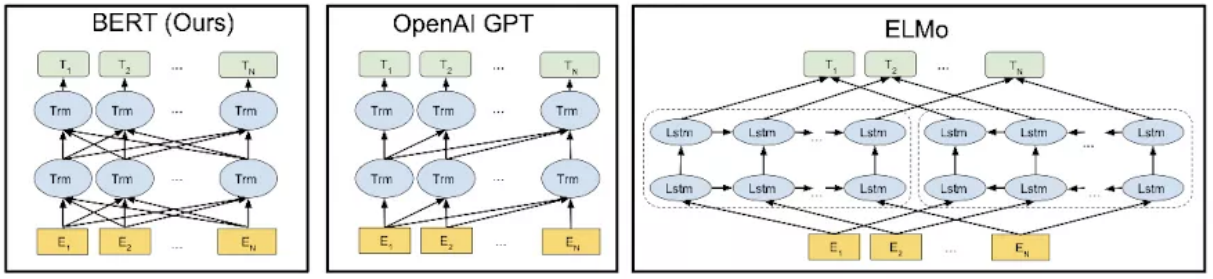


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

贡献：

- 展现了双向语言模型的作用；
- 展示了预训练表示对于降低下游任务工作量的巨大作用，并且是首个在一大把NLP任务上都取得SOTA的预训练-微调模式的表示模型；
- 代码和预训练模型都公开了。

结论

使用非监督的预训练是非常好的，对于低资源场景的任务尤其有益。主要贡献来自于使用了双向的语言模型。

相关工作

1. 无监督的feature-based pre-training，代表作ELMo
2. 无监督的fine-tuning pre-training，代表作GPT
3. 有监督的transfer learning，代表作就是CV中那些进行Imagenet进行transfer learning，这在NLP中却用的不是很多。主要是由于高质量的通用的有标签文本数据相对较少。

BERT模型设计

两个步骤：pre-training 和 fine-tuning

在pre-training阶段使用无标签的数据，在fine-tuning阶段，BERT模型使用前面预训练的权重来初始化，然后使用下游任务有标签的数据进行微调。

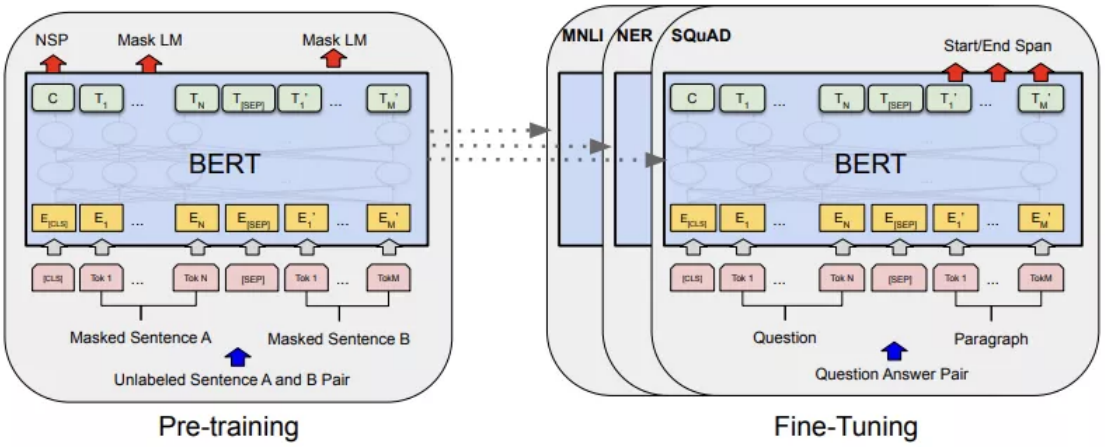


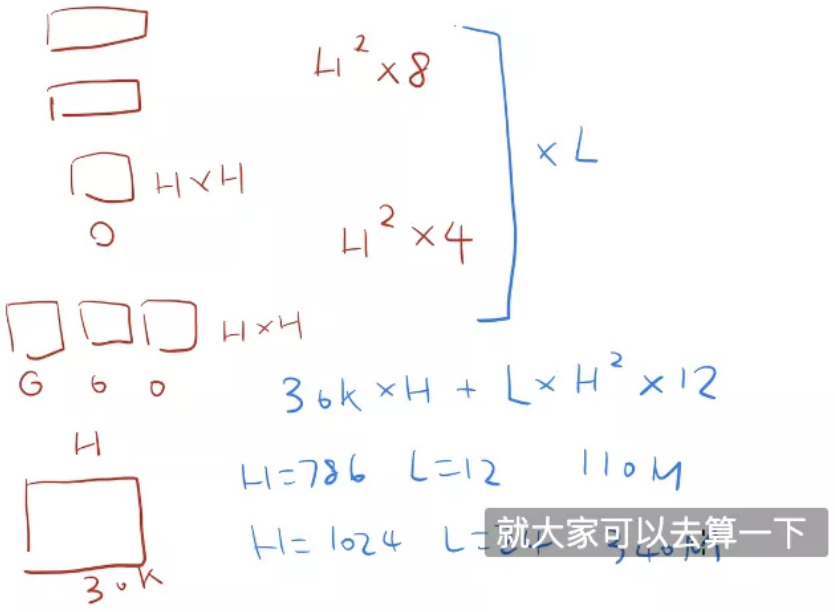
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

两阶段

模型结构和参数

模型结构是直接使用原始的Transformer。使用了两种不同架构：$BERT_{BASE}$（L=12, H=768, A=12，总参数量110M）和$BERT_{LARGE}$（L=24, H=1024, A=16，总参数量340M），其中L是Transformer的层数/block数，H是hidden size，A是头数。

后面沐神也讲解了参数量是咋算的（这部分真是太棒了）：



参数量的计算

参数的来源主要是Transformer中的embedding层、multi-head attention的投影矩阵、MLP层：

- embedding层：词汇量为V，词向量维度为H，所以这部分参数里为 $V \times H$；
- multi-head：分别是使用了A个小投影矩阵来讲原本的H维向量给降维成多个低维向量，但向量维度之和还是H，所以多个小投影矩阵合并起来就是一个 $H \times H$矩阵，然后因为self-attention会分成QKV，所以这里有3个H^2；除此之外，在经过multi-head分开后又会合并成一个H的向量，会再经过一个投影矩阵，也是H^2，所以这部分总共有$4H^2$；
- MLP层：Transformer中使用的是一个由两个全连接层构成的FNN，第一个全连接层会将维度放大4倍，第二个则降维到原始的H，因此，这里的参数量为$H \times 4H + 4H \times H = 8H^2$.
- 上面的multi-head和MLP，都属于一个Transformer block，而我们会使用L个blocks。

因此，总体参数量=$VH + 12LH^2$.

这么算下来，差不多$BERT_{BASE}$参数量是108M，$BERT_{LARGE}$是330M。（跟原文说的接近的，但相差的部分在哪儿呢？）

输入的表示

为了适应不同的下游任务，BERT的输入既可以是**单个句子**，也可以是一个句子对（例如<Question, Answer>）。

在输入token方面，使用WordPiece的embedding方式，也是sub-word tokenization的方式的一种，我们看到的那些前前面带有"##"的词就代表这是被wordpiese给切开的子词。这样可以减少词汇量，最终词汇量是30000。

每个序列的开头的token，都是一个特殊的分类token——[CLS]，这个token对应的最后一次的hidden state会被用来作为分类任务中的整个序列的表示。对于非分类任务，这个向量是被忽略的。

处理句子对时，对模型来说还是一个序列，只不过两个句子中间用一个特殊的[SEP] token进行了连接。两个句子分别还配有可学习的segment embedding；而对于仅有一个句子的输入，我们就只使用一个segment embedding.

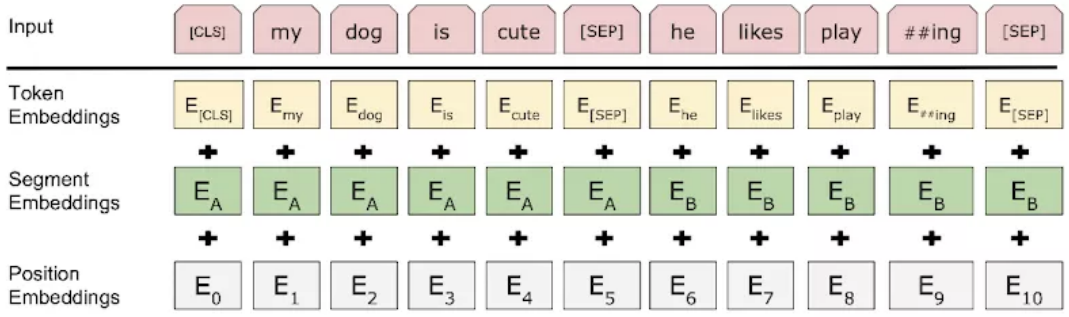


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

输入的embedding

BERT的预训练

Masked LM

随机地把原文中的15%的token给遮盖住，即用一个 [MASK] token来替换原来的词。然后把mask之后的文本输入到模型中，让模型去预测这些被mask掉的词。这样就实现了双向的语言模型。

但这样做会导致预训练和微调阶段的不一致性：预训练的时候输入都是带有 [MASK] token的，而这个token在微调阶段是看不到的，这样自然会影响微调时的效果。为了缓解这个问题，作者使用了如下的操作：

- 当挑到某个词去mask的时候，80%的概率会真的被替换成[MASK]，10%的概率会被替换成一个随机的真实token，还有10%的概率不进行任何操作。

这种做法，说实话还是挺费解的，让人感觉也不一定有多大效果，但作者说这样可以缓解一点就缓解一点吧。（实际上现在也有很多研究在解决这个问题，这部分后面补充...）

另外一个问题在于MLM在这里只使用了15%的mask比例，这会让模型需要训练更久才能收敛。但好在最终的效果非常好，所以也值了。（不知道如果使用更大的比例会怎么样？）

Next Sentence Prediction

很多的下游任务，比如QA（问答）和NLI（自然语言推理）任务，都需要模型能够理解句子之间的关系，而这种关系难以被MLM所学习到。因此作者设计了一个输入句子对的二分类的NSP任务：

- 50%的样本中，句子A和句子B是在真实文本中连续的句子，标签是 IsNext；
- 50%的样本中，B跟A不是连续的，而是随机挑选的句子，标签是 NotNext.

虽然这个任务看起来非常简单，而且作者说在预训练时这个任务可以达到97%以上的准确率，但后面的实验证明确实对QA和NLI任务有很大的帮助。

注意到pre-training的那个图，**在NSP任务中，我们使用的是[CLS] token对应的hidden state来训练**的，即我们使用这个[CLS]来代表我整个句子对的表示，用它来进行二分类任务。

BERT的微调

对于sequence-level的任务，我们可以直接使用CLS的向量作为sequence的表示，然后后面加一个简单的softmax层来进行训练；对于span-level或者token-level的任务，也只用稍微修改一下跟任务相关的输出层即可。

另外，微调跟预训练时的差别还在BERT模型训练的一些超参数上，比如learning rate，batch size等等。例如在pre-training阶段batch size=256，而在fine-tuning阶段作者推荐使用16或者32.

具体如何针对下游任务进行微调

GLUE

GLUE数据集一般都是sequence-level的任务，主要都是分类，既有单句子的，也有句子对的任务。这种就是直接用CLS配合一个softmax来跑即可。

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

GLUE

SQuAD （问答）

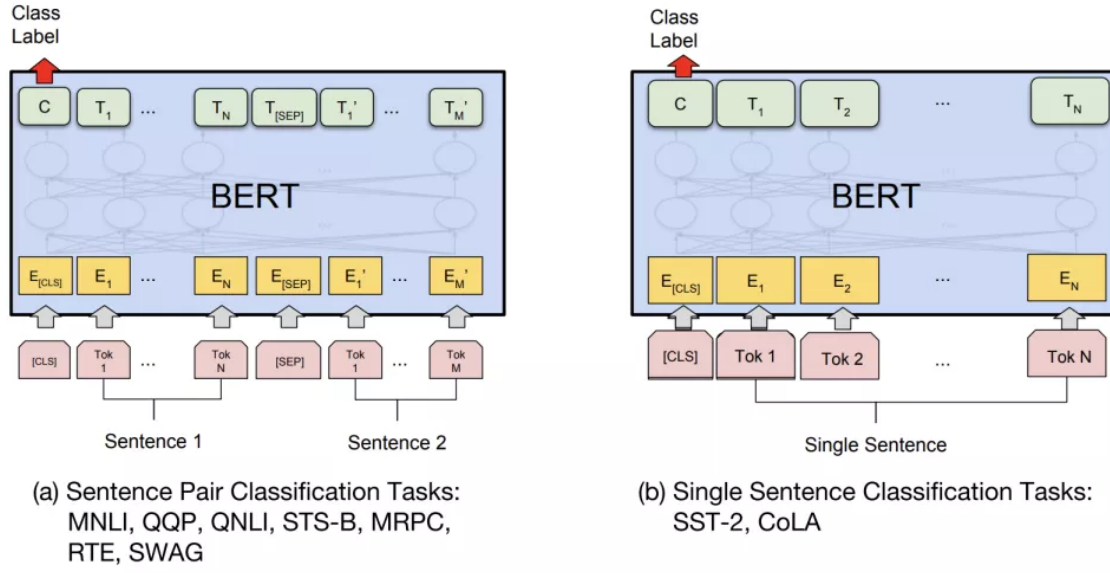
给定一个问题，从一段话中找出答案所在的片段。所以问题转化为对每个token判断是否是答案的开头或结尾。具体细节由于我不做问答，所以详情见论文吧。

SWAG

这是另外一个句子对推理任务，其实跟NSP任务比较像，所以这里也不多介绍了。

以上的任务，作者也花了几个示意图告诉我们如何做任务相关的模型调整：

对于分类任务：

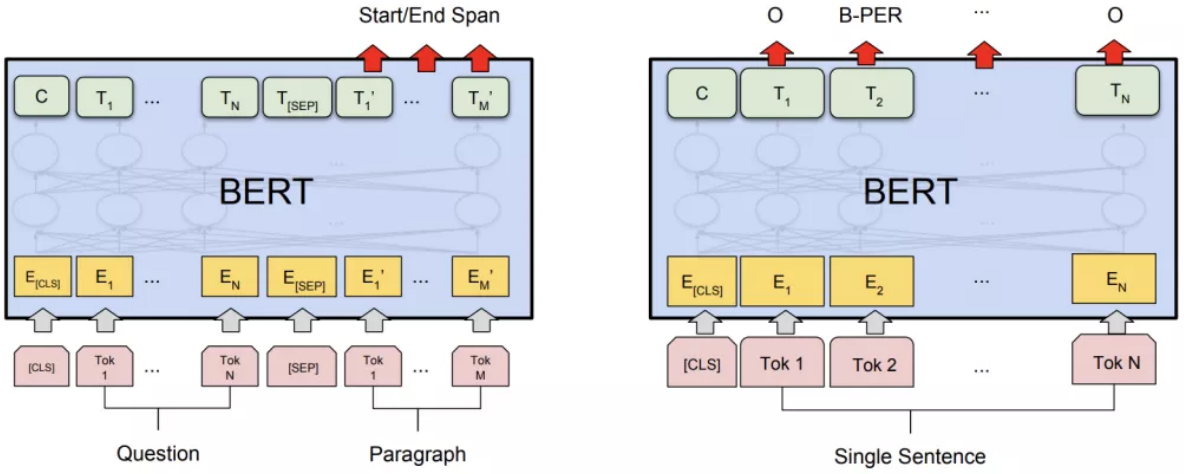


(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

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

sequence classification

对于token标注：



(c) Question Answering Tasks: SQuAD v1.1

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

token classification

所以总体上看，我们只需要做微小的调整，就可以应对各种下游任务。

消融实验

作者继续做了一些消融实验，来看看NSP、双向语言模型等的作用。

Tasks	Dev Set				
	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$BERT_{BASE}$	84.4	88.4	86.7	92.7	88.5

No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

ablation study

从上图可看到，对于各种任务来说，NSP预训练还是有帮助的，把NSP去掉的话，在很多任务上效果都会降低（但好像也没有那么明显哈）；然后如果只使用Left-to-right（LTR）的语言模型的话，效果会进一步降低，这个降低就比较明显了。

总之，这个实验论证了BERT的几个关键点都是重要的。

预训练模型的大小

BERT这个论文，证明了使用一个很大的预训练模型，可以极大地提高下游任务的表现。

从现在的眼光看，BERT也不算大了，例如GPT3的大小就是BERT的1000倍（千亿），现在甚至万亿级别的模型都在不断出现。

只使用BERT作为特征抽取器的效果

作者还探究了一下用feature-based的方式来利用BERT预训练的表示的效果，下表是在一个NER任务上的结果：

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

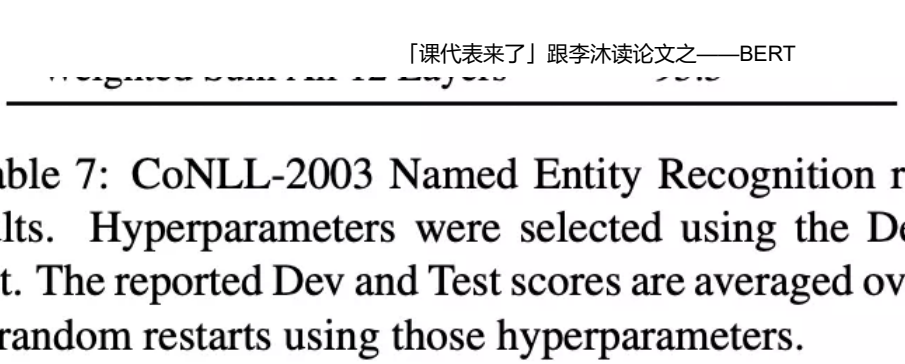


Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

使用bert作为静态特征提取器

总体结论是，如果使用BERT的话，还是尽量用fine-tuning的方式效果会更好。但是从图中看，将最后几层的hidden states拼接起来作为特征，效果在NER上也不错。

总结

作者对这个工作最大的贡献总结为BERT的双向性，然而双向语言模型和单向模型，其实只是不同的策略，使用双向的方式进行预训练，那自然在某些任务上会有些损失，比如在机器翻译、摘要生成等任务，可能BERT就没有GPT那么优秀。这其实就是有得必有失。

> Comments by Li Mu: 对于写文章，我们最好是重点突出一个卖点，不要太多了。例如这篇文章就是突出”双向语言模型“。

最后，沐神提出了一个灵魂拷问：其实BERT从整个流程上，跟GPT的工作是很类似，都是先预训练，在进行下游任务微调。为什么BERT更晚，却更出圈，现在名气和影响力是远远大于GPT的？

这个问题，在B站评论区大家也在讨论，大家的一个观点是：因为BERT做了更好的开源，把代码、预训练模型都直接公开出来了。这让广大的研究者可以直接拿来使用，体验预训练的威力，因此很快就可以传播开。

这一点告诉我们，**开源、可复现、方便后续研究者使用，对一个研究工作有很大的推动作用**。现在很多的论文，发表在顶会顶刊上，却不公开代码，或者代码公开了却写的稀烂，没有任何的文档来帮助人们复现，这必然极大影响论文最终的影响力，甚至影响作者的声誉。

“做真正有影响力、有价值的研究，而不是为了水水文章、增加自己的publications。”这句最简单、最朴素的科研工作者都应该有的价值观，在当下的环境下，尤其是国内这种长期以来的追求论文数量的价值观、高校不合理的考核机制、各大技术厂商的极端内卷等影响下，显得无比珍贵。

其他原创好文

Huggingface🤗NLP笔记8：使用PyTorch来微调模型「初级教程完结撒花ヽ(°▽°)ノ」
何时能懂你的心——图卷积神经网络（GCN）
【DL笔记6】从此明白了卷积神经网络（CNN）
Hello NLP(2)——关于word2vec你想知道的一切
整理了12个小时，只为让你可以20分钟搞懂seq2seq
博一结束后的一些反思

收录于话题 #Nice NLP Notes 20

上一篇 · 「课代表来了」跟李沐读论文之——Transformer