ACL2020 | FastBERT：放飞BERT的推理速度

原创 rumor酱 夕小瑶的卖萌屋 4月9日

来自专辑
卖萌屋@自然语言处理

一只小狐狸带你解锁 **炼丹术&NLP** 秘籍

FastBERT

自从BERT问世以来，大多数NLP任务的效果都有了一次质的飞跃。BERT Large在GLUE test上甚至提升了7个点之多。但BERT同时也开启了模型的“做大做深”之路，普通玩家根本训不起，高端玩家虽然训得起但也不一定用得起。

所以BERT之后的发展也比较清晰，一部分壕大佬们继续搞预训练提升效果，当你对BERT Large望而却步的时候，又出了GPT2，又双出了威震天Megatron-LM，又双叒出了T5，又双叒叕出了DeepSpeed。。。每次都是照着一个数量级去加，剩下的人只能默默观望，翻翻《显存不够，如何训练大型神经网络？》看哪个trick可以用上。

另一部分大佬着力于给BERT瘦身提升速度。比如剪枝，剪掉多余的连接、多余的注意力头、甚至LayerDrop[1]直接砍掉一半Transformer层；再比如量化，把FP32改成FP16或者INT8；还有蒸馏，用一个学生模型来学习大模型的知识，不仅要学logits，还要学attention score。。。

然而，大部分减肥方法都会带来精度的下降。剪枝会直接降低模型的拟合能力，量化虽然有提升但也有瓶颈，蒸馏的不确定性最大，很难预知你的BERT教出来怎样的学生。

小学生 绝对小学生

但！是！

昨天刷到了一篇让我眼前一亮的文章《FastBERT: a Self-distilling BERT with Adaptive Inference Time》[2]，是北大+腾讯+北师大

的ACL2020。作者提出了一种新的inference速度提升方式，相比单纯的student蒸馏有更高的确定性，且可以自行权衡效果与速度，简单实用。

后台回复【0409】获取论文PDF噢～

FastBERT

模型结构

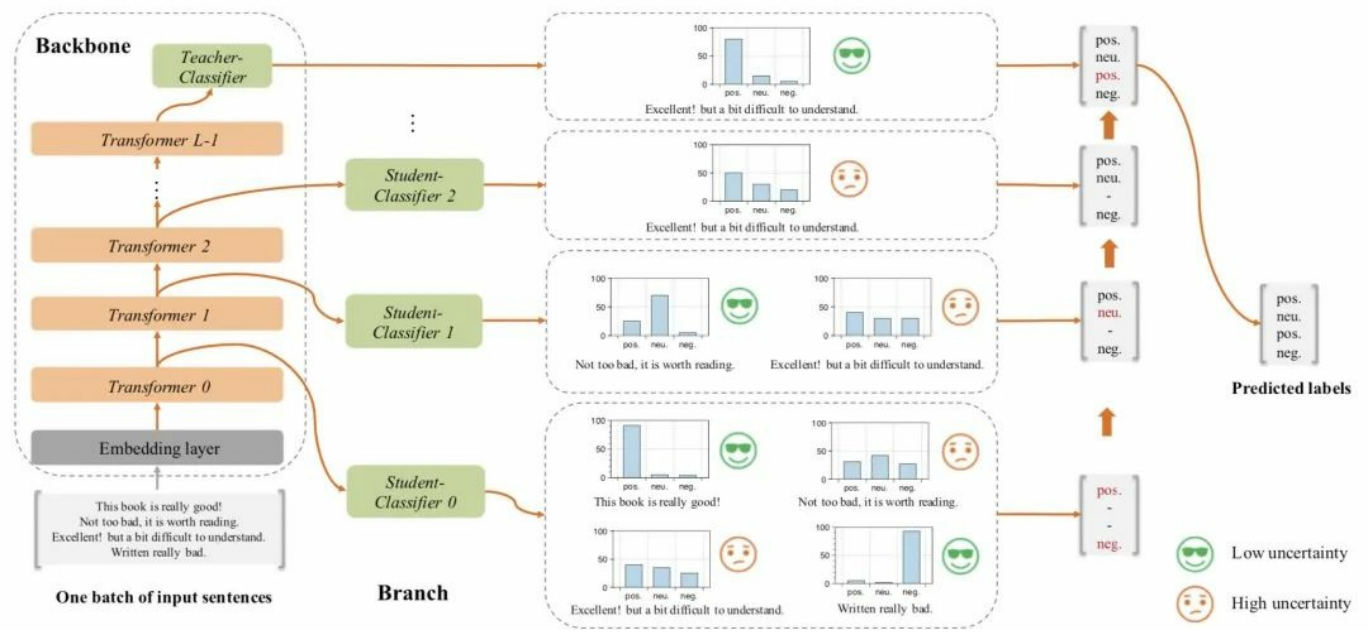
FastBERT的创新点很容易理解，就是在每层Transformer后都去预测样本标签，如果某样本预测结果的置信度很高，就不用继续计算了。论文把这个逻辑称为**样本自适应机制（Sample-wise adaptive mechanism）**，就是自适应调整每个样本的计算量，容易的样本通过一两层就可以预测出来，较难的样本则需要走完全程。

那么问题来了，用什么去预测中间层的结果呢？作者的解决方案是给每层后面接一个分类器，毕竟分类器比Transformer需要的成本小多了：

Operation	Sub-operation	FLOPs	Total FLOPs
Transformer	Self-attention ($768 \rightarrow 768$)	603.0M	1809.9M
	Feedforward ($768 \rightarrow 3072 \rightarrow 768$)	1207.9M	
Classifier	Fully-connect ($768 \rightarrow 128$)	25.1M	46.1M
	Self-attention ($128 \rightarrow 128$)	16.8M	
	Fully-connect ($128 \rightarrow 128$)	4.2M	
	Fully-connect ($128 \rightarrow N$)	-	

注：FLOPs (floating point operations)是Tensorflow中提供的浮点数计算量统计

于是模型的整体结构就呼之欲出了：



作者将原BERT模型称为主干（Backbone），每个分类器称为分支（Branch）。

要注意的是，这里的分支Classifier都是最后一层的分类器蒸馏来的，作者将这称为**自蒸馏（Self-distillation）**。就是在预训练和精调阶段都只更新主干参数，**精调完后freeze主干参数，用分支分类器（图中的student）蒸馏主干分类器（图中的teacher）的概率分布**。

之所以叫自蒸馏，是因为之前的蒸馏都是用两个模型去做，一个模型学习另一个模型的知识，而FastBERT是自己（分支）蒸馏自己(主干)的知识。值得注意的是，蒸馏时需要freeze主干部分，保证pretrain和finetune阶段学习的知识不被影响，仅用brach 来尽可能的拟合teacher的分布。

那为什么不直接用标注数据训分支分类器呢？因为直接训效果不好呗（摊手～下面是作者在消融实验给出的结果：

Config.	Book review		Yelp.P	
	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)
FastBERT				
speed=0.2	86.98	9725M (2.23x)	95.90	52783M (4.12x)
speed=0.7	85.69	3621M (6.01x)	94.67	2757M (7.90x)
FastBERT without self-distillation				
speed=0.2	86.22	9921M (2.19x)	95.55	4173M (5.22x)
speed=0.7	85.02	4282M (5.08x)	94.54	2371M (9.18x)

可以看到，非蒸馏的结果没有蒸馏要好。个人认为是合理的，因为这两种方式在精调阶段的目标不一样。非自蒸馏是在精调阶段训练所有分类器，**目标函数有所改变**，迫使前几层编码器抽取更多的任务feature。但BERT强大的能力与网络深度的相关性很大，所以过早地判断不一定准确，致使效果下降。

同时，使用自蒸馏还有一点重要的好处，就是**不再依赖于标注数据**。蒸馏的效果可以通过源源不断的无标签数据来提升。

模型训练与推理

了解模型结构之后，训练与推理也就很自然了。只比普通的BERT模型多了自蒸馏这个步骤：

1. **Pre-training**：同BERT系模型是一样的，网上那么多开源的模型也可以随意拿来～
2. **Fine-tuning for Backbone**：主干精调，也就是给BERT加上分类器，用任务数据训练，这里也用不到分支分类器，可以尽情地优化
3. **Self-distillation for branch**：分支自蒸馏，用无标签任务数据就可以，将主干分类器预测的概率分布蒸馏给分支分类器。这里使用KL散度衡量分布距离，loss是所有分支分类器与主干分类器的KL散度之和
4. **Adaptive inference**：自适应推理，及根据分支分类器的结果对样本进行层层过滤，简单的直接给结果，困难的继续预测。这里作者定义了新的不确定性指标，用预测结果的熵来衡量，熵越大则不确定性越大：

效果

对于每层分类结果，作者用“Speed”代表不确定性的阈值，和推理速度是正比关系。因为阈值越小 => 不确定性越小 => 过滤的样本越少 => 推理速度越慢。

模型最终在12个数据集（6个中文的和6个英文的）上的表现还是很好的：

Dataset/ Model	ChnSentiCorp		Book review		Shopping review		LCQMC		Weibo		THUCNews	
	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)
BERT	95.25	21785M (1.00x)	86.88	21785M (1.00x)	96.84	21785M (1.00x)	86.68	21785M (1.00x)	97.69	21785M (1.00x)	96.71	21785M (1.00x)
DistilBERT (6 layers)	88.58	10918M (2.00x)	83.31	10918M (2.00x)	95.40	10918M (2.00x)	84.12	10918M (2.00x)	97.69	10918M (2.00x)	95.54	10918M (2.00x)
DistilBERT (3 layers)	87.33	5428M (4.01x)	81.17	5428M (4.01x)	94.84	5428M (4.01x)	84.07	5428M (4.01x)	97.58	5428M (4.01x)	95.14	5428M (4.01x)
DistilBERT (1 layers)	81.33	1858M (11.72x)	77.40	1858M (11.72x)	91.35	1858M (11.72x)	71.34	1858M (11.72x)	96.90	1858M (11.72x)	91.13	1858M (11.72x)
FastBERT (speed=0.1)	95.25	10741M (2.02x)	86.88	13613M (1.60x)	96.79	4885M (4.45x)	86.59	12930M (1.68x)	97.71	3691M (5.90x)	96.71	3595M (6.05x)
FastBERT (speed=0.5)	92.00	3191M (6.82x)	86.64	5170M (4.21x)	96.42	2517M (8.65x)	84.05	6352M (3.42x)	97.72	3341M (6.51x)	95.64	1979M (11.00x)
FastBERT (speed=0.8)	89.75	2315M (9.40x)	85.14	3012M (7.23x)	95.72	2087M (10.04x)	77.45	3310M (6.57x)	97.69	1982M (10.09x)	94.97	1854M (11.74x)

Dataset/ Model	Ag.news		Amz.F		Dbpedia		Yahoo		Yelp.F		Yelp.P	
	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)	Acc.	FLOPs (speedup)
BERT	94.47	21785M (1.00x)	65.50	21785M (1.00x)	99.31	21785M (1.00x)	77.36	21785M (1.00x)	65.93	21785M (1.00x)	96.04	21785M (1.00x)
DistilBERT (6 layers)	94.64	10872M (2.00x)	64.05	10872M (2.00x)	99.10	10872M (2.00x)	76.73	10872M (2.00x)	64.25	10872M (2.00x)	95.31	10872M (2.00x)
DistilBERT (3 layers)	93.98	5436M (4.00x)	63.84	5436M (4.00x)	99.05	5436M (4.00x)	76.56	5436M (4.00x)	63.50	5436M (4.00x)	93.23	5436M (4.00x)
DistilBERT (1 layers)	92.88	1816M (12.00x)	59.48	1816M (12.00x)	98.95	1816M (12.00x)	74.93	1816M (12.00x)	58.59	1816M (12.00x)	91.59	1816M (12.00x)
FastBERT (speed=0.1)	94.38	6013M (3.62x)	65.50	21005M (1.03x)	99.28	2060M (10.57x)	77.37	16172M (1.30x)	65.93	20659M (1.05x)	95.99	6668M (3.26x)
FastBERT (speed=0.5)	93.14	2108M (10.33x)	64.64	10047M (2.16x)	99.05	1854M (11.74x)	76.57	4852M (4.48x)	64.73	9827M (2.21x)	95.32	3456M (6.30x)
FastBERT (speed=0.8)	92.53	1858M (11.72x)	61.70	2356M (9.24x)	99.04	1853M (11.75x)	75.05	1965M (11.08x)	60.66	2602M (8.37x)	94.31	2460M (8.85x)

可以看到，在Speed=0.2时速度可以提升1-10倍，且精度下降全部在0.11个点之内，甚至部分任务上还有细微提升。相比之下HuggingFace的DistillBERT的波动就比较剧烈了，6层模型速度只提升2倍，但精度下降最高会达到7个点。

总结

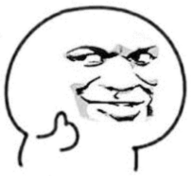
FastBERT是一个在工程上十分实用的模型，通过提前输出简单样本的预测结果，减少模型的计算负担，从而提高推理速度。虽然每层都多了一个分类器，但分类器的计算量也比Transformer小了两个数量级，对速度影响较小。后续的分支自蒸馏也设计的比较巧妙，可以利用无监督数据不断提升分支分类器的效果。

另外，**Sample-wise adaptive mechanism和Self-distillation这两个idea也是在本文首次提出**，相信会起到抛玉引玉的作用，衍

生更多此类工作。论文本身也还有一些想象空间，比如分别优化每层分类器，因为在主干被freeze的情况下各个分支是独立的；或者自蒸馏unfreeze主干，再加上数据自蒸馏一把，说不定还会有性能提升。

值得一提的是，本文的一作刘伟杰（北大）正是K-BERT[3]的作者，那也是一篇我很喜欢的文章，把知识融入到BERT的方式比较优雅，真心期待作者有更多的idea～

最后再回来夸一下，FastBERT着实很实用，而且完全不会影响到手头调好的BERT，只需要蒸馏几个浅层分类器，再把判断机制加上就可以了。而且比起不太稳定的蒸馏来说放在线上也更有底，稳稳的幸福。



兄弟，稳！

唯一的遗憾是源码要在文章发表时才会放出来，一起去催更吧～

https://github.com/autoliuweijie/FastBERT

后台回复【0409】获取论文PDFᵒᵒ～

参考文献

[1] Reducing Transformer Depth on Demand with Structured Dropout: https://arxiv.org/abs/1909.11556

[2] FastBERT: a Self-distilling BERT with Adaptive Inference Time: https://arxiv.org/abs/2004.02178

[3] K-BERT: Enabling Language Representation with Knowledge Graph: https://arxiv.org/abs/1909.07606

可能喜欢

- 卖萌屋2020 Q1季度大会
- LayerNorm是Transformer的最优解吗?
- 如何优雅地编码文本中的位置信息？三种positioanl encoding方法简述
- 详解ERNIE-Baidu进化史及应用场景
- 深入解析GBDT二分类算法（附代码实现）
- 在大厂和小厂做算法有什么不同?
- 别再喊我调参侠！夕小瑶“科学炼丹”手册了解一下