别再蒸馏3层BERT了！变矮又能变瘦的DynaBERT了解一下

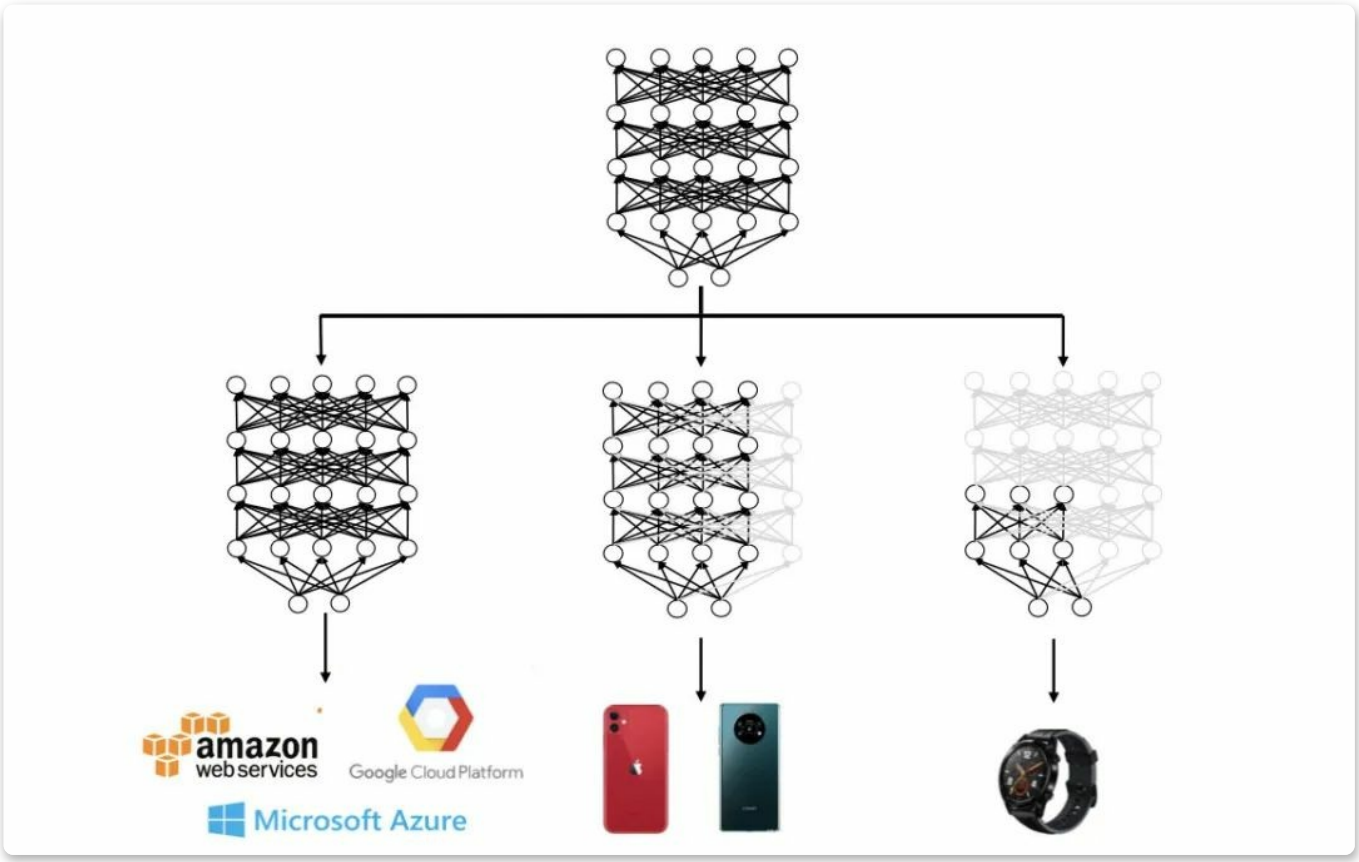
原创 rumor酱 夕小瑶的卖萌屋 5月21日

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**

神经网络模型除了部署在远程服务器之外，也会部署在手机、音响等智能硬件上。比如在自动驾驶的场景下，大部分模型都得放在车上的终端里，不然荒山野岭没有网的时候就尴尬了。对于BERT这类大模型来说，也有部署在终端的需求，但考虑到设备的运算速度和内存大小，是没法部署完整版的，必须对模型进行瘦身压缩。



说到模型压缩，常用的方法有以下几种：

1. **量化**：用FP16或者INT8代替模型参数，一是占用了更少内存，二是接近成倍地提升了计算速度。目前FP16已经很常用了，INT8由于涉及到更多的精度损失还没普及。
2. **低轶近似／权重共享**：低轶近似是用两个更小的矩阵相乘代替一个大矩阵，权重共享是12层transformer共享相同参数。这两

种方法都在ALBERT中应用了，对速度基本没有提升，主要是减少了内存占用。但通过ALBRET方式预训练出来的Transformer理论上比BERT中的层更通用，可以直接拿来初始化浅层transformer模型，相当于提升了速度。

3. **剪枝：通过去掉模型的一部分减少运算。最细粒度为权重剪枝，即将某个连接权重置为0，得到稀疏矩阵；其次为神经元剪枝，去掉矩阵中的一个vector；模型层面则为结构性剪枝，可以是去掉attention、FFN或整个层，典型的工作是LayerDrop[1]。这两种方法都是同时对速度和内存进行优化。**
4. **蒸馏：训练时让小模型学习大模型的泛化能力，预测时只是用小模型。比较有名的工作是DistillBERT[2]和TinyBERT[3]。**

实际工作中，减少BERT层数+蒸馏是一种常见且有效的提速做法。但由于不同任务对速度的要求不一样，可能任务A可以用6层的BERT，任务B就只能用3层的，因此每次都要花费不少时间对小模型进行调参蒸馏。

有没有办法一次获得多个尺寸的小模型呢?

今天rumor就给大家介绍一篇论文《**DynaBERT: Dynamic BERT with Adaptive Width and Depth**》[4]。论文中作者提出了新的训练算法，同时对不同尺寸的子网络进行训练，通过该方法训练后可以在推理阶段直接对模型裁剪。依靠新的训练算法，本文在**效果上超越了众多压缩模型，比如DistillBERT、TinyBERT以及LayerDrop后的模型**。

Arxiv访问慢的小伙伴也可以在订阅号后台回复关键词【**0521**】下载论文PDF。

原理

论文对于BERT的压缩流程是这样的：

- **训练时，对宽度和深度进行裁剪，训练不同的子网络**
- **推理时，根据速度需要直接裁剪，用裁剪后的子网络进行预测**

想法其实很简单，但如何能保证更好的效果呢？这就要看炼丹功力了 (..•˘_˘•..)，请听我下面道来～

整体的训练分为两个阶段，先进行宽度自适应训练，再进行宽度+深度自适应训练。

宽度自适应 Adaptive Width

宽度自适应的训练流程是：

1. **得到适合裁剪的teacher模型，并用它初始化student模型**
2. **裁剪得到不同尺寸的子网络作为student模型，对teacher进行蒸馏**

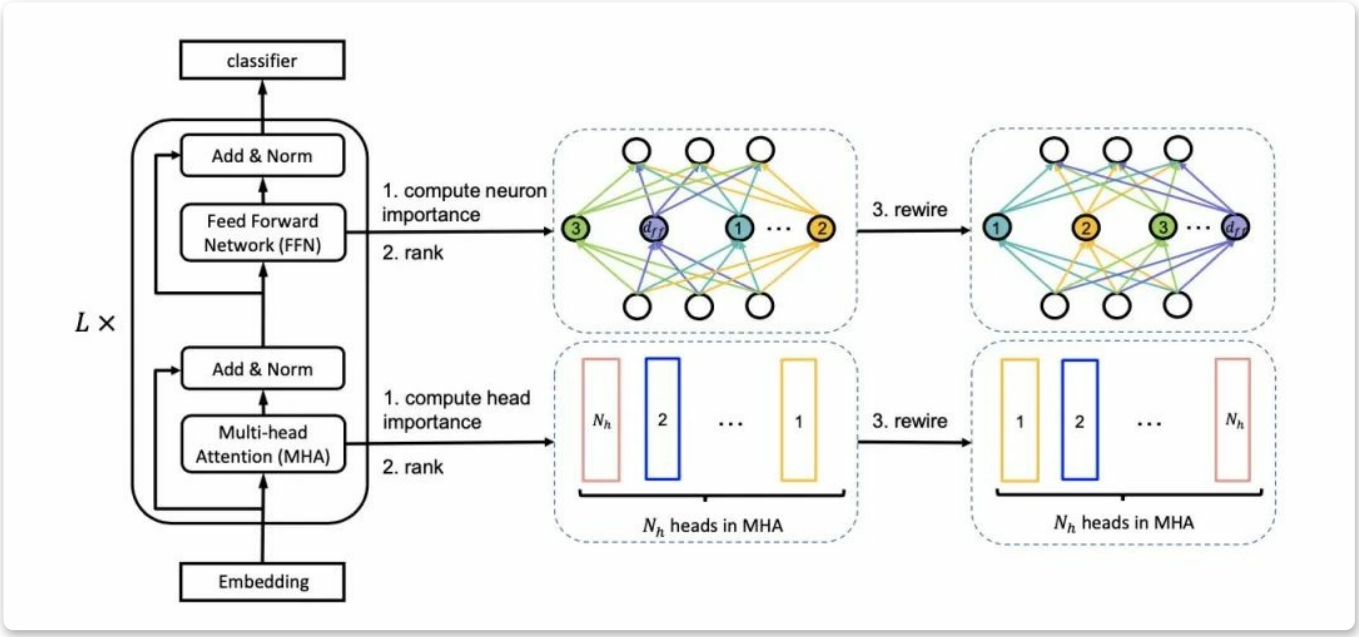
最重要的就是如何得到适合裁剪的teacher。先说一下宽度的定义和剪枝方法。Transformer中主要有Multi-head Self-

attention(MHA)和Feed Forward Network(FFN)两个模块，为了简化，作者用注意力头的个数 N_H和intermediate层神经元的个数d_{ff}来定义MHA和FFN的宽度，并使用同一个缩放系数m_w来剪枝，剪枝后注意力头减小到$\lfloor m_w N_H \rfloor$个，intermediate层神经元减少到$\lfloor m_w d_{ff} \rfloor$个。

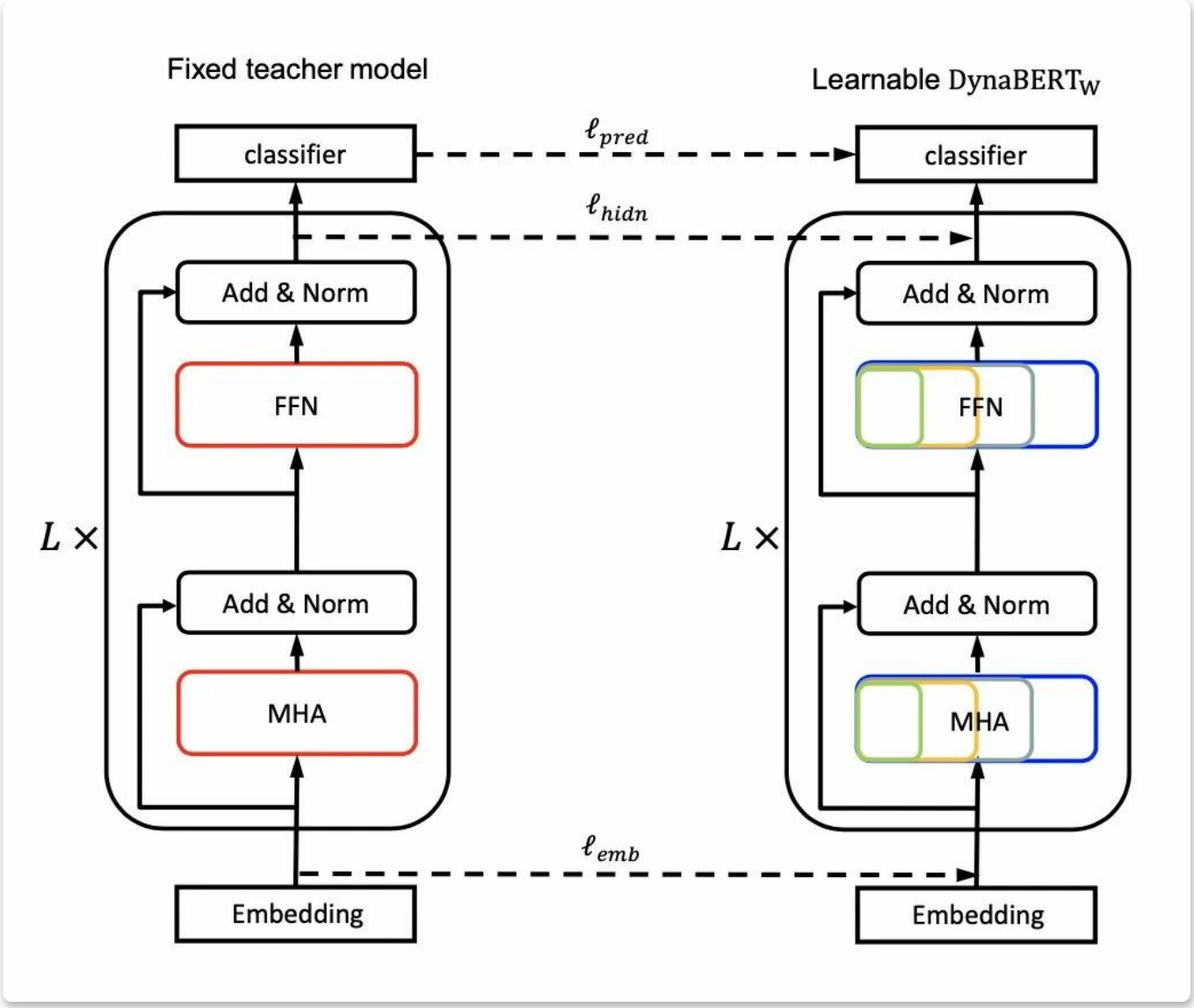
在MHA中，我们认为不同的head抽取到了不同的特征，因此每个head的作用和权重肯定也是不同的，intermediate中的神经元连接也是。如果直接按照m_w粗暴裁剪的话，大概率会丢失重要的信息，因此作者想到了一种方法，对head和神经元进行排序，每次剪枝掉不重要的部分，并称这种方法为**Netword Rewiring**。

对于重要程度的计算参考了论文[5]，核心思想是计算去掉head之前和之后的loss变化，变化越大则越重要。

利用Rewiring机制，便可以对注意力头和神经元进行排序，得到第一步的teacher模型，如图：



要注意的是，虽然随着参数更新，注意力头和神经元的权重会变化，但**teacher模型只初始化一次**（在后文有验证增加频率并没带来太大提升）。之后，每个batch会训练 $m_w = [1.0, 0.75, 0.5, 0.25]$ 四种student模型，如图：



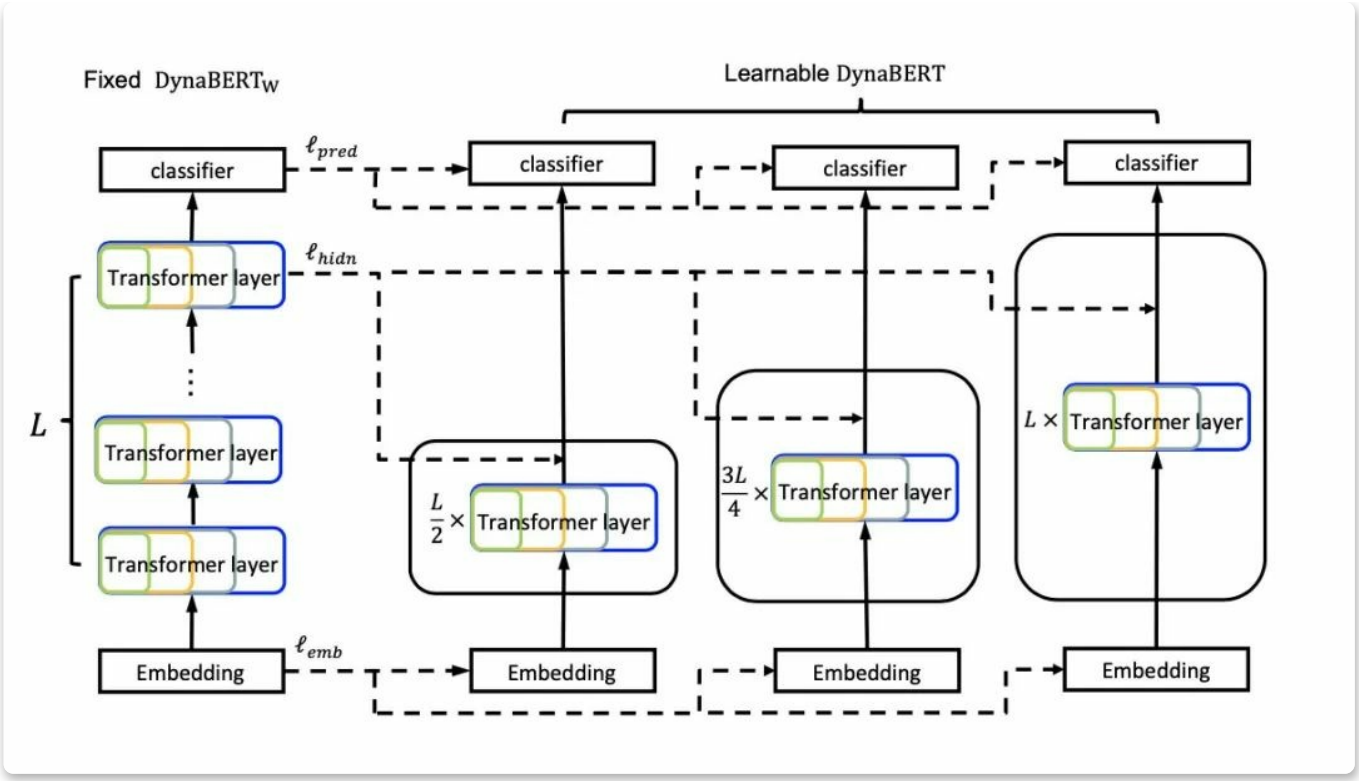
蒸馏的最终loss来源于三方面：logits、embedding和每层的hidden state。

深度自适应 Adaptive Depth

训好了width-adaptive的模型之后，就可以训自适应深度的了。浅层BERT模型的优化其实比较成熟了，主要的技巧就是蒸馏。作者直接使用训好的$DynaBERT_W$作为teacher，蒸馏裁剪深度后的小版本BERT。

对于深度，系数$m_d = [1.0, 0.75, 0.5]$，设层的深度为[1,12]，作者根据$mod(d+1, \frac{1}{m_d}) \equiv 0$去掉深度为d的层。之所以取$d+1$是因为研究表明最后一层比较重要[6]。

最后，为了避免灾难性遗忘，作者继续对宽度进行剪枝训练，第二阶段的训练方式如图：



实验

根据训练时宽度和深度的裁剪系数，作者最终可得到12个大小不同的BERT模型，在GLUE上的效果如下：

Method		CoLA			STS-B			MRPC			RTE		
BERT$_{BASE}$		58.1			89.8			87.7			71.1		
	(m_w, m_d)	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x
DynaBERT	1.0x	59.7	59.1	54.6	**90.1**	89.5	88.6	86.3	85.8	85.0	72.2	71.8	66.1
	0.75x	**60.8**	59.6	53.2	90.0	89.4	88.5	**86.5**	85.5	84.1	71.8	**73.3**	65.7
	0.5x	58.4	56.8	48.5	89.8	89.2	88.2	84.8	84.1	83.1	72.2	72.2	67.9
	0.25x	50.9	51.6	43.7	89.2	88.3	87.0	83.8	83.8	81.4	68.6	68.6	63.2
		MNLI-(m/mm)			QQP			QNLI			SST-2		
BERT$_{BASE}$		84.8/84.9			90.9			92.0			92.9		
	(m_w, m_d)	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x
DynaBERT	1.0x	**84.9/85.5**	84.4/85.1	83.7/84.6	**91.4**	**91.4**	91.1	92.1	91.7	90.6	93.2	93.3	92.7
	0.75x	84.7/85.5	84.3/85.2	83.6/84.4	**91.4**	91.3	91.2	92.2	91.8	90.7	93.0	93.1	92.8
	0.5x	84.7/85.2	84.2/84.7	83.0/83.6	91.3	91.2	91.0	**92.2**	91.5	90.0	**93.3**	92.7	91.6
	0.25x	83.9/84.2	83.4/83.7	82.0/82.3	90.7	91.1	90.4	91.5	90.8	88.5	92.8	92.0	92.0
		CoLA			STS-B			MRPC			RTE		
RoBERTa$_{BASE}$		65.1			91.2			90.7			81.2		
	(m_w, m_d)	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x
DynaRoBERTa	1.0x	63.6	61.0.7	59.5	**91.3**	91.0	90.0	88.7	89.7	88.5	**82.3**	78.7	72.9
	0.75x	**63.7**	61.4	54.9	91.0	90.7	89.7	90.0	89.2	88.2	79.4	77.3	70.8
	0.5x	61.3	58.1	52.9	90.3	90.1	88.9	**90.4**	90.0	86.5	75.1	73.6	71.5
	0.25x	54.2	46.7	39.8	89.6	89.2	87.5	88.2	88.0	84.3	70.0	70.0	66.8
		MNLI-(m/mm)			QQP			QNLI			SST-2		
RoBERTa$_{BASE}$		87.5/87.5			91.8			93.1			95.2		
	(m_w, m_d)	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x	1.0x	0.75x	0.5x
DynaRoBERTa	1.0x	**88.3/87.6**	87.7/87.2	86.2/85.8	92.0	**92.0**	91.7	**92.9**	92.5	91.4	**95.1**	94.3	93.3
	0.75x	88.0/87.3	87.5/86.7	85.8/85.4	91.9	91.8	91.6	92.8	92.4	91.3	94.6	94.3	93.3
	0.5x	87.1/86.4	86.8/85.9	84.8/84.2	91.7	91.5	91.2	92.3	91.9	90.8	93.6	94.2	92.9
	0.25x	84.6/84.7	84.0/83.7	82.1/82.0	91.2	91.0	90.5	90.9	90.9	89.3	93.9	93.2	91.6

可以看到，**剪枝的BERT效果并没有太多下降，并且在9个任务中都超越了BERT-base**。同时，这种灵活的训练方式也给BERT本身的效果带来了提升，在与BERT和RoBERTa的对比中都更胜一筹：

	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
$\text{BERT}_{\text{BASE}}$	84.6	83.6	71.9	90.7	93.4	51.5	85.2	87.5	69.6
DynaBERT $(m_w, m_d = 1, 1)$	84.5	84.1	72.1	91.3	93.0	54.9	84.4	87.9	69.9
$\text{RoBERTa}_{\text{BASE}}$	86.0	85.4	70.9	92.5	94.6	50.5	88.1	90.0	73.0
DynaRoBERTa $(m_w, m_d = 1, 1)$	86.9	86.7	71.9	92.5	94.7	54.1	88.4	90.8	73.7

另外，作者还和DistillBERT、TinyBERT、LayerDrop进行了实验对比，DynaBERT均获得了更好的效果。

在消融实验中，作者发现在加了rewiring机制后准确率**平均提升了2个点**之多：

	m_w	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	avg.
Separate network	1.0x	84.8	84.9	90.9	92.0	92.9	58.1	89.8	87.7	71.1	83.6
	0.75x	84.2	84.1	90.6	89.7	92.9	48.0	87.2	82.8	66.1	80.6
	0.5x	81.7	81.7	89.7	86	91.4	37.2	84.5	75.5	55.2	75.9
	0.25x	77.9	77.9	89.9	83.7	86.7	14.7	77.4	71.3	57.4	70.8
	avg.	82.2	82.2	90.3	87.8	91.0	39.9	84.6	78.8	61.6	77.6
Vanilla $\text{DynaBERT}_{\text{W}}$	1.0x	84.5	85.1	91.3	91.7	92.9	58.1	89.9	83.3	69.3	82.9
	0.75x	83.5	84.0	91.1	90.1	91.7	54.5	88.7	82.6	65.7	81.3
	0.5x	82.1	82.3	90.7	88.9	91.6	46.9	87.3	83.1	61	79.3
	0.25x	78.6	78.4	89.1	85.6	88.5	16.4	83.5	72.8	60.6	72.6
	avg.	82.2	82.5	90.6	89.1	91.2	44.0	87.4	80.5	64.2	79.0
+ Network rewiring	1.0x	84.9	84.9	91.4	91.6	91.9	56.3	90.0	84.6	70.0	82.8
	0.75x	84.3	84.2	91.3	91.7	92.4	56.4	89.9	86.0	71.1	83.0
	0.5x	82.9	82.9	91.0	90.6	91.9	47.7	89.2	84.1	71.5	81.3
	0.25x	80.4	80.0	90.0	87.8	90.4	45.1	87.3	80.4	66.0	78.6
	avg.	83.1	83.0	90.9	90.4	91.7	51.4	89.1	83.8	**69.7**	81.4
+ Distillation and Data Augmentation	1.0x	85.1	85.4	91.1	92.5	92.9	59.0	90.0	86.0	70.0	83.5
	0.75x	84.9	85.6	91.1	92.4	93.1	57.9	90.0	87.0	70.8	83.6
	0.5x	84.4	84.9	91.0	92.3	93.0	56.7	89.9	87.3	71.5	83.4
	0.25x	83.4	83.8	90.6	91.2	91.7	49.9	89.0	84.1	65.7	81.0
	avg.	**84.5**	**84.9**	**91.0**	**92.1**	**92.7**	**55.9**	**89.7**	**86.1**	69.5	82.9

结论

本篇论文的创新点主要在于Adaptive width的训练方式，考虑到后续的裁剪，作者对head和neuron进行了排序，并利用蒸馏让子网络学习大网络的知识。

总体来说还是有些点可以挖的，比如作者为什么选择先对宽度进行自适应，再宽度+深度自适应？这样的好处可能是在第二阶段的蒸馏中学习到宽度自适应过的子网络知识。但直接进行同时训练不可以吗？还是希望作者再验证一下不同顺序的差距。

为了简化，作者在宽度上所做的压缩比较简单，之后可以继续尝试压缩hidden dim。另外，ALBERT相比原始BERT其实更适合浅层Transformer，也可以作为之后的尝试方向。

Arxiv访问慢的小伙伴也可以在订阅号后台回复关键词【**0521**】下载论文PDF。

参考文献

[1]LayerDrop: https://arxiv.org/abs/1909.11556

[2]DistillBERT: https://arxiv.org/abs/1910.01108

[3]TinyBERT: https://arxiv.org/abs/1909.10351

[4]DynaBERT: https://www.researchgate.net/publication/340523407_DynaBERT_Dynamic_BERT_with_Adaptive_Width_and_Depth
[5]Analyzing multi-head self-attention: https://arxiv.org/abs/1905.09418
[6]Minilm: https://arxiv.org/abs/2002.10957

可能喜欢

- 献给新一代人工智能后浪——《后丹》
- 搜索中的 Query 理解及应用
- ICLR认知科学@AI workshop一览
- All in Linux：一个算法工程师的IDE断奶之路
- 卖萌屋算法岗面试手册上线！通往面试自由之路