基于Transformer的预训练模型的趋势就是越来越大，虽然这些模型在效果上有很大的提升，但是巨大的参数量也对上线这些模型提出挑战。

对于BERT的模型压缩大体上可以分为 5 种（其他模型压缩也是一样）：

- **知识蒸馏**：将 teacher 的能力蒸馏到 student上，一般 student 会比 teacher 小。我们可以把一个大而深的网络蒸馏到一个小的网络，也可以把集成的网络蒸馏到一个小的网络上。
- **参数共享**：通过共享参数，达到减少网络参数的目的，如 ALBERT 共享了 Transformer 层；
- **参数矩阵近似**：通过矩阵的低秩分解或其他方法达到降低矩阵参数的目的，例如ALBERT对embedding table做了低秩分解；
- **量化**：比如将 float32 降到 float8。
- **模型剪枝**：即移除对结果作用较小的组件，如减少 head 的数量和去除作用较少的层。

这篇文章中主要介绍知识蒸馏、参数共享和参数矩阵近似方法。

1. 使用知识蒸馏进行压缩

关于知识蒸馏的基础知识见：

MLE-interview/知识蒸馏原论文.md at master · hannawong/MLE-interviewgithub.com/hannawong/MLE-interview/blob/master/5.%E6%8E%A8%E8%8D%90%E7%B3%BB%E7%BB%9F/9.%E7%9F%A5%E8%AF%86%E8%92%B8%E9%A6%8F/%E7%9F%A5%E8%AF%86%E8%92%B8%E9%A6%8F%E5%8E%9F%E8%AE%BA%E6%96%87.md

具有代表性的论文:

1.1 DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

DistilBERT属于知识蒸馏中的**logits蒸馏**方法。

之前的很多工作都是从bert中蒸馏出一个"task-specific model", 即对**某个具体的任务**（如情感分类）蒸馏一个模型。DistilBERT不同的地方在于它是**在预训练阶段进行蒸馏**，蒸馏出来一个**通用的模型**，再在下游任务上微调。DistilBERT参数量是BERT的40%（可以在edge device上运行），保留了97%的语言理解能力。

1.1.1 损失函数设计

预训练的损失函数由三部分构成：

- 蒸馏损失：对Student和Teacher的logits都在**高温下**做softmax，求二者的KL散度

- 有监督任务损失：在这个预训练问题中就是Bert的MLM任务损失，注意此时Student模型的输出是在**温度为1**下做的softmax
- cosine embedding loss: 把Student的Teacher的隐藏向量用余弦相似度做对齐。（感觉这个类似中间层蒸馏）

1.1.2 学生模型设计

student模型只使用BERT**一半**的层；**使用teacher模型的参数进行初始化。**在训练过程中使用了动态掩码、大batchsize，然后没有使用next sentence objective（和Roberta一样）。训练数据和原始的Bert训练使用的一样，但是因为模型变小所以节省了训练资源。

在GLUE(General Language Understanding Evaluation)数据集上进行微调，测试结果：

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.3	91.3	86.9	56.3

DistilBERT保留了BERT 97%的语言理解能力

此外，作者还研究了两阶段蒸馏（跟下文TinyBERT很像），即在预训练阶段蒸馏出一个通用模型之后，再用一个**已经在SQuAD模型上微调过的BERT模型**作为Teacher，这样微调的时候除了任务本身的loss，还加上了和Teacher输出logits的KL散度loss。我理解这样相当于进行label smoothing，Student模型能够学到更多的信息，因此表现会有一个提升：

Table 2: **DistilBERT yields to comparable performance on downstream tasks.** Comparison on downstream tasks: IMDb (test accuracy) and SQuAD 1.1 (EM/F1 on dev set). D: with a second step of distillation during fine-tuning.

Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

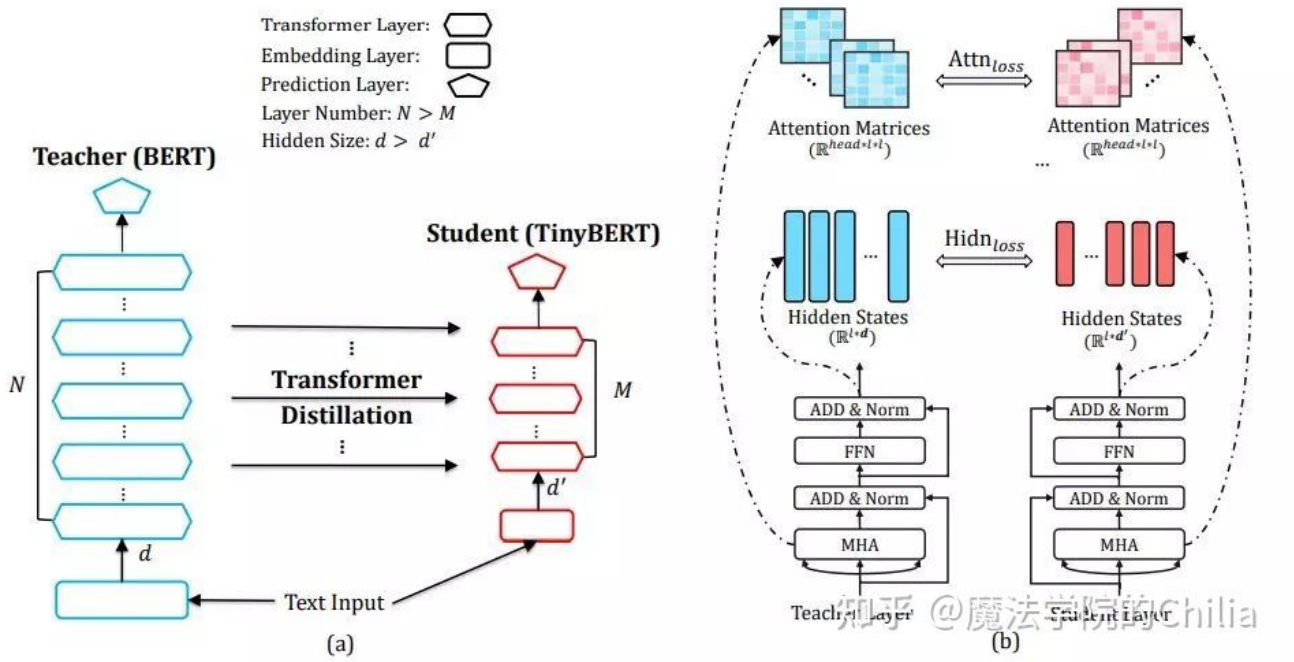
DistilBERT(D)就是两阶段蒸馏，表现优于一阶段蒸馏+微调

1.2 TinyBERT: Distilling BERT for Natural Language Understanding

TinyBERT是采用**两段式**学习框架，分别在预训练和针对特定任务的具体学习阶段执行 transformer 蒸馏。这一框架确保 TinyBERT 可以获取 teacher BERT的通用和针对特定任务的知识。

1.2.1 Transformer 蒸馏

假设 student 模型有 M 个 Transformer 层，teacher 模型有 N 个 Transformer 层。n=g(m) 是 student 层到 teacher 层的映射函数，这意味着 student 模型的第 m 层从 teacher 模型的第 n 层学习信息。把embedding层的蒸馏和预测层蒸馏也考虑进来，将embedding层看作第 0 层，预测层看作第 M+1 层，并且有映射： 0 = g(0) 和 N + 1 = g(M + 1)。这样，我们就已经**把Student的每一层和Teacher的层对应了起来**。文中尝试了4层($TinyBERT_4$)和6层($TinyBERT_6$)这个对应关系如下图(a)所示：



那么，学生对老师的蒸馏损失如下：

$$\mathcal{L}_{\text{model}} = \sum_{x \in \mathcal{X}} \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{\text{layer}}\left(f_m^S(x), f_{g(m)}^T(x)\right), \quad (6)$$

层数 Student第m层 Teacher第g(m)层 样例 相似度函数

对于<学生第m层，老师第g(m)层>，需要用 L_{layer} 计算二者的**差异**，那么这个差异如何来求呢？下面，我们来看四个损失函数：

1）注意力损失

$$L_{attn}$$

BERT的注意力头可以捕捉丰富的语言信息。基于注意力的蒸馏是为了鼓励语言知识从 teacher BERT 迁移到 student TinyBERT 模型中。具体而言，student 网络学习如何拟合 teacher 网络中多头注意力的矩阵，目标函数定义如下：

$$\mathcal{L}_{\text{attn}} = \frac{1}{h} \sum_{i=1}^{h} \text{MSE}(\boldsymbol{A}_i^S, \boldsymbol{A}_i^T), \quad (7)$$

头数

其实就是求Student的第i个注意力头与Teacher的第i个注意力头的MSE loss；一个细节是作者只使用了原始的attention矩阵A，而没有使用经过softmax之后的注意力矩阵，因为这样更好收敛。

2）hidden损失

$$L_{hidn}$$

$$\mathcal{L}_{\text{hidn}} = \text{MSE}(\boldsymbol{H}^S \boldsymbol{W}_h, \boldsymbol{H}^T), \tag{8}$$

对每一个transformer层的输出hidden state进行蒸馏。由于Student的hidden size往往小于Teacher的hidden size，所以需要一个

$$W_h$$

做适配（这也是中间层蒸馏的思想）。这也是Tinybert和DistilBERT不同的地方 -- DistilBERT只是减少了层数，而TinyBERT还缩减了hidden size。

3）Embedding层损失

$$L_{embd}$$

还是类似的方法做中间层蒸馏，用

$$W_e$$

适配：

$$\mathcal{L}_{\text{embd}} = \text{MSE}(\boldsymbol{E}^S \boldsymbol{W}_e, \boldsymbol{E}^T), \tag{9}$$

embedding size和hidden size大小一样

4）输出层损失

$$L_{pred}$$

这是logits蒸馏，在温度t下求Student的Teacher输出层的KL散度。

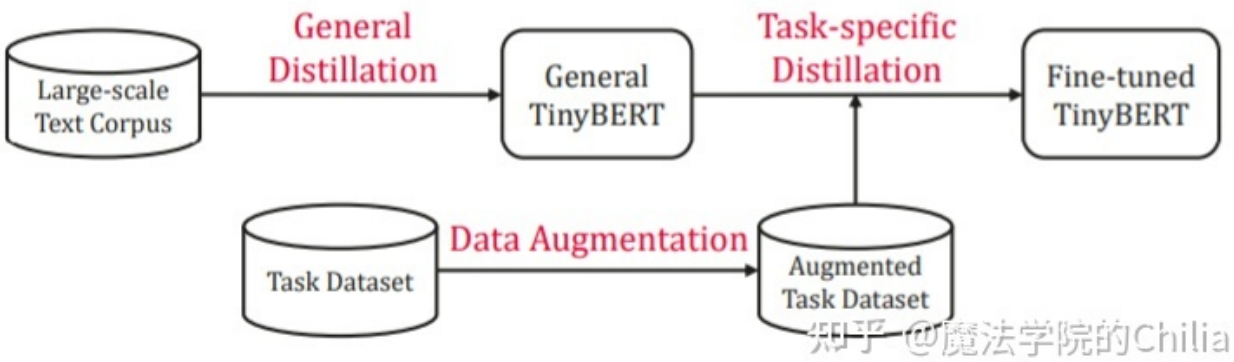
$$\mathcal{L}_{\text{pred}} = \text{CE}(\boldsymbol{z}^T/t, \boldsymbol{z}^S/t), \tag{10}$$

每一层的损失函数如下，即分embedding层、中间transformer层、输出logits层来分类讨论：

$$\mathcal{L}_{\text{layer}}(S_m, T_{g(m)}) = \begin{cases} \mathcal{L}_{\text{embd}}(S_0, T_0), & m = 0 \\ \mathcal{L}_{\text{hidn}}(S_m, T_{g(m)}) + \mathcal{L}_{\text{attn}}(S_m, T_{g(m)}), & M \geq m > 0 \\ \mathcal{L}_{\text{pred}}(S_{M+1}, T_{N+1}), & m = M+1 \end{cases} \tag{11}$$

1.2.2 两段式学习框架

BERT 的应用通常包含两个学习阶段：预训练和微调。BERT 在预训练阶段学到的知识非常重要，需要迁移到压缩的模型中去。因此，Tinybert使用两段式学习框架，包含通用蒸馏(general distillation)和特定于任务的蒸馏(task-specific distillation).



通用蒸馏（general distillation）

使用**未经过微调的预训练 BERT** 作为 teacher 模型，利用大规模文本语料库作为学习数据，执行上文所述的 Transformer 蒸馏。这样就得到了一个通用 TinyBERT。然而，由于隐藏/embedding层大小及层数显著降低，通用 TinyBERT 的表现不如 BERT。

针对特定任务的蒸馏(task-specific distillation)

之前的研究表明，像BERT这样的复杂模型在特定任务上有着**参数冗余**，所以是可以用小模型来得到相似的结果的。所以，在针对特定任务蒸馏时，使用**微调的 BERT** 用作 teacher 模型（这个和上文DistilBERT提到的方法类似，可以理解为label smoothing）。还用了数据增强方法来扩展针对特定任务的训练集。

文中的数据增强方法就是：对于multiple-piece word(就是那些做word piece得到多个子词的词语)，直接去找GloVe中和它最接近的K个词语来替换；对于single-piece word（自己就是子词的词语），先把它MASK掉，然后让预训练BERT试图恢复之，取出BERT输出的K个概率最大的词语来替换。我理解其实这个属于离散的数据增强，根据SimCSE文章中的说法，这种数据增强方法可能会引入一些噪声☹

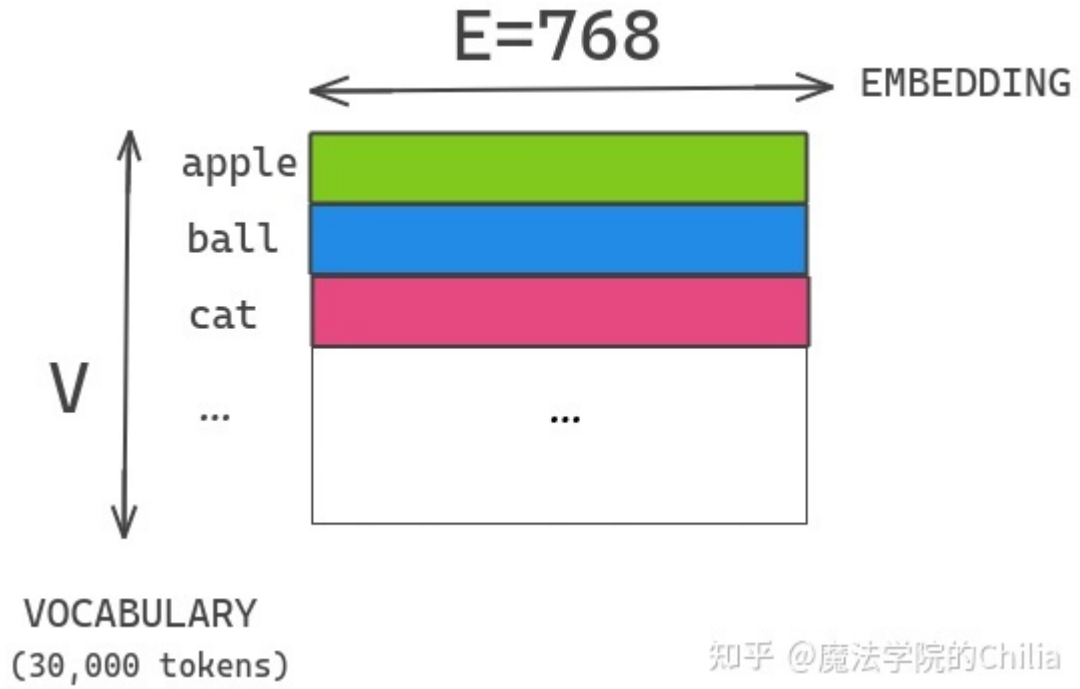
上述两个学习阶段是相辅相成的：通用蒸馏为针对特定任务的蒸馏提供良好的初始化，而针对特定任务的蒸馏通过专注于学习针对特定任务的知识来进一步提升 TinyBERT 的效果。

2. 参数共享 & 矩阵近似

这两种方法就放在一起说了，以ALBERT为例：ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

2.1 矩阵低秩分解(对Embedding Table进行分解)

ALBERT中使用和BERT大小相近的30K词汇表。假如我们的embedding size和hidden size一样，都是768，那么如果我们想增加了hidden size，就也需要相应的增加embedding size，这会导致embedding table变得很大。



> In BERT, ... the WordPiece embedding size E is tied with the hidden layer size H, i.e. E = H. This decision appears suboptimal for both modeling and practical reasons. -- ALBERT论文

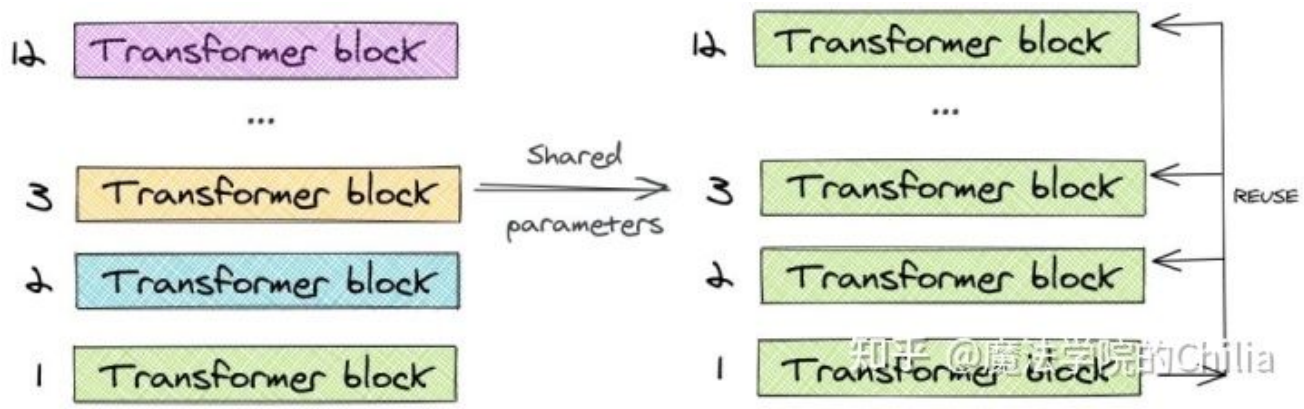
ALBERT通过将大的词汇表embedding矩阵分解成两个小矩阵来解决这个问题。这将**隐藏层的大小与词汇表嵌入的大小分开。**

- 从模型的角度来讲，因为WordPiece embedding只是要学习一些上下文无关的表示(context-independent representations), 而hidden layer是要学习上下文相关的表示(context-dependent representations). 而BERT类模型的强大之处就在于它能够建模**上下文相关**的表示。所以，理应有 H >> E.
- 从实用的角度来讲，这允许我们**在不显著增加词汇表embedding的参数大小的情况下增加隐藏的大小。**

我们将one-hot encoding向量投影到 E=100 的低维嵌入空间，然后将这个嵌入空间投影到隐含层空间H=768。其实这也可以理解为：使用E = 100的embedding table，得到每个token的embedding之后再经过一层全连接转化为768维。这样，模型参数量从原来的 $O(V \times H)$ 降低为现在的 $O(V \times E + E \times H)$.

2.2 参数共享

ALBERT使用了跨层参数共享的概念。为了说明这一点，让我们看一下12层的BERT-base模型的例子。我们只学习第一个块的参数，并在剩下的11个层中重用该块，而不是为12个层中每个层都学习不同的参数。我们可以**只共享feed-forward层的参数/只共享注意力参数/共享所有的参数。**论文中的default方法是对所有参数都进行了共享。



与BERT-base的1.1亿个参数相比，相同层数和hidden size的ALBERT模型只有3100万个参数。当hidden size为128时，对精度的影响很小。精度的主要下降是由于feed-forward层的参数共享。共享注意力参数的影响是最小的。

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base E=768	all-shared	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8
	shared-attention	83M	89.9/82.7	80.0/77.2	84.0	91.4	67.7	81.6
	shared-FFN	57M	89.2/82.1	78.2/75.4	81.5	90.8	62.6	79.5
	not-shared	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base E=128	all-shared	12M	89.3/82.3	80.0/77.1	82.0	90.3	64.0	80.1
	shared-attention	64M	89.9/82.8	80.7/77.9	83.4	91.9	67.6	81.7
	shared-FFN	38M	88.9/81.6	78.6/75.6	82.3	91.7	64.4	80.2
	not-shared	89M	89.9/82.8	80.3/77.3	83.2	91.5	67.9	81.6

ALBERT的这些降低参数的做法也可以看作一种正则化，起到稳定模型、增强泛化能力的作用。

由于进行矩阵低秩分解、共享参数并不会对模型效果产生太大影响，那么就可以增加ALBERT的参数量，使其使用小于BERT-large的参数量、但达到更好的效果。