BERT 瘦身之路：Distillation，Quantization，Pruning

AINLP 10月22日

以下文章来源于安迪的写作间 ，作者入冬前瑟瑟发抖的

安迪的写作间

就是想写点东西。 深度学习，自然语言处理研究生。 喜欢有趣的东西。

> 很多外链被微信给吃掉了，点击阅读原文查看。
>
> 原文链接: https://zhuanlan.zhihu.com/p/86900556

自 BERT 放出，各家多有改进，融入更多其他方面知识，加入更多训练数据，更复杂训练技巧，花样百出。但鉴于昂贵的训练成本，大多人也就只能看看而已，之后用开源出模型，想着怎么把它用起来。

而即使如此，BERT 家族庞大体积也让进行实时推理时，需过大空间，同时速度也会比较慢。一般线下玩玩尚好，如若想将它放入线上，作为产品。那么就需要对 BERT 进行减肥，让它体量变小，速度更快。

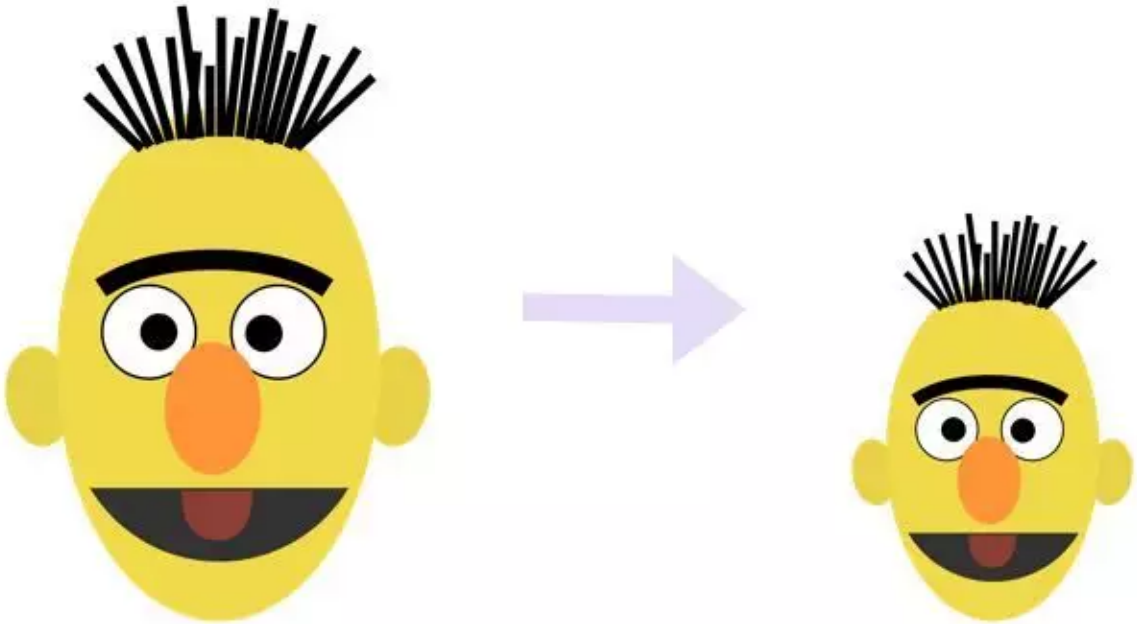
对于 BERT 瘦身，或者说目前神经网络模型，主要有三个思路：

- **Distillation（蒸馏）**：通过蒸馏技巧，将 BERT 模型知识导入小模型，之后用小模型；
- **Quantization（量化）**：将高精度模型用低精度来表示，使得模型更小；
- **Pruning（剪枝）**：将模型中作用较小部分舍弃，而让模型更小。

其中个人比较熟的是 Distillation，所以先从它讲起吧。

（多有借鉴 Rasa 的博文，感兴趣也可以看：Compressing BERT for faster prediction）

Distillation



何谓蒸馏，取其精华，去其糟粕。

模型蒸馏，是希望能将用技巧**将大模型中精华（暗知识）取出，注入到小模型**中，从而使得小模型具备大模型的好性能。而通常蒸馏出的小模型，又要比直接用相同模型训练得到模型性能要好，这也是蒸馏意义所在。

经典蒸馏法详解

最早的蒸馏法，一般认为是 Hinton 在 *Distilling the Knowledge in a Neural Network* 提出，之后得到推广。但最近 Schmidhuber 的文章 ***Deep Learning: Our Miraculous Year 1990-1991*** 里有提到，在1991年时，他就提出过类似方法来压缩模型。当然这也跟 GAN 一样都是笔糊涂账，还待各位看官自行判断，我们还是随主流。

Hinton 在论文中提出方法很简单，就是让学生模型的预测分布，来拟合老师模型（可以是集成模型）的预测分布，其中可通过用 **logits 除以 temperature 来调节分布平滑程度**，还避免一些极端情况影响。如下图，T 为 temperature，z 是 logits。

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

接下来可用获得分布作为 **soft-label（软标签）**，之后用**交叉熵**来计算损失。输入分别是除以 temperature 之后算出的学生与老师模型的概率输出。

$$CrossEntropy(s_i, t_i)$$

实际使用中，因为各个框架中交叉熵损失函数大多针对 hard-label，也就是一般 one-hot 标签，而针对软标签，只能自己手写未优化交叉熵。或者像大多实现用 **KLD（Kullback-Leibler divergence， KL散度） Loss** 来等价实现，比如 pytorch 中。

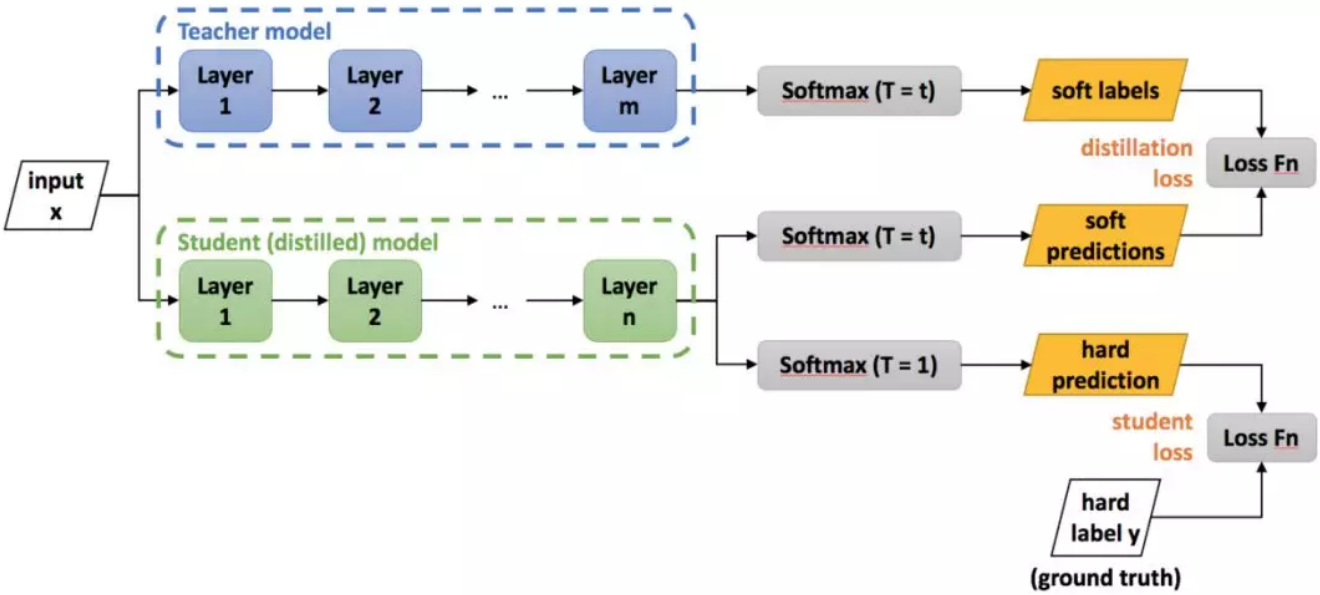
```
loss = nn.KLDivLoss(F.log_softmax(s_logits/temperature),F.softmax(t_logits/temperature))
loss = loss * (temperature)**2
```

至于之后再乘上 temperature 平方，是为了**保持梯度量级的不变**。

当然还可以用其他损失函数，比如说直接 MSE 简单粗暴的来拟合分布。

获得了拟合老师分布的损失后，还会加上实际标注数据的交叉熵损失，然后在训练过程中控制两者比例，从而使最后结果表现更好。一般刚开始设置拟合损失权重大，而在快结束时则是标注损失权重大一些。

上述过程可表示成：



BERT 蒸馏示例

这里列出几个 BERT 蒸馏例子。

首先，最完美实现上述经典方法对 BERT 蒸馏的是 HuggingFace 前段时间放出的 **DistilBERT**，将 BERT-base 从 12 层蒸馏到 6 层 BERT 模型。当然除了上述方法，还用了些其他技巧，比如用**老师模型参数初始化学生模型**，更多细节可看 HuggingFace 的博客和论文，都非常棒。

博客：Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT

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

代码：https://github.com/huggingface/transformers/tree/master/examples/distillation

接着，是之前看的一篇，关于将 BERT 模型蒸馏到 BiLSTM 来做分类任务的论文，和上述不同的时用 MSE （mean-squared-error ）来拟合学生和老师模型分布，因为作者们发现在这个任务上 MSE 要好些。

论文：Distilling Task-Specific Knowledge from BERT into Simple Neural Networks

还有一篇很有意思的博客，将 BERT 知识蒸馏到一个逻辑回归模型中去，上面这些损失函数都没有就直接**让逻辑回归输出预测 BERT 的输出结果**。

博客：Distilling BERT — How to achieve BERT performance using Logistic Regression

针对 Transformer 的蒸馏

显然看过上面例子后，就会意识到其实蒸馏自由度还是很大的，并不需要一定按照 Hinton 最初论文里一样只对最后输出进行拟合，只要能**让学生模型从老师模型中学习到东西**就行。

于是乎，除了最终的概率输出，其实很多中间结果也都包含了模型知识，词向量中也是，因此就可以思考是否能从这些里面蒸馏出知识给学生模型，特别是针对 Transformer，因为它是 BERT 的基本组件。

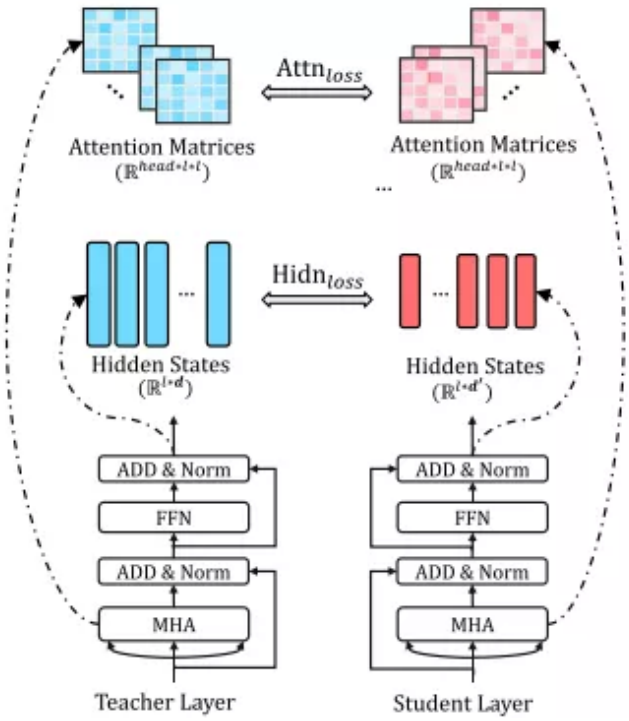
首先是 **BERT-PKD (Patient Knowledge Distillation)** 模型，它最主要是在之前提到两个损失之上，再加上一个loss, L_PT。

而这个 loss 是由**老师和学生模型中间层的 [CLS] 符的隐状态算得**，计算过程是先归一化，然后直接 MSE 求损失。之所以取 [CLS] 位置，是因为其在 BERT 分类任务中的重要性。

至于学生模型中间层如何与老师模型中间层对应，论文中发现最佳策略是直接按倍数取老师模型对应层就行，比如1对2，2对4这样.

论文：Patient Knowledge Distillation for BERT Model Compression

接着，是最近一篇华为的 TinyBERT，比起上面的 PKD 只是对中间层 [CLS] 进行拟合，它更深入了一步。对 BERT **全范围进行拟合，词向量层，中间隐层，中间注意力矩阵，最后预测层**。



因此它的损失函数比较多，值得注意的是，它在学生模型和老师模型之间，对于词向量和中间隐层会加上一个线性转换；此外它会分成两个阶段，先用 BERT 模型做通用的蒸馏，不包括预测层，之后再进行针对单独任务蒸馏，包括预测层。

该论文实验结果是比 DistilBERT 和 BERT-PKD 都要好。

论文：TinyBERT: Distilling BERT for Natural Language Understanding

蒸馏的其他用法

因为蒸馏只是一种将知识提取注入的技巧，所以它不光可以用来给模型减肥。也可以让模型大小保持不变，但通过从集成模型蒸馏，或者其他一些蒸馏技巧加强单一模型的表现。可以参考下面两个论文：

论文：Improving Multi-Task Deep Neural Networks via Knowledge Distillation for Natural Language Understanding

论文：BAM! Born-Again Multi-Task Networks for Natural Language Understanding

Quantization

何谓量化，打个比方，看 1080p 太慢，于是降到 720p 看。

同样的，如果用完整 32 位训练和保存的模型看作 1080p 的话，那么量化完后模型就可以当作是 720p，如此一来，模型自然变小，速度自然加快。

关于量化实际使用，根据实现细节涉及到好些不同分类。比如说**真量化（Real Quantization）**与**伪量化（ Pseudo Quantization）**，**训练后量化（Post Training Quantization）**与**训练中量化（During Training Quantization）**，最近 pytorch 1.3 文档中还有，动态量化（Dynamic Quantization）与静态量化（Static Quantization），看得人头晕。

真量化与伪量化

首先真量化，便是一般意义上想的，**将模型中参数表示用低精度来表示**。

比较常用的方法就是直接通过：

来将高精度（比如说32位）矩阵转换成低精度（比如说8位），之后矩阵运算使用低精度，而结果则用 scale 和 zero_point 这两个参数来还原高精度结果。

还可以更进一步，**不光矩阵运算，整个模型中的运算都用低精度（比如激活函数）**。

而关于伪量化，实际的运算过程和一般情况下跑模型没有太大区别，其实也都是 32 位运算，而增加的操作就是**将模型用低精度表示存储，然后实际运算中查表近似还原的操作**。

这里要介绍一下，量化中运用很广泛的一个算法 *k-means quantization*。具体做法是，先拿到模型完整表示的矩阵权重 W，之后用 k-means 算法将里面参数聚成 N 个簇。然后将 W 根据聚成的簇，转化成 1 到 N 的整数，每个分别指向各个簇中心点。这样就能将 32 位降到只有 log(N)位，大大减小了存储空间。而使用时只需要按照对应的 N 查表还原就行。

因为实际运算用的还是完整精度，因此也被称为**伪量化**。

训练后量化与训练中量化

首先训练后量化，其实大概就类似上面说的 *k-means quantization* 过程。

而训练中量化，一般会用一个算法 *quantization-aware training*。大概过程是：

1. 量化权重
2. 通过这个量化的网络计算损失
3. 对没量化权重计算梯度
4. 然后更新未量化权重

训练结束后，量化权重用量化后的模型直接进行预测。此过程其实有点类似混合精度训练里面的一些操作了。

量化示例

关于 BERT 的量化，开源的有因特尔放出的 NLP Architech 里的 Q8BERT。

链接：http://nlp_architect.nervanasys.com/quantized_bert.html#id7

还有一篇相关的论文，提出 group-wise quantization 加上基于二次阶Hessian信息的混合方法，用超低精度压缩 BERT：

论文：Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

最直接的方法，其实各个框架也都提供了相关函数，比如说 TensorFlow Lite 里就有自己的量化方案，而最近放出的 Pytorch 1.3 中也有关于量化的更新。

Tensorflow：https://www.tensorflow.org/lite/performance/post_training_quantization

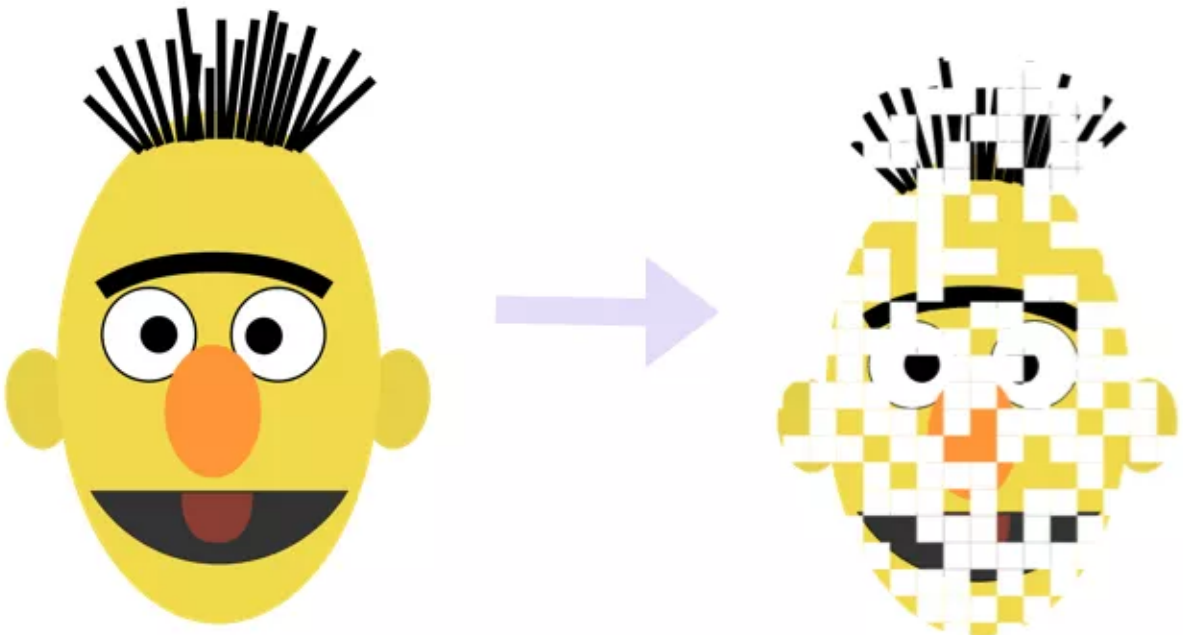
Pytorch：https://pytorch.org/docs/master/quantization.html

如果想了解量化具体实现，可参考这篇论文还有它的实现，其中不光包括量化还有后面提到的剪枝：

论文：transformers.zip: Compressing Transformers with Pruning and Quantization

代码：https://github.com/robeld/ERNIE（又是一个ERNIE）

Pruning

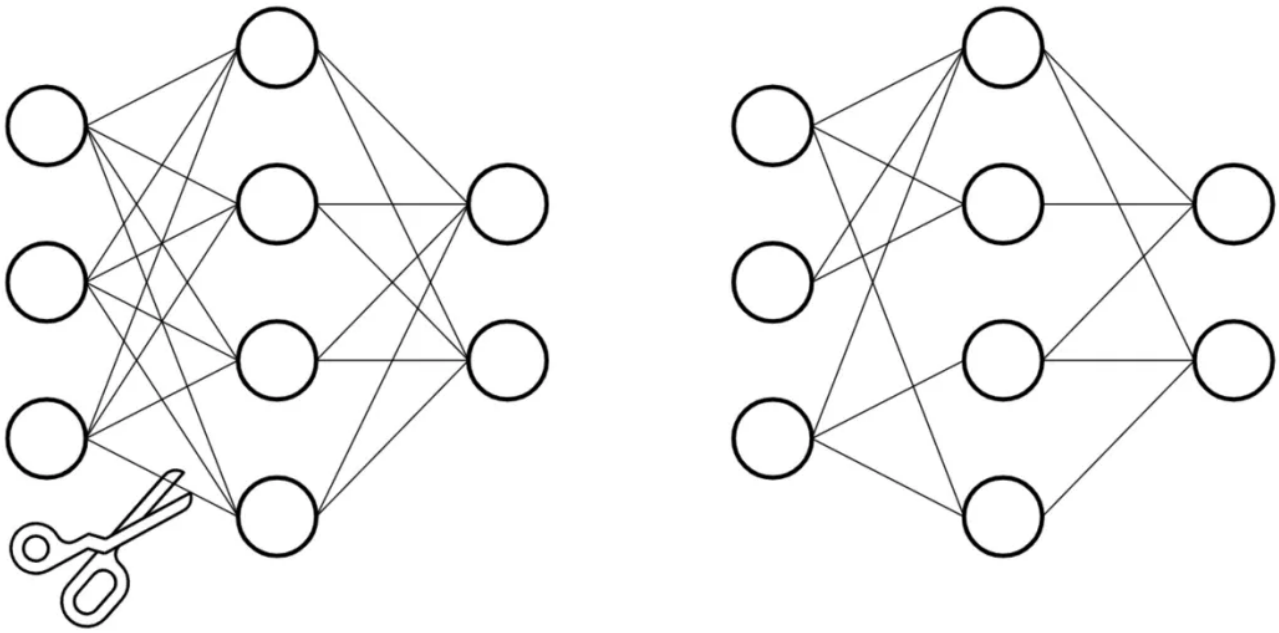


何谓剪枝，取其精华，去其糟粕，但和蒸馏不同的是，蒸馏是将精华装入一个新模型，而**剪枝则只是对原模型进行修剪，保留原模型**。

关于剪枝，具体做法简单说就是将模型中权重设为0，而根据所操作规模，可分为三个级别：

1. 对**权重连接**，其实就是权重矩阵中某个位置；
2. 对**神经元**，相当权重矩阵中某一行或一列；
3. 对**整个权重矩阵**

权重连接剪枝



常用的技巧就是 *weight pruning*，其中一个简单的做法是直接根据权重大小来剪枝，简单的将接近 0 小于某个阈值的权重连接都设为 0，这里的思想是认为**权重接近 0 的话就说明该连接在网络中重要性不大**。

因此通过该剪枝法来处理后，权重大小不变，而矩阵中会出现很多0， 而要通过该方法减小模型大小，并且加速，就要用到**稀疏矩阵**相关的知识来进行加速了。

关于稀疏矩阵或者稀疏 Transformer 相关最近也些论文与博客讲解：

1. Generative Modeling with Sparse Transformers
2. Sparse Networks from Scratch: Faster Training without Losing Performance

对于该方法 Tensorflow 的模型优化工具提供了相关教程：

链接：https://www.tensorflow.org/model_optimization/guide/pruning

神经元剪枝

和上面的权重连接剪枝相同的是，它也会设置一个标准来对每个神经元进行打分，之后根据这个标准，将分比较低的神经元给去掉，反应在矩阵上的表现就是**去掉某一行或某一列**。

于是也就带来了和前一种方法不同的一点，因为直接去掉一行一列后的话，可以在直接降低权重使用空间的情况下，仍然直接矩阵运算，而不用像上面一样使用稀疏矩阵。但同样，因为形状的改变，也会在一定程度上**影响并行运算的效率**。

权重矩阵剪枝

具体的做法和上面两个也差不多，只是一个更大范围操作。最初看到对 Bert 或者 Transformer 进行该操作是在 *Are Sixteen Heads Really Better than One?* 中看到的，因为一些研究质疑 Transformer 中**注意力头的冗余性**，于是也是根据一个打分标准（proxy importance score，模型对该参数的敏感程度）依次去掉不重要的头，最后发现有些层甚至可以从 16 个头减到只剩一个头而不太影响效果。

论文：Are Sixteen Heads Really Better than One?

代码：https://github.com/pmichel31415/are-16-heads-really-better-than-1

同样还有一篇，*The Story of Heads*，则是用一种 Layerwise Relevance Propagation (LRP) 的标准来评判头的重要性，然后他们也发现可以去掉大量头而不太影响效果：

博客：The Story of Heads

代码：https://github.com/lena-voita/the-story-of-heads

关于剪枝这一块，Rasa 也有有总结过一篇非常全的博客：

博客：Pruning BERT to accelerate inference

本文转载自公众号：安迪的写作间，作者：Andy