Applying further model compression to TinyBERT

Frederic Boileau **Yinghan Gao** **Wei Wei**
{frederic.boileau, yinghan.gao, wei.wei.2}@umontreal.ca

Abstract

Our project[1] was to apply further model compression to tinybert, a model obtained through knowledge distillation (KD) on base BERT (Jiao et al., 2019). There are quite a few approaches to this; we chose the simplest and most common ones namely global weight pruning and dynamic quantization. The scheme was thus KD followed by pruning or quantization. The two final models were evaluated on a subset of the GLUE tasks, namely CoLA, RTE, STS-B, QNLI and SST2. Our experiments show that while KD is responsible for the greatest effect on the size of the base BERT model (from 437MB to 58MB), adding pruning or quantization can improve the model without compromising significant accuracy. The pruned tinybert even compared better than its vanilla counterpart on most of the tasks.

1 Introcuction

Since the publication of the paper attention is all you need (Vaswani et al., 2017) transformers, or self-attention based models have become the de facto standard when investigating state of the art (SOA) language models. Those pretrained language models (PLM) are usually afterwards fine tuned for specific tasks. While impressive in their results we have seen progress in them mainly through making bigger models (Qiu et al., 2020). This is a challenge for deployment as fined tuned SOA PLMs require ranging from several MB to GB in extreme cases [adref], which is often unnacceptable, for mobile or IOT devices for example. Knowledge distillation is conceptually elegant and concretely an effective way to leverage large PLMs to produce smaller one which retain their performance. Initially introduced by Hinton et all (Hinton et al., 2015) Jiao et all in their tinybert paper (Jiao et al., 2019) showed how the now very common BERT PLM can be significantly reduced in size, at the same time improving the inference complexity, using KD techniques. Our first aim was to reproduce those results. Distillation proved to be computationnally very expensive and we had to decide to restrict ourselves to a subset of GLUE (the de facto standard for evaluating LMs) tasks, namely CoLA, RTE, STS-B, QNLI and SST2. The produced model performed considerably well (90% compared to our baseline, i.e. BERT) while reducing significantly the size, from 438MB to 59MB. We surveyed other compressing techniques to make the model even smaller and investigate how far we can go in compression while mixing these techniques and how they might interact.

2 Related Work

To get a good grasp of compression techniques we went over the very comprehensive survey paper by Gupta and Agrawal (Gupta and Agrawal, 2020). We found at the time of the design of the project no papers exploring combining compression techniques such as quantization and pruning with KD *in that order*. However other work such as the one by Gordon, Duh and Andrews (Gordon et al., 2020) explored the effects of weight pruning on transfer learning, however applying pruning before doing KD. Another work by Zhang et all (Zhang et al., 2020) tackles quantization and KD, but once again applying quantization (aggressive ternary one) before using KD to teach the quantized student model to perform as well as possible compared to its teacher. In what sequence, or even more boldly let's say in parallel should compression techniques be used seems to be a very open question.

[1] Code and data are stored in the link https://drive.google.com/drive/folders/1ejBAUqGMYRcRNOyQFeXhauNe8S-zePXN?usp=sharing

3 Methods

For KD of BERT base we used the code associated with the tinybert paper (Jiao et al., 2019) which can be found on github (tinybert-code). We tried to reproduce the results but like mentionned before had to restrict ourselves to a subset of the GLUE tasks. We used both our own machines in parallel and some cloud compute on google collab. We afterwards focused on dynamic quantization and global pruning. In order to implement the later two we followed the official pytorch tutorials The links to the aformentionned tutorials are dynamic-quantization-bert-tutorial and pruning-tutorial. The dynamic quantization tutorial however is still in Beta phase and the scripts had to be modified in different parts in order for the experiments to run.

4 Experiments

4.1 Datasets

We have been using Glue(Zellers et al.,2018) as a benchmark to reproduce knowledge distillation experiments. Glue contains nine English natural language generation tasks that cover fields such as Language Inference, Emotional Analysis, Semantic Similarity, etc. However, we only experiment on five sub-sets(CoLA, QNLI, RTE, SST-2, STS-B) of Glue due to the limited resources on training data.

4.2 Baselines

Our baseline models are fine-tuned BERT provided by 'TextAttack' on the Huggingface(tex). With testing the effect of distillation, we compared the performance of baseline models and distilled models on the evaluation code provided by the Noah-Huawei TinyBERT. After confirming the degradation of distilled models' performance is acceptable, we used the distilled BERT as the baseline model for further compression i.e. pruning and quantization.

4.3 Evaluation Methods

We have different evaluation metrics for different tasks as which in GLUE have different characteristics. We used the Matthews correlation coefficient to evaluate the model for the CoLA task. Matthews correlation coefficient will return a value between -1 to 1. 1 means perfect prediction. 0 means the result does not better than random prediction. -1 means the exact opposite prediction. To evaluate STS-B we used Pearson and Spearman correlation coefficients which give two values separately between -1 and 1. The better prediction has a higher absolute value of those two correlation coefficients. For SST-2, RTE, and QNLI tasks, we used accuracy.

4.4 Experiment Details

Distillation We were focusing on task-specific distillation which uses fined-tuned BERT models obtained from the Huggingface(tex) as teacher models and the general distilled model(4layer-312dim) as the student model. The latter one is provided by the Noah-Huawei(Huawei Noah) TinyBERT. The distillation contains two steps, intermediate layer distillation, and prediction later distillation. Both of them use fine-tuned BERT as the teacher model, while the former one uses the general distilled model as the student model, whichs' output will serve as the student model for the prediction layer distillation. In both intermediate layer and prediction layer distillation, we chose 32 as the batch size, 3e-5 as the learning rate. We trained 10 epochs in intermediate layer distillation and 3 epochs in prediction layer distillation. Obtaining the distilled models, we evaluate them and compare the evaluation result with the fine-tuned BERT models before the distillation.
We also tried to fine-tune the BERT by ourselves, but it requires too much computation resources. Besides, we also tried to do data augmentation for datasets in Glue and we indeed have done for several small datasets in Glue since they would not cost too many times while the others will cost a lot. The basic proceed was, get the glove embedding by Stanford Glove(Jeffrey Pennington) and run data augmentation code in TinyBERT with glove embedding vectors and pre-trained BERT as variables.

Pruning After obtaining acceptable distilled BERT models, we start pruning on their weight. We used incorporated the global pruning function in Pytorch which takes the pruning layers, the pruning amount, the pruning methods, model to be pruned as input, and return the pruned model as output. The global pruning function will not focus on the pruning rate of one specific layer instead it only concerns the total pruning amount. The mechanism of global pruning is that it will rank weights in all picked modules according to the specified ranking methods and set the low-rank weights to zero according to the specified pruning amount. Weights that were set to zero will lose the ability to engage

the forward propagation and the backward propagation.
As for the configuration that we chose for pruning, we use L1Unstructured as the pruning method. It will sort weights in selected layers by their L1 norms. Weights with low L1 norms will be set to zero. We use this sorting method because we believe a lower L1 norm has less impact on the prediction. The layers we picked for pruning contain the query, key, value, dense, normalization layer in all attention modules and all embedding layers. We tried multiple pruning rates from 0.1 to 0.4 corresponding to pruning 10 percent weights and 40 percent weights.

Quantization We used the built-in PyTorch functionality of dynamic quantization (DQ) to serve TinyBERT as a dynamic quantized model, following the tutorials of Pytorch, (beta) Dynamic Quantization on BERT (Huang). Dynamic quantization in PyTorch can convert a float model with FP32 to a quantized model with static int8. The activations and the weights are quantized dynamically to int8 in torch.quantization.quantize_dynamic API. In our project, the torch.nn.Linear modules in TinyBERT were quantized, and the weights were converted from FP32 to quantized int8 values. We compared the model size, the inference time, and the evaluation accuracy between the original TinyBERT and the quantized TinyBERT.

4.5 Result

Distillation As we can see from table 1, For all five datasets we tested on, most of the distilled model maintained approximately 90 percent performance of fine-tuned Bert. For dataset STS-b and SST-2 particularly, the degradation of the distilled model was even lower than 10%. The distilled model did indeed degrade a lit bit. However, comparing the decrease in the size of the model, the degradation is acceptable. The size of the parameter fine-tuned Bert model is 417M while that of the distilled model is 55.9M. Thus, the model was compressed a lot without sacrificing too much performance.

Pruning From Figure 1 we can conclude that the pruning rate between 10 percent to 30 percent is generally good. This value is slightly lower than that concluded in other relevant studies. We think the reason is that distillation has already increased the percentage of effective parameters.
From table1 we can find that pruning even slightly increases the accuracy of the distilled model. We believe this is because the redundant weights were removed from the forward and backward propagation, which alleviates the overfitting. Besides, we may notice that the size of -our model was not further compressed, this is because pruning can only set pruned parameters to zero instead of removing them from the model. However, from table2 we can tell that pruning also has a positive influence on reducing the time consumption on computation.

Quantization From table1, we can observe a considerable decrease 25.6% in model size from TinyBERT (FP32, 58.39MB) to quantized TinyBERT(int8, 58.39MB). Simultaneously, the evaluation accuracy of TinyBERT on each GLUE task has no obvious decrease after dynamic quantization.
From table 2, we can find that the inference time of TinyBERT on each GLUE task reduces approximately 10% after dynamic quantization.

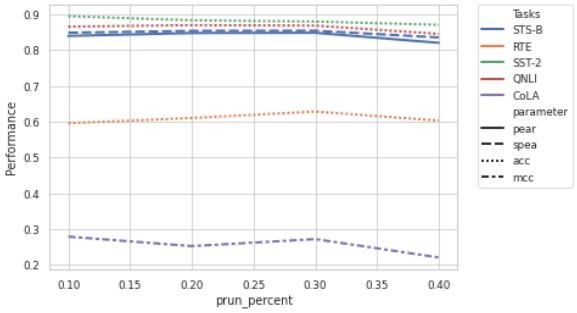

Figure 1: Pruning amount of parameter in TinyBERT on different GLUE tasks

5 Conclusion

Our experiments showed that applying further model compressions to distillied PLMs yields good results when done after the fact. In the case of quantization we observed a significant memory footprint reduction while pruning made inference more efficient, and somewhat surprisingly, more accurate in some cases. Other work such as the one found in (Zhang et al., 2020) yield good results by reversing the order of our approach, using KD to improve the performance of aggressively quantized models. We can think of all the compression techniques as nodes in a directed acyclic graph (DAG). An interesting open research question would be to explore what topologies and parameters of this DAG yield good results and what the trade offs are.

References

Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing {bert}: Studying the effects of weight pruning on transfer learning.

Manish Gupta and Puneet Agrawal. 2020. Compression of deep learning models for text: A survey.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Jianyu Huang. (beta) dynamic quantization on bert. https://pytorch.org/tutorials/intermediate/dynamic_quantization_bert_tutorial.html Accessed March 15, 2021.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert.

System	Size (MB)	CoLA Mcc	RTE Acc	STS-B Pear/Spea	QNLI Acc	SST-2 Acc
fine-tuned BERT	437.98	0.5577	0.7256	0.8791/0.8759	0.9136	0.9232
TinyBERT	58.39	0.2755	0.6101	0.8354/0.8468	0.8678	0.8979
Pruned TinyBERT	58.39	0.2789	0.6282	0.8481/0.854	0.8695	0.8945
quantized TinyBERT	43.74	0.2631	0.6426	0.8371/0.8453	0.86	0.8853

Table 1: Comparisons of fine-tuned BERT, TinyBERT, Pruned TinyBERT, and quantized TinyBERT on the set of GLUE tasks. Mcc refers to Matthews correlation and Pear/Spea refer to Pearson/Spearman.

System	CoLA	RTE	STS-B	QNLI	SST-2
fine-tuned BERT	328.3	90.8	451.3	2263.6	280.2
TinyBERT	26.8	7.6	35.6	138.9	23.2
Pruned TinyBERT	25.8	7.2	35.6	138.4	21.8
quantized TinyBERT	22.9	6.5	30.3	117.6	19.5

Table 2: Inference time of fine-tuned BERT, TinyBERT, Pruned TinyBERT, and quantized TinyBERT on the set of GLUE tasks.