Compressed SQuAD 2.0 Model With BERT

CS 224N Final Report - Winter 2020

Default Project (Alternative Goal)

(Opt-out of grading)

Andrew Deng
andrewde@stanford.edu

Wenli Looi
wlooi@stanford.edu

Abstract

The goal of our project is to design and train a SQuAD 2.0 model based on BERT that achieves reasonable performance with the smallest possible model size and limited training resources. While context-aware language models like BERT have achieved state-of-the-art performance on many NLP tasks, including SQuAD, their large size makes it difficult to host them in resource-constrained environments like mobile and edge devices for offline question answering. Starting from $\text{BERT}_{\text{BASE}}$, we implemented several size reduction techniques. Our final models achieve 96% of the original performance with 90% reduction in storage space, or 92% of the original performance with 95% reduction in storage space. Unlike many other compressed BERT-like models with high training costs, our models can be fine-tuned from BERT in a few hours on a single GPU, making it more accessible and useful to people and organizations who do not have access to large computational resources.

1 Introduction

The goal of our project is to design and train a SQuAD 2.0 model based on BERT [1] that achieves reasonable performance with the smallest possible model size and limited training resources.

While context-aware language models like BERT have achieved state-of-the-art performance on many NLP tasks, including SQuAD, their large size makes it difficult to host them in resource-constrained environments like mobile and edge devices. The smaller of the two models presented in the original BERT paper, $\text{BERT}_{\text{BASE}}$, has 110 million parameters with a total model size of 440 megabytes. A smaller model would be more ideal to avoid wasting limited storage space.

Unlike many other compressed models, we designed our model to be trainable with limited resources starting from the pre-trained BERT model. As described in the Related Work section, smaller sized BERT-like models have been designed by other authors but they often have a high training cost, even when using pre-trained BERT as a starting point. In particular, the training costs are far out of reach of a CS224n student with $150 of Azure credits. Models like ours with smaller training cost are more accessible and useful to people and organizations who do not have access to large computational resources.

A SQuAD model that fits on a mobile or edge device could have many possible applications. It could be used to provide offline question answering on e-books and documents stored on the device. This would be useful when the user does not have access to an internet connection. Even when the user does have an internet connection, an offline approach allows users to preserve their privacy when they do not wish to share their documents or questions with a third-party service. While we are choosing to focus in the SQuAD task, the techniques we use for size reduction may also be applicable to other NLP tasks that are assisted by BERT.

2 Related Work

Recent state-of-the-art NLP performance has been achieved through language model pre-training. One of the most prominent models is BERT [1], which is based on the Transformer [2] architecture and developed by Google.

Many papers have been published regarding size reduction of neural models. A summary can be found in this survey paper [3]. Techniques include quantization, parameter sharing, matrix compression, and knowledge distillation. These are all techniques that we explore.

Since BERT was published, various authors have attempted to train similar but smaller models that achieve comparable performance.

One such model is DistilBERT [4], developed by Hugging Face. On various tasks, the authors show that DistilBERT achieves about 97% of $\text{BERT}_{\text{BASE}}$ performance with 6 layers instead of 12 and about half the parameters. They trained DistilBERT by knowledge distillation from $\text{BERT}_{\text{BASE}}$ on 8 16GB V100 GPUs for approximately 90 hours.

ALBERT [5], a "lite" version of BERT developed by Google. $\text{ALBERT}_{\text{BASE}}$ achieves similar performance to $\text{BERT}_{\text{BASE}}$ while having about 90% fewer parameters because all 12 layers share the same weights. ALBERT was trained from scratch on 64 to 512 Cloud TPUs.

Another group from Google developed a model [6] that achieves similar performance to $\text{BERT}_{\text{BASE}}$ on various tasks while having 60x fewer weights. They train their model by knowledge distillation from $\text{BERT}_{\text{BASE}}$ and through a reduced "optimal" WordPiece vocabulary. This model was distilled on 32 Cloud TPUs in 2-4 days.

While the above approaches have been successful in training smaller models, one thing in common to all of them is the large amount of compute power required. In particular, they all require compute far in excess of $150 of Azure credits provided to CS224n students. This motives the main goal of this project, which is to train a compressed SQuAD 2.0 model based on BERT that achieves reasonable performance with the smallest possible model size and limited training resources.

3 Approach

We trained baselines and then implemented several techniques to compress the BERT SQuAD 2.0 model.

3.1 Baselines

Baselines were established by fine-tuning pre-trained $\text{BERT}_{\text{BASE}}$ and DistilBERT models. This fine-tuning was done with code provided by the Transformers library [7] from Hugging Face.

To answer SQuAD 2.0 questions, the method from the original BERT [1] paper is used. The question and answer are passed as a single input to BERT, separated by the `[SEP]` token. For each word with final hidden vector T_i, the probability that it is the start of the answer is computed as $P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$ where S is a newly introduced "start" vector. Similarly, we compute the probability that each word is the end vector using a newly introduced "end" vector E. Training loss during fine-tuning is the log-likelihood of the correct start and end positions. The model predicts "no-answer" by considering the span that just includes the `[CLS]` token which is always present at the start of the input.

3.2 Cross-layer parameter sharing

We implemented our own novel cross-layer parameter sharing technique, similar to ALBERT but trained from BERT with much fewer resources. The overall process is shown in Figure 1. Rather than train a new model from scratch, we first started from $\text{BERT}_{\text{BASE}}$ and fine-tuned the model on SQuAD 2.0 to get a 12-layer fine tuned model. We then forced parameter sharing between adjacent layers where the higher numbered layer replaces the lower numbered layer. For example, layers 3 and 4 both share the parameters of layer 4, and layer 3's old parameters are discarded. (Based on some informal experiments, using the upper layer performs better than the lower layer.) This model is then fine-tuned again, and the process is repeated to get smaller and smaller models.

The idea behind this method is that adjacent layers in the model should be similar, and that slowly merging layers with fine-tuning should allow the model to gradually learn to share the weights. This allows us to get parameter sharing without re-training a whole new model like ALBERT.

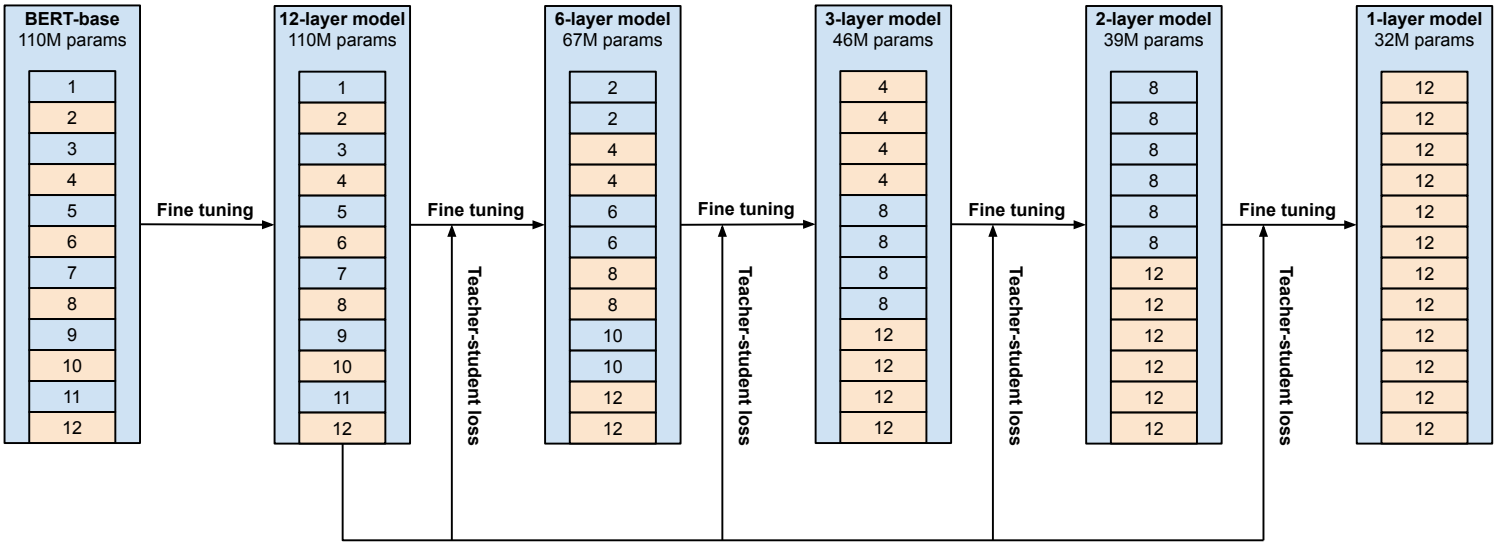

Figure 1: Cross-layer parameter sharing training procedure

3.3 Knowledge distillation during fine-tuning

Although knowledge distillation from scratch would probably have been too expensive to perform in a CS224n project, we found that the model performance is improved by adding a teacher-student loss to the fine-tuning procedure. This is inspired by the technique used in the original DistilBERT paper to achieve better performance on SQuAD 1.1. As shown in Figure 1, the initial model fine-tuned from $\text{BERT}_{\text{BASE}}$ is always used as the teacher. The teacher-student loss is computed as $\sum_i t_i \log s_i$ where t_i and s_i are the probabilities computed by the teacher and student respectively. We sum the loss for both the start-answer and end-answer probabilities, then average this with the regular fine-tuning loss (described above) to get the overall loss.

3.4 Reduced WordPiece vocabulary

Inspired by [6], we also reduced the WordPiece vocabulary used by BERT by removing the least common WordPieces. The original embedding table of 30K WordPiece tokens accounts for over 21% of the $\text{BERT}_{\text{BASE}}$ model size, so this results in a noticeable model size decrease. Note that a reduced WordPiece vocabulary is still capable of tokenizing all text, but it may require a greater number of tokens. The reduction in WordPiece vocabulary is only done at test time, as we were unable to get good results by fine-tuning with a reduced vocabulary. More exploration could be done in the future to see if fine-tuning with the reduced vocabulary can produce better results, perhaps with some tuning of hyper-parameters.

3.5 Linear quantization of tensors

We wrote our own code to perform simple linear quantization of all PyTorch tensors in the model. This allows the floating-point values to be scaled from 32-bit full precision to arbitrary-bit precision for storage. When the model is loaded, the compressed values from disk are then re-converted back to 32-bit for execution.

All values in a single tensor are scaled in the same way. We first compute minv and maxv, the minimum and maximum values in the tensor. Each value x in the tensor is compressed to a b-bit integer as $\mathrm{round}\left(\frac{x-\mathrm{minv}}{\mathrm{maxv}-\mathrm{minv}} \times \left(2^b-1\right)\right)$. This results in an integer in the range $[0, 2^b-1]$. When stored to disk, the compressed tensor needs to be stored together with minv and maxv (stored as 32-bit floats). Decompressing a compressed integer y tensor back to 32-bit floats is computed as $y \times \frac{\mathrm{maxv}-\mathrm{minv}}{2^b-1} + \mathrm{minv}$.

3.6 Random sketching of matrices

Another method for reducing the number of parameters is randomized sketching applied to the weight matrices, which has been previously applied to simple fully-connected and convolutional networks [8]. The concept of randomized sketching involves taking a high dimensional matrix and randomly projecting it to a lower dimensional matrix that preserves certain properties of original linear transformation. For example, given a weight matrix $W \in \mathbb{R}^{d\times n}$ and an input $x \in \mathbb{R}^n$, if n is large we can produce a sketching matrix $S \in \mathbb{R}^{m\times n}$ such that $(WS^T)(Sx) \approx Wx$. Then we could store weight as only $(WS^T) \in \mathbb{R}^{d\times m}$ and S. We choose S as a matrix with Rademacher random entries, i.e. each entry is ± 1 with probability $\frac{1}{2}$, and scale it by $\frac{1}{\sqrt{m}}$ since this means that $\mathbb{E}[S^TS] = I$. Additionally, storing such a matrix would be very cheap because each entry requires only one bit. We implement this operation as a custom PyTorch module that can replace any existing torch.nn.Linear layer.

4 Experimental Results

4.1 Data

We used the SQuAD 2.0 dataset provided for the CS224n project, which is different from the official SQuAD 2.0 dataset. Note that this means our results for SQuAD 2.0 are comparable with other CS224n projects but not most published papers.

4.2 Evaluation method

Our models are evaluated on the standard SQuAD 2.0 metrics, F1 and EM (exact match). F1 measures the harmonic mean of precision and recall between the predicted answer and ground-truth answer when considering them as a bag of words. EM measures the percentage of predicted answers that exactly match the ground truth.

4.3 Experimental details

All fine-tuning was done on the SQuAD 2.0 training set for two epochs. This took around 3 hours on an Nvidia P100 GPU. We used a batch size of 12, learning rate of 3×10^{-5} with the Adam optimizer. All F1/EM scores presented are for the CS224n dev set.

4.4 Results

We first obtained baselines by performing the above training procedure on pre-trained $\text{BERT}_{\text{BASE}}$ and DistilBERT. The results are shown in Table 1. DistilBERT (D) refers to DistilBERT fine-tuned with a teacher-student loss from the $\text{BERT}_{\text{BASE}}$ model, as described in the original paper and in the section above on knowledge distillation during fine-tuning. This improved the performance slightly.

	$\text{BERT}_{\text{BASE}}$	DistilBERT	DistilBERT (D)
# Params	110M	66M	66M
Size	440 MB	264 MB	264 MB
F1/EM	76.4/73.3	69.3/66.4	70.7/67.8

Table 1: SQuAD v2.0 baseline results

Next, we fine-tuned the $\text{BERT}_{\text{BASE}}$ model using our novel cross-layer parameter sharing technique shown in Figure 1. The results are shown in Table 2. As you can see, the model was able to maintain relatively good performance (>95% of original) with the 3-layer model while having a 58% reduction in the number of parameters. In particular, our models achieve higher performance than the DistilBERT baselines with fewer parameters. However, further reductions to the 2-layer and 1-layer model result in poor performance. Thus, we chose to proceed with the 3-layer model and apply the further size reduction techniques.

	12-layer model	6-layer model	3-layer model	2-layer model	1-layer model
# Params	110M	67M	46M	39M	32M
Size	440 MB	268 MB	184 MB	156 MB	128 MB
% Smaller than $\text{BERT}_{\text{BASE}}$	0%	39%	58%	65%	71%
F1/EM	76.4/73.3	76.8/72.6	72.7/69.7	57.8/54.4	50.3/46.1

Table 2: SQuAD v2.0 cross-layer parameter sharing results

Starting from the 3-layer model, we then proceeded to reduce the WordPiece vocabulary. The results are shown in Table 3. Since each vocabulary word requires a 768-dimensional embedding vector, this results in significant savings in the number of paramteters. Note that 4928 was the number of WordPiece tokens used in [6], which is why we decided to try it out specifically. The performance appears to sharply drop when less than 25% of the vocabulary is used, so we have chosen to proceed with the model with 7630 WordPiece tokens. It still achieves comparable performance to the DistilBERT baselines while having considerably fewer parameters.

# WordPiece tokens	30522 (100%)	16251 (50%)	7630 (25%)	4928 (16.1%)	3052 (10%)
# Params	46M	35M	28M	26M	25M
Size	184 MB	140 MB	112 MB	104 MB	100 MB
% Smaller than $\text{BERT}_{\text{BASE}}$	58%	68%	75%	76%	77%
F1/EM	72.7/69.7	71.7/68.6	70.2/67.0	66.8/63.3	60.7/57.9

Table 3: SQuAD v2.0 WordPiece vocabulary reduction results

The next step we did is linear quantization. Starting from the model above with 7630 WordPiece tokens, we applied varying levels of quantization and the results are shown in Table 4. 16-bit quantization did not affect the performance. Surprisingly, 10-bit quantization actually increases the performance slightly compared with the full 32-bit model, although examining the predictions, this appear to be a random effect and should be interpreted as the quantization does not affect the performance. At 6-bit and lower, the model performance drops sharply. Thus we have chosen to proceed with the 10-bit quantization for later experiments.

	Full	Quantization				
	32-bit	16-bit	10-bit	8-bit	6-bit	4-bit
# Params	28M	28M	28M	28M	28M	28M
Size	112 MB	56 MB	35 MB	28 MB	21 MB	14 MB
% Smaller than $\text{BERT}_{\text{BASE}}$	75%	87%	92%	94%	95%	97%
F1/EM	70.2/67.0	70.2/67.0	70.3/67.1	69.4/66.4	53.9/50.0	4.8/2.6

Table 4: SQuAD v2.0 linear quantization results

Finally, we tried applying the random sketching procedure to the model. The results are shown in Table 5. Since we noticed that we achieved better performance by training on the full vocabulary and testing on the reduced vocabulary, we started from our 3-layer model quantized to 10 bits and sketched each layer, then tested it on both the full and reduced vocabulary. Inside each transformer block, we used sketching to reduce the rank of the output layer to 512, reducing the 768×3072 matrix to 768×512. Additionally we sketched each of the 4 linear layer within the attention modules to 256, reducing 768×768 to 768×256. This reduced the total number of parameters by only about 9%, but many of the parameters remaining can be stored in 1 bit since we use Rademacher sketch matrices. So the storage space reduction is a bit larger. We see that with fine-tuning we can achieve essentially the same accuracy as before sketching.

	Original Full Vocab	Sketched Full Vocab	Original Reduced Vocab	Sketched Reduced Vocab
# Params	46M	42M	28M	24M
Size	58 MB	44 MB	35 MB	21 MB
% Smaller than $\text{BERT}_{\text{BASE}}$	87%	90%	92%	95%
F1/EM	72.7/69.7	72.3/69.1	70.2/67.0	70.3/66.9

Table 5: SQuAD v2.0 sketching results (all models with 10-bit quantization)

In our final test set submission, the sketched 3-layer model with 10 bit quantization using the full vocabulary achieved an F1 score of 73.079 and an EM score of 69.839.

5 Analysis

From the methods we tried, it looks like it is possible to significantly reduce the space required for the $\text{BERT}_{\text{BASE}}$ model while still retaining close to the original accuracy. We achieve a final reduction of 62% in parameter count and 90% in storage space with an F1/EM drop of about 3 points (96% of original performance), or a reduction of 74% in parameter count and 95% in storage space with an F1/EM drop of about 6 points (92% of original performance). However, we note that even though we have reduced the amount of space required, each inference still requires roughly the same amount of computation as the original $\text{BERT}_{\text{BASE}}$ since we did not actually reduce the number of layers.

In our approach, we relied heavily on modifying the pre-trained weights from the original $\text{BERT}_{\text{BASE}}$ and applying fine-tuning, which might have helped us achieve significantly better compression than the approaches used in DistilBERT and ALBERT. ALBERT was initialized and trained from scratch, and DistilBERT used the teacher weights as part of its initialization but also reduced the number of layers, which might have changed the model structure enough to require full re-training. Starting from the already-trained weights and preserving the same model structure seems to have allowed for fast convergence during the fine-tuning, whereas both of the other models required significant amounts of training to reach the same accuracy.

Additionally, aside from the vocabulary reduction the methods we used are mostly generalizable to other types of networks. The cross-layer parameter sharing might also be useful in other deep models that have multiple copies of the same layer type, which are fairly common in deep learning. Knowledge distillation may be used any time we have a fully trained original model, which is always the case in model compression. Quantization can be applied universally, since it is just a reduction in precision. Random sketching can be applied in any situation with matrix multiplication, which again is almost the entirety of deep learning.

6 Conclusion and Future Work

We implemented a variety of model compression techniques and were able to achieve F1 and EM scores close to the original $\text{BERT}_{\text{BASE}}$ model accuracy with significantly smaller storage space requirements and limited training resources. This suggests that even on such a complex model and task, there are still many redundant parameters that can be eliminated for a more lightweight model. The same likely applies to other large models in deep learning, so these compression techniques could be very useful as a preprocessing step before deploying any large model in a serious capacity. Our results also suggest that it is possible to significantly reduce BERT's model size without resorting to techniques with high training cost like those used to train DistilBERT and ALBERT.

Future work might include extending the amount of time spent on fine-tuning or tweaking the hyper-parameters, which we limited in this project due to time constraints. With further fine-tuning it might be possible to get closer to or even match the accuracy of the original model, which would mean the compressed model is strictly better than the original. It would be interesting to see if the compressed model shares the same expressiveness as the original, or if something is lost during the process. Many other model compression techniques have been described in the literature, such as in this survey paper [3], and those could be applied in the future. While we already have done some ablation analysis to determine the effects of various techniques, more analysis could be done on comparing the results of different models.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. arXiv:1810.04805.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. arXiv:1706.03762.

[3] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks, 2017. arXiv:1710.09282.

[4] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019. arXiv:1910.01108.

[5] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019. arXiv:1909.11942.

[6] Sanqiang Zhao, Raghav Gupta, Yang Song, and Denny Zhou. Extreme language model compression with optimal subwords and shared projections, 2019. arXiv:1909.11687.

[7] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[8] Shiva Prasad Kasiviswanathan, Nina Narodytska, and Hongxia Jin. Deep neural network approximation using tensor sketching, 2017. arXiv:1710.07850.