Micronet Challenge Report

WikiText-103 Language Modeling Track

Demi Guo, Song Han, Philip Isola, Hanrui Wang, Zhongxia Yan*

October 2019

1 Introduction

Natural language processing (NLP) has witnessed great success using deep neural networks. However, the explosively increased model size and computation complexity make it extremely difficult to perform the NLP tasks on edge devices with limited memory footprints and computation resources. In this challenge, we aim to design efficient language models with small models size as well as low computation complexity. Our model is based on the the general transformer family [1], and more specifically the Transformer-XL architecture [2]. The self attention layers in the transformer can receive the information from far-away tokens thus improving the ability to capture long-term dependency. We used relative positional embedding, trainable cache, Hebbian update [3], knowledge distillation to improve the performance and leveraged adaptive token embedding, adaptive softmax, pruning and quantization to shrink the model size.

2 Methods

To begin with, we apply the adaptive embedding and adaptive softmax to reduce the parameters of the first embedding layer and the softmax layer. The weights in those two layers are shared. we separate the tokens into three bins according to their occurring frequencies in the training set. The three bins contain the [0,3500), [3500, 25000) and [25000, 267735] tokens respectively. We select the embedding dimension to be 256.

The model contains eight decoder layers which are the modules proposed in [2] but modified to have fully differentiable memory at training time. Each layer has two parts, attention and Feed-Forward Network (FFN). The dimension of the $query, key, value$ vectors for each token are all 192. We use 8 attention heads. There is a trainable positional embedding added into each key vector. The attention looks the sequence of 96 tokens ahead. The FFN contains two FC layers. The inner hidden dimension is 768. This backbone model contains 8.3M parameters.

We apply Hebbian update and a trainable cache to improve the language model. Both methods are strategies to improve performance on infrequent words. Hebbian update directly interpolates the output embedding with the last hidden activation, and gradually anneals this effect to zero. This memorizes rare activations more strongly than gradient descent. Trainable cache teaches the model to refer to previously seen tokens for making predictions. This helps with rare words as well,

*Alphabetical Order. Demi Guo is with Havard Universtiy. Song Han, Philip Isola, Hanrui Wang, Zhongxia Yan are with MIT.

as the probability of a rare word reoccurring is much higher than the prior probability. With solely cache and Hebbian techniques, the test perplexity is around 34 before pruning and quantization.

To further boost the performance. We apply the knowledge distillation [4] method to transfer the knowledge from a large teacher model to the small model. The teacher model is trained on the same dataset but with much larger model size (53M parameters). The test perplexity of the teacher model is around 22. We generate the probability of the top 30 soft labels of the teacher and train the student model jointly with the hard label loss and soft label loss. The teacher annealing method is applied to gradually increase the weights for hard label loss and reduce the soft label loss. The rationale behind that is that the teacher model is imperfect, albeit it has very low perplexity. Therefore after the student learns enough knowledge from teacher, it should learn from the "truth" – the hard labels. With the knowledge distillation, we further reduce the test perplexity of the student model to 31.7.

Furthermore, we leverage pruning and quantization approaches [5] to trade accuracy for smaller model size. We first conduct a sensitivity analysis: for each parameter, we measure the perplexity performance resulted from pruning each weight or bias with different ratios. Then, for an arbitrary target total pruning ratio, we employ a heuristic method based on the sensitivity analysis. We choose the pruning ratio for each individual so that more sensitive parameters (e.g. first token bin of the adaptive embedding) will be pruned less. Specifically, given a perplexity threshold P, for each parameter, we binary search the largest pruning ratio of the parameter such that the resulting perplexity is still below P. We binary search P so that the total pruning ratio is desired. We use Automated Gradual Pruning method and initialize the model with our best model after knowledge distillation (test ppl 31.7). We started with a total pruning ratio of 20% and ended with around 35%. The final perplexity after pruning is around 34. For quantization, we apply Quantization Aware Training with linear-range symmetric fake-quantization for weights, biases, and activations to either 8 bit or 9 bits; to comply with the tournament's fake quantization rules, we ensure that the last 8 or 9 bits of the inverse scale factor's mantissa are zero, so that when multiplied with any quantized value the resulting value can still be exactly represented in fp32. We use the distiller package [6] to perform training for both pruning and quantization. With pruning ratio 33.85% and quantization to 8 bits, we achieved 34.95 perplexity and our best score 0.0475.

3 Experimental Results

3.1 Model Size

We first calculate the model size. We get the size and sparsity of each layer and then calculate the total non zeros numbers. Because we applied the quantization by ourselves, so the freebie is not applicable. Each quantized non-zero weight is counted as $\frac{bit\_width}{32}$ parameter.

Moreover, there is one bit mask for each weight and each bit-mask is counted as $\frac{1}{32}$ parameter. We marked our model size as p.

3.2 FLOPs

Comparing to the computation of the model size. The FLOPs is more difficult to calculate. Here we list the calculation of different kinds of operations in different layers of the model.

For adaptive embedding layer, embedding of different tokens has different FLOPs. For tokens in the first bin, the 256-dimension embedding is indexed directly from the matrix, therefore no FLOPs. For tokens in the second bin, a 64-dimension embedding is indexed from the matrix and scaled to 256-dimension with matmul of $64 * 256 + (64 - 1) * 256 = 32512$ FLOPs. Similarly, for

the tokens in the third bin, a 4-dimension embedding is indexed and then scaled with matmul of $4 * 256 + (4 - 1) * 256 = 1792$ FLOPs. In the test set, there are 198232, 35479 and 11858 tokens in the three bins respectively. We averaged the FLOPs among the test set for embedding layer.

After that there are 8 identical decoder layers. Firstly, a layer norm has 256, 256*2+255+1, 256 and 256 FLOPs to calculate mean, variance, apply mean, variance respectively. In total 1536 FLOPs. Then we have a matmul to compute the q, k, v of size $256*192*3+256*192*3 = 294912$ FLOPs. After that, since the attention has 8 head, each head has length q', k', v' of 24 and looks ahead for 96 tokens, the matmul of $q*k$ has in total $8*24*(96+1)+8*(24-1)*(96+1) = 36472$ FLOPs. The we calculate the positional embedding and add to the attention values. Note that the positional embedding is the same for different head. Therefore, it has $24 * (96 + 1) + (24 - 1) * (96 + 1) + (96 + 1) = 4656$ FLOPs. Then the softmax of 97 values has $(97 + (97 - 1) + 97 = 290)$. Then we multiply the softmax values with the v vector of the tokens, which costs $8 * 24 * 97 + 8 * 24 * (97 - 1) = 37056$ FLOPs. Then one matmul project the concatenation of different heads to the 256-dimension, which costs $192 * 256 + (192 - 1) * 256 = 98048$ FLOPs. Then residual connection costs 256 FLOPs. Another layer normalization costs 1536 FLOPs. Then a FC layer has a matmul with bias of $256 * 768 + 256 * 768 = 393216$. ReLU activation costs 768 FLOPs. Second FC layer projecting back to 256-dimension has $256 * 768 + 256 * 768 = 393216$ FLOPs. Finally a residual connection consumes another 256 FLOPs.

After eight decoder layers, an adaptive softmax layer is used to get the token distribution. It contains one matmuls for the first bin and it has $256 * 3502 + 256 * 3502 = 1793024$ FLOPs. Then the 3502-dimension softmax has $3502 * 3 - 1 = 10505$ FLOPs. It also has two matmuls for the second bin and they have $256 * 64 + (256 - 1) * 64 + 64 * 21500 + 64 * 21500 = 2784704$ FLOPs. The softmax for the second bin has $21500 * 3 - 1 = 64499$ FLOPs. The last bin has $256 * 4 + (256 - 1) * 4 + 4 * 242735 + 4 * 242735 = 1943924$ FLOPs. The softmax for the third bin has $242735*3-1 = 728204$ FLOPs. The trainable cache for each token takes $(256-1)*2000+2000*3-1 = 515999$ FLOPs.

Since we apply the pruning, we have different sparsity for different layers. Therefore, we profile the sparsity of the weights and bias of each layer of the model and then reduce the FLOPs according to the sparsity. Please see the details in the IPython notebooks in the GitHub repository.

Then the quantization takes effects and the number of most multiplications is reduced to $\frac{bit\_width}{32}$. Since there exists a trade-off between the pruning ratio and the quantization bit-width. Concretely, larger pruning ratio should be combined with larger quantization bit-width to preserve the perplexity. We conduct multiple combinations of different pruning ratio and bit-widths. The total FLOPs of the model is marked as q.

3.3 Score Calculation

According to the model size and FLOPs calculated above, the score t is,

$$t = \frac{p}{159} + \frac{q}{318} \tag{1}$$

For the best model with pruning ratio 33.85% and quantization bit-width 8 bits, the model size is 1.631M and the FLOPs is 11.85M. Therefore the best score is 0.0475. With pruning ratio 42.12% and quantization bit-width 9 bits, we achieved score 0.0482. With pruning ratio 40.12% and quantization bit-width 9 bits, we achieved score 0.0485.

4 Conclusion

In this report, we summarize the techniques we used in the MicroNet Challenge wikitext-103 language model track. We applied trainable cache, Hebbian update, knowledge distillation, pruning and quantization to get a model with 1.631M parameters, 11.85M FLOPs and 34.95 test perplexity. The final score of the model thus is calculated as 0.0475.

5 Acknowledgment

We sincerely thank the MicroNet Challenge organizer for organizing the event and actively answer our questions in the Google discussion group. We also thank many labmates at MIT for valuable discussions.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 1

[2] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv e-prints*, page arXiv:1901.02860, Jan 2019. 1

[3] Jack W Rae, Chris Dyer, Peter Dayan, and Timothy P Lillicrap. Fast Parametric Learning with Activation Memorization. *arXiv e-prints*, page arXiv:1803.10049, Mar 2018. 1

[4] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[5] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[6] Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. Neural network distiller, June 2018. 2