Bitmaps - Binary BERT
CSE 481 N

Tobias Rohde, Ani Canumalla

June 2020

Abstract

In this paper we introduce a method for reducing the memory usage of BERT by representing 32-bit floating point weights as 1-bit quantities; a process called binarization. Our approach leads to a decrease in memory usage by a factor of up to 30, but at the cost of decreased performance. We evaluate our binarization approach for BERT on the masked language modeling pre-training objective as well as the downstream task of binary sentiment analysis on SST-2. This project was done as part of the Natural Language Processing Capstone (CSE481N) taught by Noah Smith in Spring 2020 at the University of Washington, Seattle. All code as well as a series of blog posts describing the progression of the project can be found at `https://github.com/nlp-capstone`. A presentation of our project can be found at `https://vimeo.com/426452840`.

1 Introduction

Devlin et al. [2019] introduced the language representation model BERT, which is based on the transformer architecture by Vaswani et al. [2017]. BERT achieved state-of-the-art performance on many Natural Language Processing tasks and inspired many new pre-trained transformer models, which have pushed state-of-the-art results even further. However, pre-trained transformer models are extremely large. BERT-base and BERT-large have 110 million and 340 million 32-bit precision parameters, respectively. More recent architectures have grown to billions of parameters. This makes inference memory intensive, computationally expensive, and slow. This prevents transformer models from running on edge devices in real-time. Table 1 shows the number of parameters per component in BERT-base (uncased) and the respective size in mebibytes. Most of the parameters are in the token embeddings, the query, key and value matrices and the large feedforward networks (FFNN) at the end of each transformer layer. In this paper, we describe an approach for binarizing those parameters to reduce inference time and memory usage. Network binarization has been successfully applied to convolutional neural networks (CNN) by Rastegari et al. [2016] and we we adapt their methodology for pre-training and fine-tuning BERT. Zafrir et al. [2019] published results for an 8-bit version of BERT, but to our knowledge, no prior work on binarization for transformers exists.

2 Weight binarization

Binarization refers to the process of turning high precision weights into 1-bit quantities. We use the same approach as Rastegari et al. [2016]. Since any weight tensor can be flattened into an n-dimensional vector, we assume without loss of generality that $W \in \mathbb{R}^n$ is a weight vector. To approximate W, we find a scalar $\alpha \in \mathbb{R}^+$ and a vector $B \in \{-1, 1\}^n$ such that $\widehat{W} = \alpha B \approx W$ by solving the following optimization problem

$$\min_{\alpha, B} \|W - \alpha B\|_2^2, \quad \text{where } B_i \in \{-1, 1\} \text{ and } \alpha \in \mathbb{R}^+. \tag{1}$$

Component	Number of parameters	Size in MiB
Token embeddings	23440896	89.42
Positional embeddings	393216	1.5
Token type embedding	1536	0.01
Embeddings Norm	1536	0.01
Query Transform	12×590592	27.03
Key Transform	12×590592	27.03
Value Transform	12×590592	27.03
Attention Transform	12×590592	27.03
Attention Norm	12×1536	0.07
FFNN Transform 1	12×2362368	108.14
FFNN Transform 2	12×2360064	108.04
FFNN Norm	12×1536	0.07
Total	108891648	415.39

Table 1: Number of parameters per component in BERT-base without masked LM head (12 layers).

Expanding the objective, we get

$$\min_{\alpha,B} \|W - \alpha B\|_2^2 = \min_{\alpha,B} (W - \alpha B)^\top (W - \alpha B) = \min_{\alpha,B} W^\top W - 2\alpha B^\top W + \alpha^2 B^\top B. \tag{2}$$

By noting that $W^\top W$ is constant and that $B^\top B = \sum_{i=1}^{n} B_i^2 = n$ we can write (1) as

$$\min_{\alpha,B} -2\alpha B^\top W + \alpha^2 n. \tag{3}$$

We first find the optimal B by fixing α and dropping terms not involving B:

$$\min_{B} -2\alpha B^\top W = \max_{B} \sum_{i=1}^{n} B_i W_i \tag{4}$$

Since we can choose each B_i independently to be -1 or 1, the optimal solution is given by $B = \text{sign}(W)$, since then each $B_i W_i = |W_i| \geq 0$ is maximized. Next we find the optimal α by fixing B and equating the derivative to 0:

$$0 = -2B^\top W + 2\alpha n = -B^\top W + \alpha n. \tag{5}$$

Thus

$$\alpha = \frac{1}{n} B^\top W = \frac{1}{n} \text{sign}(W)^\top W = \frac{1}{n} \|W\|_1. \tag{6}$$

The binarization can be summarized as

$$\alpha = \frac{1}{n} \|W\|_1 \tag{7}$$

$$B = \text{sign}(W) \tag{8}$$

$$\widehat{W} = \alpha B \approx W \tag{9}$$

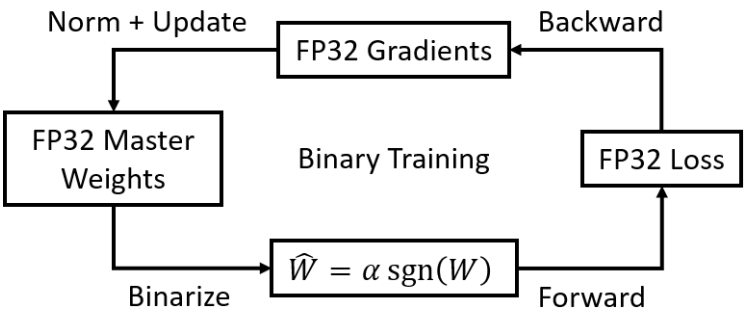

Figure 1: Binary training procedure. FP32 weights are binarized and used in the forward pass to compute the loss. Backpropagation is used to compute FP32 gradients of the loss, which are optionally normalized and used to update the FP32 weights.

The error of this approximation is given by

$$\begin{aligned}
\|W - \alpha B\|_2^2 &= W^\top W - 2\alpha B^\top W + \alpha^2 B^\top B \\
&= \|W\|_2^2 - \frac{2}{n}\|W\|_1 B^\top W + \left(\frac{1}{n}\|W\|_1\right)^2 n \\
&= \|W\|_2^2 - \frac{2}{n}\|W\|_1^2 + \frac{1}{n}\|W\|_1^2 \\
&= \|W\|_2^2 - \frac{1}{n}\|W\|_1^2 \qquad (10)
\end{aligned}$$

We binarize all layers except for the embedding and normalization layers. We use 16-bit floating point precision (FP16) to reduce the memory usage of the embeddings. We also attempt to binarize embeddings, as discussed in 7.2. Since layer norms contain few parameters we leave them in full-precision. This approach reduces memory usage from 415MiB to 56MiB. With binarized embeddings, BERT shrinks to 14MiB.

3 Training

During pre-training and fine-tuning, we do not directly update binary weights, since gradients are usually smaller than 1 and would thus have no effect on weights that can only be -1 or 1. Instead, we maintain and update 32-bit master weights (see Figure 1). The master weights are initialized with the pre-trained weights of the original BERT model. Prior to training, all α's are initialized according to equation (7). During training, all α parameters are learned and we only use equation (8) to get B and equation (9). We found that learning α significantly sped up training and led to better performance. By untying α from W during training we allow the network to learn new representations that are more suitable for binarized weights than those of the original network. Furthermore, a single weight update to α changes the entire weight tensor, which results in fast learning. In the forward pass of BERT, we binarize the master weights according to equation (9). The binarization becomes part of the computation graph and in the backward pass gradients are automatically computed with respect to the master weights and α. Note that we use the sign function in the forward pass, which has derivative 0 everywhere except at 0, where it is undefined.

To prevent gradients from becoming 0, we use

$$\frac{\partial \operatorname{sign}(w)}{\partial w} = \begin{cases} w & |w| \leq 1, \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

for the derivative of sign [Rastegari et al., 2016]. Once the binary model is trained, we compute the binary weights using equation (9) and remove the binarization code from each binarized module and replace the master weights with the binary weights. In our current implementation, the binary weights are still stored as full-precision weights to make implementation simpler. In real world use, binary weights should be stored as bits.

3.1 Pre-Training

We pre-train BERT with binary weights to see if this helps fine-tuning with binary weights on downstream tasks. We optimize the masked language modeling (MLM) objective, which is to predict a token that was masked from a sequence. We do not use the next sentence prediction (NSP) objective since it does not improve performance on downstream tasks [Liu et al., 2019]. For pre-training, we use a subset of the original data used for pre-training BERT (Wikipedia + BooksCorpus). We use 65536 sequences of length 128 as training data and a subset of 8192 sequences of length 128 as validation data. We are not using the full training data since we are transferring weights from the pre-trained model and because of a lack of computational power. As in the original BERT paper, we mask 15% of tokens. 80% of masked tokens set to the mask token, 10% are set to a random token and 10% are not changed. We follow Liu et al. [2019] and mask tokens dynamically. We use Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1\text{e}{-6}$, a batch size of 128 and a peak learning rate of $5\text{e}{-4}$. The learning rate is warmed up linearly for 8 epochs and decayed linearly for 56 epochs. We normalize gradients to have L_2-norm of 1.0. To reduce training time and decrease the size of the embeddings we train with FP16 and a loss scale of 2048.

3.2 Fine-tuning on SST-2

To test downstream performance of binary BERT, we train and evaluate it on SST-2. We binarize the pooling layer and the classification head, since we found that keeping them at full-precision does not improve performance. We use AdamW for optimization with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1\text{e}{-6}$, weight decay of 0.1, batch size of 32 and a peak learning rate of $5\text{e}{-5}$. The learning rate is linearly warmed up for the first epoch and decayed for 9 epochs. We normalize gradients to have L_2-norm of 1.0 and train with FP16 with a loss scale of 1024.

We attempt three different approaches for fine-tuning on SST-2:

1. Pre-trained binary BERT $\rightarrow$ binary fine-tuning on SST-2.
2. Pre-trained full-precision BERT $\rightarrow$ binary fine-tuning on SST-2.
3. Pre-trained full-precision BERT $\rightarrow$ full-precision fine-tuning on SST-2 $\rightarrow$ binary fine-tuning on SST-2.

We perform the above experiments with and without binarized embeddings. For binarizing embeddings we use a different α for each hidden dimension as desribed in appendix 7.2.

Approach	Loss	Accuracy	MRR	Size in MiB
Full-precision BERT	2.0653	0.6328	0.7049	415.39
Binary (FP16 embeddings)	2.9932	0.4796	0.5625	55.74
All binary	3.5398	0.4113	0.4952	13.83

Table 2: Validation metrics from pre-training binary BERT.

Fine-tuned from	Loss	Accuracy	Size in MiB
Full-precision BERT	0.3043	0.9174	415.39
Binary BERT	0.5683	0.8188	55.74
Pre-trained full precision BERT	0.4434	0.8222	55.74
SST-2 fine-tuned full precision BERT	0.5168	0.8693	55.74
Binary BERT (binary embeddings)	0.4905	0.7683	13.83
Pre-trained full precision BERT (binary embeddings)	0.4668	0.7936	13.83
SST-2 fine-tuned full precision BERT (binary embeddings)	0.4142	0.8532	13.83

Table 3: Validation metrics from fine-tuning binary BERT using different approaches. The binarized pooling layer and classification layer only consume 0.07MiB and are not included in the size column.

4 Results

4.1 Pre-training

Table 2 shows the cross entropy loss, accuracy and mean reciprocal ranking (MRR) and the size of the given model in MiB. The accuracy is the fraction of correctly predicted masked tokens. We use mean reciprocal ranking to account for cases where the model does not predict the correct token as the highest scoring token, but still assigns a high score to it. For efficiency reasons, we only check the top 32 scoring tokens when computing the MRR.

4.2 Fine-tuning on SST-2

Table 3 shows the binary cross entropy loss, accuracy and size of the given model for each fine-tuning approach. The last approach, in which we first fine-tune BERT on SST-2 with full-precision and then fine-tune again using binary weights performed better than our alternatives with and without binarized embeddings. The accuracy of our best performing binary model is only less than 5% below the accuracy of full-precision BERT.

5 Future Work

In our work we attempted to binarize either all weighs in BERT or all weights except the embeddings. Adding more binarization reduces the memory usage and computational cost of BERT, but naturally leads to a decrease in accuracy. For both pre-training and fine-tuning on SST-2, the drop in accuracy is currently too large to be practical. This tradeoff should be explored in future work to more closely match the performance of the original BERT model on a larger selection of tasks. Currently, we are only evaluating the memory usage of binary BERT. With $\{-1, 1\}$ weights, matrix multiplication can be done without multiplication but only addition and negation. This will make inference faster and is an important next step to evaluating the benefits of binary BERT.

6 Conclusion

In this work we have shown how BERT can be retrained to use only binary weights, while still achieving decent performance on the MLM pre-training objective and binary sentiment analysis. We show that for binary sentiment analysis, it is best to first fine-tune with full precision and then fine-tune again with binary weights. Binary BERT has a memory usage of 56MiB if embeddings are FP16 and 14MiB if embeddings are binarized, which corresponds to a decrease in memory usage by a factor of 7 and 30 respectively.

Acknowledgments

We thank Noah Smith, Tal August, and Kelvin Luu for their guidance and mentorship throughout this project. We also thank the members of Team SLIM (Sophie Tian, Louis Maliyam, Isaac Paang, Mitchell Otis Estberg) and Team NULL (Daniel Hernandez, Joshua Ip, Jerome Paliakkara, Norton Pengra) for providing feedback on our blog posts and this report.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

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

Jiaolong Xu, Peng Wang, Haishun Yang, and Antonio M. López. Training a binary weight object detector by knowledge transfer for autonomous driving. *2019 International Conference on Robotics and Automation (ICRA)*, pages 2379–2384, 2018.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. *ArXiv*, abs/1910.06188, 2019.

Approach	Loss	Accuracy	MRR	Size in MiB
Single-scalar	3.5497	0.4098	0.4941	13.82
Multi-scalar (hidden size)	3.5398	0.4113	0.4952	13.83
Multi-scalar (vocab size)	3.7577	0.3802	0.4648	13.88

Table 4: Validation metrics from pre-training binary BERT with binary embeddings.

7 Appendix

7.1 Alternative binarization

The current binary weight approximation $\widehat{W} = \alpha B \approx W$ cannot approximate weight matrices that have non-zero mean very well, since each weight will be approximated as either $-\alpha$ or $+\alpha$. In order to better approximate weight matrices, we attempted adding a scalar γ such that $\widehat{W} = \alpha B + \gamma \approx W$, thus each weight will be either $-\alpha + \gamma$ or $\alpha + \gamma$. We found that introducing γ initially makes learning faster, but eventually reaches the same accuracy as without γ. We believe that this is because often times parameters have a mean close to 0 and thus γ is not necessary.

7.2 Multi-scale binarization for embeddings

In addition to using a single scalar α to approximate the embedding matrix, we attempted two additional approaches. Firstly, we attempt using a different scaling factor for each hidden dimension. Let $E \in \mathbb{R}^{h \times v}$ be the embedding matrix, where h is the hidden size (768 in BERT-base) and v the vocabulary size (30522 in BERT-base). Then we use h scaling factors $\alpha \in \mathbb{R}^h$ and approximate E as:

$$\widehat{E} = \alpha \mathbf{1}^T \circ \text{sign}(E), \tag{12}$$

where $\mathbf{1} \in \mathbb{R}^v$ is a vector of 1s. Here $\circ$ represents elementwise multiplication. We also attempted using a scaling factor per token in the vocabulary using $\alpha \in \mathbb{R}^v$ and approximating E as

$$\widehat{E} = \mathbf{1} \alpha^T \circ \text{sign}(E), \tag{13}$$

where $\mathbf{1} \in \mathbb{R}^h$ is a vector of 1s. Results of this approach are shown in Table 4. As before, we learn α during training and only use (7) for initialization, where now the L_1 norm is computed along either the hidden dimension or vocabulary axis of the embedding matrix. The approach with one scalar per hidden dimension performed the best amongst our three approaches. We suspect that this because each hidden dimension can be seen as a feature of a token and this binarization approach adds the most variance across features. However, the single scalar approach performed almost equally well.

7.3 Knowledge Transfer

To improve learning we attempted transferring knowledge from the intermediate hidden states of the transformer layers. During training, we use the pre-trained full-precision BERT as the teacher model. At each step, we evaluate the teacher model and extract the hidden states at each of the 12 transformer layers. We will call those $H_i^{(t)}$, where $1 \leq i \leq 12$. Similarly, we extract the hidden states of the binary student model we are training, $H_i^{(s)}$.We then add the following transfer loss term to the MLM loss:

$$\sum_{i=1}^{12} \lambda_i \| H_i^{(t)} - H_i^{(s)} \|_2^2 \tag{14}$$

where λ_i are hyperparameters that specify how much we penalize the L_2-norm squared of the difference in hidden states at layer i. This approach was inspired by Xu et al. [2018]. We were not able to reduce the transfer loss and suspect that the binarized network learns different intermediate representations than the full-precision model. Instead of using the L_2-norm squared as a similarity metric, we also used cosine similarity, however results were similar. Instead of transferring hidden states, we also attempted transferring the intermediate attentions at each layer. For the similarity metric we tried using the L_2-norm squared, cosine similarity and the Kullback-Leibler divergence. However neither approach improved training.