CSE 481N: NLP Capstone: Blog Post 8

Team name: bitmaps - **bi**narized **t**ransformers for **m**aking f**a**st **p**rediction**s**
List of members: Tobias Rohde, Anirudh Canumalla
GitHub URL: nlp-capstone

Introduction

In this blog post, we present our most recent advances in binarizing BERT. In the last blog post we reported on the new Wikipedia + BooksCorpus subset we now use for training. In this blog post we present result from training for longer on this dataset and changes made to our binarization approach.

New baseline for training on Wikipedia + BooksCorpus

We used the same approach as before, where we keep the embedding layer and the language modeling head in full-precision and binarize all weights in the transformer layers, except for layer normalization modules. After 32 epochs, we were able to reduce the loss from 7.2543 to 4.1999 and a mean reciprocal ranking of 0.4085 and an accuracy of 0.3260, which beat any of our earlier approaches by a wide margin. This model served as our baseline for next experiments. Note that key experiments from this blog post are summarized in **Table 1** on page 3.

Binarizing multi-head (self) attention

Each layer in BERT performs multi-head attention. Each of the 12 heads computes its own attention distribution over the input sequence to the layer. In the huggingface implementation of BERT, the computation of the 12 key, query and value matrices is done efficiently in a single matrix multiplication. This lead us to accidentally assign a single scaling factor to the entire matrix. We believe that this made it more difficult for the attention heads to learn different representation. We fixed this by introducing a new linear module that can binarize a single matrix with multiple scaling factors. Note that this increases memory usage slightly, since we now have 12 weight scalars and 12 bias scalars for each multi-head attention module, rather than just 1 each. However, this is negligible since there are only 12 multi-head attention modules.

Learning scaling parameters

Recall from earlier that we approximate a weight matrix W using scaling factor $\alpha = \frac{1}{n}||W||_1$ and $B = \text{sign}(W)$. We attempted to learn α instead of tying it to the original weight matrix W. For initialization, we still use $\frac{1}{n}||W||_1$ but then just made α a parameter that gets learned as part of the model. This improved learning immensely. After only a single epoch of training, we beat the metrics from before. After 40 epochs of training, this model reduced the loss to 2.8497 and achieved a mean reciprocal rank of 0.5580 and an accuracy of 0.4687. After tuning the learning rate more, we managed to get close to this accuracy within just a few epochs. Intuitively, adjusting α now adjust the entire weight matrix, which is likely why this sped up training so significantly.

New attempts at knowledge transfer

Recall that we attempted to transfer intermediate hidden states at each layer to the new model by computing a the squared error between the hidden states of original BERT and binary BERT and penalizing the model according to a weighted sum of these losses at each layer. Formally, we can write this loss as

$$\mathcal{L}(H^{(t)}, H^{(s)}) = \sum_{i=1}^{12} \lambda_i ||H_i^{(t)} - H_i^{(s)}||_2^2,$$

where $H_i^{(t)}$ are the hidden states output by the ith layer of the teacher model and $H_i^{(s)}$ are similarly defined but for the binary student model. Adding this loss to the main masked language modeling objective did not improve performance. Even after trying many different combination of λs, this did not improve performance of the model. We also attempted to only penalize the hidden states corresponding to the tokens that are masked for prediction, but this did not work either. Next we tried to penalize according to the cosine similarity, which might make the hidden states more similar in terms of direction, while ignoring their magnitude. Again, this did not improve performance. Next we attempted transfer the attentions rather than the hidden states. We used mean squared error and kl divergence to penalize the different attention distributions corresponding to masked tokens. While we were now able to reduce this loss, neither of the two losses improved performance. This led us to set knowledge transfer aside for now.

Learning rate scheduling

We ran several experiments to tune the learning rate. We found that we can start with a large learning rate (5e-4) and reduce the loss quickly in a few epochs, but then the training slows down significantly. We attempted to reduce the learning rate if the loss did not improve after an epoch, but found this to be too slow since the loss decreases slowly for a long time. We attempted a simple step function for scheduling the learning rate, but found this to not work well either. We chose to use a similar approach as used with the original BERT, which is to linearly increase the learning rate until the peak value (warmup) and then to decrease the learning rate linearly towards 0. We chose to warmup for 4096 steps (8 epochs) and then decay for 28672 steps (56 epochs), for a total of 64 epochs of training.

Regularization, dropout and overfitting

We decided to reduce the dropout used in BERT from 0.1 to 0.05. The binarization acts a form of regularization itself, since we heavily restrict weights. We also use a dynamic masking approach as in the ROBERTA paper (unlike in the BERT paper), which means that we dynamically mask tokens during training rather than creating a fixed training set of masked tokens. Thus the network will most likely never see a particular sequence with the same tokens masked out twice, making it much more difficult to overfit.

16-bit training

Originally we believed it would be too difficult to get 16-bit training to work, since it requires keeping "master" weights in full-precision that are updated, similarly to how we train the binary weights. However, we set up NVIDIA apex and were able to get 16-bit training to work. This reduced the training time per epoch from 12 minutes to 4 minutes and allows us to use a larger batch size (now 128) without running out of CUDA memory. 16-bit training has been invaluable for quickly running experiments.

Shifting the mean of binarized weights

With our current binarization scheme of the form $\alpha B \approx W$, there are two possible values for each individual weight: $-\alpha$ and $+\alpha$. This approximation is not able to accurately represent parameters where the magnitude of the positive weights is much different from the magnitude of the negative weights. For example, consider a simple weight vector containing the two weights -2 and 8. Our current approach would approximate this as $\frac{|-2|+|8|}{2} \cdot [-1, 1] = [-5, 5]$, which is not ideal. Thus we decided to introduce a second offset parameter γ. We initialize the parameter with the mean of the given set of weights

$$\gamma = \frac{1}{n} \sum_{i=1}^{n} W_i$$

and learn it afterwards. We also adjust the initialization of α to be the mean L1 norm of the demeaned weights:

$$\alpha = \frac{1}{n} ||W - \gamma||_1$$

Our binarization scheme becomes

$$\alpha B + \gamma \approx W.$$

In the previous example, we would now have $\gamma = \frac{-2+8}{2} = 3$, and $\alpha = \frac{|-2-3|+|8-3|}{2} = 5$. Our approximation becomes $\alpha B + \gamma = 5 \cdot [-1, 1] + 3 = [-2, 8]$, which are the original weights. Note that this introduces another full precision parameter per weight matrix. We found that this approach improves accuracy. By learning α, γ, using 16-bit training and reduced dropout we achieved a cross entropy of 2.5130, an accuracy of 0.5132 and a mean reciprocal ranking of 0.6019. The memory usage was reduced to 56MB, since embeddings and language modeling head are now 16-bit. Although these numbers are very high, they are from training data and we are now seeing signs of overfitting. This is discussed further in the next steps section below. Below are the results of all experiments summarized:

Model	Cross Entropy	Accuracy	MRR	Memory (MB)
Original	2.2025	0.6210	0.6968	417
New baseline	4.1999	0.3260	0.4085	103
Learning α	2.6720	0.4932	0.5818	103
Learning α, γ + 16-bit	2.5130	0.5132	0.6019	56

Table 1: Model Performance on the BooksCorpus + Wikipedia training subset

Next steps

An important next step is to more rigorously evaluate the actual performance of our approaches using holdout data. So far our focus has been to reduce the training loss without worrying about generalization. The results above are very promising and we have evaluated some of our approaches on data not seen during training. So far we found that our approaches generalize well for new data, but especially after long training the binary BERT still seems to overfit slightly. We also think we're close to reaching the maximum performance one can achieve on masked language modeling with our current approach and would like to see how well fine-tuning binary BERT for a specific task works. Currently we are considering fine-tuning on Squad. So far we have also not performed an in-depth error analysis, since our main focus has been training instead of evaluation. However, we looked at some samples the model gets wrong and found that it often mis-predicts parts of word pieces. To counteract this we will consider always masking out full words instead of word pieces. We also found that sometimes tokens close to each other are masked, which causes the model to lack context and mis-predict the masked tokens. We will consider adjusting how we mask out tokens and perhaps not mask too many neighboring tokens. It might not be particularly useful to perform error analysis on the pre-training objective (masked language modeling), but once we fine-tune binary BERT on a specific task we will perform more in-depth analysis.