CSE481 NLP Capstone: Blog Post 5

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

Evaluation of masked language modeling task

For evaluation we are now using the validation data of the Penn Tree Bank (PTB) dataset. We are preprocessing the data in order to remove artifacts from the PTB tokenization and to prepare it for masked language modeling. We generated our evaluation dataset by masking out random tokens as described in the BERT paper:

15% of the tokens are randomly selected for masking.

- 80% of those tokens are replaced with the [MASK] token
- 10% of those tokens are replaced with a random token
- 10% remain unchanged, but the model still has to predict them

Note that the start, end and padding tokens are never masked. The model's task is now to predict the masked tokens[1]. We consider 5 metrics for evaluating the performance of the model:

1. Cross Entropy/Perplexity: We calculate the cross entropy between the true word that was masked and the predicted word. Cross entropy is also the loss function BERT uses during training.

2. Accuracy: The number of correctly predicted tokens (the token the model assigned the highest probability to) divided by the total number of tokens that were masked.

3. Mean Reciprocal Rank: We calculate the mean reciprocal rank to account for the fact that the model might predict the correct token as its second best choice, which the accuracy metric lacks. We only consider the top 32 tokens to speed up computation of this metric. Since 1/n is very small if n > 32, the effect of this optimization is minimal.

4. Inference time: Average time in milliseconds for making predictions for a sequence of length 64. We measure this time on a Titan RTX GPU. The time is measured per batch (batch size 128).

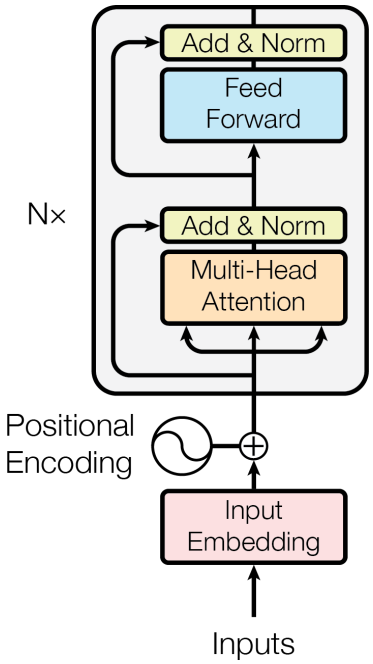
[1] We refer to all tokens that were selected for prediction as *masked tokens*, although they might be replaced with the masked token, randomly changed or left unchanged.

5. Memory usage: Memory usage of the weights of the model in MB (Mebibytes). Note that we calculate “pseudo” memory usage of the model by checking if a weight matrix contains at most two distinct values. If it does, we consider the size of each weight to just be 1 bit. Since our implementation still uses 32 bit floating numbers for the binary weights, this is necessary to calculate the hypothetical memory usage if we used a bit data type.

Baseline approaches and Results

Our baseline approach is to naively set weights to one of two values (either {0, 1} or {-1, 1}). We use 0 as the threshold for determining which of the two values the weight takes on. On the following page is a table summarizing the results from several approaches. We first used the original BERT model without any modification (*None*). Then we naively binarized all weights, as in the previous blog post (*All*).

Then we started binarizing only certain components. First we binarized only the transformer layers and left the embedding layer and language modeling head unchanged (*Layers*). Then we again binarized only the transformer layers, but left the self attention weights unchanged (*Layers except Self Attention*). Finally we tried to again only binarize the transformer layers, but leave the feed-forward neural networks that follow after the self-attention unchanged. Below is a visualization of the standard transformer encoder for reference. Note that the language modeling head is not shown.



Binarized Components	Cross Entropy	Accuracy	Mean Reciprocal Rank	Inference Time (ms)	Memory Usage (MB)
None (Orig. BERT)	4.3050	0.4017	0.4750	1.4471	417
All {0, 1}	124.3316	0.0088	0.0428	1.3065	13
All {-1, 1}	261.6284	0.0166	0.0412	1.3709	13
Layers {0, 1}	8.4759	0.0000	0.0310	1.3251	103
Layers {-1, 1}	8.6603	0.0166	0.0374	1.3042	103
Layers except Self Attention {0, 1}	10.7902	0.0000	0.0006	1.4582	208
Layers except Self Attention {-1, 1}	8.2863	0.0166	0.0595	1.3205	208
Layers except FFNN {0, 1}	9.0382	0.0000	0.0420	1.3225	312
Layers except FFNN {-1, 1}	10.5091	0.0018	0.0363	1.3225	312

Discussion of results

It is immediately clear that the naive binarization is detrimental to performance of the model. The original BERT achieved an accuracy of about 40%, while all other binarized approaches barely achieve 1% or less. However, there are still some trends.

Embeddings
Binarizing the embeddings and the language modeling hurts performance the most. The cross-entropy is 124.3316 and 261.6284 in the cases where we binarized those components. Once we stopped binarizing the embeddings, we saw a decrease in cross entropy and thus decided to not binarize those components in all of the other baseline approaches. However, it is also clear that the embedding layers are very large. When binarizing them, the memory usage was 13MB, while when not binarizing them it was 103MB. Thus we will further explore binarizing the embedding layer and language modeling head in the future, since binarizing it can reduce memory usage a lot.

Self-Attention
When only binarizing the layers, but keeping the self-attention components unbinarized, we saw an interesting difference between using {0,1} and {-1, 1}. With {0,1}, the accuracy and mean reciprocal rank (mrr) are very close to 0, but with {-1, 1} they were the highest out of all binarization approaches we have tried.

However, by leaving the self attention module unbinarized the model size also doubled as compared to the approach where we binarized the entire layers.

Feed-Forward Neural Networks (FFNN)
In each layer, there is a two layer feed forward neural network that projects the hidden states to a 4 times larger space and projects them back. In our final baseline, we tried to keep those components unbinarized. The cross-entropy was similar to the previous approach, but the accuracy and mrr were much lower. Since the FFNN's are larger than the self-attention components, the memory usage also increased to 312MB.

Inference time
The inference time is very similar across all approaches. This is not surprising, since the weights are all still 32-bit floating point numbers, just some of them take on only binary values. We expect that once we actually use 1-bit data types the inference time will go down. If we also implement more efficient operations for dealing with 1-bit weights (such as bitcount/xnor), we expect to get even more significant gains in inference time.

Error analysis

For error analysis, we took several of the binarization approaches from above and made predictions for a batch of masked sentences. BERT produces a distribution (logits) over the vocabulary for each input token. We chose the highest scoring token as the prediction and compared it against the true token that was masked out.

All {0, 1}
We quickly noticed that this model predicts the same token for every input: with high confidence (approximately 0.8 on average) it predicted the word "for" for each masked token. Thus all of the predictions the model got right were due to chance, since `for` is a common word and is masked out a few times per batch.

All {-1, 1}
Again, the model predicted the same token for every masked token. This time the model predicted "." with extremely high confidence (close to 1 for all tokens). The period token is more common than the word "for", which explains why this approach has higher accuracy. However the cross entropy is much higher, since the token takes up all of the probability mass, so all of the non-period tokens will have a high cross entropy loss.

Layers {-1, 1}
Interestingly, this model also predicted the "." token for every masked token. This explains why it has the same accuracy as the All {-1, 1} approach. However, it does so with less confidence (approximately 0.0188) on average. Thus more probability mass is spread out to other tokens, causing the other masked tokens to have lower cross entropy.

Layers except Self Attention {-1, 1}
We decided to also analyze the predictions of this approach since it performed the best among all baselines. Note that this approach has the exact same accuracy as the previous two approaches we analyzed. Thus we expected it would predict the period token again and indeed this was the case. Interestingly, the confidence for the period token varied much more than in the previous two approaches where it was almost constant across all predicted tokens. In this case it was 0.1127 on average. Notably, this approach has a high MRR, which we believe is due to the fact that again more probability mass is distributed to other tokens.

In summary, all baselines predicted the same token for every masked token. Thus many of the results are due to chance. If the model happened to predict a more common token, it had a higher chance of getting it right. Still, we were able to gain some insights by looking at the confidence of the model in those predictions.

Implementation of BERT

Lastly, we made progress on our first advanced solution attempt by restructuring Huggingface's BERT code into our own project files. This gives us a better ability to manipulate BERT components, and it gives us a better overview of the code, since in Huggingface's implementation all of BERT is contained in a single file.

Key Takeaways

- Naive binarization does not work. It makes the model predict a single token and varying the components we binarized only affected the models confidence in that single token
- Binarized embeddings dramatically deteriorate the model's performance. Embeddings are around a fifth of the total memory footprint, and we need to explore better ways of reducing their memory usage (if not binarization, maybe 16-bit floating point numbers)
- Weight assignments of {-1, 1} appear to work better than assignments of {0, 1}, since they don't assign an extremely high score to a single token
- Feed-forward networks have high potential for binarization. Feed-forward networks are especially important to binarize since they comprise a large portion of the memory usage.