CSE 481N: NLP Capstone: Blog Post 9

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 discussed training with 16-bit precision, learning rate scheduling, and shifting the mean of binarized weights. In this blog post we present our attempts at binarizing the embedding layer and provide updated metrics from evaluating the models on the new validation split we created. We also started fine-tuning with binary BERT for two sequence classification tasks: CoLA and SST-2.

Attempting to binarize Embeddings

So far we have kept the embedding matrix at full or half precision. This week, we revisited binarization for the the embedding matrix, and tried four different approaches in total. Recall that the matrix has shape `(vocab_size, hidden_size)`. Our first attempt was to binarize this matrix like any other linear layer, by using the sign of the original matrix and learning a single scaling factor α and a shift γ. This did not work well, since too much of the information in the embeddings was lost, causing them to become too similar. To diversify the token embeddings, we tried using `hidden_size` scaling factors and shifts, one for each hidden dimension. This binarization approach worked best, and we display the results in Table 1. We also tried learning different scaling and shift parameters for each token (total of `vocab_size` parameters). However, this approach produced a higher loss than the previous approach.

A natural extension is to combine the last two approaches and have a scaling parameter for each token and for each hidden dimension, for a total of `vocab_size + hidden_size`. We then scale the i, jth entry of the binarized (-1, 1) embedding matrix by the product of the ith token scalar and the jth hidden dimension scalar. Mathematically, this corresponds to the outer product of the vector of token scalars and the vector of hidden dimension scalars. Since this corresponds to a rank-1 approximation of the weight matrix, we initialize the weights with the first components of the SVD of the weight matrix. This worked well, however computing the outer product with full precision weights and then performing elementwise multiplication with the binary matrix is to our knowledge not possible to be implemented efficiently with binary operations, which is an essential requirement for our binarization process. Thus we disregard this approach, but leave the idea of approximating weight matrices with low-rank approximations for other people.

For each of the aforementioned approaches, the loss stopped decreasing after the first few epochs, preventing us from training the binarized embeddings for very long. While the above approaches reduced the memory usage to only 14MB, the trade-off in accuracy might be too high. If time allows, we will revisit binarizing embeddings in the following week.

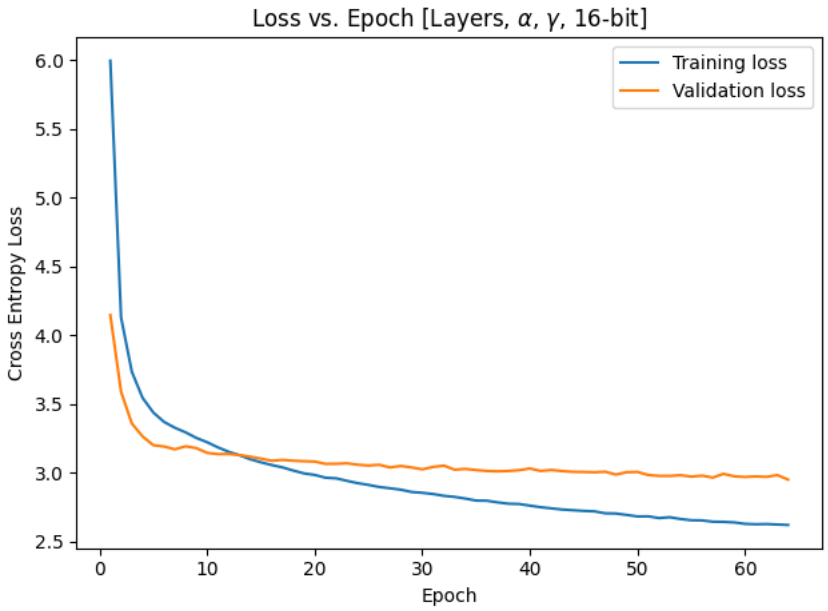
Updated Metrics for Validation Data

In the past blog posts, we evaluated model performance only on the training data, since our focus was reducing the loss, even if we overfit. Our original belief was that binary BERT could not overfit significantly to the training data, since the binarization acts as a form of regularization. We created a validation split and trained the model again while monitoring validation metrics. We found that if we disabled dropout and weight decay during training, the model was overfitting significantly, contradicting our original belief. Below are the updated metrics from evaluating original BERT, our current best approach and our new approach with binarized embeddings on the validation data.

Approach	Loss	Accuracy	MRR-32	Memory
Original BERT	2.0752	0.6315	0.7036	417MB
Layers (α, γ, 16-bit)	2.9660	0.4819	0.56507	56MB
Layers + Embeddings (α, γ, 16-bit)	3.8115	0.3702	0.4553	14MB

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

Below is a plot of the training loss and validation loss from training the best model (row 2 above):



One can see that the validation loss goes down very quickly. We tried tuning training parameters to reduce the loss even more, but after training with different settings for many epochs we found that we get the same final loss as when just training with our current parameters (learning rate, batch size) for a few epochs.

Fine-Tuning with Binary BERT

We managed to pre-train BERT on a subset of the original data (BooksCorpus + Wikipedia) and achieve a low cross entropy loss, high accuracy and high mean reciprocal ranking. Although we didn't match BERT's original performance, we think we are close to the best metrics we can achieve with our current binarization approach and the amount of data we are using. A masked language model is not particularly useful (in fact we

had difficulty coming up with use cases, one potential one being text restoration) and it is almost exclusively used for pre-training. To better evaluate the performance and usefulness of our binarization approach, we decided to start on one of our original stretch goals: to fine-tune binary BERT on a downstream task. We decided to fine-tune for binary classification on the CoLA and SST-2 tasks from GLUE. We did not choose tasks from SuperGLUE or SQuAD since we did not pre-train binary BERT with sequence pairs. Question answering tasks like SQuAD, language inference tasks and all tasks from SuperGLUE rely on the model being able to understand relationships between two sequences which are fed into the model simultaneously. Fine-tuning our model for CoLA or SST-2 just involves replacing the language modeling head with a linear layer to predict a single score for each sequence. We describe the two tasks in more detail below.

CoLA

The Corpus of Linguistic Acceptability (CoLA) consists of 10657 sentences collected from 23 different linguistics publications. Each of the sentences is accompanied by a label classifying the text as being either grammatically 'acceptable' or 'unacceptable'. For example,

- "The professor talked us" : labeled as 0, meaning ungrammatical
- "The dog barked its way out of the room" : labeled as 1, meaning grammatical

SST-2

The Stanford Sentiment Treebank contains fine grained sentiment labels for 215,154 phrases in the parse trees of 11,855 sentences. Phrases are labeled as either positive or negative. We ignore the tree structure in the data. Below are two examples from the dataset:

- "Good morning thespians a bright and sunny day in UK, Spring at last" labeled as 1, meaning positive sentiment.
- "Mine is broken too! I miss my sidekick" labeled as 0, meaning negative.

Initial training results

We started with fine-tuning on the CoLA dataset. Unfortunately we did not manage to successfully train binary BERT for this task yet. As our initial set of hyperparameters for learning we chose those suggested in the RoBERTa paper, however the loss did not decrease. Since the dataset is small, we could run training experiments quickly and try out many combinations of hyperparameters. Still, none of the hyperparameters we tried work for fine-tuning. As a sanity check, we fine-tuned original BERT on the dataset to ensure that our data setup is correct. Indeed, we were able to reduce the loss. Next we performed a simple error analysis, which showed that the model predicts the same label (0 or 1) for every sequence. We attempted to overfit the model to a small subset of 64 sequences, but this still did not work. Our current approach is to freeze the embeddings and transformer layers and just fine-tune the pooling layer (BERT has a layer that transforms the hidden states of the CLS token and applies the tanh activation) and the classification layer. This helped reduce the loss slightly, but the model is still predicting the same label for the majority of sequences with the exception of a few sequences.

Next steps

We will continue to attempt fine-tuning BERT on CoLA. We will not continue to work on pre-training, unless during fine-tuning we discover that there is an issue related to our pre-training procedure that prevents us from successfully fine-tuning the model. If we successfully fine-tune on CoLA, we will try fine-tuning in the same way on SST-2.