CSE481 NLP Capstone: Blog Post 3

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

Project: Binarized Transformers

In recent years, Transformer neural networks have led to state of the art results in a wide variety of natural language processing tasks. Still, Transformers have high memory usage and are not fast enough to be run in real-time on edge devices. The goal of this project is to speed up the inference time and to reduce the memory usage of transformers by using binary weights. This project was inspired by the XNOR-Net paper (XNOR-Net), in which the authors have binarized convolutional neural networks leading to a reduction in memory usage by a factor of 32 and a speed up by a factor or 58 (in the most extreme case of binarization). Of course binarization comes at a performance cost and the main challenge will be to binarize the transformer while still maintaining high performance. To our knowledge there are currently no publications on binarizing transformers. This makes for a novel and very promising capstone project. If successful, we believe transformer NLP models could run much more cost-effectively on edge devices in real-time. This is one of the main motivations for this project.

In the past, there has been some work on making Transformers more efficient. However, to our knowledge no work has been done on applying binarization to Transformers. For instance, in the paper Reformer: The Efficient Transformer, authors have reimplemented scaled dot-product attention to reduce its time complexity and reversible residual layers instead of standard residual connections, which allows them to further speed up training. There are several existing papers about binarizing other architectures. In the XNOR-Net, authors binarized convolutional neural networks. In Learning Recurrent Binary/Ternary Weights authors have binarized recurrent neural networks, including LSTMs and GRUs. Those papers will serve as a starting point for our experiments. Since Linear layers can be phrased as 1-by-1 convolutions, the binarized convolutions from the XNOR-Net paper will be helpful for binarizing the linear components in the Transformer.

Our minimum viable action plan is to take a pre-trained Transformer model, such as BERT, and naively change all of the FP32 weights to only one of two values. That is, the weights will remain floating point tensors for now, but they will only be allowed to assume one of two values. We will experiment with different values (for instance 0 and 1 or -1 and 1) and we will attempt using different methods of determining the binary value of a full precision weight. The most simple methods are thresholding functions. For instance, one could use the sign function (with 0 mapping to 1).

If time allows, there are several ideas we would like to explore. One important issue is that simply setting weights to 0s and 1s will not improve performance or reduce memory usage. That is because we are still storing FP32 weights and we are still using the same floating point operations. In order to get the model to perform well and use less memory we would actually have to store single bits and use bitwise operations near the machine-level, which as far as we know is impossible in PyTorch. Thus a stretch goal would be to implement binarized Transformers in a lower level language such as C/C++. It seems like this would be a crucial aspect of a project, but we believe that if we can retain high performance with all binary weights, we are guaranteed to have reduced memory usage and (depending on the operations used) also reduced inference time once implemented in a low-level language. However, this will be pure engineering work and thus remains a stretch goal. As an extended stretch goal, we would also like to try running our binarized transformer model on an edge-device, such as a Jetson Nano or Raspberry Pi to see how well it performs in a real-world scenario. Another evaluation related stretch goal is to implement other SOTA models using our binarized transformer and compare the performance against the original, non-binarized version. We would like to make this comparison for different models and different NLP tasks, such as QA or machine translation. For the comparisons, we would not only compare performance metrics, but also inference time, FLOPs, and memory usage.

While Transformers are large and seem complex, they are highly modular architectures. That is, they can be naturally broken into the individual components making up the larger architecture. We plan to conduct our research by binarizing each of the component modules making up the larger Transformer encoder and decoder layers. Some examples of modules we would need to binarize include linear layers and multi-head attention layers (which can be further decomposed into smaller components). At each stage of the binarization process, we will evaluate metrics and test performance to see if the model still functions. Our plan is to start with binarizing linear layers, since there already exist reliable methods we can use.

To summarize, the objective is to make transformers faster and use less memory through binarization, while still retaining as much performance as possible.

In order to evaluate the models performance, we are planning on using the Penn Treebank or WikiText datasets, on which we will calculate model perplexity for the masked language modeling task. Originally we wanted to evaluate it on regular language modeling, but since we decided to use BERT we realized it makes more sense to evaluate the model on the task BERT was trained on – masked language modeling. The metric for this task is also simple, we can just mask out tokens in our corpus and then predict the masked tokens and calculate the cross-entropy between the predicted tokens and the true tokens. Masked language modeling and language modeling in general are simple tasks that do not require annotated data and have simple metrics. Thus we likely use it as a task to determine model performance throughout the project. We will also measure inference time, relative to the GPU or CPU speed and memory usage. The three metrics cross-entropy, inference time and memory usage directly reflect our project objectives and thus seem ideal.

Training Transformers is expensive and requires a lot of computation power, thus we need to carefully think about the resources we have available. We are planning on primarily using the GPUs allocated for the capstone, Google colab and our GPUs at home. We will never pre-train a Transformer from scratch and will likely only retrain it for a few epochs using pre-trained weights, so the available resources should be sufficient.