CS247 Project Final Report
Accelerating Transformers by Pruning Edges

Kwang Jun Kim
UCLA
UID: 005297920
justkj7@cs.ucla.edu

Chiao Lu
UCLA
UID: 204848946
chiao@cs.ucla.edu

Howard Xie
UCLA
UID: 005354166
howardx@cs.ucla.edu

Liunian Harold Li
UCLA
UID: 005271406
liunian.harold.li@cs.ucla.edu

ABSTRACT

Transformer is the SOTA model for text modeling in NLP and traditionally viewed as a stand-alone model. However, it can also be formalized as a graph neural network. It models a sentence as a fully connected graph, where words are nodes and they are connected to each other. We will implement BERT, the bidirectional encoder representations from transformers, as a graph neural network. The GNN will be large and inefficient, so we will explore ways to prune the graph.

ACM Reference Format:
Kwang Jun Kim, Chiao Lu, Howard Xie, and Liunian Harold Li. 2020. CS247 Project Final Report Accelerating Transformers by Pruning Edges. In *Proceedings of ACM Conference (Conference'20).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Pre-trained language models such as ELMo [5], GPT, BERT [3], and RoBERTa [4] have achieved state-of-the-art performance on various NLP tasks, serving as the indispensable backbone of many downstream NLP models. They are first pre-trained on billion-scale corpus and then fine-tuned to a particular NLP task. Though powerful, the computational cost of using these pre-trained language models is arguably large. In this project, we look into the popular BERT model, which is built upon the Transformer, and propose a way to reduce the computational cost of BERT when fine-tuning it for a downstream task.

The basis of our project is to accelerate transformers for NLP use. Transformers can often be seen or thought of as a graph neural network. It is large and complex with n^2 edges, thus requiring large computational resources. By finding ways to prune the underlying graph model, we can achieve a healthy speedup, while preserving accuracy to a degree.

Conference'20, July 2020, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Transformer relies on the attention mechanism, which could be viewed as a way of information propagation. Thus, Transformer can be seen as a graph neural network operating on a fully-connected graph, where each word connects to every other words and attention propagates information through every word. With this approach, however, we run into the problem of having to deal with fully connected graphs. Running GNN on fully connected graphs is only possible with a small graph. However, since we are dealing with NLP, we might be dealing with a graph with thousands of nodes. In this case, the time and space complexity becomes $O(N^2)$ which doesn't scale.

Because of the intolerable complexity of the naive GNN which models the transformer as fully connected graphs, we argue that not every word needs to pay attention to every other word. Rather, there might exist a way of strategically removing certain edges in the graph (i.e., masking out certain attention flows) while maintaining much of the model performance. Building upon this idea, we propose two research questions:

RQ1: how will the performance change when we keep increasing the ratio of the edges removed? Intuitively, more edges we remove, more speedup we could achieve but the performance will accordingly drop. Thus, it is interesting to study empirically how many edges we could remove without losing much of the performance.

RQ2: how should we remove the edges? The most obvious way to remove the edges is to remove them randomly. However, we argue that parsing trees, which shows the grammar structures of sentences, might serve as a reasonable prior of how the remaining graph after pruning should look like. In the first case, we could still be at $O(N^2)$ complexity, but in the second case we have $O(N)$. We would like to benchmark how much drop in performance we suffer after removing some edges with the strategies outlined above.

We test our idea on a standard sentiment analysis dataset SST-2 [6]. We compare the original model BERT Base, a model with randomly pruned edges, and a model with syntax pruning. We find that the model with syntax pruning achieves great balance between computation cost and performance, verifying the efficacy of our proposed method.

2 RELATED WORK

2.1 Graph Neural Networks

In Graph Neural Networks(GNN), sentences are represented as nodes and edges where words in vertices are interconnected with each other by edges as graph data. The buildup of such graph modeling is achieved through neighborhood aggregation which enables a node to collect features of neighboring nodes to update its own representation. The graphical model of a sentence is fully-connected when every word represented by node has relational edges to every other word in the sentence. However, a sentence graph is not necessarily to be fully-connected because there might be unnecessary or less meaningful relationships with regard to sentence representation. The complexity of graph gets intensified as a sentence has more words, and it takes longer and longer time to process input graph in neural network as the size of sentence becomes bigger and bigger (Figure 1). Our focus is to handle such unnecessary edges in a sentence graph and generate an optimized GNN model.

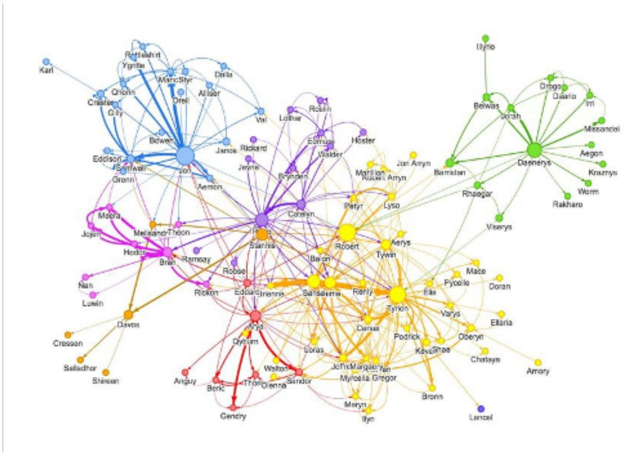

Figure 1: A sentence graph becomes more and more complicated as the number of words in a sentence increases.

2.2 Transformer

Transformer is a neural network architecture upon which BERT is built, and [7] it firstly introduced attention-mechanisms in which input sequence determines other following steps. Transformer utilizes three matrices which are Query, Keys, and Values to calculate attention value for tokens in a sequence. Besides, encoder and decoder of Transformer helps transform a sequence to another sequence and capture timely dependency, and such feature is useful to translate different languages. Based on theoretical concepts, pytorch-transformers has been implemented as an open source platform for NLP, and it provides a-state-of-the-arts architectures such as BERT, GPT-2, XLNet, and XLM. Our task is not directly related to Transformer. Since BERT runs upon Transformers, however, it would be at least beneficial to understand background and mechanism of Transformer for BERT implementation.

2.3 BERT

BERT which is acronym for Bidirectional Encoder Representations from Transformers is a state-of-the-art language model for various NLP tasks. [2] The main feature of BERT is pre-trained contextual representation, and it has been implemented running upon Transformers which supports attention-mechanisms. Unlike other models of unidirectional representation reading word, BERT embeds bidirectional representation based on previous and next contexts. Key techniques for such bidirectional representation are Mask Language Model(MLM) and Next Sentence Prediction(NSP) both combined to intensify model training (Figure 2). In addition, BERT tokenizes words into sub-tokens, and this feature is helpful for representing a sentence more deeply. Basically, we are going to simulate GNN on BERT as efficiently as possible with reduced edges. So, we need to understand how BERT works and then find optimization room in our GNN reimplementation. Also, our focus includes tokenizer feature as well in order to keep performance good enough while reducing edges from fully-connected graph.

2.4 Accelerating Transformers for Long Documents

Basically, Transformers are not appropriate for processing long documents because its self-attention operation scales quadratically with the length of sequence. BERT handles this issue by setting a hard limit of 512 tokens per sentence assuming that majority of sentences to be processed in within the limit, but it does not solve the issue genuinely for long documents. As one of solutions, [1] Longformer tries to tackle this issue through handling only relevant information. In Longformer, specifically, a dilated sliding window combined with global attention makes attention mechanism operate in linear time even with the growing size of sentence. Although methodology is somewhat different, our idea was inspired from Longformer in that it is possible to reduce self-attention time complexity from $O(N^2)$ to $O(N)$ if we only process relevant edges from a fully-connected graph. However, there are numerous ways to distinguish relevancy of each edge in graph, and this task would be further step for advanced optimization regarding better performance.

3 METHOD

3.1 Transformer as GNN

https://graphdeeplearning.github.io/post/transformers-are-gnns/

Compared to the traditional RNN approach where each word in a sentence is modeled as points in a time series (i.e. sequential), transformers on the other hand construct feature for each word using an attention mechanism to discover the importance of each word with respect to every other word. With this in mind, at each iteration in the training, we can update the feature of each word by some kind of aggregation of all other words.

The update rules are described as follows:
$h_i^{l+1} = \text{Attention}(Q^l h_i^l, K^l h_j^l, V^l h_j^l)$
i.e. $h_i^{l+1} = \sum_{j \in S} w_{ij}(V^l h_J^l)$
where $w_{ij} = \text{softmax}_j(Q^l h_i^l \cdot K^l h_j^l)$.

Here, S is the set of words in the sentence, l is layer, h is hidden feature, and Q^l, K^l, and V^l are weights to be learned.

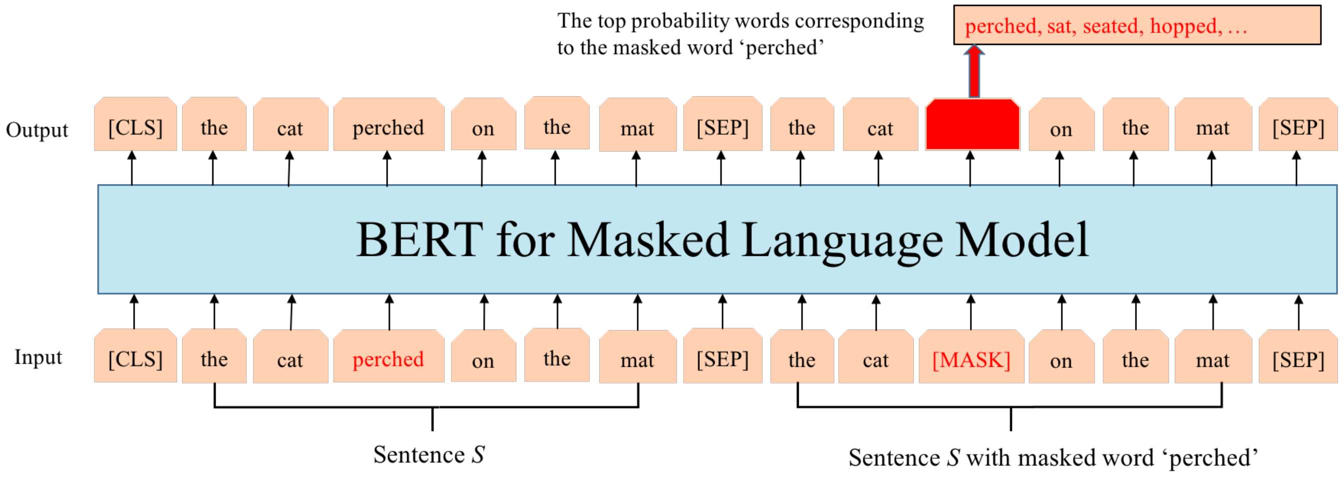
The top probability words corresponding to the masked word 'perched'
perched, sat, seated, hopped, ...
Output
[CLS]
the
cat
perched
on
the
mat
[SEP]
the
cat
on
the
mat
[SEP]
BERT for Masked Language Model
Input
[CLS]
the
cat
perched
on
the
mat
[SEP]
the
cat
[MASK]
on
the
mat
[SEP]
Sentence S
Sentence S with masked word 'perched'

Figure 2: Conceptual depiction of mask language model and next sentence prediction in BERT

Notice that the above formulation of transformer resembles the structure of Graph Neural Networks (GNN). This suggests that we can view transformers as GNN operating on fully connected graphs.

Briefly, GNN update the feature of each node i at layer l by aggregating it's own feature and the features of its neighbors. Specifically, $h_i^{l+1} = \sigma(U^l h_i^l + \sum_{j \in N(i)} (V^l h_j^l))$,

where U^l and V^l can be learned and σ is some non linear activation function. Here, the summation $\sum_{j \in N(i)}$ can be replaced by other aggregators such as mean, max, or attention mechanism.

3.1.1 Sentences, Transformers, and Fully Connected Graphs. We briefly discussed how transformers relate to GNN. With this in mind, we can now think of sentences as a fully connected graph, where each word is a node in the graph, and each node connects to every other node (hence a fully connected graph). We then apply multi-head attention mechanism as the neighbor aggregation function. See figure below for visualization.

However, by viewing transformers as GNN, we notice a serious problem: a fully connected graph is $O(N^2)$ both in time and space. Imagine operating on a long sentence with 1000 words, then approximately the GNN model would require 1 million operations and (unit) storage.

Our project tries to tackle the above issue by exploring various strategies of pruning edges. It should be obvious that as more and more edges get pruned, we would expect an increase in speed at the cost of performance. Here, we investigate how different edge pruning strategies affect the performance.

3.2 Edge Pruning for Accelerating

As stated in the section above, $O(N^2)$ time and space footprint is undesired. So, we devise three different strategies for pruning edges.

- Untouched
- Random
- Syntax Tree

We describe the three strategies in detail below.

3.2.1 Untouched. As discussed above, GNN on fully connected graph is expensive. So, in the untouched strategy, we do not modified the adjacency matrix and utilize the fully connected graph for GNN. This is actually our baseline for benchmarking the two strategies below.

3.2.2 Random. In the random strategy, we specify a percentage and randomly mask out edges. In detail, suppose we have a graph consisting of n nodes and we want to randomly prune out p% of the edges. This mean that at the end we would have $\frac{n(n-1)}{2} \times (1 - p\%)$ edges left. Notice, the complexity is still $O(N^2)$ according to the equation above.

3.2.3 Dependence Tree. Dependence tree captures the semantic relationship between words in a sentence using a tree. For example, in the sentence "I drink water" we can make a simple tree with root "drink" having two children, "I" and "water". This tree tells us that "drink" connect "I" and "water" to form a semantically reasonable sentence. Figure 3 shows an example of a more complicated syntax tree of the sentence "James ate some cheese whilst thinking about the play."

4 EXPERIMENT

4.1 Setting

We test our model on SST-2, a widely-used sentiment analysis dataset. The task is to classify the sentiment of a given sentence as either positive or negative. BERT has been shown to exhibit great performance on this task. Below, we introduce the models we compare:

- BERT Base: We use the pre-trained BERT Base model provided by [3]. A softmax layer is added over the last layer representation of a "[CLS]" token to classify the sentiment label of the input sentence.
- Syntax $X\%$: We prune the fully-connected attention maps down to the syntax tree of the sentence and train the same

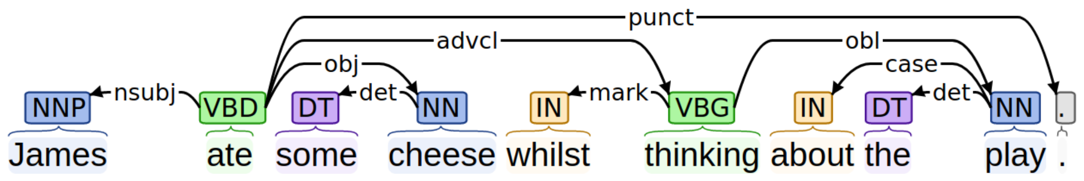

Figure 3: The dependence tree of sentence "James ate some cheese whilst thinking about the play.". For example, "ate" is the root node.

Model	ELMo	GPT	BERT Base (Official)	BERT Base
Performance	90.4 (Test)	91.3 (Test)	93.5 (Test)	93.2

Table 1: Performance of baseline models on SST-2.

Model	Syntax 20%	Syntax 50%	Syntax 70%
Performance	89.7	91.3	92.2
Model	Random 20%	Random 50%	Random 70%
Performance	85.2	87.4	90.7

Table 2: Performance of the proposed models on SST-2.

BERT Base model on the dataset but with the pruned attention maps. In practice, 20% of the original edges are kept and we denoted this model as Syntax 20%. Additionally, we randomly add back a certain percentage of the edges to reveal the trade-off we make when we remove more and more edges of the attention maps.

- Random $X\%$: We randomly remove edges from the fully-connected attention maps and keep $X\%$ of the edges. Random $X\%$ has the same number of edges as Syntax $X\%$.

We use Adam as the optimizer with a linear warm up and decay learning rate schedule. We use the same hyper-parameters for all models. We use a max learning rate of $2e-5$, a batch size of 32, and a warm up ratio of 0.1. We train the model for 3 epochs and then evaluate on the development set.

4.2 Results

Here, we report the performance of previously reported models and our own models in Table 1 and Table 2. We make the following observations:

Syntax $X\%$ shows great trade-off between performance and speed. . With 30% less edges, which in theory translates to 30% less computational load, the performance only drops mildly. The performance still holds up pretty well with half the computation load (Syntax 50%) while being decent with 20% of the computational load (Syntax 20 %).

Pruning the attention maps according to the syntax tree greatly outperforms random pruning. Syntax $X\%$ is consistently better than Random $X\%$, given the same computational budget. This aligns with our assumption and is consistent with previous works suggesting the BERT automatically learns syntax during pre-training.

Syntax 50% has similar performance as GPT.. Interestingly, Syntax 50% exhibits similar performance as GPT. GPT is a unidirectional language model, as the words in a sentence could only attend to its previous words. Viewing GPT as also a graph neural network, the number of attention edges in GPT is exactly the half of that of BERT, equaling that of Syntax 50%. Since GPT is of the same size as BERT, it is interesting to see Syntax 50% achieving similar performance as GPT. It not only verifies again the efficacy of the syntax pruning method, but also partially hints that the superiority of BERT over GPT largely come from its bi-directional modeling, i.e. doubled edge numbers.

5 CONCLUSION

In this project, we develop upon the idea of viewing the state-of-the-art NLP model, the Transformer, as a graph neural network and propose to prune the edges in the input graph to reduce the computational load. We investigate two ways to prune the edge, 1) random pruning and 2) pruning according to a syntax tree. Experiments on SST-2 verify that syntax pruning achieves good balance between performance and speed and outperforms random pruning significantly. One caveat is that the current implementation we use do not implement the Transformer as a real graph neural network and the actually running time on GPU does not correlated well with the edge number. Thus the computation cost reduction is hard to observe directly using the current implementation.

6 FOOT PRINT

6.1 Task Distribution Form

1. Run BERT on Colab/the server on SST-2	Chiao, Howard
2. Creating adjacency matrix algorithm	Chiao, Kwang
3. Creating model	Harold
4. Creating presentation slides	Chiao, Kwang, Harold, Howard
5. Writing final report	Chiao, Kwang, Harold, Howard
6. Writing midterm report	Chiao, Kwang, Harold, Howard

6.2 Timetable

Due to computational consideration, we used SST-2 [6], a sentiment classification dataset as our diagnostic dataset. We tested the performance difference between directly fine-tune a pre-trained BERT on SST-2 and fine-tuning BERT but with certain attention edges removed. The detailed plan we took is detailed below:

- First week: Download SST-2 dataset and setup environment.
- Second week: Implement the edge pruning algorithm (i.e. random pruning and pruning by the syntactic tree).
- Third week: test the models' performance on SST-2.
- Fourth week: write up the report and presentation slides.

REFERENCES

[1] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. *arXiv preprint arXiv:1906.04341* (2020).

[2] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2017. What Does BERT Look At? An Analysis of BERT's Attention. *arXiv preprint arXiv:1906.04341* (2017).

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[5] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* (2018).

[6] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*. 1631–1642.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Lukasz Gomez, Aidan N.and Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *arXiv preprint arXiv:1706.03762* (2017).