Fine-Tuning BERT via Submodular Maximization

Andrew Downie
University of Waterloo
acdownie@uwaterloo.ca

Abstract

The BERT Model released 2019 is an exciting advancement in contextual language modeling and has already shown strong performance on many benchmarks. Modeling problems with submodular functions has also been a topic of interest for many machine learning problems because of its ability to model representativeness of a subset of a data set which has applications in data set selection for training. This report focuses on gaining insights of how submodular summarization can be leveraged to more efficiently train and fine-tune a language model such as BERT. The proposed ideas were be tested and validated using the Story Cloze Test, which is a test used to evaluate the contextual understanding of a language model.

1 Introduction

Natural Language Processing (NLP) is a growing sub-field of machine learning and linguistics that can be used to accomplish a variety of goals from text categorization and summarization to language translation. Many of these problem require finding sentence representations and then utilizing them for our objectives [1]. Solutions to these problems have applications in many areas such as machine translation, market advertising analysis and speech recognition. Some of the older solutions utilized properties of the text such as n-grams between sentences, word to vector models, and sentiment analysis to analyze text [2]. In 2017 in a paper called "Attention Is All You Need", the Transformer network was introduced which utilized self-attention layers in and a encoder-decoder structure that was able to solve many of the problems that previous networks where facing at the time, as well as performing very well on many benchmarks [1]. More recently Google's AI Language team has released a pre-trained model called "Bidirectional Encoder Representations from Transformers" or BERT for short which leveraged the encoder structure from the transformer architecture to create a powerful language model that can be used in many applications [3]. The BERT model that was pre-trained on a large text corpus to generate a basic language model that takes text and embedded them into a vector space where the contextual information is encoded. To use the model for other applications, all that needs to be done is to add output layers to the model and fine tune these layers for a specific task.

In another field of NLP, there have been mathematical advancements in text summarization and identification of key information in text. In Lin and Bilmes [4], the authors were able to formulate a function that takes a subset of a text in a document and output a measure of how much the subset summarizes the entire document. This function has the key property of being submodular which allows for a greedy strategy to optimize the function efficiently.

One of the problems with BERT is that is it can be data hungry and thus expensive to train. One way to reduce resources required to train is to select a subset of the data set to train on that summarizes or characterizes the unique training examples to reduce the size of the problem [5]. Since when fine-tuning the BERT Model the data set being trained on is a set of labeled text documents, then the summarization function can be applied to the data set and a subset of the data can be identified efficiently to reduce the size of the training set. This subset should include the a diverse set of

Submitted to Introduction to Machine Learning (CS480/680, 2020 Spring). Do not distribute.

training examples that also summarize the total data set. This should be able to reduce computational resources as well possible improve performance. A framework for accomplishing this will be explored in this paper.

2 Background

2.1 Submodulairity and Deep Learning

Submodular set functions are use to model problems with diminishing returns properties. More formally, function $f : 2^V \rightarrow \mathcal{R}$ is submodular if $\forall S, T \subseteq V$ and $S \subseteq T$ where V is the ground set then $\forall x \in V$.

$$f(S \cup \{x\}) - f(S) \geq f(T \cup \{x\}) - f(T) \tag{1}$$

For the rest of the report we will denote the marginal return given a subset as the following.

$$f(x|S) = f(S \cup \{x\}) - f(S) \tag{2}$$

In general finding the exact solution to maximizing a submodular function is an NP-Hard problem but a greedy algorithm can compute an constant factor approximation of $(1 - e^{-1})$ in polynomial time [6]. In many data sets there is often samples that are redundant and provide little information in the context of the whole set. The amount of information or how representative each element of the data set can be expressed as a submodular function, therefore finding a subset that provides almost all of the information can be computed efficiently and to near optimally. The important data points can be retrieved and used for training, this improves training efficiency. This method was explored in Wei et al. [5], where they provided an adaptive algorithm for selecting data points using a submodular objective function, and it showed positive results for classifiers such as Naive Bayes, Nearest Neighbors and Deep Convolutional Neural Networks but not for a language model such as BERT. There has also been research into summarizing and finding representative elements for text specifically [4]. The submodular function used for these this particular problem for summarizing text is and adapted graph cut-function. The weights in the graph refer to the similarity between elements in the data set. In a paper by Lin and Bilmes [4] the authors use this method to effectively summarized multiple documents quickly comparatively to other methods. Finally, representative element selection using submodularity can be done in a distributed map reduce style of computation which allows for the selections to be computed on much larger data sets [7].

2.2 Story Cloze Test

To evaluated the effectiveness of this reports contributions, the model trained will be tested on the Story Cloze Test benchmark. The Story Cloze Test is a method of determining how well a language model can relate multiple sentences together. The goal of the task is that given a four sentence story, model must selected correct ending out of two possibilities. Both ending make grammatical sense with previous sentences but differ based on what happens to the subjects in the stories [2]. When the test and data set were released in 2016 some of the state of the art methods for this problem were N-gram Overlaps, Sentiment Analysis, Narrative chains and Deep Structured Semantic Models. In 2019 a paper by Li et al. [8] showed that BERT could accomplish this task through transfer learning. The proposed transBERT which combined fine-tuning using some unsupervised tasks and existing supervised task to train BERT for this specific task. They did this by using a large data set from Twitter and IMDB movie data and showed very good results on the Story Cloze Test from Mostafazadeh et al. [2]. This work will focus on this problem but the training will be complete on examples specifically from the data set provided [2] and will focus more on training example selection to reduce resource consumption than trying to train a BERT model to set a record score on the benchmarks.

2.3 BERT Fine-tuning

In Sun et al. [9] the authors explores methods for fine tuning BERT using single and multitask fine tuning for the model. They showed that the model can reduce the error rate for a particular task by

training the model on a different tasks. The paper also shows that BERT can be effectively fine-tuned on reduced data sets. More specifically they were able to show that BERT can perform well on the Story Cloze Test. What differs from this is that they were testing how effective BERT was at transfer learning and not exploring how data set selection specifically affects the fine-tuning of the model.

3 Main Results

The main objective is to take a data set and efficiently extract a subset of the data set that is representative and diverse, that when used for training will give similar performance to training on the whole set. As previously explained these qualities of a subset can be characterized by a submodular function. This function then can be efficiently maximized over the data set subject to a cardinality constraint to produce this subset. The specific submodular function was used to characterize the quality of the subset of the data set. Let V be the total data set to select from. Let w_{ij} represent the similarity between $i, j \in V$. Finally let $S \subseteq V$ be the subset to evaluate. Then the submodular function we are maximizing is formulated as follows.

$$f(S) = \sum_{i \in V \setminus S} \sum_{j \in S} w_{ij} - \lambda \sum_{i,j \in S : i \neq j} w_{i,j} \tag{3}$$

This function is the same submodular function that was used in Lin and Bilmes [4] to summarize documents. The double sum in the equation calculates the value of the graph cut of a fully connected graph between all the data points, where the weights of each edge is a value that represents the similarity between the two nodes. The graph cut function is can be used to measure how much a subset summarizes the data set. The subtracted sum represents a penalty for elements in the subsets being similar. This term is to promote selection of diverse elements. λ is a tune-able trade off parameter that is adjusted for specific problems.

Now using this function the goal is to solve the following problem and then train a the model with the solution generated by solving the following problem.

$$\begin{aligned} &\max_{S \subseteq V} f(S) \\ &s.t\ |S| \leq k \end{aligned} \tag{4}$$

It will now be shown how this function can be integrated to a data pipeline for training

3.1 Proposed Solution

The following psuedocode outlines the set of steps to make use of the submodular function described in equation (3). A key note about this is that for this to be a practical solution, the total time to computer the pre-processing and the training must be shorter than training the model on the complete data set. Otherwise the solution would not provide any efficiency benefits.

Algorithm 1: Data Pre-processing Framework

Input: D, k
Output: Model S

```
     // D is the data set and k is the cardinality of the output
 subset
```
$X \leftarrow$ GenerateVectorEmbeddings(D)
```
     // each row of X is a d dimensional vector embedding of the
 datapoint
```
$W \leftarrow$ GenerateSimilarityMatrix(X)
```
    // i,jth entry represents the similarity between data points i
 and j
```
$S \leftarrow$ GreedlySelectElements(W, k)
```
                        // Efficiently select datapoints to train on
```
Model $\leftarrow$ TrainModel(S);
```
          // Train model using the reduced subset
```

On a higher level the proposed algorithm takes the data set that is provided and converts it into a matrix $X \in \mathbb{R}^{n \times d}$ where each row is the vector representation of one of the data points in the data set. It then uses the embedded vectors to compute similarities between each of the vectors. Finally it uses utilizes a greedy algorithm to find the subset of data points that approximately solves (4). Each of these following steps will be described in more detail below.

GenerateVectorEmbeddings: For this particular problem the data points were common four sentence common sentence stories and these need to be converted to into d dimensional vectors. Two methods were experimented with one, being taking each story from the data set and using the vector embeddings from evaluating the story with the base pre-trained BERT model provided in Devlin et al. [3]. After each story was run though the this base pre-trained model and the last output layer was passed through an average pooling layer to produce a 768 dimensional vector embedding of the data point. The other method used was the GenSim's Doc2Vec algorithm which is another popular method to convert paragraphs into vector embeddings, which is described in Le and Mikolov [10].

GenerateSimilarityMatrix: Once the vector embedding matrix X is computed it can now be used to compute the similarities between the data points. For this particular problem the cosine similarity between two vectors were used to compare the vectors. The output is a similarity matrix W where each entry is described as follows

$$w_{ij} = \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2} \tag{5}$$

In general this matrix is expensive to compute because it requires calculating approximately $|V|^2$ similarities between the vectors but with a GPU can be leveraged to compute W quickly. This is done by noticing that.

$$w_{ij} = \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2} \tag{6}$$

$$= \left\langle \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|_2}, \frac{\mathbf{x}_j}{\|\mathbf{x}_j\|_2} \right\rangle \tag{7}$$

We can then calculate $\bar{X}$ where each row of bar calculated by $\bar{X}_i = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|_2}$ where X_i is the ith row of X. Then to compute W all that needs to be done is the following matrix operation.

$$W = \bar{X}^\top \bar{X} \tag{8}$$

Each of these operations can be quickly computed using a GPU because they are all simple matrix multiplications.

GreedlySelectElements: This algorithm will select the subsets of elements by picking the elements with largest marginal return at each iteration. The marginal given a selected set S denoted as $f(x|S)$ can be calculated as follows.

$$f(x|S) = \left(\sum_{j \in V \setminus S \cup \{x\}} w_{xj} - \sum_{j \in S} w_{xj} \right) - \lambda \sum_{j \in S, j \neq x} w_{xj} \tag{9}$$

To select each of the elements to maximize the objective function the algorithm repeats the following two steps until the selected set is of size k. Let x_i and S_i be the data point selected and the total selected set at the ith iteration.

$$x_i \leftarrow \underset{x \in V \setminus S_{i-1}}{\arg\max}\ f(x|S_{i-1}) \tag{10}$$

$$S_i \leftarrow S_{i-1} \cup \{x_i\} \tag{11}$$

At each stage the each $f(x|S_i)$ needs to be computed to select x_i but naively this would be an expensive operation because for each of x a sum of at least $|V|$ elements would have to be computed. Under further inspection the difference in each each marginal between iteration can be found as follows.

$$f(x|S_i) - f(x|S_{i-1}) \tag{12}$$

$$= \left(\sum_{j \in V \setminus S_{i-1} \cup \{x_i\} \cup \{x\}} w_{xj} - \sum_{j \in S_{i-1} \cup \{x_i\}} w_{xj} \right) - \lambda \sum_{j \in S_{i-1} \cup \{x_i\}, j \neq x} w_{xj} \tag{13}$$

$$- \left(\sum_{j \in V \setminus S_{i-1} \cup \{x\}} w_{xj} - \sum_{j \in S_{i-1}} w_{xj} \right) + \lambda \sum_{j \in S_{i-1}, j \neq x} w_{xj} \tag{14}$$

$$= -w_{xx_i} - w_{xx_i} - \lambda w_{xx_i} \tag{15}$$

Knowing this means they only differ by a the similarity of the most recently selected element. To avoid computing the whole sum each time we can store the marginals in a variable and then after each new element is selected $f(x|S_i) = f(x|S_{i-1}) - (2 + \lambda)w_{xx_i}$. We can further improve the performance by vectorizing these equations. Let W_i denote the ith column where the similarities between x_i and all the other elements are storied. Then we can write the update for each marginal as follows.

$$\begin{bmatrix} f(x_1|S_i) \\ \vdots \\ f(x_n|S_i) \end{bmatrix} = \begin{bmatrix} f(x_1|S_{i-1}) \\ \vdots \\ f(x_n|S_{i-1}) \end{bmatrix} - (2 + \lambda)W_i \tag{16}$$

Where

$$\begin{bmatrix} f(x_1|S_0) \\ \vdots \\ f(x_n|S_0) \end{bmatrix} = \begin{bmatrix} \sum_{j \neq 1} w_{x_1, x_j} \\ \vdots \\ \sum_{j \neq n} w_{x_n, x_j} \end{bmatrix} \tag{17}$$

Which can be efficiently computed on a GPU and make this solution feasible.

4 Experimental Results

These processing steps take the data set and encoded the data into a form where the data points can be compared using a similarity metric. Then using the the greedy submodular maximization algorithm a diverse and summarizing subset of the data can be computed that can be more feasibly trained on. These three pre-processing steps can all be computed efficiently and in the following section we will show that the results for training fine tuning BERT are promising.

The code for this project can be found using this link.

https://drive.google.com/file/d/1KYx_4Gqa_6R4-GQMhC_bmASX-fMrnAMe/view?usp=sharing

4.1 Experiment Set Up

To test the effectiveness of the proposed pre-processing method to improve training speed the following experiment was used. A test and data set for the Story Cloze Test produced by Mostafazadeh et al. [2] was used for training and testing. To test the effectiveness of the pre-processing was tested by fine-tuning a BERT multiple choice model with with subsets of the data sets with varying sizes were selected using the pre-processing algorithm. To compare the effectiveness the accuracy on the test set was computed after 5 epochs of training as well as the time to compute both the pre-processing steps as well as the model training. The accuracy was simply computed using the following equation.

$$Accuracy = \frac{\#Correct}{\#Total\ Test\ Cases} \tag{18}$$

The HuggingFace's implementation of the base BERT model as well as the BERT for Mulitple Choice model was used to test the pre-processing algorithm [11]. An AdamW Optimizer was used with an initial learning rate of $2 * 10^{-5}$ was used to fine-tune the models.

4.2 Data Augmentation

The data set was initially had 1871 labeled data points of 5 Sentence stories, and after some preliminary testing it was found that the test points were all sufficiently diverse and pre-processing has little effect. To determine if the algorithm could effectively select a diverse and information dense subset, redundant data points were added to data to mimic data that was collected and not hand picked as in Mostafazadeh et al. [2]. More specifically, one percent of the data set was randomly sampled and duplicated one hundred times and added to the data set to add a significant subset of redundant information. In total, a data set of 3742, was used to train on.

4.3 Results

For comparison a model was trained using the full augmented data set with redundant information and the results on the test sets are summarized in table 1. The accuracy was computed after 5 epochs of traing and the time is the time to compute all 5 epochs and test.

Accuracy (%)	time(s)
76.86	946.17

Table 1: Results for Training Model using Full Data Set

4.4 Accuracy Analysis

The pre-processing was compared with subsets selected of sizes of %5, %10, %25, %40, %60 and %80 of the total data set. The pre-processing was tested using embedding generated using a base pre-trained BERT Model and the Gensim Doc2Vec algorthm. Then for comparison a randomly sample data set with out replacement of the same sizes was also trained. The accuracy results are recorded in table 2.

Size of Data Set (%)	BERT Vector Embeddings (%)	Doc2Vec Vector Embeddings(%)	Randomly Selected Data(%)
5.0	72.10	58.36	68.89
10.0	80.49	63.12	77.66
25.0	84.07	77.82	81.67
40.0	83.27	83.11	72.10
60.0	84.07	82.84	79.85
80.0	84.07	85.46	82.42

Table 2: Accuracy Results for Training Using the Proposed Pre-processing

For this experiment a $\lambda = 10$ was used and this is because from some minor testing preformed the best as well it was similar to the value use by the authors in Lin and Bilmes [4] for summarizing text. From the graph it is clear that using the BERT embedding performs the best when selecting smaller data sets. As the subsets get larger the embedding from the Doc2Vec Algorithm start performing similarly and both the BERT Embeddings and Doc2Vec outperform just randomly selecting the data points. An interesting observation is that the using the BERT Embedding with only selecting %10 of the data points can actually out perform the model training on the full data set. This is likely due to the fact that with the highly redundant data the model has issues of over fitting to particular data points causing the model to have difficulty generalizing.

4.5 Time Analysis

Note the experiments were run using Google Colab that was assigned a NIVIDA Tesla T4 GPU. Table 3 summarizes the time for the model to train using the varying sizes of data points. The models relatively took the same time if the subsets were the same size. These are the average values for the time to complete the traing for each model given the percentage of the total data set they are training on. It is clear that reducing the data set can significantly reduce the training time for a model. Now we want to evaluate the cost of overhead to do the pre-processing. We will compare

Size (%)	time(s)
5.0	185.17
10.0	225.52
25.0	323.05
40.0	416.82
60.0	510.48
80.0	575.85

Table 3: Training Times for data sets of Varying Sizes

the overhead for prepossessing the BERT Embeddings to the pre-processing using the Doc2Vec algorithm. The time to compute each step of the pre-processing will be found in table 4.

Step	BERT Vector Embeddings (s)	Doc2Vec Vector Embeddings (s)
Compute Embeddings	25.17	46.83
Compute Similarity Matrix	0.063	0.035
Select %0.05 of the data points	0.049	0.047
Select %0.10 of the data points	0.078	0.086
Select %0.25 of the data points	0.261	0.185
Select %0.40 of the data points	0.316	0.300
Select %0.60 of the data points	0.448	0.451
Select %0.80 of the data points	0.591	0.610

Table 4: Times to Complete Pre-processing Steps

From the data computation of the embeddings is the most expensive step of the process for both generating the BERT Embeddings and Doc2Vec Embeddings. Both computing the Similarity Matrix and data point selection both took under 1 second for both methods. If the input the data was already encoded into vectors, then the cost of computing the embedding can be ignored.

5 Disscussion and Next Steps

It is clear from the data collected that using this method on data sets with highly redundant information using this pre-processing frame work can provide improvements in terms of both accuracy and time. Using this method for fine tuning BERT with this specific data set showed that accuracy can be improved by %9 as well as reduce training time by %64. This is a strong improvements but the trade of is increased space complexity. The size of the similarity matrix grows at a rate $|V|^2$ which for larger data sets can become prohibitively large. In Mirzasoleiman et al. [7] they mitigate some of these issues by splitting the work onto a distributed network in a Map-Reduce style algorithm while still providing guarantees for performance. Also note that this pre-processing highly utilizes GPU acceleration for linear algebra operations to make the algorithm feasible. With out the GPU this method can quickly become infeasible.

Given more time to explore the topic, there are still more nuances to be explored. In terms of experimental results, it would be preferable to explore how much the density of redundancy affects the performance as well as how much the the trade off factor λ affects performance. It would also be preferable to test the performance of the pre-possessing on other language models and network architectures.

6 Conclusion

This report contributes a framework for data pre-processing for machine learning models to combat pit falls from highly redundant data sets. The introduces some trade offs between space complexity and model training efficiency. This work could be built off by apply this methods to other problems and adapt this algorithm to operate online to filter out poor data points for active learning applications. This frame work would prove useful if many models must be rapidly be trained such as

recommendation systems where they might want to train a model for each user. Using this framework could potential reduce the cost of computing each of these models.

Acknowledgement

I would like to thank my supervisors Stephen L. Smith and Bahman Gharesifard as well as our Professor Yaoliang Yu for the guidance.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[2] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1098. URL https://www.aclweb.org/anthology/N16-1098.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[4] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, Los Angeles, California, June 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N10-1134.

[5] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1954–1963, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/wei15.html.

[6] George Nemhauser, Laurence Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14:265–294, 12 1978. doi: 10.1007/BF01588971.

[7] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization. *Journal of Machine Learning Research*, 17(235):1–44, 2016. URL http://jmlr.org/papers/v17/mirzasoleiman16a.html.

[8] Zhongyang Li, Xiao Ding, and Ting Liu. Story ending prediction by transferable BERT. *CoRR*, abs/1905.07504, 2019. URL http://arxiv.org/abs/1905.07504.

[9] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583, 2019. URL http://arxiv.org/abs/1905.05583.

[10] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.

[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2019.