minBERT, NLP Tasks, and More

Stanford CS224N Default Project Milestone

Tiankai Yan
Institution of Computational and Mathematical Engineering(ICME)
Stanford University
ytk2304@stanford.edu

Abstract

In this project, I worked with minBERT, which is a streamlined version of of BERT (Devlin et al., 2018) and examined its state-of-the-art performance in solving a variety of natural language downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. To achieve enhanced performance, I employed multiple modeling strategies, beyond the standard implementation of BERT, demonstrating advantages in training compared to the conventional approach.

1 Key information to include

- **Title:** minBERT, NLP Tasks, and More
- **Team member names:** Tiankai Yan(ytk2304@stanford.edu)
- **Default Project**
- **Note:** "The accuracy result" in part 2 mentioned in this project refers to the final dev/test accuracy and Pearson correlation result produced by predictions. Saying "accuracy" does not mean there is only accuracy as evaluation metric.

2 Introduction

Natural Language Processing (NLP) is a branch of artificial intelligence (AI) that focuses on understanding, interpreting, and generating human languages. It is the one of the most popular research field, and every breakthrough it made captures the attention of millions of people around the world. Various methods have been developed for NLP, including statistical method and traditional time-series models (including LSTM) (Wang et al., 2016). But nowadays, the most effective and popular method is transformer-architecture pre-train models. Pre-trained models employ two strategies: feature-based and fine-tuning. BERT (Devlin et al., 2018), or Bidirectional Encoder Representations from Transformers (BERT), incorporates fine-tune strategy. It is the first model utilizing unsupervised bidirectional methods instead of traditional unidirectional tasks so that it achieves state-of-the-art performance in various NLP tasks. This project will further explore the application of BERT and aim to improve the model specifically for different downstream tasks.

3 Related Works

I reviewed a variety of works to construct the architecture of the models and extension strategies.

The attention idea is widely applied in many NLP tasks, including the attention LSTM (Wang et al., 2016) and transformer model (Vaswani et al., 2017). Attention mechanism allows a model to assign different weights to different parts of the input data, making the model focus more on more relevant information. Transformer (Vaswani et al., 2017) is the architecture for the layer of the BERT. Based on the idea of attention, it is a revolutionary approach to sequence-to-sequence task.

It explains the formulation of encoders and decoders, multi-head self-attention and feed-forward network. Capitalizing on the transformer, the BERT model (Devlin et al., 2018) is another revolutionary model in pre-training and fine-tuning models. Adopting bidirectional idea, the paper articulates its pre-training strategies: masked LM and next sentence prediction. The BERT paper also fully introduces multiple downstream tasks, which can be handled by BERT, and the state-of-the-art results achieved by BERT.

The model in this paper also incorporates other stratgies, inspired by multiple works. Bregman Proximal Point Optimization, (Gutman and Pena, 2018), proposed by Gutman and Pena, is a efficient approach handling non-smooth optimization landscapes and enhancing convergence rates and model performance on complex tasks. It is able to prevent aggressive updates. The Bregman paper articulates Proximal Point Optimization and application of Bregman divergence function. PALs and Multi-Task Learning (Stickland and Murray, 2019)is another strategy. The paper proposed PALs (projected attention layers), a low-dimensional multi-head attention layer and multi-task learning through sampling. This strategy will bring significant benefits in training through multiple downstream tasks. Moreover, Gradient Episodic Memory, articulated in (Lopez-Paz and Ranzato, 2017), is used to solve the catastrophic "forgetting" issue and improve the performance of continual learning through constraints checking and gradient projection.

4 Approach

4.1 BERT Implementation

The BERT model incorporate the ideas of transformers, including query, key, value, and attention. The implementation focuses on attention mechanism and transformer layer. BERT employs scaled dot-product attention in a multi-headed context, defined as:
MultiHeadAttention(Q, K, V) = Concat(head1, head2, ..., headh)W^o, while headi = Attention(QW_i^Q, KW_i^K, VW_i^V) = softmax($\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}}VW_i^V$). Each transformer block includes such attention, addition/normalization, feed forward and another add/norm.

AdamW Optimizer The AdamW optimizer, which is used within the BERT model, is an extension of the original Adam Optimizer. It is used to compute the gradient and to update the parameters/weights through strategies of estimation of the first and second moments and adaptive learning rate adjustment. In addition to the original Adam Optimizer, AdamW modifies the standard Adam Optimizer by decoupling the weight decay from the gradient updates, applying it directly to the parameters

4.2 Downstream Task

During the fine-tune stage, we address three downstream tasks:

- Sentiment Analysis: predict the positive/negative sentiment scale for a given sentence
- Paraphrase Detection: predict whether two given sentences are paraphrases
- Semantic Textual Similarity (STS): predict the scale of relevance of two given sentences

4.3 Extension

4.3.1 Bregman Proximal Point Optimization

Bregman Divergence Bregman divergences are a family of functions representing the distance between two points. The well-known Euclidean distance is an example of Bregman divergence. However, Bregman divergence functions can be asymmetric and do not follow triangle inequality.

In this project, two types Bregman divergence are employed.

- **Euclidean Distance** $d(\theta, \phi) = \sqrt{\sum_{i=1}^{n}(\theta_i - \phi_i)^2}$
- **KL((Kullback-Leibler)) Divergence** $D_{KL}(\theta \| \phi) = \sum_i \theta_i \log\left(\frac{\theta_i}{\phi_i}\right)$

Proximal Point Optimization using Bregman Divergence The proximal Point method, widely applied in many optimization problems, is used to find the minimum of a convex function. It operates by iteratively refining the solution using the current estimate to build a simpler subproblem whose solution serves as the next estimate. In our project, I incorporate the idea of proximal point optimization in our AdamW optimization by adding a "regularization term" to the original objective function. The regularization term is calculated by using a function to calculate the divergence between the current parameters and previous parameters **notated using** θ, ϕ. That function is Bregman divergence: it penalize the aggressive updating.

The key idea of the Bregman Proximal Point Optimization is to iteratively minimize the objective function augmented with a Bregman divergence term. It is able to help a quicker convergence for the parameters and avoid aggressive updating, working as a kind of "regularization".

The project will incorporate Bregman Proximal Point Optimization using two Bregman divergences sepeately and test their result and training performance. The hyperparameters will be tuned, and the details will be covered in the "Experiment" part.

4.3.2 Multi-task learning

Multitask Learning Multitask learning is the one of the core methods adopted in the project. Instead of training a separate model for each task, one model is trained simultaneously on multiple related tasks. The underlying principle is to share representations between tasks, leveraging commonalities and differences across tasks: sentiment analysis, paraphrase detection, and STS - to enhance learning efficiency and effectiveness. The training loop iterates over each task, computing gradients for each task within each epoch. Moreover, the following strategies are incorporated in the multi-task learning. Notice that the output in the three tasks are different.

1. **Sentiment Analysis**. For each sentence, output contain 5 logits, while each logit represents a score of the sentence to one of the five sentiment labels (0 -negative, 1-somewhat negative, 2-neutral, 3-somewhat positive, 4-positive). Thus, cross-entropy loss is utilized for sentiment analysis.
2. **Paraphrase Detection**. For each pair of sentences, the output is a single logit, as a score to determine whehter the two sentences are paraphrase or not. Hence, binary cross-entropy loss is employed, which is a function available in PyTorch:
   $BinaryCrossEntropyLoss(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$
3. **STS**.For each pair of sentences, STS task involves predicting a 5-labels scale, akin to the scale of sentiment analysis, rather than the binary scale used by paraphrase detection tasks. But, the output in STS is a single logit instead of 5-logits outputs in the sentiment analysis tasks. Consequently, the task is akin to a "regression" task, for which I use MSE loss for STS task.

4.3.3 GEM: Gradient Episodic Memory

Gradient Episodic Memory, or GEM, proposed in (Lopez-Paz and Ranzato, 2017), is a technique mitigating the problem of catastrophic forgetting in continual learning scenarios. "Episodic Memory" is psychological term, refering the ability to recall and mentally re-experience specific episodes from one's life. In my model, forgetting is that BERT loss the knowledge it captured from the previous round during a new round. Considering that BERT model in my project handles three different tasks simultaneously, a strong solution to prevent forgetting is necessary. The basic for GEM is to store gradients from previous tasks and use them to constrain the updates for the current task. The implementation of GEM is achieved through

- **Episodic Memory**. In AdamW optimization, we add a new argument for the class: memory, used for storing gradients.
- **Gradient Updates** When the model learns a new task, GEM compares the gradients of the loss with respect to the new task against the gradients with respect to the tasks stored in episodic memory. Given a loss function $L(\theta))$ and the gradient with respect to the parameters θ, denoted as $(\nabla L(\theta))$, the GEM constraint can be written as:
  $\nabla L(\theta^*)^T \nabla L(\theta) \geq 0$. If the constraint satisfied, the proposed parameter update g is unlikely

to increase the loss at previous tasks according to the paper, (Lopez-Paz and Ranzato, 2017). If the constraint isn't satisfied, on the other hand, the gradient should be projected onto the orthogonal complement of the previous gradients to prevent the loss increasing.

4.4 Other Extension

I have also experimented with the following extension methods:

- **Cosine-similarity** Inspired by (Reimers and Gurevych, 2019), Cosine-similarity, a measure to quantify the degree of similarity of two sentences, is also experimented. It was used in paraphrase and STS tasks. The equation for cosine-similarity is: given the embeddings for both tested sentences A, B, we have
  $cosine\_similarity(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$
  However, this method was ultimately discarded due to poor accuracy results.
- **PAL: Projected Attention Layers** Projected Attention Layers method is also proposed in the Stickland and Murray (2019). PAL did not contribute to improvements in accuracy, but it resulted in reduced training time. So some versions of my models adopt PALs with its parameter: projection size. Its architecture, defined in the PAL class, consists three parts
  1. **Projection** A layer to reduce the dimension from the hidden size to the projection size, used to focusing the model on the most salient features
  2. **Activation** Incorporates the ReLU activation function to introduce non-linearity.
  3. **Projection Back** A layer that projects the output back to the original hidden size, ensuring the output of PAL is compatible with subsequent steps in training.

4.5 Baselines

The following baselines are utilized to evaluate the results of my models.

- The accuracy result given by the prediction of binary-scale sentiment analysis through Attention-based LSTM articulated in the paper (Wang et al., 2016). It was compared with the my accuracy result of the single classifier implementation, including another binary-scale sentiment analysis task, for (part 1), and used to test the performance of my BERT modeling.
- Sentiment classification scores listed in handout was compared with my scores. It is also for part 1.
- For part 2, the result generated by a normal BERT model without any extension strategies was utilized as a baseline and was compared with the result provided by my final model.

The following comparisons were also conducted to explore the "optimal model". examining their training efficiencies as well as their accuracy results.

- Results of modelw with KL divergence and Euclidean Divergence.
- Results of models with and without PALs

5 Experiment

5.1 Dataset

In this project, these datasets are capitalized.

- **The CFIMDB dataset**. It contains movie reviews with their positive/negative labels. It is used for binary-scale sentiment analysis. The dataset is structured into three columns: ids for each sentence, the sentence, and its binary label.
- **Stanford Sentiment Treebank (SST) dataset** It contains movie reviews with their labels. There are in total 5 kinds of labels: negative, somewhat negative, neutral, somewhat positive, or positive, marked from 0 to 4. It is used for five-labels-scale sentiment analysis. The dataset is structured into four columns: ids for each sentence, the sentence, and its sentiment score, (a score from 0 to 4)

- **Quora Dataset** It includes pairs of sentences with binary labels indicating whether they are paraphrases or not. The dataset has four columns: ids for each pair of sentences, the first sentence, the second sentence, and their binary label.
- **SemEval STS Benchmark Dataset** It contains pairs of sentences with similarity scores indicating how similar they are, ranging 0 (unrelated) to 5 (equivalent meaning). The dataset is structured into four columns: ids for each pair of sentences, the first sentence, the second sentence, and their similarity score.

5.2 Evaluation Method

I evaluate the performance of my model through these metrics

- Accuracy and Pearson correlation. For the sentiment analysis and the paraphrase detection task, I evaluate my model using accuracy. It is the ratio of total number of correct predictions to total number of predictions, through *np.mean(np.array(pred) == np.array(observed))*. The result is a range from 0 to 1. For the semantic textual similarity (STS), I evaluate my model using Pearson correlation,
  $r = \frac{\sum_{i=1}^{n}(pred_i - \overline{pred})(observed_i - \overline{observed})}{\sqrt{\sum_{i=1}^{n}(pred_i - \overline{pred})^2}\sqrt{\sum_{i=1}^{n}(observed_i - \overline{observed})^2}}$,
  in coding, it is implemented through *np.corrcoef(pred,observed)*
- During the comparison of implementation with two strategies, I will also compare their computing time and consumed computing units. A tiny improvement at the cost of huge increase in computing time and computing units is not worthwhile.

5.3 Experiment Details

The following hyper-parameters are used in my model (and for part 2): hidden size of 768, drop out probability of 0.3, learning rate of 1e-5, with scheduler using *ReduceLROnPlateau*, weight decay=1e-4. The Bregman ratio, which is multiplied with Bregman divergence, is 0.01. The projection size used in PAL is 256. For every epoch, the model will be trianed, then iterate through tasks, then through batches, executing forward and backward process. The Bregman divergence, multiplied with Bregman ratio, will be added to a part of the loss. Then, during the optimizer step, the constraint in GEM is checked to determine whether it is necessary to project gradients.

5.4 Results

5.4.1 Part 1 Results

The following is my result for part 1 comparing with the baseline listed in the handout:

	My result	Handout Result
Pretraining for SST, Dev Accuracy	0.402	0.390
Pretraining for CFIMDB, Dev Accuracy	0.784	0.780
Finetuning for SST, Dev Accuracy	0.520	0.515
Finetuning for CFIMDB, Dev Accuracy	0.967	0.966

Also, the result of binary-scale sentiment analysis (equivalent to "Finetuning for CFIMDB", another binary-scale sentiment analysis, which is 0.967 in my model) listed in the Attention-based LSTM paper (Wang et al., 2016): 0.899, my model displays an obvious advantages comparing to these baselines.

5.4.2 Part 2 Results

The final results for part 2 is:

- Sentiment Analysis: Test Accuracy of 0.448
- Paraphrase Detection: Test Accuracy of 0.807
- SST: Test Correlation: of 0.547

- Overall test score: 0.676
- Sentiment Analysis: Dev Accuracy of 0.454
- Paraphrase Detection: Dev Accuracy of 0.802
- SST: Dev Correlation: of 0.544

The final model is implemented with Bregman Proximal Point Optimization of Euclidean Divergence, Multi-task learning, and Gradient Episodic Memory, without PAL. It used about 10 hours for training, consuming about 20 compute units through training in Google Colab using T4 GPU.

It shows a significant advantages comparing to the result without any extension strategies, which is the baseline model for part 2:

	The final model	Model without any extensions
Sentiment Analysis, Dev Accuracy	0.454	0.321
araphrase Detection, Dev Accuracy	0.802	0.376
SST, Dev Correlation	0.544	0.025

I also collected those following results.

1. Model with Bregman Proximal Point Optimization of **KL Divergence**, multi-task learning, GEM and without PAL:

	The final model	Model with KL Divergence
Sentiment Analysis, Dev Accuracy	0.454	0.460
araphrase Detection, Dev Accuracy	0.802	0.818
SST, Dev Correlation	0.544	0.540

   The model with KL Divergence is slightly better than the model with Euclidean Divergence. However, to achieve such a slight improvement, 30 hours of training and 60 compute units were used for training. It is not worthwhile.

2. Model with with Bregman Proximal Point Optimization of Euclidean Divergence, multi-task learning, GEM and **PAL**

	The final model	Model with PAL
Sentiment Analysis, Dev Accuracy	0.454	0.450
araphrase Detection, Dev Accuracy	0.802	0.801
SST, Dev Correlation	0.544	0.529

   The model with PAL displays a small disadvantage in accuracy result, however, it reduced the training time by half an hour. PAL implementation failed to improve the accuracy, but it has an advantage in training time. So whether to incorporate PAL is depending on a trade-off between training time and accuracy, even though the difference is not significant.

5.5 Analysis

The BERT implementations and the extension strategies generates very promising results in understanding natural languages and handling multi tasks. However, there are still huge space for improvement. By carefully inspecting the implementation of my models and the results they achieved, the following qualitative points are concluded.

- The multitask learning plays an very important role. Without multi task learning, the probability of correct labeling prediction, even for binary scale, is very limited, (considering that the independent random prediction will also achieve the approximate 0.5 accuracy). It indicates that the similarities among different tasks are very limited, and the "knowledge" generated from single task A learning have little effect on handling task B.

- For the model without any extension, the training suffers considerably from aggressive updating. It was very common that the current epoch achieved substantial decreasing in dev accuracy comparing to the previous epoch. The usage of Bregman Proximal Point Optimization and GEM solve, which carefully check for aggressive updating and loss increasing, solved that issue. However, the "regularization" might be too strong: the models achieved the approximate accuracy and correlation in the first 3 epochs (while the original model oscillated across epochs). In the following epochs, there was neither substantial accuracy loss, nor great accuracy improvements. It turns out that a robust shield against loss in accuracy should be paired with a even stronger engine empowering accuracy gain, which my project currently lacks.
- Whether to incorporate a model is not solely determined by the accuracy, but also by a trade-off between computing efficiency and accuracy results. An impractical model, such as one that uses KL divergence, achieves tiny improvements in accuracy at a huge cost in computing time and resources. Although the big-O complexity of Euclidean divergence and KL divergence is the same, in practice, squares and additions are computationally simpler, while divisions and logarithm are computationally more consuming. During the Euclidean divergence, the complex square root is computed only once for each pair, whereas KL divergence requires complex calculations for every iteration, leading to considerable differences in practical application, 10 hours vs 30 hours.
- According to the paper (Stickland and Murray, 2019), for multi-task learning, more resources should be allocated to complex tasks and fewer to simpler tasks. The current multi-task learning method I used is called "Round-Robin" according to the paper, (Stickland and Murray, 2019), and it caused an imbalance: during training, the size for paraphrase detection receive the largest amount of size: 17688. Paraphrase detection is the simplest task, predicting a binary scale. Sentiment analysis tasks and STS, adopting 5-label scale, has only 1068 and 755. That means, in our model, the simplest task is allocated with the largest amount of resources, which is not inefficient. This could be a fundamental reason for the comparatively poor performance in the other two tasks.

6 Conclusion

This project has primarily accomplished the following:

- (Part 1)Implementation of BERT and single classifier of sentiment analysis, yielding strong prediction results compared with baseline on the handout and baseline presented in Attention-LSTM paper (Wang et al., 2016).
- The implementation of multi-task classifier and the extension strategies: Bregman Proximal Point Optimization, Multi-task learning, Gradient Episodic Memory, and Project Attention Layers.
- The multi-task classifier was trained and tested, revealing substantial improvements in prediction results through the extension strategies. In addition, the comparison of different model versions facilitated the identification of the optimal final model and explored a trade-off between computing efficiency and accuracy result.
- The project also articulates and investigates both the improvements and the obstacles, paving the way for the future progress.

Limitations and Future Work The project does encounter following limitations and challenges:

- A Rapid convergence was observed, but the accuracy results were not as promising as expected.
- The project lacks an efficient weight balancing methods. Currently , the training dataset for the simplest task is disproportionately large.
- The project also lacks in an through exploration of task relationship. It operated as tasks were entirely separate/

To address these limitations, especially the accuracy issue, the following methods could be adopted in future works:

- **Dynamic Task Weighting** An efficient, dynamic task weighting method, adjusting the weights based on the performance of the model on each task during training. The following strategies could be implemented, including linear scaling, exponential scaling, or performance threshold. The paper (Gong et al., 2019) would be helpful.
- **Cross-Task Learning** Relating the update results among tasks so that the progress in one task could benefit the other based on their similarities or differences. Techniques like stacking, chaining, or voting could be employed to combine tasks more effectively.
- **Change in Learning Architecture** I plan to test a variety of models in order to improve accuracy, with a focus on **Ensemble Models**: training multiple models and combining their predictions for each task. Additionally, expanding the dataset, addition pre-training would also likely prove beneficial.

Thank you for a wonderful quarter. This project is immensely interesting, and it provides me with extensive learning opportunities in modeling, implementation, and the exploration of new methods and ideas. It also trains my skills in academic reading, coding, and technical writing. Moreover, the numerous discussions with TAs are invaluable to me. I wish you will have a nice Spring Break.

References

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.

Ting Gong, Tyler Lee, and et. al. 2019. A comparison of loss weighting strategies for multi task learning in deep neural networks. IEEE Access 7 (2019): 141627-141632.

D. H. Gutman and J. F. Pena. 2018. A unified framework for bregman proximal methods: subgradient, gradient, and accelerated gradient schemes. arXiv preprint arXiv:1812.10198.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. Advances in neural information processing systems 30.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. International Conference on Machine Learning. PMLR.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. Advances in Neural Information Processing Systems 30.

Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. 2016. Attention-based lstm for aspect-level sentiment classification. Proceedings of the 2016 conference on empirical methods in natural language processing.