Walk Less and Only Down Smooth Valleys

Stanford CS224N Default Project

Quinn Hollister
Stanford University
bh9vw@stanford.edu

Julian Cooper
Stanford University
jelc@stanford.edu

Thomas Brink
Stanford University
tbrink@stanford.edu

Abstract

Transfer learning has become a valuable tool for handling a variety of downstream NLP tasks given a single generic pre-trained model [1]. Since most of the downstream tasks are more specialized, quality training data is a scarce resource, making training large models from scratch very difficult. In this project, we investigate how the performance of a pre-trained BERT encoder model changes for three different downstream prediction tasks when we include (1) additional pre-training on target-domain data, (2) additional fine-tuning on more representative mix of downstream tasks, and (3) regularization in our fine-tuning loss function and parameter (SMART by Jiang et al. [2]). We find that SMART prevents over-fitting that is common in fine-tuning BERT models to downstream tasks. This adversarial regularization is especially effective when combined with multi-task learning.

1 Approach

Model extensions. We plan to extend the default minBERT model in three ways:

1. **Additional pre-training for target domain** [3]: While we expect our pre-trained minBERT model weights to be effective for paraphrasing and semantic similarity analyses, we assume it will struggle to classify sentiment. This is because the Wikipedia corpus is largely filled with non-emotive, informational language. Because of this, we want to try adding an incremental pre-training layer that includes more emotive language (e.g. editorial corpus MULTIOpEd [4]) trained on masked language modeling and next sentence prediction (original BERT objectives [5]).
2. **Round-robin multitask fine-tuning** [6]: The default implementation assumes that fine-tuning only on sentiment classification for SST dataset will generalize well to paraphrasing and similarity prediction tasks. Even superficial experiments proved to us that this was not true! To address, we have implemented a batch-level round-robin routine. For each batch iteration, we cycle through training dataloaders for SST, Quora and STS and perform one update for each so our fine-tuned model will see examples for all tasks. The updates use cross entropy, binary cross entropy and cosine similarity loss respectively.
3. **Regularization of fine-tuning loss and optimizer step** [2]: Many fine-tuning routines suffer from overfitting, which lead to poor performance on test sets of downstream prediction tasks (see literature review). Having carefully included corpus text from relevant domains in our pre-training steps, we want to make sure our fine-tuning does not diverge too rapidly from the pre-trained weights. For this, we make use of regularization techniques by implementing SMART (see below for details).

We have implemented a version (2) and (3) from the planned extensions listed above. These results are shown in the experiments section. Given the details of SMART regularization are the most complex, we include a section summarizing the math and logic.

SMART implementation. To effectively control the **extremely high complexity** of the LLM, Jiang et al. [2] propose a smoothness-inducing adversarial regularization technique. The desired property is

that when the input x is perturbed by a small amount, the output should not change much. To achieve this, Jiang et al. [2] optimize loss $\mathcal{F}(\theta)$ using: $\min_\theta \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta)$ where,

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i;\theta), y_i) \qquad \text{regular loss function}$$

$$\mathcal{R}_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{\|\tilde{x_i} - x_i\|_p \leq \epsilon} l_s\left(f(\tilde{x_i};\theta), f(x_i;\theta)\right) \qquad \text{regularization term}$$

$$l_s(P,Q) = \mathcal{D}_{KL}(P\|Q) + \mathcal{D}_{KL}(Q\|P) \qquad \text{symmetric KL-divergence}$$

Note that this regularizer term is measuring the local Lipschitz continuity under the symmetrized KL-divergence metric. So, the output of our model does not change much if we inject a small perturbation (constrained to be ϵ small in the p-euclidean metric) to the input. Thus, we can encourage our model f to be smooth within the neighborhoods of our inputs. This is particularly helpful when working in a low-resource domain task.

To prevent **aggressive updating** , the optimization routine is changed so that we introduce a trust-region-type regularization at each iteration, so we only update the model within a small neighborhood of the previous iterate. In order to solve the regularizer term, Jiang et al. [2] develop a class of Bregman proximal point optimization methods.

Specifically, we update parameters θ by: $\theta_{t+1} = \text{argmin}_\theta \mathcal{F}(\theta) + \mu \mathcal{D}_{Breg}(\theta, \tilde{\theta}_t)$, where,

$$\mathcal{D}_{Breg}(\theta, \tilde{\theta}_t) = \frac{1}{n}\sum_{i=1}^{n} l_s\left(f(x_i;\theta), f(x_i;\tilde{\theta}_t)\right) \qquad \text{Bregman divergence}$$

$$\tilde{\theta}_t = (1-\beta)\theta_t + \beta\tilde{\theta}_{t-1} \qquad \text{regularized update step}$$

Baselines. First, we compare our sentiment analysis accuracy with the baseline scores presented in the default project handout. These scores include baseline accuracies for both pre-trained and finetuned models on the SST and CFIMDB datasets. We also verify that multitask classification for default pretrain and finetune matches our single-task classifier results for sentiment analysis of SST dataset. Mostly, we can view the handout benchmarks as 'correctness' tests for our implementation. (See Table 1 in results section.)

Second, we compare all of our models (i.e. pretrain default, finetune default, finetune with extensions) against each other across the three prediction tasks evaluated by the multitask classifier. Here, we treat the finetune default BERT model defined in section 3 of the handout as our baseline. Since the idea behind the regularization used for our extensions is to prevent overfitting during fine-tuning, we would hope our final model improves on our benchmarks.

2 Experiments

Data. For fine-tuning and evaluating our model on downstream prediction tasks, we will use the Stanford Sentiment Treebank, Quora Dataset, and SemEval STS Benchmark Dataset. The *Stanford Sentiment Treebank* (SST) dataset consists of 11,855 single sentence reviews, where a review is labelled categorically {negative, somewhat negative, neutral, somewhat positive, positive}. The *Quora Dataset* (Quora) consists of 400,000 question pairs with binary labels (true if one question is paraphrasing the other). And, finally, the *SemEval STS Benchmark Dataset* (STS) consists of 8,628 different sentence pairs, with each given a score from 0 (unrelated) to 5 (equivalent meaning).

Evaluation method. We use accuracy for the sentiment analysis and paraphrase detection tasks, and Pearson correlation for semantic textual similarity. We note that the sentiment analysis accuracy is based on multi-class classification, while the paraphrase detection task is binary.

Experimental details. Beyond our baseline results (default pretrain and finetune), we ran experiments for each model extension we have implemented so far (extensions 2 and 3). The following describes configuration choices made for each:

- **Ext 2: Round-robin fine-tuning** (rrobin): Set number of iterations (over batches from all three datasets) to be `floor(len(sts_train_data) / args.batch_size)`, where STS

represents the smallest task-specific training dataset. In this way, we guarantee training equally across all three tasks, which helps balance our finetuning process (otherwise Quora may dominate), but comes at the cost of throwing away potentially informative training data.

- **Ext 3: SMART regularization** (smart): The hyperparameter values specific to the SMART implementation were $\lambda = 1$, $\epsilon = 1e-5$, $\sigma = 1e-6$, $\beta = 0.995$, $\mu = 1$, $\eta = 1e-3$ and K = 1. We also found this technique increased memory usage and so needed to restrict batch size to 8 rather than 64.

The learning rate only differed between the finetune and pretraining stages, where pretrain had a learning rate of 1e-3 and finetune had a learning rate of 1e-5.

Multitask classifier training times per epoch (on aws ec2 instance): pretrain default 0:09 min, default finetune default 0:25 min, finetune rrobin 1:07, finetune smart 2:41, and finetune rrobin+smart 7:44 min. The need to reduce batch size for models that used SMART regularization (due to memory usage) was a major time cost.

Results Table 1 compares dev accuracy of our default model with benchmarks provided in the handout for the single-task classifier. Our results are all close to the benchmarks which gave us confidence in the 'correctness' of our default implementation.

Prediction task	Sentiment (SST)		Sentiment (CFIMDB)	
Model type	Our model	Benchmark	Our model	Benchmark
Pretrain default	0.393	0.390 (0.007)	0.788	0.780 (0.002)
Finetune default	0.522	0.515 (0.004)	0.963	0.966 (0.007)

Table 1: Dev accuracy of default model vs benchmarks for single-task classifier

Table 2 compares dev accuracies of our default implementation (treated as our baseline from now on) with our different model extensions. As hoped, both our batch-level round-robin and SMART regularization extensions independently improved our model's overall performance, particularly its ability to generalize to paraphrase and similarity tasks. Our best result came from combining both of these extensions (rrobin+smart).

Model type	Sentiment (SST)	Paraphrase (Quora)	Similarity (STS)
Pretrain default	0.396 (*)	0.380	0.019
Finetune default	0.525 (*)	0.522	0.240
Finetune rrobin	0.524	0.726	0.583
Finetune smart	0.520	0.501	0.382
Finetune rrobin+smart	0.532	0.741	0.680

Table 2: Dev accuracies of default vs model extensions for multi-task classifier

3 Future work

In their development of the SMART framework, Jiang et al. found that combining SMART with multi-task learning achieved the best benchmark performance results [2]. In a similar vein, we would like to explore how MTL could improve our model and combine it with gradient surgery in order to optimize our model on all of the tasks instead of just the semantic similarity task. Also, our model has a number of hyperparameters and trying to find the optimal combination of these is computationally expensive and time-consuming. One idea we are eager to try is extending the bayseian optimization framework described by Nando de Freitas et. al. [7].

References

[1] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[2] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. SMART: robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *CoRR*, abs/1911.03437, 2019.

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

[4] Siyi Liu, Sihao Chen, Xander Uyttendaele, and Dan Roth. MultiOpEd: A Corpus of Multi-Perspective News Editorials. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

[5] 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.

[6] Cs 224n: Default final project: minbert and downstream tasks.

[7] Nando de Freitas et. al. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.