Improving the performance of miniBERT and BERTaaR (BERT as a Recruiter)

Stanford CS224N Default Project

Prabhjot Singh Rai
Department of Computer Science
Stanford University
prabhjot@stanford.edu

Jared Isobe
Department of Computer Science
Stanford University
jtisobe@stanford.edu

Abstract

We use a miniBERT architecture on sentiment analysis, paraphrase detection, and semantic textual similarity tasks. To boost performance, we used cosine-similairty fine-tunning and multi-task learning to get generalizable sentence embeddings that perform better on all three tasks. We achieved an overall score of 0.589, a sentiment accuracy of 0.282, a paraphrase accuracy of 0.803, and a STS correctness of 0.365. In addition, we attempted to use BERT as a Recruiter (BERTaaR) on a public Yahoo! jobs dataset to perform information retrieval.

1 Key Information to include

- TA mentor: Arvind Mahankali
- External collaborators (if no, indicate "No"): No
- External mentor (if no, indicate "No"): No
- Sharing project (if no, indicate "No"): I (Jared) used this project for inspiration for my CS281: Ethics of AI project, which looks into bias using BERT on the Equity Evaluation Corpus.

2 Introduction

BERT (Bidirectional Encode Representations from Transformers) was an innovative NLP model developed by Devlin et al. (2019) at Google AI Language in 2018. The following are the key ideas and components of the BERT model:

1. Transformer Architecture: BERT relies on self-attention mechanisms rather than Recurrent Neural Networks (RNNs). This enables the model to process input data in parallel thereby improving the efficiency by capturing complex dependencies in text.
2. Bidirectional Context: One of the significant aspects of BERT is bidirectional training of the Transformer. Rather than reading the text input sequentially (left-to-right or right-to-left), BERT reads the entire sequence of words at once which allows the model to learn the context of a word based on all of its surroundings.

3 Related Work

Sun et al., 2020 outlines a three step solution to fine-tune BERT (Figure 1) is to conduct further pre-training on BERT with within-task training data or in-domain data, fine-tune with multi-task learning, and then fine tune for the actual task. Some other methods to improve performance include

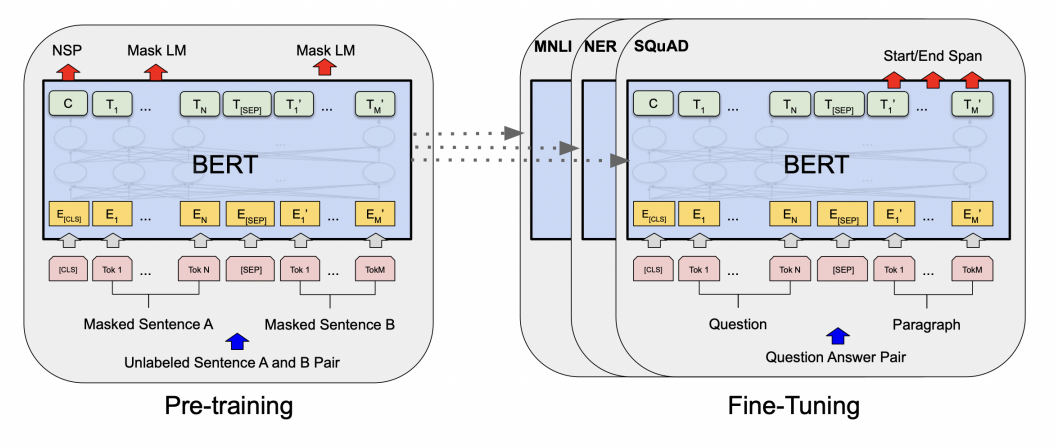

Figure 1: The method for training BERT using pre-training and fine-tuning in Devlin et al. (2019)
.

pre-processing training data, layer selection and learning, catastrophic forgetting, and low-shot problems. Through these methods, SOTA results were achieved in 2020 on several tasks.

One example of job recruiting using NLP is in Abdollahnejad et al. (2021), which created a BERT model that outputs the chance that an application will be a good fit for a job. The model takes as input a combination of the job description and the applicant's application and outputs a value between 0 and 1 that represents the probability that the applicant will be hired, with values closer to 1 representing a higher probability of the applicant being hired (Figure 2).

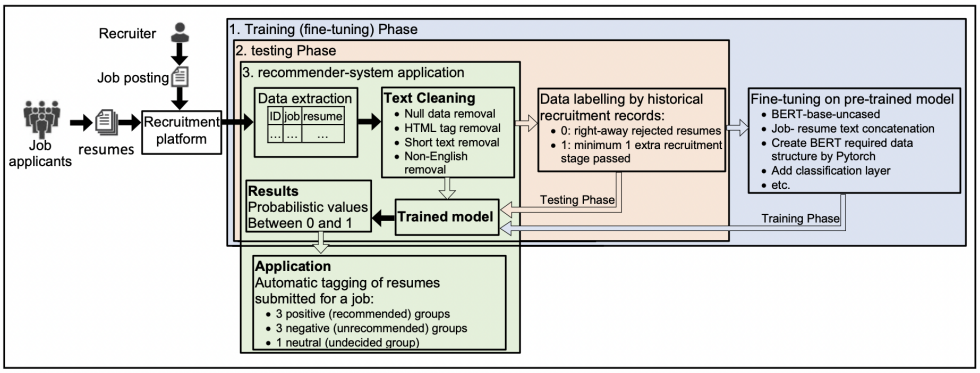

Figure 2: An example of how a BERT model can be used in the recruitment process from Abdollahnejad et al. (2021). The BERT model is used to output a score that represents the probability that they would get the job.

4 Approach

As a part of the default project, the approaches were divided into the following sections:

4.1 Implement BERT Multi-head Self-attention, Transformer Layer and Adam Optimizer

As a part of the default project, we implemented the BERT model from scratch. We implemented the multi-head self-attention mechanism, which is the core of the transformer architecture. We also implemented the BERT Transformer Layer and performed sanity checks to ensure that the model was implemented correctly.

This was followed by implementing the Adam optimizer, which is a variant of the stochastic gradient descent algorithm. We used the Adam optimizer to train the BERT model on the downstream tasks.

4.2 Fine-tuning BERT on sentiment analysis

We fine-tuned the BERT model on the Stanford Sentiment Treebank (SST) dataset and CFIMDB dataset and obtained the results as defined in the handout.

4.3 Fine-tuning BERT on Additional Downstream Tasks

We fine-tuned the BERT model on the Quora Paraphrase Dataset and the Semantic Textual Similarity (STS) dataset along with the Sentiment Analysis task. The high level goal that we aimed to achieve was to get generalizable sentence embeddings that perform better on all three tasks by finetuning the entire model first on the dataset for which we have the most data (Paraphrase), followed by the similarity training task (again on full model), and finally last linear layer training on the sentiment analysis task.

This was our approach to multi-task learning based on the algorithm in Liu et al. (2019). We also implemented cosine similarity fine-tuning to further boost performance of our model, as described in Reimers and Gurevych (2019).

4.4 BERT as a Recruiter

After we completed these tasks, we attempted to do information retrieval on an Online Job Postings dataset from Yahoo! mailing group on Kaggle. However, we were not able to make substantial progress due to time constraints.

5 Model Architecture

5.1 Pooler Output

We had implemented 3 vanilla NNs (each single layered with ReLU activation) trained on only the SST task for the baseline submission. The following was the architecture for the last linear layer of the model for the three tasks:

```
self.sentiment_classifier = nn.Sequential(
       nn.Linear(BERT_HIDDEN_SIZE, BERT_HIDDEN_SIZE // 2),
       nn.ReLU(),
       nn.Linear(BERT_HIDDEN_SIZE // 2, N_SENTIMENT_CLASSES)
   )

   self.paraphrase_classifier = nn.Sequential(
       nn.Linear(BERT_HIDDEN_SIZE * 2, BERT_HIDDEN_SIZE),
       nn.ReLU(),
       nn.Linear(BERT_HIDDEN_SIZE, 1)
   )
   self.similarity_classifier = nn.Sequential(
       nn.Linear(BERT_HIDDEN_SIZE * 2, BERT_HIDDEN_SIZE),
       nn.ReLU(),
       nn.Linear(BERT_HIDDEN_SIZE, 1)
   )
```

These linear layers were attached to the pooler output (CLS token) of the BERT model as follows:

```
def forward(self, input_ids, attention_mask):
    x = self.bert(input_ids, attention_mask)['pooler_output']
    x = self.dropout(x)
    return x
```

```
    def predict_sentiment(self, input_ids, attention_mask):
        hidden_states = self.forward(input_ids, attention_mask)
        return self.sentiment_classifier(hidden_states)

    def predict_paraphrase(self,
                           input_ids_1, attention_mask_1,
                           input_ids_2, attention_mask_2):
        hidden_states_1 = self.forward(input_ids_1, attention_mask_1)
        hidden_states_2 = self.forward(input_ids_2, attention_mask_2)
        hidden_states = torch.cat((hidden_states_1, hidden_states_2), dim=1)
        return self.paraphrase_classifier(hidden_states)

    def predict_similarity(self,
                           input_ids_1, attention_mask_1,
                           input_ids_2, attention_mask_2):
        hidden_states_1 = self.forward(input_ids_1, attention_mask_1)
        hidden_states_2 = self.forward(input_ids_2, attention_mask_2)
        hidden_states = torch.cat((hidden_states_1, hidden_states_2), dim=1)
        return self.similarity_classifier(hidden_states)
```

The basic idea was to take the CLS token embedding output by the BERT model as input to the sentiment classification, output two CLS token embeddings for the two sentences in the paraphrase task, and output two CLS token embeddings for the two sentences in the STS task and then concatenate them to get a single embedding for the two sentences.

5.2 Last Linear Layer

As we will discuss in the experiments, in order to improve the performance of the model, we also tried to incorporate a bidirectional LSTM layer after the BERT last layer output for sentiment classification. The model architecture for Sentiment Analysis was as follows:

```
class BidirectionalLSTM(nn.Module):

    def __init__(self, input_size, hidden_size):
        super(BidirectionalLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, bidirectional=True, batch_first=True)

    def forward(self, x):
        out, _ = self.lstm(x)
        return out

class CustomModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(CustomModel, self).__init__()
        self.lstm = BidirectionalLSTM(input_size, hidden_size)
        self.fc1 = nn.Linear(hidden_size*2, 50)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(50, num_classes)

    def forward(self, x):
        x = self.lstm(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

This was integrated with the existing bert model by padding the output of the BERT sequence to the maximum sequence length and then passing it through the LSTM layer. The output of the LSTM layer was then passed through the linear layers to get the final sentiment classification.

6 Training

In our codebase, We used Ray with PyTorch Lightning to scale up training on multiple GPUs. The approach for distributed training came in because the training on the full model was taking a long time on a single GPU. Moreover, we wanted to experiment with different hyperparameters and training strategies to improve the performance of the model and reduce the training time. Therefore, since PyTorch Lightning provides a simple API for distributed training, we decided to use it for our experiments.

Through multiple experiments, we found that our model was able to fit in a single GPU and the only overhead was the time taken to train the paraphrase datasets, therefore, we decided to shard the data with DDP and train on multiple GPUs. The multitask_classifier_lightning.py[Trainer] was created to extensively use and experiment with the PyTorch Lightning's API (ex FP32, FP16 etc.) under provisioning by Ray. Since the cluster was setup using cloud provider, it gave us flexibility to choose the right GPU instance as well. There was one caveat configuring cloud storage while setting up ray with pytorch lightning as listed in the caution section here. All workers need to have access to a shared storage.

7 Experiments

7.1 Data

7.1.1 Stanford Sentiment Treebank Dataset

The SST dataset is a collection of over 11,000 sentence with human annotations of sentiment, ranging from negative, somewhat negative, neutral, somewhat positive, to positive Socher et al. (2013).

7.1.2 Quora Paraphrase Dataset

The Quora Paraphrase Dataset is a collection of 400,000 lines of potential duplicate questions. The label is a binary "yes" or "no" to if it is a paraphrase Iyer et al. (2017).

7.1.3 Semantic Textual Similarity Dataset

The STS dataset ranks similarity between two pieces of text between 0 (not at all related) and 5 (same meaning) Agirre et al. (2013).

7.1.4 CFIMDB

The CFIMDB dataset is a collection over 2,000 sentences from highly polar movie reviews with a binary label for positive or negative IMDB.

7.1.5 Yahoo! mailing group Dataset from Kaggle

We will be incorporating existing public dataset on Online Job Postings found on Kaggle. The dataset consists of 19,000 job postings that were posted through the Armenian human resource portal CareerCenter. This dataset was extracted from the Yahoo! mailing group which was the only human resource portal in the early 2000s. The purpose of this dataset will be to attempt to do informational retrieval.

7.2 Evaluation method

- Sentiment Classification - Accuracy of label of positive or negative sentiment (can be binary or a range, such as in the SST case).
- Paraphrase detection - Accuracy of binary labels of if the two sentences are paraphrases.
- Sentiment Analysis - Pearson value of the actual vs predicted sentiment score.

7.3 Experimental details

We used a learning rate of of 1e-5 for full model pretraining and 1e-3 for final layer. The training time was 1.5 days for all three tasks on a 40 core, m3 max chip GPU.

7.4 Results

	Sentiment Acc.	Paraphrase Acc.	STS. Correct	Overall
Basic implementation (Baseline)	0.520	0.384	0.080	0.482
Para, STS (w/o cosine), SST Full (dev)	0.485	0.471	0.335	0.541
Para Full, STS (w/ cosine) Full, SST Last	0.282	0.803	0.365	0.589

8 Analysis

8.1 Paraphrase full model training

We stared by training for 10 epochs on paraphrase task, stopping at 9 due to what appears to be a minimum being reached (Figure 3). In the chart, it shows that the loss (binary cross entropy with logits) decreased and the weights converged as expected. We were able to resume the experiment after stopping, allowing us to continue on other tasks. With DDP, we were able to reduce the overall training time for this from 1.5 days to 5 hours. This experiment used 6 works (Nvidia Tesla T4s) each with 16GB vram.
Experiment Name: cosine-ddp-full-model-10-epochs-per-task
Trainer: TorchTrainer_086a8_00000_0_2024-06-09_09-59-59
Logs: job-driver-raysubmit_XxzyPncrRSBr1ckR.log

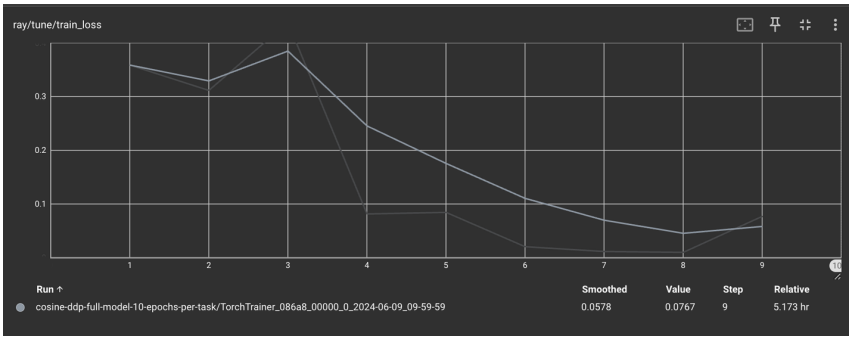

Figure 3: Loss over 9 epochs on the paraphrase task (cosine-ddp-full-model-10-epochs-per-task)

8.2 Paraphrase, STS (without cosine) and SST full model training

We see the validation set evaluation metrics in Figure 4 when we run STS (without cosine) and SST full model training. STS accuracy increased when training on STS but then decreased when training on the SST tasks. This leads us to believe that the model isn't retaining much information when training on STS, and subsequently can lose it by training on a different task. So, we believe we should retrain only the final layer for SST to prevent this loss of STS learning. STS learning increased consistently, which is partially due to the fact that there are no other learning tasks after.

Experiment Name: resume-from-para
Trainer:
STS training: TorchTrainer_119d5_00000_0_2024-06-09_11-55-26
SST training: TorchTrainer_51619_00000_0_2024-06-09_12-11-31
Logs: job-driver-raysubmit_d2KrQJRNFECSbGTK.log

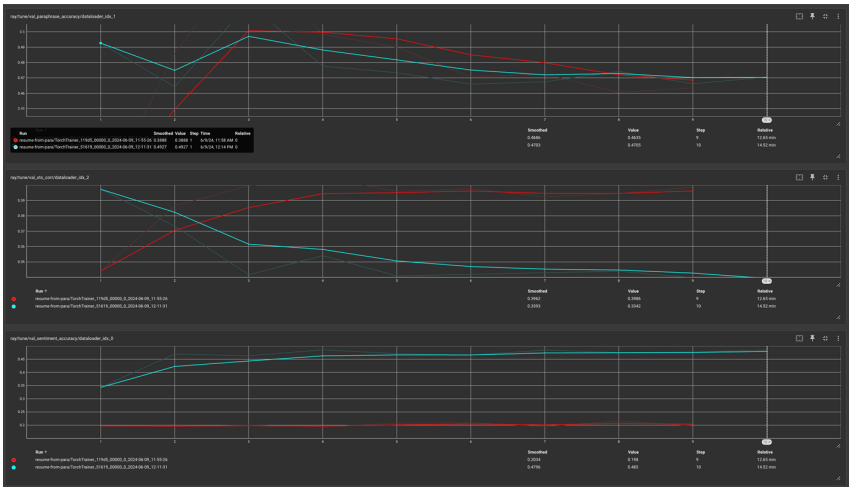

Figure 4: Training on STS and SST full model over 9 epochs with a learning rate of 1e-5 and a batch size of 32.

8.3 Paraphrase, STS (with cosine) and SST Full model training

When using cosine-similarity for STS, we see an increase in performance for all tasks (Figure 5).
Experiment Name: resume-from-para-cosine
Trainers:
STS training: TorchTrainer_b0055_00000_0_2024-06-09_13-25-45
SST training: TorchTrainer_f32e8_00000_0_2024-06-09_13-41-57
Logs: job-driver-raysubmit_Z5QuyrCvcgiuvJ99.log

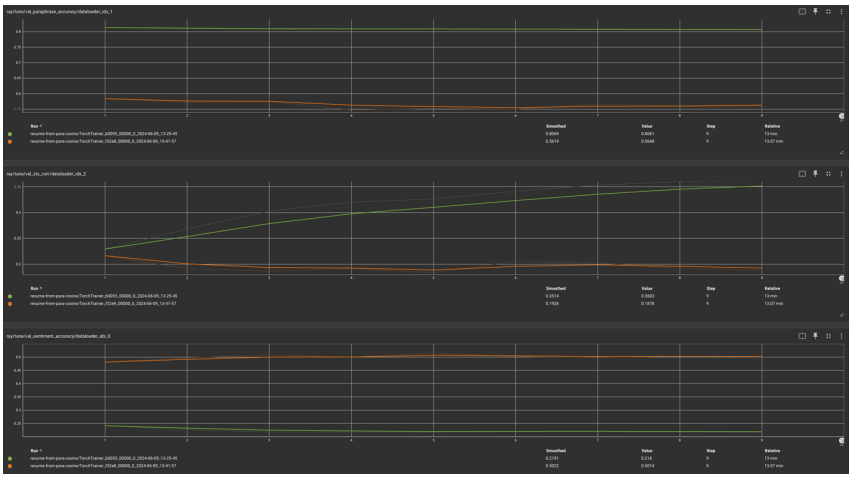

Figure 5: Training on STS with cosine-similarity and SST full model over 9 epochs with a learning rate of 1e-5 and a batch size of 32.

8.4 Full model trained on Paraphrase, Full Model on STS (with cosine), and Last Linear Layer on SST

We see the best results when running full model training on paraphrase, full model on STS with cosine and last linear layer training on SST (Figure 6). This allows the sentiment classification accuracy to increase without changing the accuracy of the other two metrics when conducting training on the last liner layer on SST (Figure 7).

The full model on STS is trained by:
Experiment name: resume-from-para-cosine
Trainer: TorchTrainer_e4df5_00000_0_2024-06-09_15-00-17
Logs: job-driver-raysubmit_zNdccSCBKG1XTkxL.log

And then SST last layer is trained by:
Experiment name: resume-from-para-after-cosine-sts-sst
Trainer: TorchTrainer_c5b48_00000_0_2024-06-09_15-20-54
Logs: job-driver-raysubmit_QcDj6U39sBPpkw7q.log

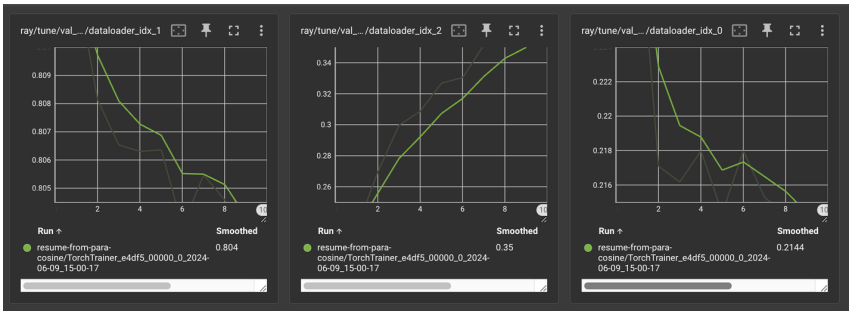

Figure 6: Accuracy metrics full model on STS (with cosine) showing a slight decrease in the other two tasks with a 10% increase in STS training.

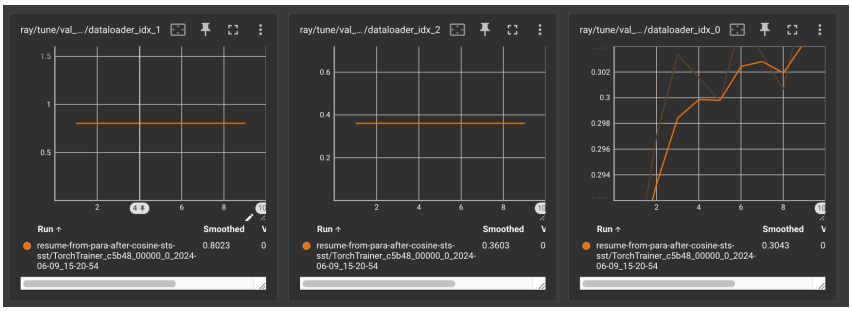

Figure 7: Accuracy metrics showing no change in the metrics for the first two tasks and only SST when applying tunning to only the last layer.

8.5 BERT last linear layer + LSTM for sentiment classification

Using a LSTM and finetuning only for the last linear layer, we observe better results on the sentiment analysis task, which when integrated with the sequence of training that we did for previous runs improve the accuracy of the overall multi-task classification (Figure 8). This was not incorporated into our leaderboard position due to time constraints.
Experiment Name: test
Trainer: TorchTrainer_bbe6f_00000_0_2024-06-09_20-14-06
Logs: job-driver-raysubmit_Awanpn5yj4KgUPQw.log

9 Conclusion

We have shown that a miniBert architecture can be pre-trained and then undergo fine-tuning and other methods to perform on downstream tasks, such as sentiment analysis, paraphrase detection and STS. We see a bump in performance from multi-task tunning training and cosine similarity fine-tuning. Creating a custom model using a LSTM and fine-tuning only for the last linear layer for sentiment analysis showed great potential, however, due to time constraints we were not able to submit to the leader board incorporating this information.

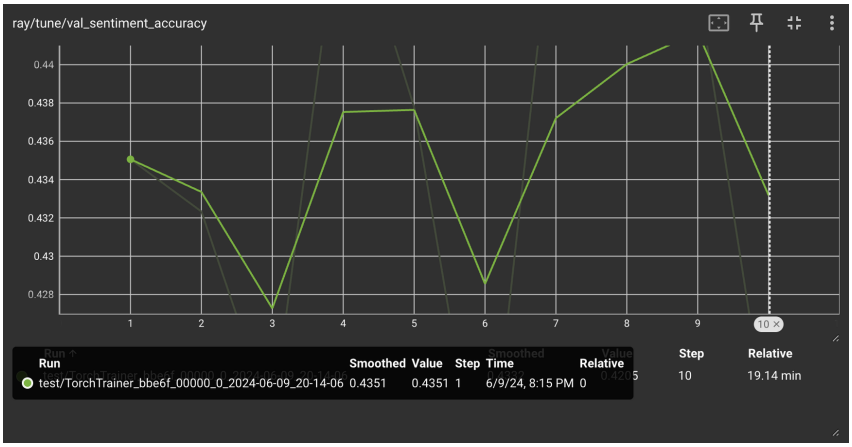

Figure 8: Bump in performance from using a LSTM and finetuning only for the last layer for sentiment analysis task.

Additional training and methods, such as fine-tuning with regularized optimization and contrastive learning, could be conducted to further improve the accuracy on the tasks. Further work could also be done to build on using BERT as a recruiter.

One approach to perform information retrieval on job postings could be to generate text using bert encodings and train on the loss of the generated encodings and the actual encodings for the given task. For example, starting with the prompt "The role in this job description is" prepended with the job description and letting the model complete the sentence. We can then train on the loss of response generated and the actual outputs from the labelled datasets (for example Kaggle). For suitability score, it would be a regression problem on the labelled dataset. Due to time constraints, we could not implement it on our end but we are looking to collaborate together on this project in near future.

10 Ethics Statement

One ethical consideration of building LLMs for tasks such as STS is that extensions of the work could be used to speculate about the identity of an anonymously published piece of work.This is a societal risk as it could be used to expose whistle blowers, political activists, or authors who would simply like to remain anonymous. In addition, an LLM would be outputting a prediction based on training data that may or may not be relevant to the user data. Innocent people could be accused of being the author of a piece of work. One mitigation strategy could be train on STS tasks that focus on meaning of text, not the style of the writing. In addition, training to identify the author of one piece of text based on a large corpus (such as determining an author based on a snippet of their writing out of several authors) should be avoided or handled with scrutiny.

Another ethical consideration for using BERT for job-seeking tasks is that it could be used to automate recruiting roles. This project does not intend to impact any existing roles or automate human intervention entirely. Through this extension of BERT, we are delving into the possibility of BERT being helpful reducing the effort necessary to hire quality candidates. Thinking about mitigation strategies for recruiting LLMs, one method could be to train models to see if candidates are qualified or not (binary) vs output rankings (scale 1-10). Although binary classification is on a spectrum, having the output be binary would prevent applicants from being ranked if a binary result is returned. This would allow recruiters to have an objective method to complement existing methods, such as keyword matching. The time saved could allow recruiters to spend more time getting to know applicants who have the qualifications for the role or researching select candidates who could be a better fit if they had certain qualifications.

11 Collaboration

Prabhjot led code development and Jared led report and poster development.

References

Elias Abdollahnejad, Marilynn Kalman, and Behrouz H. Far. 2021. A deep learning bert-based approach to person-job fit in talent recruitment. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 98–104.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. Sem 2013 shared task: Semantic textual similarity. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics (*SEM)*, volume 1, pages 32–43.

GitHub codebase. `https://github.com/psr-ai/CS224N-default-project`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

IMDB. `https://developer.imdb.com/non-commercial-datasets/`.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First quora dataset release: Question pairs. `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`.

Kaggle. Online job postings dataset. `https://www.kaggle.com/datasets/madhab/jobposts`.

Xiaodong Liu, Pengcheng Hea, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. In *Association for Computational Linguistics (ACL)*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP),*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew 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*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification. `https://arxiv.org/pdf/1905.05583`.

Multitask Trainer. `https://github.com/psr-ai/CS224N-default-project/blob/main/multitask_classifier_lightning.py`.

12 Appendix

*All the following experiment results are available here. On the tensorboard page, please search on the left hand side by the "Trainer Name" for each experiment. For example, paraphrase full model training experiment has Trainer

`''TorchTrainer_086a8_00000_0_2024-06-09_09-59-59''`

associated with it, hence searching by that name will show the metrics corresponding to that trainer. The corresponding experiment logs files are available here.