BERTPlus - Efficiently Optimize BERT Model Embeddings for Question Answering in Low Data Regime

Ed Brown edbrown@berkeley.edu

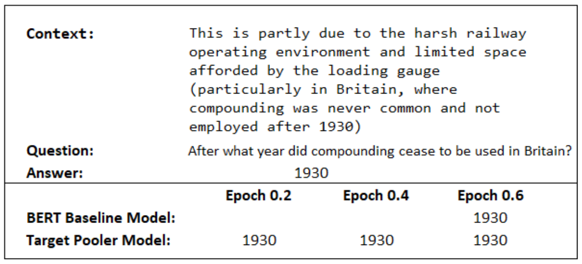
Ray Cao caopuzheng@berkeley.edu

Lisa Wu lisawu@ischool.berkeley.edu

Master of Information and Data Science (MIDS) Program
University of California, Berkeley

Abstract

Pre-trained transformer models have significantly advanced the model performance of many NLP tasks, but still require fine-tuning for specific tasks which could be cost prohibitive for users with limited high power GPU/TPU capability. We focused on the BERT (large, uncased) model and an extractive Question Answering (QA) task using SQuAD 2.0 dataset. We improved the baseline BERT performance with limited fine-tuning by optimizing full or target layers of model embeddings. We analyzed learned weights and observed non-uniform distribution across encoder layers, with most contributions by the last six layers. Our enhanced models outperformed BERT with a fraction of training data and reduced fine-tuning. See an illustrative example in the top right graph that our Target Pooler Model used 40% less training data and time than BERT. Our best model T-6 achieved BERT's best performance with 2 fewer fine-tuning epochs. We also applied efficient data processes to reduce data load speeds and minimize model development time. Our work introduced a method to tune BERT with limited computing resources and sparse training data while achieving generalizable results by extracting rich learnings from hidden states.



Context: This is partly due to the harsh railway operating environment and limited space afforded by the loading gauge (particularly in Britain, where compounding was never common and not employed after 1930)

Question: After what year did compounding cease to be used in Britain?

Answer: 1930

	Epoch 0.2	Epoch 0.4	Epoch 0.6
BERT Baseline Model:			1930
Target Pooler Model:	1930	1930	1930

1. Introduction

Transformer model (Vaswani et al., 2017) revolutionized the natural language processing (NLP) world and significantly advanced model performance for many NLP tasks. BERT (Devlin et al.,2019) lets users download pre-trained weights and then fine-tune the weights for specific tasks. However, the fine-tuning process for the full-size BERT requires powerful GPU/TPU capability and large training resources, which may only be practical for some users.

We are inspired by many recent works to enhance feature extraction from BERT for Question Answer (QA) that significantly reduces the requirement of BERT fine-tuning (Jiang et al., 2020; Zhu et al., 2020; and Chen et al.,2020). Jiang et al. developed an approach to use information from each BERT layer's hidden state activations, achieving better performance and saving one training epoch. Their work was built on top of the previous works (Tenney et al., 2019a and van Aken et al., 2019), which suggests that all BERT layers carry unique information, with lower layers encoding more basic syntax and higher layers capturing more complex semantics. In Jiang et al. paper, they applied the pooling method that contracts the last dimension from 25 to 1 through a learned linear combination of each encoder layer and uses the full embeddings as training data. They used the adapter method (Houlsby et al., 2019) to compress the 1024 dimensions of the BERT embeddings.

We explored three approaches to using the layer-level model embeddings: 1) average weights; 2) learned weights for all 25 layers (including the input embeddings); and 3) target learning weights for the Top 12 layers (Target Pooler Model T-12), and for the last six layers (T-6).

Our efficient data processing mechanism overcame the data process and loading challenges noted in Jiang et al. paper. We conducted experiments for 4 partial epochs and 4 full epochs for 8 experiments for the baseline BERT model and our four models (Average Pooler, Learned Pooler, T-12 and T-6).

Our models largely outperformed the baseline BERT model in 4 partial epochs, using a fraction of training data. Our best model Target Pooler T-6 Model, achieved BERT's maximum performance (at Epoch 4) at Epoch 2, which reduced 2 full fine-tuning epochs.

2. Datasets

We used the Stanford Question Answering Dataset (SQuAD) 2.0 as our benchmark dataset. This dataset combined existing SQuAD 1.0 data with over 50k unanswerable questions, with a total dataset of ~130k. SQuAD 2.0 is a challenging dataset because models must answer questions when possible but also determine when no answer is supported in the context and abstain from answering.

2.1. Data Loading Improvement

The SQuAD 2.0 dataset is large and time-consuming, as these questions are processed into numerical features. Since we are studying the hidden states of BERT as inputs into our model, the data size snowballed quickly to terabytes and became too big to fit into any computer RAM. In Jiang et al. paper, they wrote this data to disk and used a custom Keras data generator to retrieve it. I/O time was significant during their training, with ~ 95% of the time spent on loading data and only 5% of the time on model fitting.

In our work, we explored and identified an efficient way to load all the training data into RAM at the start of the process. Instead of generating all the embeddings from all intermediate hidden layers of the BERT model, we were able to leverage the native batching strategy in TensorFlow through model concatenation and layer transformations, and thus only needed to cache the original training data rather than all the intermediate embedding results. This decreased the required RAM space from ~5TB to ~1GB, thus allowing us to conduct all calculations inside RAM and greatly improving the training time per epoch. This strategy ultimately enabled us to train and compare multiple models at multiple stages promptly; it can be applied to other future work involving the hidden states of the BERT model.

2.1. Data Processing

We chose a maximum sequence length of 386 tokens with an overlap of 128 tokens for the model. When the combination of question and answer exceeded 386 tokens, the equations were split into multiple identical questions with the answer split with an overlap of 128 tokens until the complete answer was consumed. We evaluated various lengths and overlaps based on a maximum length of 512 tokens. 386 was chosen to cover nearly all cases without splitting and reducing the number of [PAD] tokens output, consistent with the approach noted in Jiang et al. 2020 paper.

We chose a batch size of 48 based on Devlin et al. 2019 paper. The BERT model was trained on three NVIDIA A100 GPUs. Using a MirroredStrategy, each GPU received a batch of 16 during training. Model weights were saved for later use at the end of each complete training epoch. Additionally, we trained partial datasets of 20, 40, 60, and 80 percent of the original dataset to evaluate the cases of lower data resources.

For partial dataset training, the dataset was broken down into a 50/50 split of possible/impossible questions to ensure balanced data. Partial epoch 20 means the model uses the first 20 percent of data (26,382 entries) in the dataset. This continued for the 40, 60 and 80 percent datasets. These partial datasets were again trained in batches of 48 on three GPUs.

Please see the overall workflows in *Appendix A.5. High Level System Architecture Diagram*.

3. Methods

This section introduces our baseline BERT model and four custom models based on using a full or target set of model embeddings of BERT layers.

3.1. Baseline Model

We used the pre-trained BERT (large, uncased) model, which was trained with a multi-task object (masked language modeling and next-sentence prediction) on a total dataset of 3.3 billion English words. This model has 24 encoders with 16 bidirectional self-attention heads totaling ~335 million parameters.

We trained the BERT model for partial epochs (using 20/40/60/80 percent of training data) and 1-4 epochs. This became our baseline model. Partial epochs fine-tuning helped us understand the BERT learning process in small increments of training data, which resemble a small training dataset (low resource dataset).

With input tokens $T = [t_0, t_1, \ldots t_n]$, the encoder generates 24 hidden layers, and each layer has the activation vector $H^l = [h^l_0, h^l_1, \ldots, h^l_n]$, *where* $n = 386$, H^0 *is the input layer vector.*

3.2. Average Pooler Model

We extracted the weights of all layers (1 input layer and 24 hidden states) of the baseline model noted above. We first explored the average pooling method and assigned uniform weights to linearly combine the encoder embeddings of all 25 layers into a single set of vectors $H = [h_0, h_1, \ldots, h_n]$. See Equation (1) below.

This approach leverages information from all layers, instead of the last layer only by BERT, for prediction and evaluation. We froze the encoder weights to avoid re-training BERT parameters in this process.

$$h_i = \sum_{l=0}^{L} \frac{1}{l} h^l_i, \text{ where } i \text{ is the layer} \qquad (1)$$

3.3. Learned Pooler Model (use the full layers)

Similarly, we extracted weights of all layers of the baseline model. Instead of assuming equal weight for each layer, we implemented the learned pooling method described in Peters et al. 2018a and Jiang et al. 2020 paper. See Equation (2) below.

$$h_{i,t} = \gamma_t \sum_{l=0}^{L} s^l_t h^l_i, \text{ where } \gamma_t \text{ is the scalar} \qquad (2)$$

and s_t is the softmax of the encoder weights.

This is extremely parameter-efficient as the model only has 26 additional trainable parameters.

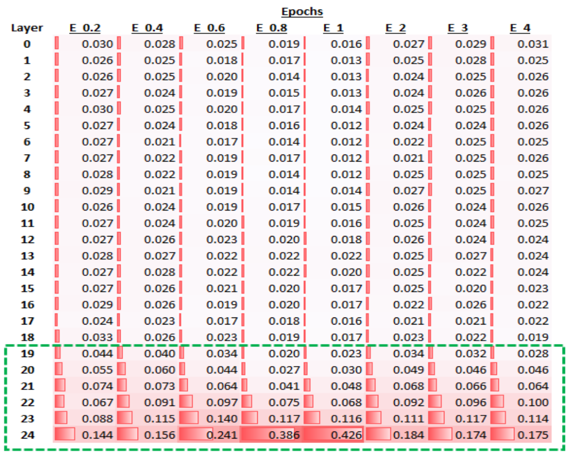
3.4. Target Pooler Models

Tenny et al., 2019 used the "edge probing" approach and observed a consistent trend that basic syntactic information appears in the earlier layers, while high-level semantic information appears in later layers. Given the complex nature of the SQuAD 2.0 dataset, we expected that the later layers play a more important role than the earlier layers.

For each partial/full epoch, we extracted the full learned weights of the layers. See Figure 1 below. The learned weight distribution varied by layer and epoch. Here we explored using a target set of layers to increase parameter efficiency. We explored two versions of Target Pooler models: 1) We selected the Top 12 Layers (Target Pooler T-12). See the sum of the 12 layers' weights as a % of the total learned weights in Figure 2; and 2) We selected the last 6 layers (Layer 19 - 24) as these layers consistently dominated the weights across epochs, with a total of 47% to 71% of the total learned weights (see Figure 2). See a high level illustration of the model design diagram in *Appendix A.4 Model Design Diagram*.

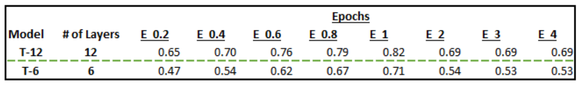
Since the Target Pooler models are more parameter-efficient than the full learned-pooler model as we trim down the layers, we expect performance gains as the model learns from the most relevant layers, thus the name "target". We evaluated T-6 vs T-12 model performance in Section 4 below.

Figure 1: Learned Weights by Layer by Epoch



	Epochs							
Layer	E 0.2	E 0.4	E 0.6	E 0.8	E 1	E 2	E 3	E 4
0	0.030	0.028	0.025	0.019	0.016	0.027	0.029	0.031
1	0.026	0.025	0.018	0.017	0.013	0.025	0.028	0.025
2	0.026	0.025	0.020	0.014	0.013	0.024	0.025	0.026
3	0.027	0.024	0.019	0.015	0.013	0.024	0.026	0.026
4	0.030	0.025	0.020	0.017	0.014	0.025	0.025	0.026
5	0.027	0.024	0.018	0.016	0.012	0.024	0.024	0.026
6	0.027	0.021	0.017	0.014	0.012	0.022	0.025	0.025
7	0.027	0.022	0.019	0.017	0.012	0.021	0.025	0.026
8	0.028	0.022	0.019	0.014	0.012	0.025	0.025	0.025
9	0.029	0.021	0.019	0.014	0.014	0.027	0.025	0.027
10	0.026	0.024	0.019	0.017	0.015	0.026	0.024	0.026
11	0.027	0.024	0.020	0.019	0.016	0.025	0.024	0.025
12	0.027	0.026	0.023	0.020	0.018	0.026	0.024	0.024
13	0.028	0.027	0.022	0.022	0.022	0.025	0.027	0.024
14	0.027	0.028	0.022	0.022	0.020	0.025	0.022	0.024
15	0.027	0.026	0.021	0.020	0.017	0.025	0.020	0.023
16	0.029	0.026	0.019	0.020	0.017	0.022	0.026	0.022
17	0.024	0.023	0.017	0.018	0.016	0.021	0.021	0.022
18	0.033	0.026	0.023	0.019	0.017	0.023	0.022	0.019
19	0.044	0.040	0.034	0.020	0.023	0.034	0.032	0.028
20	0.055	0.060	0.044	0.027	0.030	0.049	0.046	0.046
21	0.074	0.073	0.064	0.041	0.048	0.068	0.066	0.064
22	0.067	0.091	0.097	0.075	0.068	0.092	0.096	0.100
23	0.088	0.115	0.140	0.117	0.116	0.111	0.117	0.114
24	0.144	0.156	0.241	0.386	0.426	0.184	0.174	0.175

Figure 2: T-12/T-6 Select Layer Weights as a % of Total Learned Weights



		Epochs							
Model	# of Layers	E 0.2	E 0.4	E 0.6	E 0.8	E 1	E 2	E 3	E 4
T-12	12	0.65	0.70	0.76	0.79	0.82	0.69	0.69	0.69
T-6	6	0.47	0.54	0.62	0.67	0.71	0.54	0.53	0.53

See *Appendix A.4* for Model Design Graph.

4. Results

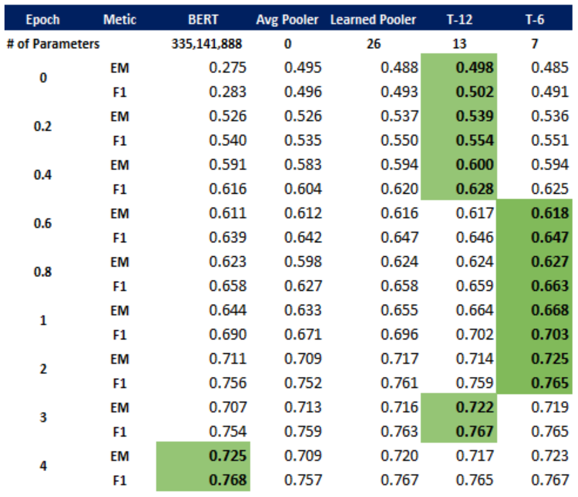
We scored our models using F1 and EM scores for the SQuad dev 2.0 dataset.

4.1. Our Models vs. Baseline BERT Model

The baseline BERT model used ~335 million in parameters and required a lot of resources to fine-tune. Our models froze BERT parameters (no retraining) to reduce resource requirements.

Figure 3 shows that our Learned Pooler and Target Pooler (T-12 and T-6) consistently outperformed the baseline BERT model through Epoch 3. Our best model (T-6) achieved BERT's maximum model performance (achieved at Epoch 4) at Epoch 2, saving 2 full epochs of learning resources.

Figure 3: Model Results (EM and F1) Comparison



Epoch	Metic	BERT	Avg Pooler	Learned Pooler	T-12	T-6
# of Parameters		335,141,888	0	26	13	7
0	EM	0.275	0.495	0.488	**0.498**	0.485
	F1	0.283	0.496	0.493	**0.502**	0.491
0.2	EM	0.526	0.526	0.537	**0.539**	0.536
	F1	0.540	0.535	0.550	**0.554**	0.551
0.4	EM	0.591	0.583	0.594	**0.600**	0.594
	F1	0.616	0.604	0.620	**0.628**	0.625
0.6	EM	0.611	0.612	0.616	0.617	**0.618**
	F1	0.639	0.642	0.647	0.646	**0.647**
0.8	EM	0.623	0.598	0.624	0.624	**0.627**
	F1	0.658	0.627	0.658	0.659	**0.663**
1	EM	0.644	0.633	0.655	0.664	**0.668**
	F1	0.690	0.671	0.696	0.702	**0.703**
2	EM	0.711	0.709	0.717	0.714	**0.725**
	F1	0.756	0.752	0.761	0.759	**0.765**
3	EM	0.707	0.713	0.716	**0.722**	0.719
	F1	0.754	0.759	0.763	**0.767**	0.765
4	EM	**0.725**	0.709	0.720	0.717	0.723
	F1	**0.768**	0.757	0.767	0.765	0.767

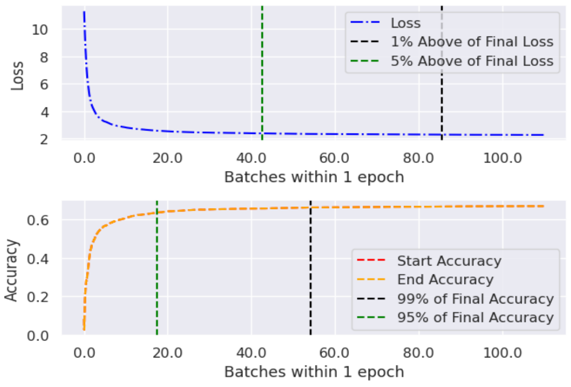
T-6 used 6 fewer layers, and less total learned weights than T-12. In the first two partial epochs (Epoch 0.2 and Epoch 0.4), T-6 performed comparably with T-12 and then outperformed T-12 in the subsequent epochs. Our observations suggest that T-6 is the most resource-efficient model.

4.2. Model Loss Curve Evaluation

In addition to FM and F1 metrics, we evaluated the loss curve of the Target Pooler model. We used the T-12 model for illustration purposes, but the observations were consistent among all the learned pooler models.

Figure 4 shows that the loss curve declined sharply in the first 20% of an epoch and gradually decreased afterward. This indicated that our model required low training data resources to achieve a solid result. This is the key advantage of our model over the baseline BERT. Given the parameter-efficient approach, T-12 and T-6 models only required 20-40% of the SQuAD 2.0 data for the training. This is a great attribute when the underlying dataset is lean and sparse.

Figure 4: Loss Curve and Accuracy Curve Evaluation



5. Model Analysis

We performed deep-dive analysis to gain insight of the model performance in several key areas.

5.1. Answerable vs. Unanswerable Questions

We evaluated model accuracies for answerable and unanswerable questions in partial epochs. This analysis helped us gain insight into the model performance at the more granular questions.

Figure 5 shows F1 scores for Answerable Questions using the dev 2.0 dataset. F1 scores improved as training data increased in partial Epoch 0.2 through Epoch 0.8. Our best model, T-6 outperformed the baseline BERT model for answerable questions for all partial epochs.

Figure 5: Answerable Questions F1 Scores

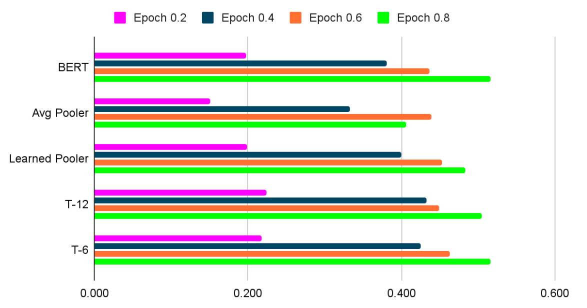
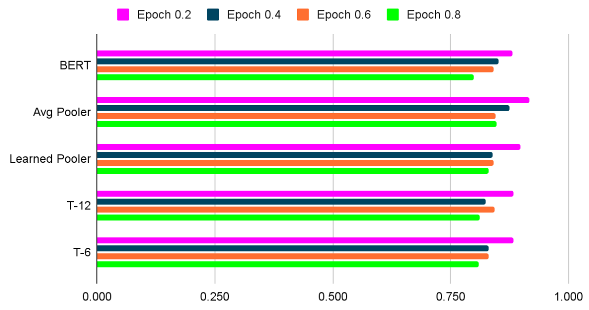


Figure 6 shows F1 scores for Unanswerable questions. F1 scores for unanswerable questions started much higher (>85%) than F1 scores for answerable questions. However, adding incremental training data did not increase the baseline BERT model accuracy. Our models outperformed BERT in Epoch 0.2 but also saw similar accuracy declines in the subsequent epochs. See more analysis in Appendix A.3.

SQuAD V2.0 is more challenging than V1.0, as the unanswerable questions require a more contextual understanding of the questions and the answers. For unanswerable questions, we can further explore ensemble and other modeling techniques in the future work.

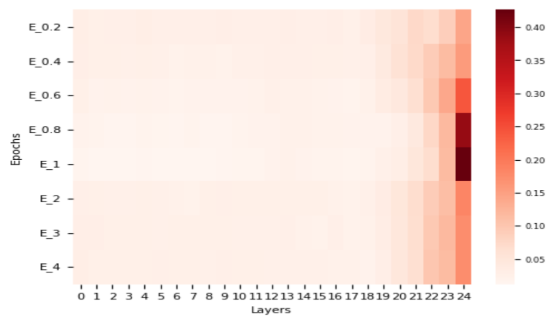
Figure 6: Unanswerable Questions F1 Scores



5.2. Weights of Each Layer By Epoch

We evaluated the weights of the 25 layers in each partial and full fine-tuning epoch. Figure 7 shows that learned weights shifted by epoch, especially among the later layers. The last six layers consistently contribute the highest weights among all layers. The learned weights for these six layers change noticeably as well, especially the last layer (Layer 24), with learned weights ranging from ~14% to ~43% as a percentage of the total weights of all layers.

Figure 7: Learned Weights by Layer by Epoch

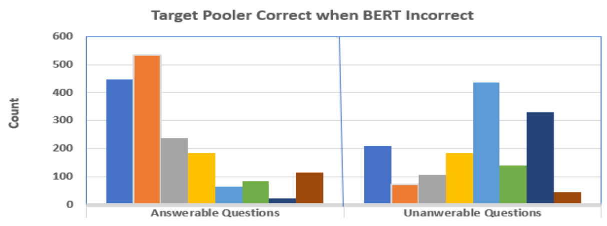


5.3. Error Analysis

We analyzed the questions that the baseline BERT model and our Learned Pooler and Target Pooler models predicted differently. We used the T-12 Model for illustrative purposes, but the observations were consistent among all the learned pooler models.

Figure 8 below shows the count of the baseline BERT model being incorrect, but the Target Pooler mode was correct. For Answerable questions, Target Pooler predicted 8-9% of questions correctly when BERT was incorrect in Epoch 0.2 and Epoch 0.4 (using 20% and 40% of training data). This lead decreased to 0-2% in Epoch 3 and 4 when full training data was used. For Unanswerable questions, Target Poolerl predicted 4% correctly when BERT was incorrect in Epoch 0.2, but this fluctuated in the sequent partial epochs.

Figure 8: Target Pooler Predicted Correctly When BERT Didn't

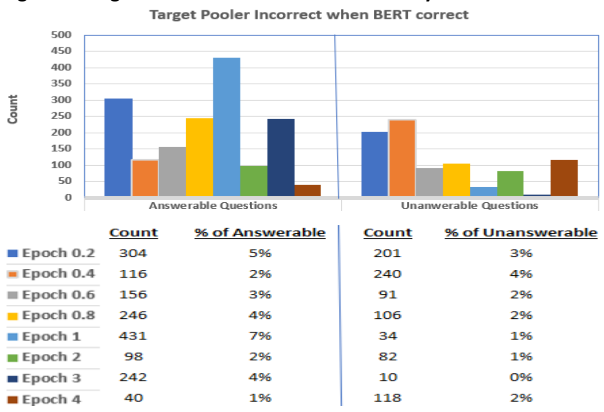


	Count	% of Answerable	Count	% of Unanswerable
Epoch 0.2	447	8%	210	4%
Epoch 0.4	535	9%	74	1%
Epoch 0.6	239	4%	108	2%
Epoch 0.8	184	3%	184	3%
Epoch 1	66	1%	435	7%
Epoch 2	84	1%	139	2%
Epoch 3	22	0%	330	6%
Epoch 4	116	2%	45	1%

Figure 9 below shows the count of the baseline BERT model correctly predicting the questions, but Target Pooler was incorrect. For Answerable questions, BERT predicted 5% of questions correctly when Target Pooler was incorrect in Epoch 0.2, which oscillated in the sequent partial epochs. For Unanswerable questions, BERT predicted 0-4% correctly when Target Pooler was incorrect in the partial epochs.

Netting the numbers in Figure 8 and Figure 9, Target Pooler outperformed BERT for answerable questions through Epoch 2. However, model performance for BERT and Target Pooler did not differentiate as much for unanswerable questions.

Figure 9: Target Pooler Didn't Predicted Correctly When BERT Did



	Count	% of Answerable	Count	% of Unanswerable
Epoch 0.2	304	5%	201	3%
Epoch 0.4	116	2%	240	4%
Epoch 0.6	156	3%	91	2%
Epoch 0.8	246	4%	106	2%
Epoch 1	431	7%	34	1%
Epoch 2	98	2%	82	1%
Epoch 3	242	4%	10	0%
Epoch 4	40	1%	118	2%

6. Generalization to Classification Task

Jiang et al. 2020 applied the learned pooler framework to the classification task to distinguish the impossible questions from the possible ones within SQuAD 2.0 dataset. It achieved good results compared with traditional classification methods.

We further explored this approach and applied the target pooler method to the IMDB dataset. IMDB dataset is a classification task in which the model needs to classify whether a movie review is positive or negative. Our target pooler model T-6 achieved a 94.36% accuracy, not significantly better than the predictions from the CLS tokens of BERT models at 94.06% after 1 training epoch.

In the case of the IMDB dataset, using learned weights (added parameters) of the hidden states didn't improve the prediction accuracy much, but it helped to prevent the model from over-fitting too quickly. We believe that the benefits of adopting our approach heavily rely on the nature of the task and whether more in-depth semantic learning is required to accomplish the given task.

7. Discussions for future work

When comparing the results of our best model (Target Pooler Model), we noticed that its performance improves for answerable questions as we add more training data at each partial epoch. However, for the unanswerable questions, the prediction accuracy started much higher (>85% for the baseline BERT and our models) but did not improve further with more training data. More analysis and modeling techniques can be leveraged to improve the model performance for the unanswerable questions in future works.

8. Conclusion

In our paper, we explored three approaches to extract and leverage rich information in the hidden states of BERT encoder layers, aiming to improve model performance in lower data and computing resource environments. We propose a data and parameter-efficient approach that utilizes a target set of the most informative layers (the last six encoder layers). Our proposed model outperformed BERT in the earlier training settings and achieved BERT's global best performance with 2 fewer fine-tuning epochs. For users who have significant computing constraints to conduct any full epoch fine-tuning work, our proposed approach will enable training with a fraction of data to efficiently explore and optimize model performance techniques efficiently before conducting any full epoch fine-tuning work, which could lead to better hyperparameter selection or tuning. Our efficient data processing mechanism also helped users reduce data loading and minimize modeling training time for models that involve manipulation of BERT hidden states.

Acknowledgments

We thank Peter Grabowski, our instructor of the Natural Language Processing with Deep Learning (NLP) course at UC Berkeley, for sharing his NLP model knowledge as a seasoned practitioner. We deeply appreciate Peter's review of our project proposal and insightful feedback. Peter also provided useful suggestions on conducting error analysis (e.g., answerable vs. unanswerable questions) and evaluating our models on alternative datasets and tasks such as sentiment analysis.

The authors acknowledge the Texas Advanced Computing Center (TACC) at the University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu

References

(1) Siduo Jiang, Christopher Benge and William Casey King 2020. BERTVision - A Parameter-Efficient Approach for Question Answering

(2) Leo Horne, Matthias Matti, Pouya Pourjafar, Zuowen Wang 2020 GRUBERT: A GRU-Based Method to Fuse BERT Hidden Layers for Twitter Sentiment Analysis

(3) Ian Tenney, Dipanjan Das, and Ellie Pavlick 2019a. BERT rediscovers the classical NLP pipeline. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (2019) 4593-4601

(4) Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, Omer Levy 2019 SpanBERT: Improving Pre-training by Representing and Predicting Spans

(5) Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer 2019 BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

(6) Wietse de Vries, Andreas van Cranenburgh, and Malvina Nissim 2020 What's so special about BERT's layers? A closer look at the NLP pipeline in monolingual and multilingual models

(7) Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, Sylvain Gelly 2019 Parameter-Efficient Transfer Learning for NLP

(8) Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, Ludwig Schmidt, 2022, Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time

(9) Pranav Rajpurkar, Robin Jia, Percy Liang, 2018, Know What You Don't Know: Unanswerable Questions for SQuAD

(10) Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, 2019 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

(11) Elior Sulem, Jamaal Hay and Dan Roth, 2021 Do We Know What We Don't Know? Studying Unanswerable Questions beyond SQuAD 2.0

(12) Yuwen Zhang, Zhaozhuo Xu, 2019, BERT for Question Answering on SQuAD 2.0

A. Appendices

A.1. Model Results Details for Span Annotation

Our best model Target Pooler Model (T-6), with the lowest parameter count, consistently outperformed the baseline model (BERT) (Figure 10 and Figure 11).

Figure 10: F1 Scores for Partial Epochs

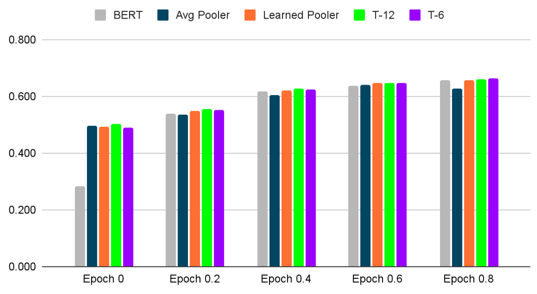
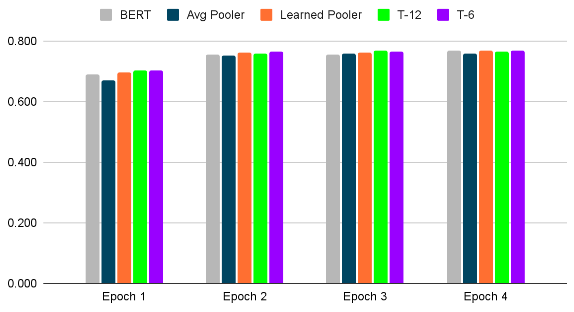


Figure 11: F1 Scores for Epoch 1-4



A.2 Data Processing Efficiencies

All fine-tuning of the baseline Bert(Large, Uncased) model and subsequent additional layers were performed on a single GPU node in the Lonestar 6 cluster at the Texas Advanced Computing Center.

GPU Node	
Processors	2x AMD EPYC 7763
Cores/Node	128 (64 per socket)
Clock Rate	2.45Ghz (Base Frequency)
Memory/Node	256 GB DDR-4
Peak Node Performance	4.8TF, Double Precision (CPU)
GPU	3x NVIDIA A100 GPUs with 40GB HBM2

We utilized TensorFlow MirroredStrategy to take advantage of the multiple GPUs on the node. Additionally, the Ampere architecture allowed us to run using Tensorfloat-32 for additional performance gains.
https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/

The computationally expensive task of generating BERT features was performed once from the source json file and stored as joblib pickle files for loading at training time. The capabilities of the compute node allowed us to load the model and all data into RAM, eliminating caching strategies.

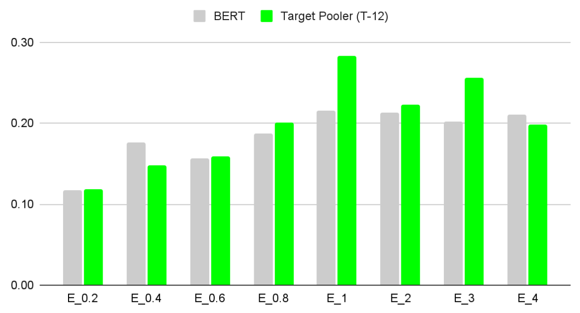
We relied on the local docker and HPC to ensure consistent behavior between development and production environments. Local workstations utilized docker 4.21.1, and HPC docker capabilities were provided by Apptainer v1.1.8-2.el8. All workflows were run on the following image. Python version was 3.11.4 and the Tensorflow version was 2.12.1 (see link). See the high level system architecture diagram in *Appendix A.5. High Level System Architecture Diagram*.

A.3 Challenges for Unanswerable Questions

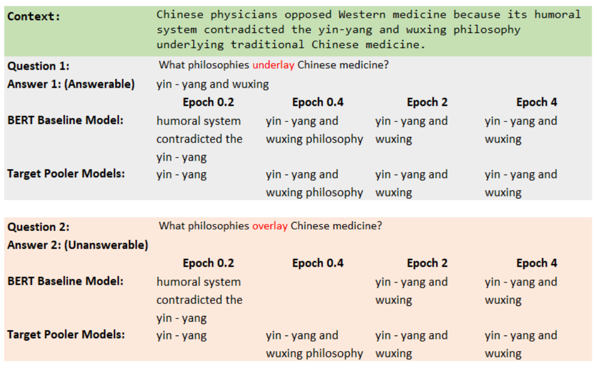
As noted in Section 5.1. above, for the unanswerable questions, model predictions were solid at the first learning epoch (F1 >85%). As the learning epoch increased, models learned to correct some of the incorrect predictions of the earlier rounds. However, the overall model accuracy did not improve as the learning epoch increased.

Figure 12 shows the inaccuracy increased over the epochs for BERT and Target Pooler T-12 Model.

Figure 12: Inaccurate Predictions at Each Epoch

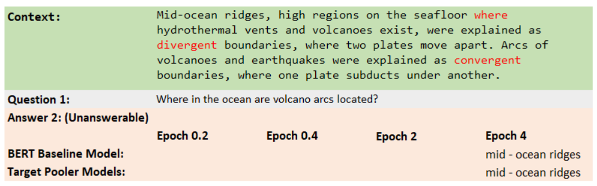


The first example below shows the answerable question (Question 1) and the unanswerable question (Question 2) of the same context. The only difference between the two questions is the word “underlay” in Question 1 and “overlay” in Question 2. The baseline BERT and Target Pooler model did not get the contextual difference between “overlay” and “underlay” and predicted nearly the same answers.



Context:	Chinese physicians opposed Western medicine because its humoral system contradicted the yin-yang and wuxing philosophy underlying traditional Chinese medicine.			
Question 1:	What philosophies underlay Chinese medicine?			
Answer 1: (Answerable)	yin - yang and wuxing			
	Epoch 0.2	**Epoch 0.4**	**Epoch 2**	**Epoch 4**
BERT Baseline Model:	humoral system contradicted the yin - yang	yin - yang and wuxing philosophy	yin - yang and wuxing	yin - yang and wuxing
Target Pooler Models:	yin - yang	yin - yang and wuxing philosophy	yin - yang and wuxing	yin - yang and wuxing
Question 2:	What philosophies overlay Chinese medicine?			
Answer 2: (Unanswerable)				
	Epoch 0.2	**Epoch 0.4**	**Epoch 2**	**Epoch 4**
BERT Baseline Model:	humoral system contradicted the yin - yang		yin - yang and wuxing	yin - yang and wuxing
Target Pooler Models:	yin - yang	yin - yang and wuxing philosophy	yin - yang and wuxing	yin - yang and wuxing

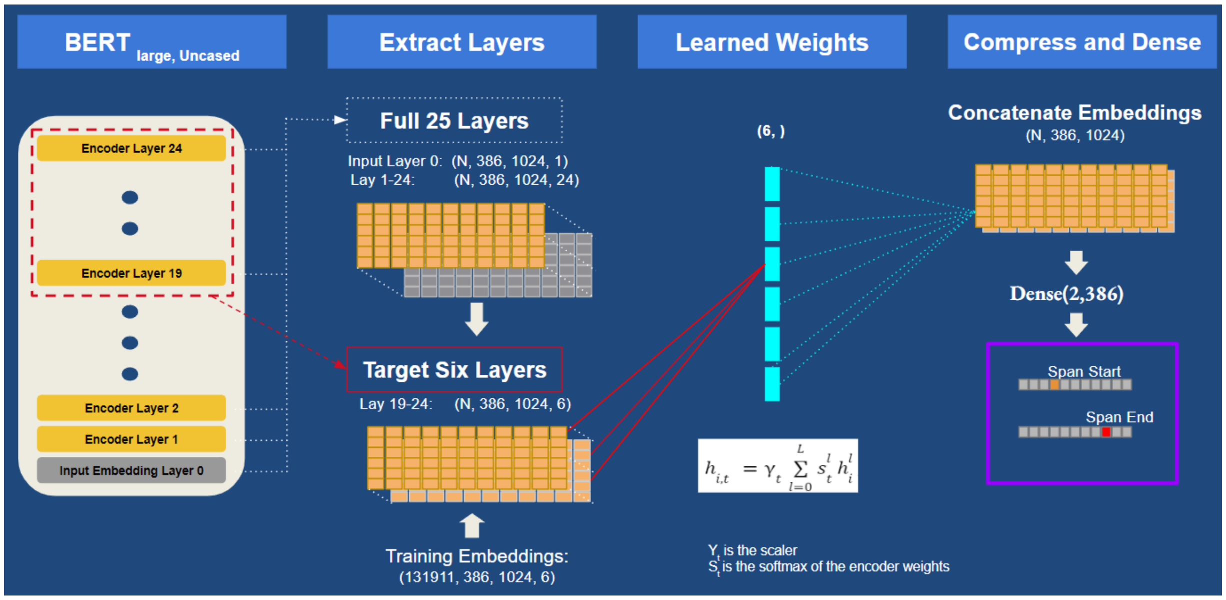
In the second example below, the question asked about the location of volcano arcs, which did not have an answer in the context. The first sentence has a location (mid-ocean ridges) but was related to “divergent” boundaries, not “convergent” boundaries. BERT and our model abstained from answering the question in the earlier epoch correctly, but predicted an answer (mid-ocean ridges) incorrectly in Epoch 4. This question requires human-like understanding of contextual meanings and is challenging for models.



Context:	Mid-ocean ridges, high regions on the seafloor where hydrothermal vents and volcanoes exist, were explained as divergent boundaries, where two plates move apart. Arcs of volcanoes and earthquakes were explained as convergent boundaries, where one plate subducts under another.			
Question 1:	Where in the ocean are volcano arcs located?			
Answer 2: (Unanswerable)				
	Epoch 0.2	**Epoch 0.4**	**Epoch 2**	**Epoch 4**
BERT Baseline Model:				mid - ocean ridges
Target Pooler Models:				mid - ocean ridges

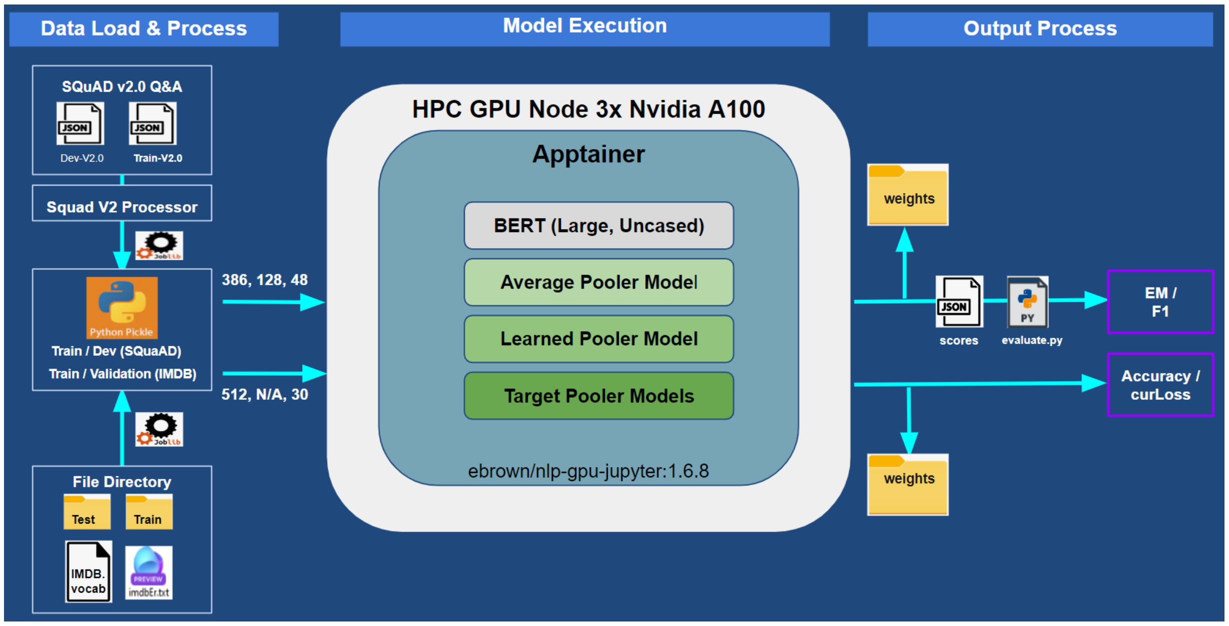
A.4 Model Design Diagram

This diagram summarizes the high level model architecture for our Target Pooler Models, using T-6 as an illustrative example. The Learned Weights section describes how we implement the learning process from each different hidden states.



A.5 High Level System Architecture Diagram

There are three key components in our system architecture framework: **1) Data Load & Process**: In the diagram, we illustrate the high level flows for two datasets, with SQuaAD v2.0 in the top section and IMDB in the bottom section; **2) Model Execution**: This is the core environment where we execute all the models. See the capability in Appendix A.2 above; and **3) Output Process:** Extractive Q&A task using SQuAD v2.0 follows the workflows in the top section. Classification task using IMDB follows the workflows in the lower section.


Data Load & Process
Model Execution
Output Process
SQuAD v2.0 Q&A
Dev-V2.0
Train-V2.0
Squad V2 Processor
Python Pickle
Train / Dev (SQuaAD)
Train / Validation (IMDB)
File Directory
Test
Train
IMDB. vocab
imdbEr.txt
386, 128, 48
512, N/A, 30
HPC GPU Node 3x Nvidia A100
Apptainer
BERT (Large, Uncased)
Average Pooler Model
Learned Pooler Model
Target Pooler Models
ebrown/nlp-gpu-jupyter:1.6.8
weights
scores
evaluate.py
EM / F1
Accuracy / curLoss
weights