BERTVision - A Parameter-Efficient Approach for Question Answering

Siduo Jiang
UC Berkeley
siduojiang@berkeley.edu

Cristopher Benge
UC Berkeley
cris.benge@berkeley.edu

William Casey King
UC Berkeley
caseyking@berkeley.edu

Abstract

We present a highly parameter-efficient approach for Question Answering (QA) that significantly reduces the need for extended BERT fine-tuning. Our method uses information from the hidden state activations of each BERT transformer layer, which is discarded during typical BERT inference. Our best model achieves maximal BERT performance at a fraction of the training time and GPU/TPU expense. Performance is further improved by ensembling our model with BERT's predictions. Furthermore, we find that near optimal performance can be achieved for QA span annotation using less training data. Our experiments show that this approach works well not only for span annotation, but also for classification, suggesting that it may be extensible to a wider range of tasks[1].

1 Introduction

The introduction of Transformers (Vaswani et al., 2017) has significantly advanced the state-of-the-art for many NLP tasks. The most well-known Transformer-based model is BERT (Devlin et al., 2019). The standard way to use BERT on a specific task is to first download pre-trained weights for the model, then fine-tune these weights on a supervised dataset. However, this procedure can be quite slow, and at times prohibitive for those without a powerful GPU/TPU, or those with limited CPU capacity. Smaller Transformers, such as DistilBERT (Sanh et al., 2019), can fine-tune up to 60% faster. However, such models tend to consistently underperform full-size BERT on a wide range of tasks. A method that reduces fine-tuning but maintains the same or better performance would make BERT more accessible for practical applications.

To develop our method, we draw inspiration from previous works that use BERT for feature extraction rather than for fine-tuning (Zhu et al., 2020; Chen et al., 2020). For example, Zhu et al. showed that the sequence outputs from the final BERT layer can be used as contextualized embeddings to supplement the self-attention mechanism in an encoder/decoder neural translation model. This led to an improvement over the standard Transformer model in all tested language pairs on standard metrics (BLEU score (Papineni et al., 2002)).

One characteristic these studies share with typical BERT inference is that only information from the final layer of BERT is used. However, a study by (Tenney et al., 2019a) suggests that all layers of BERT carry unique information. By training a series of classifiers within the edge probing framework (Tenney et al., 2019b), the authors computed how much each layer changes performance on eight labeling tasks, and the expected layer at which the model predicts the correct labels. The discovery is that syntactic information is encoded earlier in the network, while higher level semantic information comes later. Furthermore, classifier performance generally increases for all tasks when more layers are accounted for, starting from layer 0, suggesting that useful information is being incorporated at each progressive layer. Others, such as (van Aken et al., 2019), looked specifically at QA with SQuAD and published similar findings. Their work suggests that different layers encode different information important for QA.

(Ma et al., 2019) showed that a simple averaging of only the first and last layers of BERT results in contextualized embeddings that performed better than only using the final layer. The authors evaluated this approach on a variety of tasks such as text classification and question-answering. Together, these works suggest that the hidden state activations within BERT may contain unique information that can be used to augment the final layer. That said, the exact way of doing this requires further exploration.

In this work, we leverage the findings by Tenney

[1] See GitHub repository: BERTVision

and Ma as inspiration for developing a solution with a two-fold goal: 1. Reduce expensive BERT fine-tuning, and 2. Maintain or exceed BERT-level performance in the process. To do so, we extract the information-rich hidden states from each encoder layer and use the full embeddings as training data. We demonstrate that, for two question-answering tasks, even our simple architectures can match BERT performance at a fraction of the fine-tuning cost. Our best model saves on one full epoch of fine-tuning, and performs better than BERT, suggesting that our approach may be a desirable substitute to fine-tuning until convergence.

2 Methods

This section introduces our baseline BERT model, and custom models trained on BERT embeddings. We also describe how we use the SQuAD 2.0 question-answering dataset for both span annotation and classification.

2.1 Modeling approach

Our custom models use the BERT activations from each encoder layer as input data. For a single SQuAD 2.0 example, our data point has a shape of (X,1024, 25), where $X = 386$ for span annotation, and 1 for classification (see appendix [A] for details). We term this representation of the data as *"embeddings."*

2.2 Learned and average pooling

We implemented the pooling method described in (Tenney et al., 2019a), which contracts the last dimension from 25 to 1 through a learned linear combination of each layer. We call this approach learned pooling (LP). We also evaluate the pooling approach reported in (Ma et al., 2019), average pooling (AP), where each encoder layer is given equal weights.

2.3 Adapter compression

We use the term "compression" to refer to methods that reduce the 1024 dimension of the BERT embeddings. The method proposed in (Houlsby et al., 2019) is termed *"adapter"*, which is an auto-encoder type architecture wherein the embeddings are first projected into a lower dimension using a fully-connected layer. For our purposes, the adapter serves as a bridge between BERT embeddings and downstream layers. The architecture in Houlsby et. al. can only handle 3D tensors (including batch), because each adapter handles data from a single transformer layer. Since we work with the activations from multiple layers at once, we need to be able to handle 4D tensors. To do this, our adapter implementation can treat each transformer layer as independent, learning separate compression weights for each layer. Alternatively, we can also employ weight-sharing, so that the compression for each encoder layer follows the same set of transformations.

2.4 Custom CNN's

We also implemented various novel CNN architectures, as well as modified existing models such as Inception and Xception (Szegedy et al., 2014; Chollet, 2016). For span annotation, the key is that the CNN models must preserve the text length dimension of 386 (see appendix [A] for rationale). To view a notebook of all models we tried and their summaries, see: BERT Vision GitHub repo.

2.5 Fine-tuning BERT

For all experiments, we use the $BERT_{LARGE}$ uncased implementation from Hugging Face. To establish a baseline for our QA tasks, we fine-tuned BERT for 6 epochs with a setup similar to that described in (Devlin et al., 2019). For QA span annotation, questions that do not have answers have the start and end span positions assigned at the CLS token (position 0). We used the Adam optimizer with an initial learning rate of 1e-5. Due to hardware constraints, we use batch size of 8 rather than 48. At inference time, the most likely complete span is predicted based on the maximum softmax probability of the start- and end-span position. The setup is identical in classification, except that we used the pooled CLS token rather than the sequence outputs.

2.6 Ensembling

We evaluated multiple ensemble approaches for span annotation. The most successful method takes the element-wise max of the softmax probabilities output by each model in the ensemble. Let Z and Y be two model softmax probabilities for start or end span position. Then, the predicted position is:

$$\arg\max\left(\max_{Z_i,Y_i}, \quad \text{for i in } \{1,2,...,386\}\right) \tag{1}$$

2.7 Data processing and evaluation

We use SQuAD 2.0 for our QA dataset. Standard Exact Match (EM) and F1-score (F1) are used for

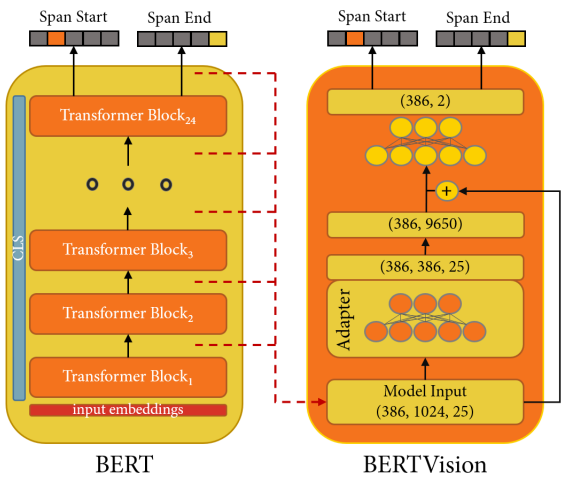


Figure 1: Architecture of our best model in relation to BERT. **Left**: BERT and its span annotation inference process. **Right**: For our model, BERT embeddings are first transformed through our custom adapter layer. Next, the last two dimensions are flattened. Optionally, a skip connection is added between the sequence outputs of the final BERT layer and this flattened representation. This is present in the best model discovered at 3/10 of an epoch, but was not necessary for the best model discovered at 1 full epoch. This tensor is then projected down to (386, 2) with a densely connected layer and split on the last axis into two model heads. These represent the logits of the start-span and end-span position.

evaluation as outlined in (Rajpurkar et al., 2018). For classification, EM is equivalent to accuracy as we are predicting a single binary outcome. For our BERT models, we restrict the maximum token length to 386 (See appendix [A] for rationale). Question-context pairs that exceed this maximum sequence are split into multiple segments (as many as needed) with an overlap between each segment of 128 tokens. For the splits that do not contain the answer, the labels for that split are set to "no answer". At inference time, we take the argmax of the span probabilities predicted by each split as the final prediction for the example.

3 Results

3.1 Span Annotation

For span annotation, we sought to generate useful embeddings with as little fine-tuning as possible. However, we were unable to find models that can fit BERT embeddings without some fine-tuning (see appendix [A] for details). As a result, we decided that a small amount of fine-tuning would be necessary.

3.1.1 BERT fine-tuning as baseline

To establish a baseline, we fine-tuned BERT for up to 6 epochs with results shown in Figure [6]. We measure performance for every 10th of a fractional epoch between 0 and 1 epochs, as well as full epochs up to 6. We observed that performance peaked at 2 epochs, achieving an Exact Match (EM) of 0.747, and an F1 score of 0.792. Between 0 and 1 epochs, performance consistently increased as measured by both EM and F1. (For comparison with other published works, see appendix [A]) For span detection we extracted embeddings at 3/10 of an epoch and at one full epoch. (for a rationale for this decision see appendix [A])

3.1.2 Models trained at 3/10 epoch embeddings

Using the embeddings at 3/10 of an epoch, we explored over 20 parameter-efficient models using a combination of CNNs, pooling, and compression techniques[2]. Table [1] shows that our best models outperform baseline BERT fine-tuned to 3/10 of an epoch. Results of all models are in appendix [A].

First, we compare two models that use different pooling strategies, LP (learned pooling) and AP (average pooling) [2]. We found that the LP model achieved similar performance as BERT itself, while AP significantly reduced performance. An analysis of the learned weights suggests that a non-uniform distribution of pooling weights is optimal, with the distribution slightly favoring later layers of BERT compared to earlier layers (see appendix [A]).

Next, we evaluated models leveraging adapters (see Methods). We found that our modified adapters of all flavors improved model performance compared to baseline BERT at 3/10 of an epoch, with our best model improving EM by 4.5 percentage points, and F1 by 3.8. As indicated in Table [1], without weight-sharing, the number of parameters increases by ∼24x, with a penalty to model performance, making weight-sharing superior in every respect. In addition, (Houlsby et al., 2019) found that an adapter size of 64 provides the best F1-score when used between transformer blocks. Our modified implementation did not perform well at this size with or without weight-sharing.

While we extensively explored stacking pooling with adapters and CNNs, we did not find a model

[2]See appendix [A] for training time and strategy

which performed better than our best adapter model. Table [1] shows the results, and the appendix [A] contains the rest. For one model where we stacked a modified Xception network, the number of parameters increased by 16x, but performance was no better than pooling alone.

Model	% Params BERT$_{large}$	SQuAD2.0 EM	SQuAD2.0 F1
BERT $\frac{3}{10}e$	100%	0.654	0.702
learned pooling (LP)	0.001%	0.660	0.703
average pooling (AP)	0.001%	0.657	0.700
adapter size 386	**0.124%**	**0.699**	**0.740**
- weights not shared	2.957%	0.676	0.723
adapter size 64	0.021%	0.680	0.732
- weights not shared	0.491%	0.684	0.716
LP adapter size 386 & xception	1.970%	0.668	0.711

Table 1: Models trained on embeddings at $\frac{3}{10}$ epochs

3.1.3 Best models on top at 1 epoch

Using 1 epoch embeddings as our training data, we fit the same set of models as with the 3/10 epoch embeddings (see the appendix [A] for full results). In most cases, we found that our models outperformed BERT at the same level of fine-tuning. The best model is nearly identical to that found with 3/10 epoch embeddings (see table [2]), outperforming BERT by 2.1 percentage points in EM, and 1.3 in F1. This model's performance is also competitive with maximum performance achieved with BERT on this task, which is BERT fine-tuned to 2 epochs. This result shows that we are able to reduce BERT fine-tuning by 1 epoch and still achieve comparable performance with a model of only about 0.124% of the number of BERT parameters. The implications of our findings suggest that near-optimal BERT performance can be achieved in a fraction of the training time and GPU/TPU expense.

Model	% Params BERT$_{large}$	SQuAD2.0 EM	SQuAD2.0 F1
BERT $1e$	100%	0.728	0.777
BERT $2e$ (*best*)	100%	0.747	0.792
adapter size 386	**0.124%**	**0.749**	**0.790**
- weights not shared	2.957%	0.716	0.767
adapter size 64	0.021%	0.739	0.785
- weights not shared	0.491%	0.736	0.784
LP adapter size 386 & xception	1.970%	0.741	0.786

Table 2: Models trained on embeddings at 1 epoch

3.1.4 Training with less data

The previous sections show that our models perform well compared to BERT when trained on the full dataset. However, since supervised data can be difficult to obtain, we explore whether the same performance can be achieved with significantly less data. To answer this question, we trained our best models on varying amounts of data, ranging from 0% to 100%. As with the full dataset, all models were trained for 1 epoch. Figure [2] shows the results. In both cases, we rapidly approach strong performance early, beating BERT at ∼30% of the data. Even by 10%, we approach peak performance. This shows that we can achieve strong performance even when using less training data.

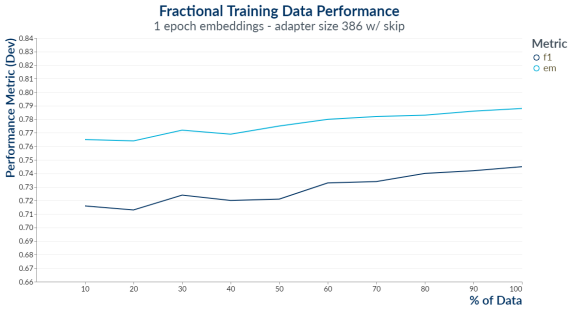


Figure 2: Training with less data

3.1.5 Ensembling our models with BERT

Since our approach requires fine-tuning BERT in order to derive workable embeddings, the BERT predictions come for free. Thus, we can ensemble our models with BERT predictions with no additional training required. Table [3] presents the results of this exercise. Ensembling the models at 3/10 of an epoch did not lead to an improvement, but ensembling our 1 epoch model with BERT at 1 epoch led to more than a half percentage point improvement. This suggests that our model and BERT supply complementary information that together leads to better predictions.

Model	SQuAD2.0 EM	SQuAD2.0 F1
BERT $\frac{3}{10}e$	0.654	0.702
our model $\frac{3}{10}e$	0.699	0.740
BERT $1e$	0.728	0.777
our model $1e$	0.749	0.790
ensemble BERT+our model $\frac{3}{10}e$	0.691	0.734
ensemble BERT+our model $1e$	**0.756**	**0.798**

Table 3: QA ensembling results

3.2 Classification

The previous section on QA span annotation uses the full sequence embeddings generated by BERT. Another common way to apply BERT is to text classification, which only uses the CLS token. Our goal in this section is to explore the efficacy of leveraging the CLS token hidden state activations in an analogous manner to our approach with the span annotation task.

3.2.1 BERT fine-tuning as baseline

Similar to QA span annotation, we use BERT fine-tuning as a baseline. We fine-tuned BERT for 6 epochs using the CLS token, with the EM and F1 for each epoch shown in Figure [5]. Our best performance was achieved at 3 epochs. We utilized embeddings at 2/10 of an epoch and 1 full epoch for the binary classification task (see appendix [A] for rationale).

3.2.2 Models trained at 2/10 epoch embeddings

At 2/10ths of an epoch, BERT achieved an F1 of 0.635 and EM of 0.687 [5]. We tried training various parameter-efficient model architectures similar to span annotation, most of which were CNN-based or simple linear. Our best performing model leverages a simple linear weighting that contracts across the channels dimension through learned weights inspired by (Tenney et al., 2019a). We trained for 10 epochs and recorded performance of the model at each epoch evaluated against the dev set. At every epoch, the model outperformed the BERT baseline at 2/10 of an epoch [3].

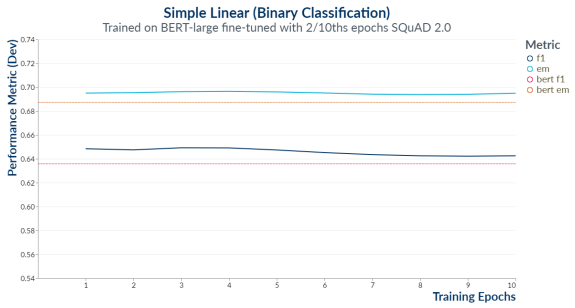


Figure 3: 2/10ths epoch model vs BERT

3.2.3 Models trained at 1 epoch embeddings

With 1 epoch embeddings, our results show significant improvement over the 2/10ths epoch models. Surprisingly, the best results were achieved at 1 epoch of training with the simple linear model consisting of only 0.001% of BERT's parameters. This model outperforms BERT at both 1 and 2 epochs of fine-tuning. This implies that our parameter efficient model can save on 1 full epoch of BERT fine-tuning. However, performance does fall short of maximal BERT performance achieved at 3 epochs, and we were unable to find a model architecture that achieves this [4]. Nevertheless, this suggests that with text classification, we can use the CLS token hidden state activations in the same way we used the full sequence hidden state activations in the span annotation task.

e	BERT F1	BERT EM	Our F1	Our EM	F1 Δ	EM Δ
1	0.720	0.761	0.763	0.782	+0.043	+0.021
3	0.790	0.804	0.795	0.811	+0.005	+0.007

Table 4: Comparison of BERT and our models performance at 1 and 3 epochs on binary classification task

3.2.4 Modeling with 3 full epoch embeddings

BERT achieved its best performance against the SQuAD 2.0 binary classification task at 3 full epochs (0.79 F1, 0.804 EM) [5]. Given that the simpler linear model outperformed BERT at the same level of fine-tuning, we wanted to evaluate if the simple model's trend in performance would continue as BERT exhausts its ability to learn the binary classification task. In this final evaluation, the linear model again outperformed BERT, achieving the top overall score for our task, albeit at a smaller margin [4].

4 Model analysis

4.1 Question types

Model performance on SQuAD 2.0 can be categorized based on whether the question-context pair has an answer (Has Answer versus No Answer). The dev set is very balanced in this sense as it contains 5,928 (∼49.93%) pairs with answers and 5,945 (∼50.07%) pairs without answers. Figure [4] shows the fraction of questions of each type that BERT and our best-performing models correctly answers. As BERT is fine-tuned, the number of correctly answered questions in both categories increases, but questions with "No Answer" increases more rapidly. Our best models answer more "No Answer" questions correctly than BERT, but underperform BERT in terms of "Has Answer" questions.

To investigate further, we looked at our 3/10 epoch model's incorrect predictions on "Has An-

swer" questions, and found that a large percentage, ∼65%, were predicted as having no answer (rather than with the wrong answer). These results suggest that our models may be more liberal than BERT at predicting "no answer".

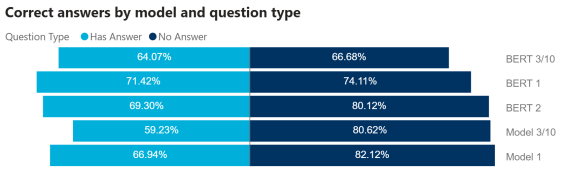


Figure 4: Correctly answered percentages by model

4.2 Second most likely answer

For questions where our 3/10 epoch model's most likely answer was incorrect, we considered the second most likely answer predicted by the model. We found that the second most likely answer is correct 44% of the time when the most likely answer is incorrect. We also note 47% of these were also "Has Answer" questions, which is a more balanced distribution compared to the most likely answer (where only 42% of correctly answered questions have an answer). This suggests that even though our model at 3/10 is liberal at predicting no answer, its second most likely answer is quite frequently correct and less biased.

4.3 Context Length

The previous section shows that our models are better at recognizing questions with no answers compared to BERT. Here, we look at which models perform better on longer sequences. At 3/10 of an epoch of fine-tuning, BERT and our models performed similarly (BERT average length: 165.86, our model average length: 164.50). However, we believe this is because both models answered correctly many of the same questions. When looking at questions that were uniquely answered correctly by each model, a clear difference emerges: the average sequence length for BERT is longer (BERT: 175.30, our model: 161.68). This suggests that our models are better at answering questions with shorter contexts than BERT. The same trend holds at 1 epoch of fine-tuning.

4.4 Do 2 epochs perform even better?

Our previous results show that models built on BERT embeddings at 1 epoch achieves maximal BERT performance at 2 epochs. Here, we investigate if we can further improve performance by training directly on embeddings derived at 2 epochs. While this approach does not reduce BERT fine-tuning time, it does provide a way to investigate whether our approach can achieve even better performance with longer fine-tuning.

Table [5] shows the results. As expected, the 2-epoch models outperform the same models trained on 1-epoch embeddings. Since ensembling improved performance at 1 epoch, we applied the same approach to the models at 2 epochs. Surprisingly, the 2 epoch ensemble does not outperform the 1 epoch ensemble in terms of either EM (0.749 vs 0.756) or F1 (0.795 vs 0.798). This suggests that additional fine-tuning does not guarantee better performance. Our 1-epoch models, in addition with ensembling, not only saves on BERT fine-tuning time, but also achieves the best possible performance, suggesting a total lack of need for fine-tuning beyond 1 epoch.

Model	**SQuAD2.0**	
	EM	**F1**
BERT $2e$ (*best*)	0.747	0.792
best model $1e$	0.753	0.797
best model $\frac{3}{10}e$	**0.749**	**0.795**
ensemble BERT $2e$ + best $1e$	0.751	0.796

Table 5: 2 epochs embeddings evaluation results

5 Conclusion and future work

In this paper, we propose a parameter-efficient approach that achieves maximal BERT performance for QA span annotation and greatly reduces fine-tuning time required for BERT. Our models are trained on BERT hidden state activations (embeddings), and consistently outperform BERT at the same level of fine-tuning. By using an ensemble of our model with BERT's predictions, we further surpass BERT performance, reducing the need for fine-tuning by 1 epoch. We achieved similarly promising results for QA classification, which suggests that this approach works well with both the full sequence BERT embeddings and with the CLS token embedding. Future work might look at reducing fine-tuning even further and applying this approach to other NLP tasks.

Acknowledgments

We are incredibly thankful for the faculty and instructors of W266 : *Natural Language Processing with Deep Learning* for their guidance, patience, and detailed feedback over the course of this

project. In particular, we call out special thanks to Daniel Cer, PhD and Joachim Rahmfeld without whom we would not have escaped the tangled maze of BERT hidden activations in time.

References

Betty van Aken, Benjamin Winter, Alexander Löser, and Felix A. Gers. 2019. How does bert answer questions? *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D. Davison. 2020. Table search using a deep contextualized language model. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

François Chollet. 2016. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357.

Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751.

Xiaofei Ma, Zhiguo Wang, Patrick Ng, Ramesh Nallapati, and Bing Xiang. 2019. Universal text representation from bert: An empirical study.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. pages 311–318.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going deeper with convolutions. *CoRR*, abs/1409.4842.

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*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. 2019b. What do you learn from context? probing for sentence structure in contextualized word representations. *CoRR*, abs/1905.06316.

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

Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. 2020. Incorporating bert into neural machine translation. *ArXiv*, abs/2002.06823.

A Appendices

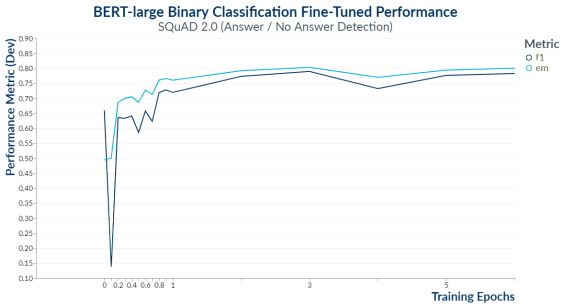
Included in this section are facts, figures, and tables that support our research but were not strictly required to communicate our findings effectively in the main paper body.

A.1 BERT fine-tuning figures

Model : BERT-large
Task : Binary Classification (SQuAD v2 - Answer / No Answer)
Parameters : 335,141,888

Baseline	Condition	Epochs	EM	F1	Time to Train (sec)
	BERT-large (no tuning)	0	0.4942306073	0.6599467694	n/a
	BERT-large fine-tuning	1/10	0.5008843595	0.1389131067	1421
X X	BERT-large fine-tuning	2/10	0.6871894214	0.6357395057	2840
	BERT-large fine-tuning	3/10	0.7004969258	0.6337796087	4258
	BERT-large fine-tuning	4/10	0.7052977344	0.6414591659	5676
	BERT-large fine-tuning	5/10	0.6871051967	0.5860724234	7096
	BERT-large fine-tuning	6/10	0.7279541818	0.6579839051	8514
	BERT-large fine-tuning	7/10	0.7129621831	0.6230922362	9932
	BERT-large fine-tuning	8/10	0.7612229428	0.7192791365	11350
	BERT-large fine-tuning	9/10	0.7659395267	0.7282151589	12769
X X	BERT-large fine-tuning	1	0.7609702687	0.7201735358	14105
	BERT-large fine-tuning	2	0.7925545355	0.7736421285	28293
	BERT-large fine-tuning	3	0.8035879727	0.7900990099	42478
	BERT-large fine-tuning	4	0.7704876611	0.7327645386	56665
	BERT-large fine-tuning	5	0.7944917039	0.7768836869	70864
	BERT-large fine-tuning	6	0.8006401078	0.7831027215	85054

(a) BERT$_{large}$ fine-tuning table : classification



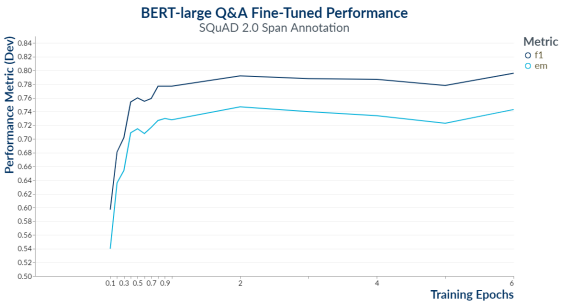
(b) BERT$_{large}$ fine-tuning plot : classification

Figure 5: BERT$_{large}$ fine-tuning for classification

Model : BERT-large
Task : Span Annotation (SQuAD v2 - Start/End Span Positions)
Parameters : 335,143,938

Baseline	Condition	Epochs	EM	F1	Time to Train (sec)
	BERT-large (no tuning)	0	0.036	0.051	n/a
	BERT-large fine-tuning	1/10	0.728	0.777	1375
	BERT-large fine-tuning	2/10	0.747	0.792	2750
X X	BERT-large fine-tuning	3/10	0.74	0.788	4125
	BERT-large fine-tuning	4/10	0.734	0.787	5500
	BERT-large fine-tuning	5/10	0.723	0.778	6875
	BERT-large fine-tuning	6/10	0.743	0.796	8250
	BERT-large fine-tuning	7/10	0.54	0.597	9625
	BERT-large fine-tuning	8/10	0.636	0.681	11000
	BERT-large fine-tuning	9/10	0.654	0.702	12375
X X	BERT-large fine-tuning	1	0.709	0.754	13750
	BERT-large fine-tuning	2	0.715	0.76	27537
	BERT-large fine-tuning	3	0.708	0.755	41324
	BERT-large fine-tuning	4	0.717	0.759	55111
	BERT-large fine-tuning	5	0.727	0.777	68898
	BERT-large fine-tuning	6	0.73	0.777	82685

(a) BERT$_{large}$ fine-tuning table : span annotation



(b) BERT$_{large}$ fine-tuning plot : span annotation

Figure 6: BERT$_{large}$ fine-tuning for span annotation