Enhancing BERT with Adapter Fusion: A Strategy for Multi-Task Knowledge Transfer

Stanford CS224N Default Project

Hao Xu
Department of Computer Science
Stanford University
xariaxu@stanford.edu

Abstract

This project delves into advanced methodologies aimed at enhancing the efficacy of the BERT model across three distinct downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. To harness insights from multiple tasks, the study explores sequential fine-tuning, multi-task learning, and adapter fusion techniques. Through experimentation, it becomes evident that sequential fine-tuning yields performance improvements solely when the involved tasks exhibit congruence in dataset characteristics, architectural nuances, and objectives. Conversely, adapter fusion emerges as a promising strategy for mitigating issues such as catastrophic forgetting and dataset imbalances, thereby offering a more robust approach for facilitating knowledge transfer across diverse multi-task settings.

1 Key Information to include

- Mentor: None
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

The prevailing approach for addressing natural language tasks involves the utilization of pretrained models. Transfer to a specific task is accomplished through fine-tuning all parameters of the pretrained model solely on that task, frequently leading to state-of-the-art performance outcomes. Nonetheless, this method necessitates the fine-tuning of all network parameters for each task, thereby culminating in the creation of task-specific models.

To disseminating information across multiple tasks, sequential fine-tuning entails commencing with the pretrained language model and subsequently fine-tuning it on each task individually. However, this approach is susceptible to the issue of catastrophic forgetting, leading to the erasure of previously acquired knowledge from all preceding tasks. This, coupled with the intricate determination of the optimal task order for fine-tuning the model, impedes the seamless transfer of knowledge across tasks.

In this study, I explore the efficacy of adapter fusion as a means of effectively disseminating knowledge across multiple tasks while circumventing the challenges associated with catastrophic forgetting and task balancing. adapter fusion adeptly tackles this challenge by delineating the knowledge extraction and composition stages. Through this segregation, adapter fusion unveils its potential to surpass the performance achieved by conventional full model fine-tuning. Notably, this

superiority is contingent upon the attainment of commensurate performance levels by the adapters during the knowledge extraction phase for each respective task.

3 Related Work

3.1 Current Approaches to Transfer Learning

Suppose we are given a model that is pretrained on a task with training data D_0 and a loss function L_0, the weights Θ_0 of this model are learned as follows:

$$D0 := \text{Large corpus of unlabelled text}$$
$$L0 := \text{Masked language modelling loss}$$
$$\Theta_0 \leftarrow \textbf{argmin}_{\Theta} L_0(D_0; \Theta)$$

We define C as the set of N downstream tasks having labelled data of varying sizes and different loss functions:

$$C = \{(D_1, L_1), \ldots, (D_N, L_N)\}$$

The aim is to be able to leverage a set of N tasks to improve on a target task n with $C_n = (D_n, L_n)$. In this work we focus on the setting where $n \in \{1, \ldots, N\}$. We wish to learn a parameterization Θ_n that is defined as follows:

$$\Theta_n = \textbf{argmin}_{\Theta'} L_n(D_n; \Theta')$$

where Θ' is expected to have encapsulated relevant information from all the N tasks. The target model for task n is initialized with Θ' for which we learn the optimal parameters Θ_n through minimizing the task's loss on its training data.

3.1.1 Sequential Fine-Tuning

Sequential fine-tuning involves iteratively adapting a model to each task by sequentially updating its weights. In this process, the model is fine-tuned for one task at a time, applying all weight updates before proceeding to the next task.

$$\Theta_{(n-1)\rightarrow n} \leftarrow \textbf{argmin}_{\Theta'} L_n(D_n; \Theta')$$

where $\Theta_{(n-1)\rightarrow n}$ indicates that we start with the previous $\Theta_{(n-1)}$.

However, this approach is susceptible to the issue of catastrophic forgetting (Phang et al. (2019)), where the model progressively loses knowledge acquired from previously learned tasks as it is fine-tuned on subsequent ones. This challenge is compounded by the complexity of determining the optimal sequence in which tasks should be addressed. Consequently, these factors significantly impede the effective transfer of knowledge across tasks, undermining the overall efficacy of sequential fine-tuning.

3.1.2 Multi-Task Learning (MTL)

Multi-Task Learning entails the fine-tuning of a pretrained language model by optimizing a weighted aggregate of the objective functions corresponding to each target task concurrently. This simultaneous training approach seeks to cultivate a shared representation across all tasks, thereby enhancing the model's ability to generalize effectively to each specific task.

$$\Theta_{0\rightarrow\{1,\ldots,N\}} \leftarrow \textbf{argmin}_{\Theta} \left(\sum_{n=1}^{N} L_n(D_n; \Theta_0) \right)$$

where $\Theta_{0\rightarrow\{1,\ldots,N\}}$ indicates that we start with Θ_0 and fine-tune on a set of tasks $\{1, \ldots, N\}$.

However, Multi-Task Learning necessitates simultaneous access to all tasks during the training phase, which imposes the requirement for complete joint retraining whenever new tasks are introduced.

Moreover, this approach presents a significant challenge in balancing the performance across multiple tasks, making it difficult to train a model that performs optimally on each task. As demonstrated by Lee et al. (2017), MTL models often exhibit overfitting on tasks with limited data and underfitting on tasks with abundant data. This disparity hinders the effective transfer of knowledge across tasks, thereby limiting the practical applicability of MTL in diverse real-world scenarios.

3.1.3 Adapter

Adapters tailor a limited set of task-specific parameters to each task while preserving the underlying pretrained language model intact. Consequently, adapters for multiple tasks can be trained independently and simultaneously.

$$\Phi_n \leftarrow \textbf{argmin}_{\Phi} L_n(D_n; \Theta_0, \Phi)$$

In this framework, for each of the N tasks, the core parameters Θ_0 remain constant, and only the task-specific parameters, Φ_n, are optimized. Φ contains considerably fewer parameters than Θ_0, only 3.6% of the pretrained model parameters in Houlsby et al. (2019). This approach enables efficient parallelization of the training process for adapters across all N tasks, facilitating the compartmentalization of learned knowledge within discrete components of the model.

3.1.4 Adapter Fusion

In the initial phase of knowledge extraction, as described by Pfeiffer et al. (2021), adapters are trained independently for each of the N tasks. In the subsequent phase of knowledge composition, these adapters are integrated through a process known as Adapter Fusion. During this fusion stage, the core model parameters, Θ, and the pre-trained adapters, Φ, are kept fixed. New parameters, Ψ, are introduced to learn how to effectively combine the outputs of the adapters, facilitating the model's ability to address the target task.

$$\Psi_n \leftarrow \textbf{argmin}_{\Psi} L_n(D_n; \Theta, \Phi_1, \ldots, \Phi_N, \Psi)$$

The training dataset for each task n is utilized in two distinct stages. First, it is employed to train the task-specific adapters, denoted as Φ_n. Subsequently, the same dataset is used to train the fusion parameters, Ψ_n, during the Adapter Fusion phase. By delineating these stages — knowledge extraction through adapter training and knowledge composition via Adapter Fusion — this methodology effectively mitigates issues such as catastrophic forgetting, task interference, and training instabilities.

4 Approach

4.1 Architecture

For sentiment analysis, a linear layer is appended to the BERT model. This layer takes an input of 768, corresponding to the base BERT embedding dimensionality, and produces an output matching the number of label categories, which is 5 (ranging from 0 for negative to 4 for positive). The model's prediction corresponds to the label with the highest output. Training is conducted using the cross-entropy loss function.

In the domain of paraphrase detection, I adopt a siamese BERT network structure (Reimers and Gurevych (2019)) to produce sentence embeddings. The cosine embedding loss is employed as the objective function, with the margin set to 0. Paraphrases are characterized by a cosine similarity larger than 0, whereas non-paraphrases yield a cosine similarity of 0 or less, as shown in Figure 1.

In the context of semantic textual similarity, Reimers and Gurevych (2019) employ a siamese BERT network architecture to derive semantically meaningful sentence embeddings, denoted as u and v. They compute the cosine similarity between these embeddings and utilize mean squared error (MSE) as the objective function. In this framework, sentences that are semantically equivalent yield a cosine similarity of 1, while those that are unrelated produce a cosine similarity of 0 or less. To quantify the degree of semantic equivalence on a scale from 0 (not at all related) to 5 (identical meaning), I adjust all negative cosine similarity values to 0 and scale the cosine similarity by a factor of 5, as illustrated in Figure 2.

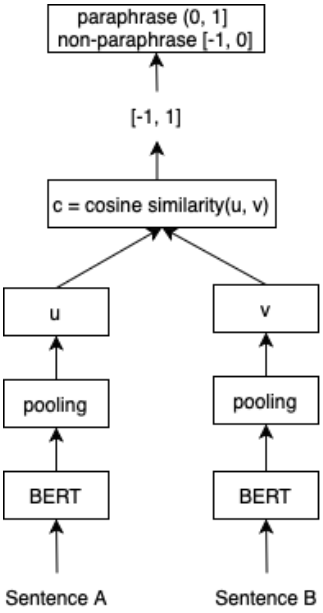

Figure 1: Paraphrase detection task head

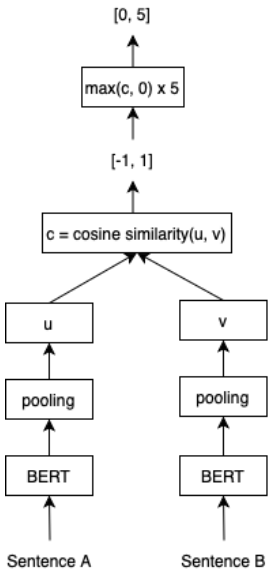

Figure 2: Semantic textual similarity task head

4.2 Adapter

Adapters offer a significant advantage in that they obviate the need to fine-tune the entire parameter set of a pretrained model. Instead, they introduce a limited number of task-specific parameters, allowing the pretrained language model to remain unchanged. The process employed by adapters involves an initial projection of the original d-dimensional features into a reduced dimension m, followed by the application of a non-linearity, and subsequently, a projection back to the original d-dimensional space. The total number of additional parameters introduced per layer, accounting for biases, can be quantified as $2md + d + m$ (Houlsby et al. (2019)). This parameter-efficient approach significantly reduces the computational cost and complexity associated with adapting pretrained models to new downstream tasks.

The configuration and integration of adapter within a pretrained model are intricate and critical to its performance. Houlsby et al. (2019) conducted experiments with various adapter architectures, concluding that incorporating adapter modules twice within each transformer layer yields optimal results. As illustrated in Figure 3, the adapters are applied directly to the outputs of both the attention and feed forward layers, precisely after these outputs are projected back to the input size, yet prior to the skip connection. Subsequently, the output from the adapter flows into the subsequent layer normalization. Each adapter module comprises a sequence of operations: a feed forward down-projection that reduces the input dimensionality, followed by a non-linearity using the Gelu activation function, and concluding with a feed forward up-projection that restores the original dimensionality.

4.3 Adapter Fusion

Adapter Fusion, as proposed by Pfeiffer et al. (2021), facilitates the dynamic integration of N task-specific adapters, Φ_n, with the shared pretrained model, Θ, by introducing a novel set of parameters Ψ. These parameters are designed to synthesize the outputs of the adapters based on the target task's data.

As illustrated in Figure 4, the Adapter Fusion parameters Ψ consist of Key, Value, and Query matrices at each layer l, denoted as K_l, V_l, and Q_l respectively. In this architecture, the Query matrix receives input from the pretrained transformer's output. In contrast, the Key and Value matrices are fed the outputs from the respective task-specific adapters. The product of the Query and Key matrices is

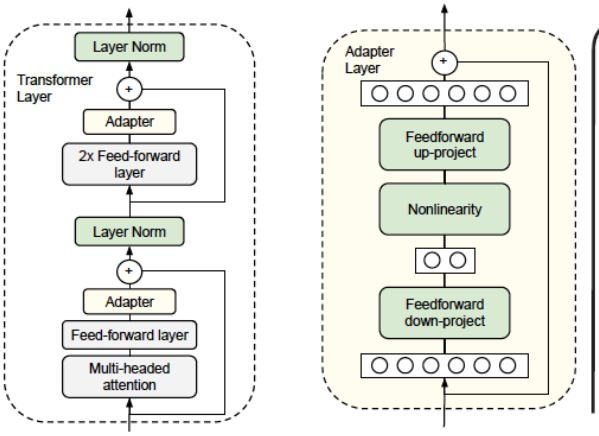

Figure 3: Adapter (Houlsby et al. (2019))

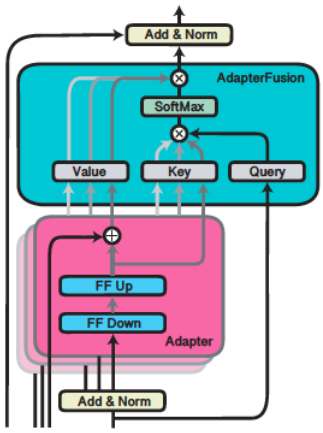

Figure 4: Adapter Fusion (Pfeiffer et al. (2021))

processed through a softmax function, which determines the adaptive weighting of each adapter relative to the given context.

$$
\begin{aligned}
s_{l,t} &= \textbf{softmax}(h_{l,t}^T Q_l \cdot z_{l,t,n}^T K_l), n \in \{1, \ldots, N\} \\
z'_{l,t,n} &= z_{l,t,n}^T V_l, n \in \{1, \ldots, N\} \\
Z'_{l,t} &= [z'_{l,t,0}, \ldots, z'_{l,t,N}] \\
o_{l,t} &= s_{l,t}^T Z'_{l,t}
\end{aligned}
$$

5 Experiments

5.1 Data

5.1.1 Large Movie Review Dataset (Maas et al. (2011))

The dataset consists of $50,000$ movie reviews, evenly divided into $25,000$ training and $25,000$ test samples. The overall label distribution is balanced, with $25,000$ positive and $25,000$ negative. Reviews scoring ≤ 4 out of 10 are classified as negative, while those scoring ≥ 7 are classified as positive. I fine-tune the model on this dataset to develop richer and more robust embeddings for sentiment analysis in the movie reviews domain.

Both the Large Movie Review Dataset (LMRD) and the Stanford Sentiment Treebank (SST) share the same model architecture. However, the labels in the LMRD are scaled from 1 to 10. To align these with the SST's 0 to 4 label range, I normalize the labels by subtracting 1 and then dividing by 2. This rescaling ensures consistency in label representation across datasets.

5.1.2 Stanford Sentiment Treebank (Socher et al. (2013))

The Stanford Sentiment Treebank, used for sentiment analysis, comprises $11,855$ individual sentences extracted from movie reviews. Each sentence is assigned a label representing its sentiment on a scale from 0 to 4: 0 for negative, 1 for somewhat negative, 2 for neutral, 3 for somewhat positive, and 4 for positive.

5.1.3 Quora Question Pairs Dataset (Iyer et al. (2012))

The Quora dataset, designed for paraphrase detection, consists of $404,298$ question pairs with labels indicating whether pairs of questions are paraphrases of each other.

5.1.4 SemEval STS Benchmark Dataset (Agirre et al. (2013))

The SemEval STS Benchmark dataset comprises $8,628$ sentence pairs, each assigned a similarity score ranging from 0 (unrelated) to 5 (equivalent meaning).

5.2 Evaluation method

To evaluate the overall quality of the model, I utilize an averaged composite metric comprising three key performance indicators: accuracy on the Stanford Sentiment Treebank, accuracy on the Quora Question Pairs Dataset, and the Pearson correlation coefficient on the SemEval STS Benchmark Dataset. This holistic measure aligns the model's performance with the project's objectives and conforms to the evaluation criteria set forth by the leaderboard.

5.3 Experimental details

In all experiments, I employ BERT-base-uncased Devlin et al. (2019) as the pretrained language model. A dropout rate of 0.3 is applied across linear layers. The fine-tuning process is conducted with a learning rate of $1e-5$ using the AdamW optimizer.

5.4 Results

All the dev results are presented in Table 1 and all the test results are presented in Table 2.

Table 1: Models Dev Results

Model	SST Dev Accuracy	QD Dev Accuracy	STS Dev Correlation
Last Linear Layer	0.	0.368	0.260
Full Model + CLS Token	0.514	0.626	0.785
Full Model + Mean Token	0.518	0.739	0.814
Sequential Fine-Tuning	0.523 (LMRD $\rightarrow$ SST) $\uparrow$	0.702 (LMRD $\rightarrow$ QD) $\rightarrow$	0.802 (LMRD $\rightarrow$ STS) $\rightarrow$
Sequential Fine-Tuning	-	0.698 (SST $\rightarrow$ QD) $\rightarrow$	0.789 (SST $\rightarrow$ STS) $\downarrow$
Sequential Fine-Tuning	0.427 (QD $\rightarrow$ SST) $\downarrow$	-	0.419 (QD $\rightarrow$ STS) $\downarrow$
Sequential Fine-Tuning	0.491 (STS $\rightarrow$ SST) $\rightarrow$	0.718 (STS $\rightarrow$ QD) $\rightarrow$	-
Adapter	0.459	-	0.638
Adapter Fusion (SST + STS)	0.419	-	0.675

Table 2: Models Test Results

SST Test Accuracy	QD Test Accuracy	STS Test Correlation	Overall Test Score
0.548	0.735	0.815	0.730

6 Analysis

6.0.1 Last Linear Layer

Due to the architecture of the model, only the sentiment analysis task head incorporates an additional linear layer, which maps the embedding dimensionality of 768 to 5 sentiment labels. This layer is the sole component that can be trained when the BERT parameters are frozen. In contrast, the paraphrase detection and semantic textual similarity tasks do not have any trainable task-specific head parameters. Consequently, this setup achieves a dev accuracy of 0.514 (Table **??**) in sentiment analysis, a dev accuracy of 0.368 in paraphrase detection, and a dev correlation of 0.260 in semantic textual similarity. These low values are primarily due to the reliance on the pretrained language model, which has not been fine-tuned for these specific downstream tasks yet.

6.0.2 Full Model

Each downstream task requires fine-tuning all model parameters specifically for that task, leading to a specialized model tailored to each individual task. A pooling operation generates a fixed-size

sentence embedding from the output of BERT. I experiment with two pooling strategies: using the output of the CLS token and computing the mean of all output vectors (mean strategy).

The results indicate that the CLS token achieves a dev accuracy of 0.514 on the Stanford Sentiment Treebank (Table 3), 0.626 on the Quora Question Pairs Dataset (Table 4), and a dev correlation of 0.785 on the SemEval STS Benchmark Dataset (Table 5).

In contrast, the mean strategy attains a dev accuracy of 0.518 on the Stanford Sentiment Treebank (Table 6), 0.739 on the Quora Question Pairs Dataset (Table 7), and a dev correlation of 0.814 on the SemEval STS Benchmark Dataset (Table 8). Notably, the mean strategy yields substantial improvements, averaging approximately 5%. Furthermore, the mean strategy achieves a dev accuracy of 0.721 on the Large Movie Review Dataset (Table 9).

6.0.3 Sequential Fine-Tuning

After sequentially training on two datasets across all possible combinations, I discover several noteworthy patterns:

(1) Training on the LMRD before the SST enhances performance. This improvement is observed because both LMRD and SST share a similar domain — movie reviews, and a common objective of sentiment analysis. This is the only instance where sequential fine-tuning leads to a performance boost. Conversely, training on the LMRD prior to the STS neither enhances nor diminishes performance. This neutrality can be attributed to the fact that LMRD's focus on sentiment analysis is largely irrelevant to the task of evaluating sentence similarity in STS, especially in cases where neither sentence conveys explicit sentiment.

(2) Training on the QD significantly degrades performance on all other tasks. This adverse effect can be attributed to two main factors. Firstly, the QD is substantially larger, containing $283,010$ training examples, compared to the SST and STS, which have $8,544$ and $6,040$ respectively. This size disparity means that any deviation in the training objectives of QD can cause considerable damage to performance when applied to the other tasks. Secondly, the nature of QD's task — paraphrase detection — differs fundamentally from those of SST and STS. Although STS also aims to assess semantic equivalence, it differs from QD in that it measures degrees of similarity rather than making a binary paraphrase decision. The QD employs a cosine embedding loss function, which tends to reward complete agreement when paraphrases are detected. This inclination makes the model more likely to assign a full score (e.g., 5 for "same meaning") upon detecting a paraphrase, thus failing to adequately handle the nuanced and varied degrees of similarity that are crucial in STS evaluations.

(3) Training on any dataset has a negligible impact on the performance of QD. The role of dataset imbalance is significant in sequential fine-tuning. Both the SST and the STS are 40 times smaller than QD, thus discrepancy in objective does not substantially affect the model's performance on QD.

6.0.4 Adapter Fusion

The introduction of supplementary adapter components while maintaining the frozen status of BERT parameters resulted in a marginal decline in performance. This phenomenon can be attributed to the relatively limited parameterization of adapter modules, accounting for merely 3.6% compared to the BERT model, thereby predisposing the observed performance decrement. However, subsequent to the knowledge composition phase, facilitated by adapter fusion, a slight enhancement in performance was discerned, particularly evident in the Semantic Textual Similarity (STS) task. This observation underscores the potential for surpassing the performance of the full model fine-tuning, contingent upon the optimization of adapter performance during the knowledge extraction phase.

7 Conclusion

In summary, my study highlights that the effectiveness of sequential fine-tuning is highly dependent on the similarity between tasks in terms of both datasets and objectives. When tasks share similar datasets and goals, sequential fine-tuning can significantly enhance performance by effectively transferring knowledge between them. However, if the tasks differ markedly in either dataset characteristics or objectives, sequential fine-tuning often fails to surpass the performance of training each model independently. These findings emphasize the critical role of task alignment in the success of sequential fine-tuning approaches.

In contrast, adapter fusion adeptly tackles this challenge by delineating the knowledge extraction and composition stages. Through this segregation, adapter fusion unveils its potential to surpass the performance achieved by conventional full model fine-tuning. Notably, this superiority is contingent upon the attainment of commensurate performance levels by the adapters during the knowledge extraction phase for each respective task.

The principal constraints of this study stem from the restricted timeframe within which the research was conducted. Specifically, the paraphrase detection model, characterized by its extensive database and prolonged training durations (approximately 2 hours per epoch), may not have undergone exhaustive fine-tuning to attain optimal performance levels. These circumstances impede the comprehensive exploration of the model's capabilities and potential enhancements, underscoring the need for extended experimentation periods to refine its efficacy further.

8 Ethics Statement

Deploying models like BERT in real-world applications entails ethical considerations, particularly concerning potential biases, misinterpretations, and privacy infringements. For instance, in sentiment analysis, the model can perpetuate or amplify existing societal biases present in the training data. Similarly, in paraphrase detection and semantic textual similarity tasks, biases in the training data may result in misrepresentations or misinterpretations of semantic equivalence. Moreover, user data may be leaked.

To mitigate these ethical risks, several practical risk mitigation strategies can be implemented. Firstly, it is imperative to employ diverse and representative datasets during model training to reduce biases. Additionally, ongoing monitoring and evaluation of model performance, particularly for sensitive tasks, can help identify and rectify biases as they arise. Furthermore, privacy protection measures should be implemented to safeguard user data, especially in models where personal information may be processed.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

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. arXiv preprint arXiv:1902.00751.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2012. First quora dataset release: Question pairs. Quora.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully character-level neural machine translation without explicit segmentation. volume 5, pages 365–378, Cambridge, MA. MIT Press.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Jason Phang, Thibault Févry, and Samuel R. Bowman. 2019. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. arXiv preprint arXiv:1811.01088.

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*. Association for Computational Linguistics.

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.

A Appendix (optional)

Table 3: Full Model + CLS Token - Stanford Sentiment Treebank (Socher et al. (2013))

Epoch	Train Loss	Accuracy (Train)	Accuracy (Dev)
1	1.220	0.622	0.505
2	0.954	0.735	0.502
3	0.731	0.815	0.490
4	0.529	0.912	0.514
5	0.364	0.923	0.496
6	0.247	0.928	0.493
7	0.173	0.976	0.498
8	0.150	0.978	0.507
9	0.116	0.986	0.494
10	0.088	0.984	0.490

Table 4: Full Model + CLS Token - Quora Question Pairs Dataset (Iyer et al. (2012))

Epoch	Train Loss	Accuracy (Train)	Accuracy (Dev)
1	0.298	0.557	0.543
2	0.266	0.581	0.556
3	0.251	0.604	0.572
4	0.239	0.637	0.592
5	0.229	0.665	0.611
6	0.222	0.662	0.601
7	0.216	0.681	0.611
8	0.211	0.698	0.626
9	0.207	0.694	0.617
10	0.204	0.696	0.617

Table 5: Full Model + CLS Token - SemEval STS Benchmark Dataset (Agirre et al. (2013))

Epoch	Train Loss	Pearson Correlation (Train)	Pearson Correlation (Dev)
1	0.173	0.822	0.778
2	0.127	0.862	0.781
3	0.106	0.890	0.781
4	0.093	0.910	0.783
5	0.083	0.921	0.785
6	0.076	0.927	0.782
7	0.070	0.928	0.778
8	0.066	0.933	0.779
9	0.063	0.933	0.779
10	0.062	0.937	0.779

Table 6: Full Model + Mean Token - Stanford Sentiment Treebank (Socher et al. (2013))

Epoch	Train Loss	Accuracy (Train)	Accuracy (Dev)
1	0.016	0.998	0.513
2	0.013	0.999	0.502
3	0.016	0.991	0.507
4	0.021	0.997	0.518
5	0.018	0.996	0.504
6	0.019	0.997	0.508
7	0.023	0.999	0.501
8	0.012	0.998	0.518
9	0.019	0.999	0.505
10	0.014	0.994	0.490

Table 7: Full Model + Mean Token - Quora Question Pairs Dataset (Iyer et al. (2012))

Epoch	Train Loss	Accuracy (Train)	Accuracy (Dev)
1	0.293	0.589	0.567
2	0.260	0.633	0.589
3	0.244	0.671	0.617
4	0.232	0.697	0.634
5	0.223	0.725	0.643
6	0.215	0.738	0.649
7	0.208	0.750	0.653
8	0.203	0.760	0.650
9	0.199	0.778	0.674
10	0.195	0.779	0.675

Table 8: Full Model + Mean Token - SemEval STS Benchmark Dataset (Agirre et al. (2013))

Epoch	Train Loss	Pearson Correlation (Train)	Pearson Correlation (Dev)
1	0.140	0.863	0.807
2	0.100	0.903	0.813
3	0.082	0.925	0.814
4	0.070	0.934	0.810
5	0.062	0.943	0.813
6	0.057	0.945	0.810
7	0.054	0.947	0.807
8	0.051	0.949	0.809
9	0.049	0.949	0.810
10	0.048	0.952	0.809

Table 9: Full Model + Mean Token - Large Movie Review Dataset (Maas et al. (2011))

Epoch	Train Loss	Accuracy (Train)	Accuracy (Dev)
1	0.791	0.747	0.704
2	0.586	0.830	0.721
3	0.450	0.888	0.719
4	0.330	0.956	0.708
5	0.214	0.977	0.706