Efficient Structured Prediction with Transformer Encoders

Ali Basirat, Centre for Language Technology, University of Copenhagen alib@hum.ku.dk

Abstract Finetuning is a useful method for adapting Transformer-based text encoders to new tasks but can be computationally expensive for structured prediction tasks that require tuning at the token level. Furthermore, finetuning is inherently inefficient in updating all base model parameters, which prevents parameter sharing across tasks. To address these issues, we propose a method for efficient task adaptation of frozen Transformer encoders based on the local contribution of their intermediate layers to token representations. Our adapter uses a novel attention mechanism to aggregate intermediate layers and tailor the resulting representations to a target task. Experiments on several structured prediction tasks demonstrate that our method outperforms previous approaches, retaining over 99% of the finetuning performance at a fraction of the training cost. Our proposed method offers an efficient solution for adapting frozen Transformer encoders to new tasks, improving performance and enabling parameter sharing across different tasks.

1 Introduction

The text encoder models evolved from the Transformer architecture (Vaswani et al., 2017) have extensively influenced natural language processing. The standard workflow of these models is based on transfer learning from a model pre-trained on vast amounts of text to a target task. Finetuning is a commonly used technique adjusting the parameters of a pre-trained encoder to a target task using the standard backpropagation algorithm. Despite its simplicity and tremendous success, finetuning can be computationally expensive, particularly for structured prediction tasks in which the parameters are updated at the token level, as opposed to document classification tasks in which the parameters are updated for each document.[1]

In addition, due to in-place parameter updates, finetuning limits the reusability of Transformer encoders, particularly in cloud environments where resources are shared between users. Additionally, finetuning a Transformer encoder for a specific task or a limited number of tasks does not necessarily perform well on other tasks that are not similar to the target task. This is because of catastrophic forgetting, which reduces the generalizability of neural network performance on out-of-domain data (McCloskey and Cohen, 1989). Consequently, a finetuned Transformer encoder becomes a massive computational block specified for a target task, which limits the scalability, modularity, and compositionality of the base encoder (Pfeiffer et al., 2020).

Recent attempts to resolve the shortcomings of finetuning are based on the adapter mechanism (Houlsby et al., 2019) that injects learning blocks into a frozen encoder to facilitate knowledge transfer from pre-training to target tasks (Pfeiffer et al., 2021; Stickland and Murray, 2019; Guo et al., 2021; Hu et al., 2022). While this approach can effectively replicate the finetuning performance (Hu et al., 2022), it still requires careful considerations to balance the encoder's sharability and inference efficiency in a cloud environment, as it tends to sacrifice inference efficiency for sharability, and vice versa, as we will empirically show in this study.

We adopt a different strategy based on the early studies of Peters et al. (2018); Kondratyuk and Straka (2019); Hao et al. (2020) for Transformer encoder adaptation. In contrast to the adapter approach (Houlsby et al., 2019), which adds trainable parameters to the encoder, this method pipes the encoder into an aggregation block that adapts the representations obtained from the encoder's intermediate layers to a target task through linear interpolation. Accordingly, this approach does not necessitate changing the base encoder architecture, making it easier to be shared when compared to the adapter solution.

Although the layer aggregation approach is easy to implement, it requires further consideration when applied to structured prediction. The primary reason for this is that the method assumes that the layers' linear weights are a function of the target task solely and independent of input tokens. This means that a layer weight remains constant for all tokens during inference.

[1] For example, finetuning a BERT model can take 25 days for parsing (Kondratyuk and Straka, 2019) and 2 days for relation extraction (Huguet Cabot and Navigli, 2021).

While this assumption might be acceptable for an encoder that is *finetuned* for *document classification*, as (1) finetuning allows for training the substantial parametric capacity of the original encoder, and (2) classification often relies on a single token vector that represents the entire document, it may not hold true for a frozen model employed for structured prediction. In such cases, it is necessary to model the word dynamics based on the available intermediate representations because the model parameters remain fixed throughout training. Furthermore, Peters et al. (2019) suggest that the linear combination of intermediate representations in a frozen model can match the performance of a finetuned model only if the pre-training and target tasks are sufficiently similar. This implies the need for additional customization of the aggregated representations to account for any disparities between the pre-training and target tasks.

Our paper presents an encoder adaptation model that effectively combines the efficiency of a frozen model with the effectiveness of a finetuned model by addressing the weaknesses of the linear aggregation method. Our approach includes two key mechanisms: an aggregation block and a tailoring block. The aggregation block models the dynamics of words by utilizing the intermediate representations of the encoder and introduces an attention mechanism that trains token representations based on both the target task and the local contribution of intermediate layers to tokens. The tailoring block reduces the impact of dissimilarity between the pretraining and target tasks by refining the aggregated representation, thus allowing for more effective knowledge transfer from pre-training to the target task.

We are not the first to propose the intermediate layer aggregation at the token level. Cao et al. (2022) have also explored a similar approach. They train the token-layer attention weights based on a task-specific query vector used to measure the similarity between intermediate representations locally. Nonetheless, our experiments demonstrate the practical benefits of our approach in downstream tasks in structured prediction and document classification.

We evaluated the effectiveness of our adaptation mechanism on two major classes of Transformer encoders: BERT-based (Devlin et al., 2019) and GPT-based (Radford et al., 2019) models. Our evaluation of the proposed adaptation mechanism builds upon the promising results of ablation studies of the two key blocks. Our experimental results confirm that the adaptation mechanism can perform as well as finetuning while being approximately 13 times more efficient in training time and consuming only 0.3% of the memory required for finetuning, with almost no harm to the inference efficiency. Compared to other approaches, our technique performs significantly better on the majority of structured prediction tasks and remains on par with the best-performing models for document classification.

2 Related Work

The initial investigation of BERT showed that it captures a rich hierarchy of linguistic knowledge in its intermediate layers (Tenney et al., 2019b,a; Jawahar et al., 2019; Lin et al., 2019; de Vries et al., 2020); This leads to the effective use of BERT by linearly aggregating the middle layers based on a target task (e.g. sentiment analysis (Horne et al., 2020; Xiao et al., 2021), morphosyntactic parsing (Kondratyuk and Straka, 2019), gender debiasing and coreference resolution (Abzaliev, 2019), and cross-lingual transfer learning (Chen et al., 2022)). Building on this approach, Cao et al. (2022) expand the task-oriented layer aggregation to encompass both token and task aspects. This extension is achieved through the introduction of an attention fusion model that leverages the local features of a token. We extend Cao et al. (2022)'s method by introducing an attention mechanism that incorporate global views of intermediate representations.

Other techniques that combine intermediate representations include Su and Cheng (2019), who apply the squeeze and excitation technique (Hu et al., 2018), and Yang and Zhao (2019), who use a bidirectional GRU layer to calculate the linear weights of the intermediate representations. From an architectural point of view, our adapter mechanism is based on a dynamic aggregation of the intermediate representations and preserves the parallel encoding functionality of the original encoder. This is in contrast to the linear method of Kondratyuk and Straka (2019), which is based on a static weighting of the intermediate representations, as well as the methods of Yang and Zhao (2019), Horne et al. (2020) and Xiao et al. (2021), which add overhead sequential units to the encoder, hampering parallel computing.

When it comes to improving efficiency, the literature has proposed strategies such as knowledge distillation (Hinton et al., 2015), attention pruning (Michel et al., 2019), model quantization (Zafrir et al., 2019), low-rank adaptation (Hu et al., 2022), shallow finetuning (Ben Zaken et al., 2022) and prompt tuning (Li and Liang, 2021). Our method aligns with the adaptation techniques category, augmenting a frozen model with a few learning blocks to facilitate knowledge transfer to a target task. A related approach by Houlsby et al. (2019) injects adapter layers into a frozen BERT model to enable model sharing in a cloud environment, specifically for efficient sequential multitask learning. Pfeiffer et al. (2021) address the catastrophic forgetting issue and balancing of different tasks in Houlsby et al. (2019)'s ap-

proach. Additionally, Stickland and Murray (2019) enhance the efficiency of the adaptation technique by incorporating a low-rank approximation of the model's key operations. In connection to this work, Hu et al. (2022) introduce an efficient method centered around the low-rank factorization of the expected changes in attention matrices in a transformer encoder.

Our approach diverges from that of Houlsby et al. (2019), Pfeiffer et al. (2021), and Hu et al. (2022) in several ways. In terms of model integration, we envelop the adapter around a pre-trained encoder model, unlike the strategies employed by Houlsby et al. (2019), Pfeiffer et al. (2021), and Hu et al. (2022), who embed the adapter blocks within the original encoder architecture. This distinction allows us to exclude the encoder during training, resulting in a substantial enhancement of training efficiency and facilitating model sharing. In addition, the number of trainable parameters in the models of Houlsby et al. (2019) and Hu et al. (2022) scales with the number of intermediate layers of the original encoder, as opposed to our model in which the number of trainable parameters is almost independent of the number of middle layers. Moreover, our encoder adaptation mechanism uses a dedicated tailoring block to explicitly address the differences between the pre-training and target tasks.

3 Encoder Adaptation

An encoder $\mathcal{B}$ consisting of $l-1$ intermediate Transformer layers plus one input embedding layer transforms an input document $s = (t_1, \ldots, t_n)$ into a three-dimensional tensor:

$$\mathcal{B} : V^n \rightarrow \mathbb{R}^{l \times n \times d}$$

where V is a vocabulary, d is the number of encoder dimensions, and l is the number of intermediate layers. The output tensor $B = \mathcal{B}(s)$ has three views corresponding to its dimensions: A layer view $B_{i,:,:}$ is an $n \times d$ matrix sliced along the layer dimension of B at index i representing a state matrix for the ith layer. A token view $B_{:,j,:}$ is an $l \times d$ matrix sliced along the token dimension of B at index j representing the token at position j. Finally, a feature view $B_{:,:,k}$ is an $l \times n$ matrix sliced along the third dimension of B at index k representing the kth embedding sub-space of encoder feature space.

An adapter function takes an encoding tensor B and merges its layer views (i.e.,) into a matrix:

$$\mathcal{A} : \mathbb{R}^{l \times n \times d} \rightarrow \mathbb{R}^{n \times d}$$

The resulting matrix includes distilled information of the intermediate representations adapted to a target task. We propose a parametric adaptation function consisting of two blocks: an *aggregation block* that merges the layer views and a *tailoring block* that adjusts the aggregated representations to a target task. The final adapted output is constructed by a residual connection, enabling the model to control the tailoring influence on the aggregation. The architecture is shown in Figure 1.

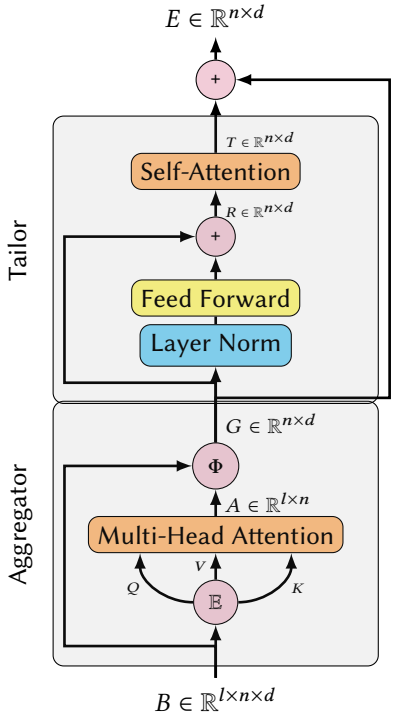

Figure 1: The proposed adapter architecture. The input tensor B contains the intermediate representations taken from a Transformer-based encoder for a sequence of tokens, and the output is a token embedding matrix.

3.1 Aggregation

The aggregation block takes the input tensor and merges the layer views. It uses a multi-head attention layer (Vaswani et al., 2017) to calculate the attention weights between pairs of layers and tokens based on the global views of B along with the layer and token dimensions. It then pools the feature vectors that the layers produce for every token based on the attention weights of the token and layers.

Breaking it down step by step, the input tensor B is first passed through the block $\mathbb{E}$ that calculates the attention's query, key, and value matrices. The query matrix Q is a linear projection of the mean matrix $\mathbb{E}_l = \frac{1}{l} \sum B_{i,:,:}$:[2] More formally,

$$Q = \mathbb{E}_l W_Q + b_Q \,,$$

[2]Our preliminary results with uniform and weighted averaging shows that both models perform equally on our development set. Therefore, we select uniform averaging because of its simplicity.

where W_Q is a $d \times k$ trainable matrix, and b_Q is a k-dimensional bias vector with $k \ll d$. Similarly, the key and value matrices are based on a linear projection of $\mathbb{E}_t = \frac{1}{n} \sum B_{:,j,:}$:

$$K = \mathbb{E}_t W_K + b_K \qquad \text{and} \qquad V = K\,,$$

where the $d \times k$ matrix W_K and the k-dimensional bias vector b_K ($k \ll d$) are learnable parameters.[3] We refer to the parameter k as the *attention dimensionality* of the adapter.

Next, the Multi-Head Attention layer constructs an $l \times n$ matrix A whose columns define weight distributions over the layer views for each token. Intuitively, the attention value $A_{i,j}$ indicates the importance of layer view $B_{i,:,:}$ in the representation of the token t_j. Finally, the aggregated representation for a token at position j is calculated in the computational block Φ based on the weighted sum of the corresponding intermediate representations:

$$G_{j,:} = \sum_{i=1}^{l} A_{i,j} B_{i,j,:}$$

This sum pooling is equivalent to $\text{diag}(A^T B)$ where the diag operator is applied on the first and second dimensions of the product $A^T B$.

3.2 Tailoring

The tailor block further specifies the aggregated token vectors to the target task. Inspired by the Transformer architecture (Vaswani et al., 2017), the tailor block adopts residual learning (He et al., 2016) with a residual mapping consisting of a layer normalization (Ba et al., 2016) followed by a position-wise feed-forward network:

$$R = G + \text{ReLU}(\text{Dropout}(L(\text{Norm}(G))))\,,$$

where L is a linear layer that adjusts the aggregated token vectors to the target task. Following Xiong et al. (2020), we use pre-layer normalization in which the normalization layer is placed before the feed-forward network.[4]

Finally, the self-attention layer compensates for the lack of trainable parameters to model task-specific dependencies between tokens. We calculate a self-attention matrix based on a linear transformation of the input vectors:

$$V = \text{Dropout}(R) W_T + b_T \qquad Q = K = V$$

where W_T is a $d \times k$ trainable matrix, and b_T is a k-dimensional bias vector. We then utilize a MultiheadAttention layer (Vaswani et al., 2017) to construct an $n \times n$ attention matrix M based on the query (Q), key (K), and value (V) matrices. The reason for performing the linear transformation on R is to reduce the dimensionality of the token vectors, which determine the size of the learning parameters in the MultiheadAttention block. To calculate the tailored matrix T of size $n \times d$, we multiply the attention matrix by the input matrix R:

$$T = M \times R$$

The rows of T are the sum of the token representations in R weighted by their attention scores.

4 Experiments

We study the adapter performance on downstream tasks and investigate the contribution of the aggregation and tailoring blocks to the performance gain. The experiments are based on the cased versions of BERT-Large (Devlin et al., 2019) and ROBERTA-Large (Liu et al., 2020) models as representatives of encoder-only models, and different variants of the GPT2, including GPT2-Small, Medium, and Large, as representatives of decoder-only models. All models are provided by Huggingface.

The test benchmark includes the two major types of classification tasks in NLP, including structured prediction and document classification. The evaluation benchmark for structured prediction includes tasks defined on the following datasets:

- CoNLL-2003 (Tjong Kim Sang and De Meulder, 2003): part-of-speech tagging (POS), chunking (CHK) and named-entity recognition (NER)
- Universal Dependencies (Nivre et al., 2017) (English EWT, v2.3): part-of-speech tagging (UPOS and XPOS), dependency label prediction (DEPREL), and dependency parsing (LAS)
- English Penn Treebank (Marcus et al., 1993) converted to Stanford Dependencies: dependency parsing (LAS)
- WEBNLG (Gardent et al., 2017): named-entity recognition (NER) and relation extraction (RE)

We use the standard data splits for training, validation, and testing the models. The reason for incorporating multiple datasets is to demonstrate the method's robustness not only across tasks but also in varied data and domains. The document classification benchmark is based on the GLUE tasks (Wang et al., 2018) including grammaticality (CoLA), sentiment textual similarity

[3] We set V equal to K based on our preliminary experiments showing no significant difference in the results with and without a dedicated transformation of the values.

[4] This is in contrast to post-layer normalization, which is used in the original Transformer architecture (Vaswani et al., 2017) and the BERT implementation of Devlin et al. (2019).

(STS-B), semantic equivalence (MRPC), textual entailment recognition (RTE), and sentiment analysis (SST-2). For these tasks, we train and validate models on their training data, and keep the benchmark's validation data for final model testing.

We set the attention dimension k to 16 and the number of attention heads to 2 in both attention layers in the aggregator and adapter blocks. This decision is based on our preliminary results on the CoNLL development data. For each task, we train five classifiers with different random seeds and report the average results on the test sets. As our optimizer, we use Adam with a 1cycle learning rate scheduler with cosine annealing. Our implementations are based on PyTorch (Paszke et al., 2019) and the experiments are carried out on an NVIDIA A100 40GB Tensor Core GPU. An implementation of the encoder model is available here `https://github.com/abasirat/llm-adapter`.

5 Performance

This section summarizes the results collected from the structured prediction and document classification tasks. We compare our adapter performance, named Adapted, with other techniques, namely:

- Frozen: returns the output of the last layer of a frozen encoder (i.e., $\mathcal{A}(B) = B_{l,:,:}$).
- Linear: returns a scaled linear aggregation of intermediate representations of a frozen model, i.e., $\mathcal{A}(B) = c\sum_{i=1}^{l} w_i B_{i,:,:}$ where $w_0, \ldots, w_l = \text{Softmax}(\alpha_0, \ldots, \alpha_l)$ (Kondratyuk and Straka, 2019). We train the parameters α_i and c together with other trainable parameters of each task's classifier.
- Fusion: returns a linear aggregation of the intermediate representations of a frozen model for each token and task, i.e., $\mathcal{A}(B) = \sum_{i=1}^{l}(A \odot B)_{i,:,:}$ where A is an $l \times n$ attention matrix and $A \odot B$ is the Hadamard product between A and every feature view of B. The attention matrix A is based on a d-dimensional task-specific attention vector Q learned during training, i.e., $A_{i,j} = \frac{\exp(Q \times B_{i,j,:})}{\sum_{k=1}^{l}\exp(Q \times B_{k,j,:})}$ (Cao et al., 2022).
- Lora: returns the output of the last layer of an encoder adapted to a target task based on the Lora factorization technique of Hu et al. (2022). Lora is an efficient technique that improves state-of-the-art on most downstream tasks. It extends the adaptation technique of Houlsby et al. (2019) using a simplified version of the method of Aghajanyan et al. (2020). We train Lora with its recommended setting.
- Finetune: returns the output of the last layer of a fine-tuned encoder model (Devlin et al., 2019).

We freeze the encoder parameters in all of the above techniques except for the Finetune, in which all encoder parameters are updated during training. The outputs taken from these encoders are subsequently employed for the target tasks outlined in Section 4. The following sections provide detailed insights into the modules employed for each task.

5.1 Structured Prediction

In this section, we study the adapter performance on BERT and GPT encoder families. The token classifiers for POS tagging, chunking (CHK), and DEPREL prediction consist of an encoder (e.g., BERT) with a light head block mapping a token vector to a probability distribution over target tags. The head block consists of a dropout layer followed by a dense layer of size $d \times d$ (d is the encoder's dimensionality) with *tanh* activation connected to another dropout and a dense layer of size $d \times |\text{tag set}|$ with a softmax activation. Following Ács et al. (2021), we classify tokens based on their first subword for semantic tasks such as NER in CoNLL2003 and their last subword for syntactic tasks such as POS and CHK in CoNLL2003 and UPOS, XPOS, and DEPREL in UD. We integrate the encoder adapters into the parser of Dozat and Manning (2017) for the parsing experiments and the joint named entity and relation extraction system of Yan et al. (2021) for NER and RE in WEBNLG.[5]

We set the batch size to 32 sentences for the tasks in CoNLL, UD (excluding dependency parsing), and WEBNLG and 5000 samples for dependency parsing on UD and WSJ. We train the CoNLL and UD taggers for 10 epochs, and the parsers and relation extractors for 100 epochs. We schedule the learning rate using the 1cycle policy (Smith and Topin, 2017) with a cosine annealing strategy. The maximum learning rate for parsing is set to $2e-3$, and for other tasks is selected among $\{1e-5, 1e-4, 1e-3\}$ based on the models' performance on the development sets. We disable the parser's character embedding module to better study the adaptation effect. The other parameters in the parser and relation extractor are set to their default values.

5.1.1 BERT Family

Table 1 summarizes the results obtained from each adaptation technique on the structured prediction tasks. The Adapted models based on our proposed approach perform better than Frozen, Linear, and Fusion models in all tasks,[6] and on par with Lora and Finetune

[5] Our parsing experiments are based on the parser implementation available at `https://github.com/Unipisa/biaffine-parser`

[6] One exception is dependency parsing, in which the linear models perform slightly better than the Adapted models.

models. On average, the Adapted models result in extensive improvements over the baseline frozen models by 3.7%, which is significantly larger than the improvement made by the Linear (2.6%) and Fusion (1.4%) models , and is on a par with Lora (3.8%) and Finetune (3.8%) models (see Figure 2 for detailed information).

A deeper look into the results shows that the absolute improvement made by the Adapted models over the baseline Frozen models, as shown in Figure 2, is higher than that of the other techniques on the syntactic tasks, such as POS tagging and chunking, but slightly lower than Lora on the more semantic tasks NER and RE. This suggests that the Adapted models are more effective than other techniques in encoding the syntactic information. A further detailed study on the contribution of the aggregation and tailoring blocks to syntactic and semantic encoding is presented in Section 6.

Also, we see that Adapted models outperform other models in retaining 99.5% of the BERT's finetuning performance (excluding dependency parsing in which the finetuned models perform significantly lower than the frozen baseline) and level with Lora on the ROBERTA's finetuning performance (i.e., the Adapted and Lora models retain 99.2% and 99.4% of finetuned ROBERTA models, respectively.[7]

5.1.2 GPT Family

GPT models are generative language models known for their strong performance in text generation tasks. However, they are less commonly used for discriminative tasks, included in our test benchmarks. The purpose of our experiments with GPT models is to present empirical evidence on the utility of different adapting techniques for the GPT family. We leverage a GPT model as a feature extractor, bypassing its final generator layer. After tokenizing an input string, we perform a forward pass through the GPT model with the tokenized input and extract the hidden states from all layers. These hidden states serve as the encoder output, denoted as B in our adapter formulation in Section 3, which is subsequently adapted for downstream tasks. To manage computational costs, the experiments focus on a smaller number of tasks and exclude Finetune experiments.

Table 2 summarizes the results collected from GPT models. First, compared to the BERT-based models, the results show that the adapted GPT models perform weaker. This performance drop is expected due to the discriminative nature of the tasks that are not generally considered for GPT models. However, within the adapted GPT models, our adapter performs better than other techniques in all tasks.[8] This contrast with the relative model performance on the BERT family, where the Adapted model performs on par with Lora on most tasks. We are uncertain about the cause of this discrepancy in the better relative performance of the Adapted model on the GPT family compared to the BERT family, and we leave it for further investigation in future research.

5.2 Document Classification

While our adaptation technique primarily emphasizes individual token representations rather than document representation, we still find it important to examine the adapter performance on standard document classification tasks, even though it may not be the optimal use case for our technique. We apply the adaptation technique on a subset of GLUE tasks (Wang et al., 2018) and compare its performance with other methods. The experiments are based on the CoLA, STS-B, RTE, SST-2, MRPC, and RTE tasks. We train five models with different random seeds and report the average classification score on the development sets. We select the learning rate from $\{1e-4, 3e-4, 7e-4\}$ for Frozen, Linear, Fusion, Adapted, and Lora techniques and from $\{1e-5, 2e-5, 3e-5\}$ for Finetune models.

We follow the standard approach for document classification that encodes an input document (e.g., a sentence or a pair of sentences) into a vector representation and then passes it into a header block that maps the vector to a class distribution. The document vector is often a dedicated vector (e.g., [CLS] in BERT) or the mean of the token vectors comprising the input document. In this study, we use the mean vector, which relies on the token representations, to represent a document. The classification head is similar to the header block we use for structured prediction networks. Except for SST-2 classification models, we train other models up to 100 epochs with a learning patience of 20. The SST-2 models are trained for 20 iterations and the learning patience of 10. The batch size in all training setups is 32.

Figure 3 summarizes the results of document classification. On average, the Adapted models (79.6) outperform the Frozen (69.7), Linear (71.3), and Fusion (74.3) models but perform slightly weaker than Lora (81.4) and Finetune (81.7) models on the document classification. The higher performance of the Adapted to the Fusion models indicates that our token-layer attention mechanism based on the global contextual information is more meaningful to the document classification tasks

[7] We exclude dependency parsing from the finetuning analysis because it does not perform as expected. We considered different finetuning strategies (finetuning top $n = 1, \ldots, 4$ layers, finetuning middle layers ($9-17$), followed by linear aggregation, and finetuning only during the first and second epochs. In all experiments, we could get maximum LAS of 94.4 for BERT and 93.8 for ROBERTA on WSJ, which is still significantly below the frozen baseline.

[8] One exception is Lora used for DEPREL prediction.

BERT-Large (345M)

	CoNLL-2003 (English)			UD (English-EWT)				WSJ	WEBNLG	
	POS	CHK	NER	UPOS	XPOS	DEPREL	LAS	LAS	NER	RE
Frozen	89.4 ±0.06	83.9 ±0.05	79.9 ±0.23	92.7 ±0.06	92.8 ±0.06	80.4 ±0.04	95.04 ±0.53	94.83 ±0.05	96.2 ±0.19	89.8 ±0.25
Linear	93.2 ±0.07	89.6 ±0.04	85.6 ±0.09	96.2 ±0.02	96.1 ±0.04	87.8 ±0.02	96.37 ±0.43	**95.24** ±0.09	96.6 ±0.21	91.4 ±0.47
Fusion	92.5 ±0.30	87.9 ±1.32	87.2 ±0.15	95.9 ±0.07	95.8 ±0.14	87.9 ±0.04	95.76 ±0.51	95.07 ±0.07	96.4 ±0.43	90.2 ±1.02
Adapted	93.6 ±0.04	90.4 ±0.17	89.5 ±0.27	96.4 ±0.07	96.3 ±0.06	91.7 ±0.05	**96.63** ±0.36	95.17 ±0.05	97.3 ±0.08	**92.2** ±0.35
Lora	92.8 ±0.17	89.4 ±0.22	89.8 ±0.22	96.0 ±0.07	96.0 ±0.07	91.6 ±0.42	95.09 ±0.50	94.90 ±0.07	97.4 ±0.07	92.0 ±0.21
Finetune	**93.9** ±0.07	**91.1** ±0.07	**91.3** ±0.13	**96.8** ±0.04	**96.8** ±0.04	**93.6** ±0.07	–	94.30 ±0.21	**97.8** ±0.07	91.4 ±0.10

ROBERTA-Large (345M)

	CoNLL-2003 (English)			UD (English-EWT)				WSJ	WEBNLG	
	POS	CHK	NER	UPOS	XPOS	DEPREL	LAS	LAS	NER	RE
Frozen	91.3 ±0.05	87.8 ±0.05	82.8 ±0.12	94.0 ±0.04	93.8 ±0.03	83.8 ±0.08	96.29 ±0.46	95.40 ±0.13	95.6 ±0.11	89.8 ±0.21
Linear	93.2 ±0.03	89.6 ±0.04	86.1 ±0.10	96.3 ±0.02	96.3 ±0.05	87.5 ±0.06	**96.68** ±0.41	**95.53** ±0.04	96.5 ±0.11	91.4 ±0.34
Fusion	91.2 ±0.93	86.2 ±3.50	85.1 ±2.58	95.2 ±0.31	94.8 ±0.43	85.6 ±1.15	96.14 ±0.43	95.31 ±0.15	94.8 ±1.52	88.1 ±2.02
Adapted	93.6 ±0.03	90.5 ±0.12	89.7 ±0.23	96.6 ±0.09	96.4 ±0.04	91.8 ±0.07	96.25 ±0.32	95.45 ±0.12	96.8 ±0.17	91.9 ±0.48
Lora	93.0 ±0.04	89.7 ±0.10	91.6 ±0.13	97.0 ±0.10	97.0 ±0.07	93.1 ±0.07	96.29 ±0.43	95.39 ±0.09	97.4 ±0.16	**92.1** ±0.26
Finetune	**93.7** ±0.10	**91.1** ±0.17	**92.6** ±0.11	**97.6** ±0.04	**97.5** ±0.04	**94.5** ±0.10	–	93.66 ±0.42	**97.8** ±0.06	91.4 ±0.34

Table 1: Encoder adaptation performance on downstream tasks averaged over five trials with different random seeds. All results are based on the F_1 score on the test sets. Bold: the results of best-performing models. Gray highlights: the best-performing model among the non-finetuned models, i.e., the models that preserve the base encoder frozen during training. The comparisons are based on a two-tailed t-test with p-value<0.05

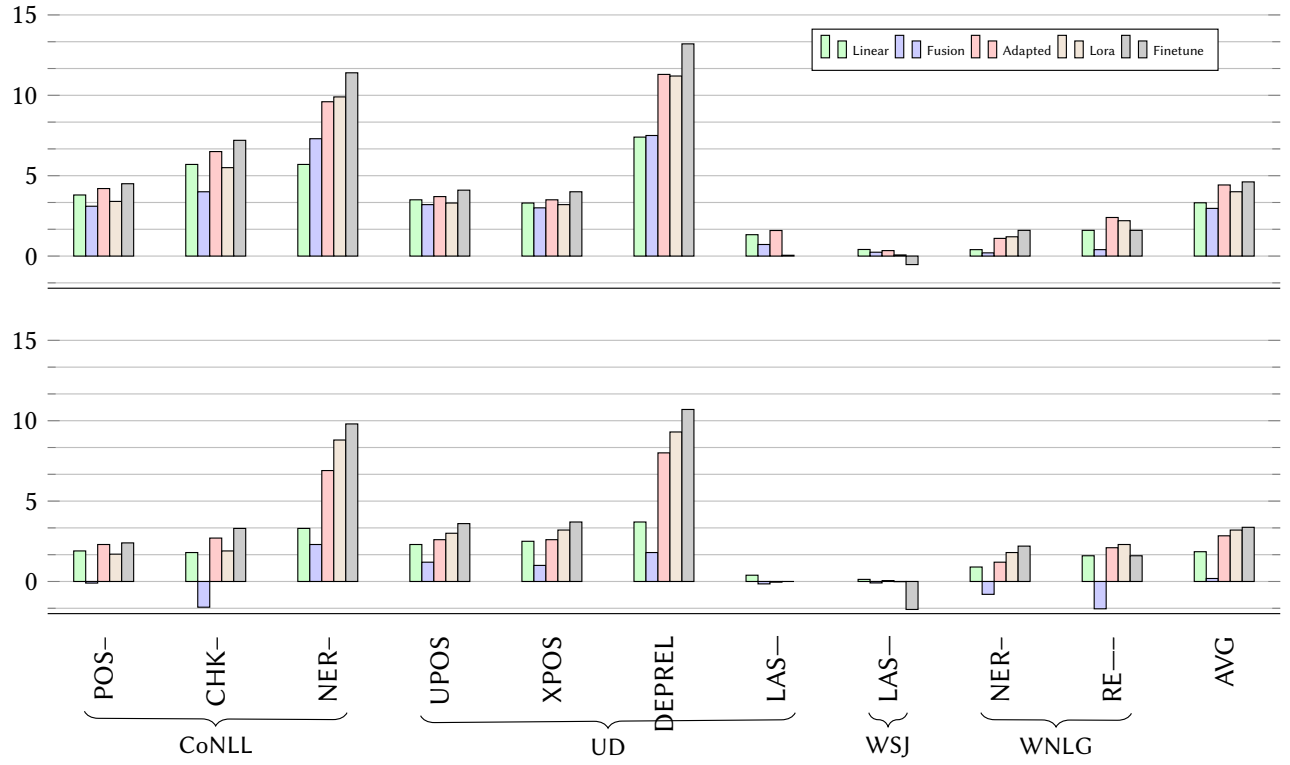

Figure 2: Absolute performance improvement (or degradation) over the frozen baseline. Top: BERT, bottom: ROBERTA.

	CoNLL-2003 (English)			UD (English-EWT)		
	POS	CHK	NER	UPOS	XPOS	DEPREL
			GPT2-small (124M)			
Frozen	90.7 ±0.06	86.5 ±0.07	68.8 ±0.27	92.6 ±0.05	92.8 ±0.04	78.6 ±0.06
Linear	91.4 ±0.29	88.7 ±0.60	63.1 ±0.09	92.9 ±0.22	93.6 ±0.44	80.4 ±0.19
Fusion	89.1 ±0.11	86.1 ±0.30	62.1 ±0.55	89.6 ±1.10	89.4 ±0.84	76.4 ±0.87
Adapted	**93.4** ±0.07	**89.7** ±0.18	**81.0** ±1.50	**94.3** ±0.06	**95.0** ±0.09	**82.8** ±0.11
Lora	92.0 ±0.06	87.6 ±0.07	70.1 ±0.19	92.8 ±0.08	93.4 ±0.05	80.7 ±0.07
			GPT2-medium (355M)			
Frozen	90.3 ±0.07	86.6 ±0.09	71.8 ±0.22	92.7 ±0.05	92.8 ±0.15	78.8 ±0.13
Linear	91.8 ±0.26	88.9 ±0.21	64.2 ±0.13	94.2 ±0.04	94.5 ±0.29	81.4 ±0.56
Fusion	90.0 ±0.32	85.0 ±2.64	70.3 ±4.60	92.7 ±0.20	92.8 ±0.22	78.4 ±0.37
Adapted	**93.5** ±0.07	**89.3** ±0.66	**80.0** ±0.74	**94.3** ±0.22	**95.2** ±0.18	**82.8** ±0.72
Lora	92.7 ±0.06	88.6 ±0.06	73.2 ±0.10	93.5 ±0.03	94.4 ±0.04	**83.3** ±0.04
			GPT2-large (774M)			
Frozen	90.8 ±0.05	86.7 ±0.08	72.5 ±0.16	93.2 ±0.06	93.2 ±0.05	79.5 ±0.10
Linear	93.2 ±0.04	89.4 ±0.15	68.5 ±0.47	94.5 ±0.07	95.2 ±0.02	83.0 ±0.06
Fusion	92.1 ±0.15	87.5 ±0.32	72.8 ±0.48	93.7 ±0.15	94.1 ±0.09	80.8 ±0.10
Adapted	**93.4** ±0.09	**89.6** ±0.10	**83.3** ±0.37	**94.7** ±0.08	**95.4** ±0.12	**83.9** ±0.12
Lora	93.0 ±0.03	89.1 ±0.09	75.7 ±0.40	93.7 ±0.04	94.7 ±0.03	**84.0** ±0.07

Table 2: The performance of the adapter techniques on GPT models. Bold numbers are significantly higher than other results in a column (two-tailed t-test with p-value<0.05).

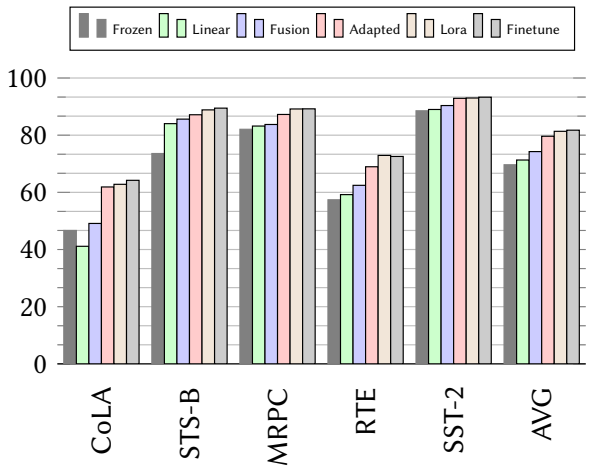

Figure 3: Document classification results based on the BERT-Large model.

than the locally trained attention weights trained of the Fusion models. The Adapted models perform comparably to Lora on the sentiment tasks (STS-B and SST-2) and the grammaticality task CoLA but weaker on other tasks. This observation indicates that tuning the internal attention weights of the encoder, as done by Lora, is more beneficial to the document classification tasks than adapting the intermediate representations, as our Adapted model considers.

5.3 Model Efficiency

The adapting techniques we consider in this study can be classified into two categories regarding their integration into the base model. The first are those techniques that chain a learning block on top of the encoder, and the second are those that are infused into the encoder. The piped models treat the base encoder as a black box and rely only on the intermediate representations produced by the encoder. However, the infused techniques must carefully modify the base architecture and insert their parameters in the encoder. In our test benchmark, the learning blocks of the Linear, Fusion, and Adapted techniques are chained onto the base encoder while the parameters of Lora are infused.

When it comes to the standard model training with backpropagation, the piped techniques are more advantageous since they allow us to perform the forward pass

only once, store the intermediate representation (i.e., the layer activations), and reuse them during training. This trick significantly improves training time at the cost of storage, which is much cheaper than the GPU cost required. However, both piped and infused techniques perform the backward pass similarly on all trainable parameters. Therefore, in order to improve training efficiency, we adopt the activation storing trick for Frozen, Linear, Fusion, and Adapted models.

Table 3 summarized the training and inference efficiency of the adapter techniques. The results are averaged over the efficiency statistics of CoNLL tagging models. Training the piped model is significantly more efficient than training the infused models. The Frozen, Linear, and Fusion models are the most efficient techniques in both computational time and trainable parameters. The Adapted models are more than 10× faster than Finetune and Lora models but 2× slower than Frozen, Linear, and Fusion models, which is due to the large number of trainable parameters it uses. The training efficiency gain of the piped models comes from the activation storing trick.

The activation storing trick is also responsible for the shorter training time in piped models compared to their inference time. As mentioned earlier, we forward every training example only once through the base encoder of these models during their first training epoch. Later, we only pass it through the adapter parameters, which are relatively smaller than the encoder. Assuming t_e as the time required for a forward pass through the encoder, t_f, and t_b as the time required for a forward and backward pass through the adapter parameters, then the average training time for a batch of sentences is $t_{\text{tr}} = \frac{t_e + e(t_f + t_b)}{e}$ for e training epochs. This fraction becomes smaller as e increases. However, the inference time for a test example is $t_{\text{inf}} = t_e + t_f$, which is larger than the average training time if the number of training epochs (e) is larger than 1. This is because $t_e + et_b < et_e$ for $e > 1$ given that $t_b \ll t_e$, which implies $t_{\text{tr}} < t_{\text{inf}}$. It is important to note that the training times reported in Table 3 are averaged over the training epochs, and the total training time is longer than the inference time.

The Adapted models perform as efficiently as the other piped models at the inference time while still being more efficient than Lora. Lora's lower inference efficiency is because of the infused attention weights distributed across all encoder layers. Although the added parameters are relatively small, they still cause a computational delay because they are coupled to the encoder's attention weights in each layer and must be processed sequentially in the same order as the encoder's layer. This latency can be improved by merging the infused weights into the encoder's attention parameters. In this case, Lora will be as efficient as the other models but at the cost of losing its reusability because it will become a large model specified for a target task. In contrast, our adaptation mechanism includes a small number of relatively large computational blocks to the encoder, enabling more efficient use of the GPU parallel processing capability.

	Train	Inference	#Params
Frozen	0.004	0.024	0.0
Linear	0.004	0.024	0.0
Fusion	0.004	0.024	0.0
Adapted	0.008	0.025	1.1
Lora	0.095	0.062	0.8
Finetune	0.106	0.024	333.6

Table 3: The model efficiency. Train/Inference: average training/inference time (seconds) for a batch of 32 sentences. The time does not include the tokenization and data loading time, which is independent of the actual training. The training time is averaged over training epochs. #Params: number of trainable parameters in each adaptation mechanism ($\times 10^6$).

6 Ablation Study

This section studies the importance of the aggregation and tailoring blocks to our adapter architecture. Figure 4 represents the improvements made by each block over the baseline Frozen models. A significant part of the improvement in the structured prediction tasks comes from the aggregation that accounts for 85% of the average improvements over frozen BERT models. However, both blocks contribute almost equally to the average improvement in document classification. This indicates the importance of the token-wised layer aggregation for structured prediction tasks that search for the interconnections between tokens. On the other hand, the tailoring block contributes significantly to the document classification tasks whose objectives differ from the encoder's pretraining objective (i.e., masked token prediction).

We also see that the necessity for tailoring in the structured prediction becomes more evident as the task complexity increases. The results suggest that the required information for syntactic tasks such as POS tagging, chunking, and parsing is already available in the intermediate representations and we only need to aggregate the information properly. However, more complex tasks that rely on both syntactic and semantic inference (e.g., DEPREL, NER, and RE) can benefit from both aggregation and tailoring blocks.

Next, we study the importance of the residual connection between the aggregation and tailoring blocks.

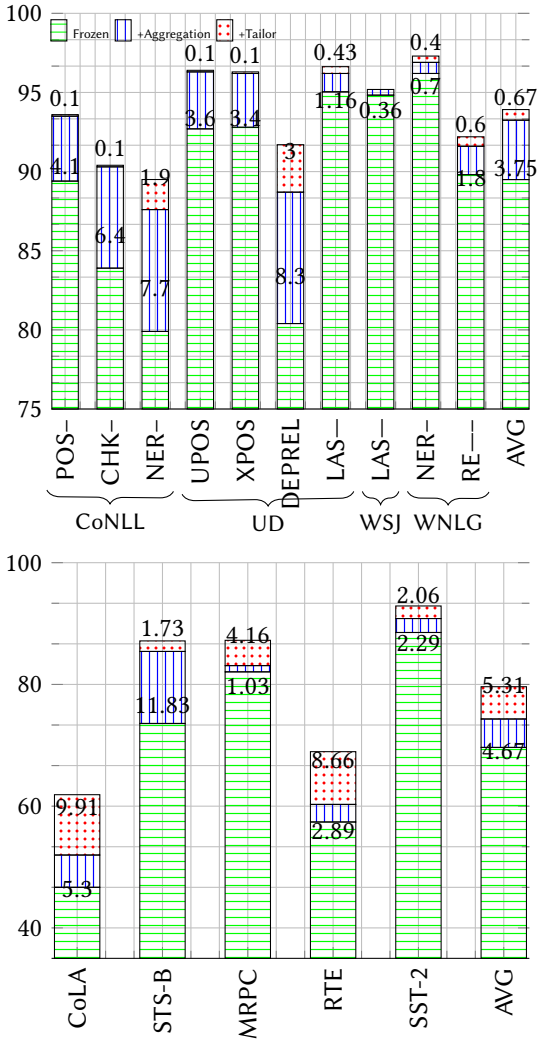

Figure 4: The accumulative contribution of the aggregation and tailoring blocks to the frozen BERT model. The adapted output is taken from the residual connector, adding the aggregator and tailor outputs.

Table 4 summarizes the base results on a subset of the structured prediction tasks. The base results in the Agg column are from the aggregator block. The other columns summarize the improvement or degradation caused by the tailoring block and the residual connection between the two blocks. First, compared with the results reported in Table 1, the aggregator block (see Column Agg) surpasses the Linear and Fusion aggregation by an average score of 0.6 and 1.9, respectively. This empirically supports our assumption about the local contribution of the intermediate representations to tokens within a task, as opposed to the token-independent aggregations of the Linear model. It also shows that our global modeling of the token-layer attention mechanism performs better than the local modeling of the Attention Fusion mechanism. Second, the tailoring block hurts the model performance when piped to the aggregation block without the residual link (see Column +Tailor). The residual information cancels out the tailor negative effect and significantly improves performance. We speculate that this is due to the controlling effect of the residual connection that lets the tailoring block affect the aggregated information only in specific contexts if needed.

	Agg	+Tailor	+Residual	Adapted
POS	93.6	0.0	0.1	0.1
CHK	90.3	−1.0	1.1	0.1
NER	87.4	0.0	1.6	1.6
DEP	95.2	−0.2	0.2	0.0
RE	91.6	−0.3	0.9	0.6
AVG	91.6	−0.3	0.8	0.5

Table 4: The performance improvement (or degradation) after adding the tailoring block and the residual link to the aggregation block. The results are based on the BERT model.

7 Conclusion

We have introduced a task adaptation mechanism for Transformer encoders to address the reusability and efficiency issues of finetuning in structured prediction. The proposed model aggregates the intermediate representations of a frozen encoder based on the input tokens and tailors them to a target task. Empirical results confirmed that the adaptation mechanism improves the training efficiency significantly while being on par with the finetuning performance. Further ablation studies confirmed the importance of both the aggregation and tailoring blocks. In future work, we want to study the adapter performance within different languages and analyze attention weights in more detail across different tasks in multilingual benchmarks.

Acknowledgements

We are grateful to the unknown reviewers who provided invaluable feedback on this work and to Marco Kuhlmann for his interest, contributions, and support for this research. We would also like to express our thanks to the Danish National Life Science Supercomputing Center for granting access to Computerome 2.0 through Project ku-00223 and to the National Supercomputer Centre in Sweden (NSC) for allowing access to Berzelius through Project Berzelius-2021-51. This project was supported by the Excellence Center at

Linköping-Lund in Information Technology (ELLIIT), Project A15.

References

Abzaliev, Artem. 2019. On GAP coreference resolution shared task: Insights from the 3rd place solution. In *Proceedings of the First Workshop on Gender Bias in Natural Language Processing*, pages 107–112, Florence, Italy. Association for Computational Linguistics.

Ács, Judit, Ákos Kádár, and Andras Kornai. 2021. Subword pooling makes a difference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2284–2295, Online. Association for Computational Linguistics.

Aghajanyan, Armen, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning.

Ba, Lei Jimmy, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Ben Zaken, Elad, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Cao, Jin, Chandana Satya Prakash, and Wael Hamza. 2022. Attention fusion: a light yet efficient late fusion mechanism for task adaptation in NLU. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 857–866, Seattle, United States. Association for Computational Linguistics.

Chen, Beiduo, Wu Guo, Quan Liu, and Kun Tao. 2022. Feature aggregation in zero-shot cross-lingual transfer using multilingual bert.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dozat, Timothy and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Gardent, Claire, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. Creating training corpora for NLG micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 179–188, Vancouver, Canada. Association for Computational Linguistics.

Guo, Demi, Alexander Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4884–4896, Online. Association for Computational Linguistics.

Hao, Yaru, Li Dong, Furu Wei, and Ke Xu. 2020. Investigating learning dynamics of BERT fine-tuning. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 87–92, Suzhou, China. Association for Computational Linguistics.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE Computer Society.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Horne, Leo, Matthias Matti, Pouya Pourjafar, and Zuowen Wang. 2020. GRUBERT: A GRU-based method to fuse BERT hidden layers for Twitter sentiment analysis. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 130–138, Suzhou, China. Association for Computational Linguistics.

Houlsby, Neil, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Hu, Edward J, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu

Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hu, Jie, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141.

Huguet Cabot, Pere-Lluís and Roberto Navigli. 2021. REBEL: Relation extraction by end-to-end language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2370–2381, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jawahar, Ganesh, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Kondratyuk, Dan and Milan Straka. 2019. 75 languages, 1 model: Parsing Universal Dependencies universally. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Li, Xiang Lisa and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Lin, Yongjie, Yi Chern Tan, and Robert Frank. 2019. Open sesame: Getting inside BERT's linguistic knowledge. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 241–253, Florence, Italy. Association for Computational Linguistics.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. RoBERTa: a robustly optimized BERT pretraining approach.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

McCloskey, Michael and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Michel, Paul, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Nivre, Joakim, Daniel Zeman, Filip Ginter, and Francis Tyers. 2017. Universal Dependencies. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, Valencia, Spain. Association for Computational Linguistics.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Peters, Matthew E., Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy. Association for Computational Linguistics.

Pfeiffer, Jonas, 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.

Pfeiffer, Jonas, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterHub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*,

pages 46–54, Online. Association for Computational Linguistics.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1.

Smith, Leslie N. and Nicholay Topin. 2017. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120.

Stickland, Asa Cooper and Iain Murray. 2019. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5986–5995. PMLR.

Su, Ta-Chun and Hsiang-Chih Cheng. 2019. Sesame-bert: Attention for anywhere.

Tenney, Ian, 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.

Tenney, Ian, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019b. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations (ICLR 2019)*.

Tjong Kim Sang, Erik F. and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

de Vries, Wietse, 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. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4339–4350, Online. Association for Computational Linguistics.

Wang, Alex, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Xiao, Zeguan, Jiarun Wu, Qingliang Chen, and Congjian Deng. 2021. BERT4GCN: Using BERT intermediate layers to augment GCN for aspect-based sentiment classification. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9193–9200, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiong, Ruibin, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR.

Yan, Zhiheng, Chong Zhang, Jinlan Fu, Qi Zhang, and Zhongyu Wei. 2021. A partition filter network for joint entity and relation extraction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 185–197, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yang, Junjie and Hai Zhao. 2019. Deepening hidden representations from pre-trained language models.

Zafrir, Ofir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, volume 5, pages 36–39.