Efficient Learning of Multiple NLP Tasks via Collective Weight Factorization on BERT

Christos C. Papadopoulos* **Yannis Panagakis***
Manolis Koubarakis* **Mihalis A. Nicolaou†**
*Dept. of Informatics and Telecommunications,
National and Kapodistrian University of Athens
christos.c.papad@gmail.com yannisp, koubarak@di.uoa.gr
†Computation-based Science and Technology Research Center, The Cyprus Institute
m.nicolaou@cyi.ac.cy

Abstract

The Transformer architecture continues to show remarkable performance gains in many Natural Language Processing tasks. However, obtaining such state-of-the-art performance in different tasks requires fine-tuning the same model separately for each task. Clearly, such an approach is demanding in terms of both memory requirements and computing power. In this paper, aiming to improve training efficiency across multiple tasks, we propose to collectively factorize the weighs of the multi-head attention module of a pre-trained Transformer. We test our proposed method on finetuning multiple natural language understanding tasks by employing BERT-Large as an instantiation of the Transformer and the GLUE as the evaluation benchmark. Experimental results show that our method requires training and storing only 1% of the initial model parameters for each task and matches or improves the original fine-tuned model's performance for each task while effectively decreasing the parameter requirements by two orders of magnitude. Furthermore, compared to well-known adapter-based alternatives on the GLUE benchmark, our method consistently reaches the same levels of performance while requiring approximately four times fewer total and trainable parameters per task.

1 Introduction

Transformer-based language models such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019b) and T5 (Raffel et al., 2020) have shown remarkable performance in many Natural Language Processing (NLP) tasks, including language understanding (Liu et al., 2019a), machine translation (Vaswani et al., 2017) and text generation (Brown et al., 2020), to mention but a few examples. Such deep language models are first pretrained on a large-scale unlabelled text dataset, and are then fine-tuned on various downstream tasks by employing labeled data. To this end, sequential transfer learning is adopted, where all the pretrained model parameters are optimized on the downstream task-specific loss. Despite the simplicity and the practical success of such a training scheme, the exponential increase in the size of the models and training data can make it difficult and costly for researchers and practitioners with limited computational resources to benefit from these models in domain or task-specific applications.

To mitigate this issue, many recent papers have focused on learning multiple tasks simultaneously by sharing most of the parameters of a given model between tasks, and introducing a task-specific classifier on top of the shared network (Liu et al., 2019a). Along with cutting down on parameter costs through the aforementioned parameter sharing scheme, multi-task learning also improves the overall performance of the pretrained model across all tasks. However, training on multiple tasks requires access to all task-specific data simultaneously, hindering the extension of such models to new domains incrementally.

An alternative approach to parameter sharing is the introduction of task specific non-shared parameters, also known as adapters, to the initial model architecture. Adapters (Rosenfeld and Tsotsos, 2020; Rebuffi et al., 2017) are typically small non-shared feedforward networks (FFN) inserted at each layer of a main network (e.g., BERT). For each task, a separate set of adapters needs to be trained. In particular, the standard practice is to freeze the main model during task training and update only the adapter parameters, introducing a huge parameter cost reduction and the ability to incrementally train a model for an arbitrary number of tasks without having access to each dataset at the same time. However, the effectiveness of adapters is subject to extensive architecture search, that includes not only the adapter structure, but also their position in the overall network architecture, as well as the evaluation of parameter reduction and performance trade-

Findings of the Association for Computational Linguistics: NAACL 2022, pages 882 - 890

offs (Houlsby et al., 2019; Pfeiffer et al., 2020a).

Motivated by the above mentioned shortcomings, we introduce a *tensor-based method for adapting a pre-trained Transformer to new unseen NLP tasks by collective weight factorization of network weights*. In particular, our method relies on collecting the Multi Head Attention (MHA) module weight matrices into a high-order tensor and represent it in Tucker format (Tucker, 1963), by applying a full-rank Tucker decomposition to it (see Fig. 1). This allows training only the decomposition factor matrices while keeping the rest of the network frozen, leading to *significant parameter cost savings without any architecture search*. In short, our contributions are the following:

- We formulate a method for efficient learning of multiple NLP tasks, utilising a Tucker decomposition of the tensor consisting of the weights of a pre-trained Transformer. Compared to adapter-based alternatives, we are able to achieve high parameter cost reductions without resorting to a costly architecture search.
- We experimentally show that our method matches adapter-based alternatives and full model fine-tuning in terms of performance by evaluating it on the GLUE benchmark. Compared to adapters, we achieve over four times less total parameters and only need to train 1% of the model's parameters for each task.
- We study the effect of applying our method to different parts of the Transformer architecture, similarly to other studies, e.g., (Lu et al., 2021; Houlsby et al., 2019; Tay et al., 2020b) . Interestingly, our results suggest that applying adaptation methods to the MHA mechanism can still perform well or even better than when applied to fully connected layers.

The organization of the rest of the paper is as follows. Section 2 discusses closely related work. In Section 3, we introduce the proposed methodology. In Section 4, we provide implementation details of our method and present experimental results. Conclusions are drawn in Section 5.

2 Related Work

This work lies at the intersection of multi-source domain adaption, while further incorporating tensor decompositions for the parametrization of the Transformer architecture.

A primary approach for fine-tuning deep learning models in the context of multi-source domain adaptation is adapter learning. Adapters were introduced in Computer Vision as a way to efficiently fine-tune a model by adding extra adapter modules to the network (Rosenfeld and Tsotsos, 2020; Rebuffi et al., 2017) and have been extended to the NLP domain as well. Specifically, involving the Transformer architecture (Vaswani et al., 2017), Stickland and Murray (2019) fine-tune the pre-trained transformer BERT (Devlin et al., 2019) with a multi-task strategy by training task-specific adapters along with the rest of the model. Houlsby et al. (2019) propose using adapters to efficiently learn multiple tasks in an incremental manner by fine-tuning only the task specific model adapters each time. Likewise, Pfeiffer et al. (2020b) adapt the pretrained multilingual model XLM-R (Conneau et al., 2020) to other languages through training language-specific adapters. In (Pfeiffer et al., 2021), a simpler architecture is presented where adapters are only added after the feed-forward part of the attention module. The placement and internal structure of adapters is non-trivial and may impact model efficiency, thus adapters require extensive experimentation before they are deployed (Pfeiffer et al., 2020a). In contrast, the method proposed in this paper is straightforward to implement, without requiring any architecture search. Additionally, the number of trainable parameters is even less than adapter-based counterparts, while performance is comparable or better.

Instead of adding extra task-specific parameters to the model as in the adapter case, a number of works aim to directly modulate the layer weights for each task. These methods are mostly based on the hypernetwork scheme (Ha et al., 2017) in which the weights of a network are generated dynamically by using another network. An alternative strategy is modulating the activations of the network layers as seen in (Perez et al., 2018) where question encodings are used to modulate the activation maps of a convolutional network. Tay et al. (2020b) use a hypernetwork scheme based on decomposable projections to create efficient multi-task transformers utilising training strategies similar to (Stickland and Murray, 2019).

In the context of deep learning, tensor methods are often used to compress a model or make its operations faster (Panagakis et al., 2021). A prominent idea in this area is reshaping the weight matri-

ces of network layers into high order tensors and decomposing them to achieve high compression rates. Similar to the decomposition of convolutional (Bulat et al., 2020) and fully connected layers (Novikov et al., 2015), Khrulkov et al. (2019) propose to compress the embedding matrices of NLP architectures by reshaping them into tensors and then use the Tensor-Train (TT) Decomposition (Oseledets, 2011) on them. The same strategy is used in (Tjandra et al., 2017) where all the weight matrices of an RNN or an LSTM unit are again decomposed into a TT format.

The main novelty behind our work lies in utilizing the Tucker decomposition in a collective weight factorization context, in order to efficiently extend pretrained Transformer models to new tasks within the NLP domain. We show that selectively grouping and decomposing specific layers (such as the MHA modules) can lead to efficient NLP task adaptation with little to no architecture search, while retaining strong performance.

3 Collective Weight Factorization for Incremental Task Learning

In this section, we present our method which is based on the collection of a pretrained network's weight matrices into a single high-order tensor. We then introduce task-specific factor matrices by decomposing the said tensor using a Tucker decomposition on the collected weight matrices and training the factor components of the decomposition via backpropagation. A visual overview of the proposed method is presented in Fig. 1.

3.1 Preliminaries

We consider learning models $h_1, h_2, \ldots, h_N$ for tasks $T_1, T_2, \ldots, T_N$ that become available in any order, with each model h_i has θ_i parameters. This is in contrast to multi-task learning where all the tasks are available at train-time, with some or all parameters being jointly trained and shared between tasks, i.e., $\theta_1 = \theta_2 = \theta_N$. Our hypothesis is that the weights of a big model trained on a generic task using huge amounts of data can be successfully utilized to fine-tune new models on new tasks, while keeping the number of newly introduced weights for training to a minimum.

Notation. We denote matrices by capital letters in italics (e.g., M) and tensors by caligraphic capital letters (e.g., $\mathcal{T}$). We use a colon to denote all the elements of a mode, e.g., $\mathcal{T}_{:,:,i_2}$ for the frontal slices of a 3-way tensor. The *mode-n unfolding* of a tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times \ldots \times I_N}$ is defined as a matrix $P_{[n]} \in \mathbb{R}^{I_n \times I_M}$ with $I_M = \prod_{k=0, k \neq n}^{N} I_k$. Finally, we define the *n-mode product* of a tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times \ldots \times I_N}$ with a matrix $M \in \mathbb{R}^{J \times I_M}$ as the result of multiplying the matrix with the mode-n unfolding of the tensor: $\mathcal{T} \times_n M = MT_{[n]} \in \mathbb{R}^{I_0 \times \ldots \times I_{n-1} \times J \times I_{n+1} \times \ldots \times I_N}$

Tucker Decomposition. The Tucker decomposition (Tucker, 1963) of an n-way tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times \ldots \times I_N}$ decomposes the tensor $\mathcal{T}$ into a core tensor $\mathcal{G} \in \mathbb{R}^{R_0 \times R_1 \times \ldots \times R_N}$ and a set of factor matrices $F^{(n)} \in \mathbb{R}^{R_k \times I_k}$ and can be regarded as a higher order Principal Component Analysis (PCA). The decomposed tensor is expressed as the n-mode product of the core tensor with the corresponding factor matrix:

$$\mathcal{T} = \mathcal{G} \times_1 F^{(1)} \times_2 F^{(2)} \times \ldots \times_n F^{(n)} \quad (1)$$

Computing the core and factor matrices (De Lathauwer et al., 2000) requires unfolding the tensor along each mode, performing Singular Value Decomposition (SVD) and storing the left singular vectors of the decomposition. The core is then computed by multiplying the initial tensor $\mathcal{T}$ with the transpose of each factor matrix:

$$\mathcal{G} = \mathcal{T} \times_1 F^{(1)^T} \times_2 F^{(2)^T} \times \ldots \times_n F^{(n)^T} \quad (2)$$

It is well known that *low-rank* Tucker decomposition can reduce parameters (Kolda and Bader, 2009). In this work however, we do not use Tucker for compression of an individual model, but rather to capture task-agnostic components of model weights that are shared amongst tasks, making finetuning to multiple tasks much more efficient.

3.2 Collective weight factorization

Throughout the rest of the paper, we consider the collective factorization of the MHA module of a transformer which consists of multiple attention modules as in (Vaswani et al., 2017). Concretely, the attention operation consists of queries and keys of dimension d_k stacked into matrices Q, K and values of dimension d_v stacked into the matrix V:

$$Att(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (3)$$

The multi-head attention is then formed by concatenating multiple attention heads and parameterized with a *hidden size* h. Each head linearly projects each Q, K, V input with matrices

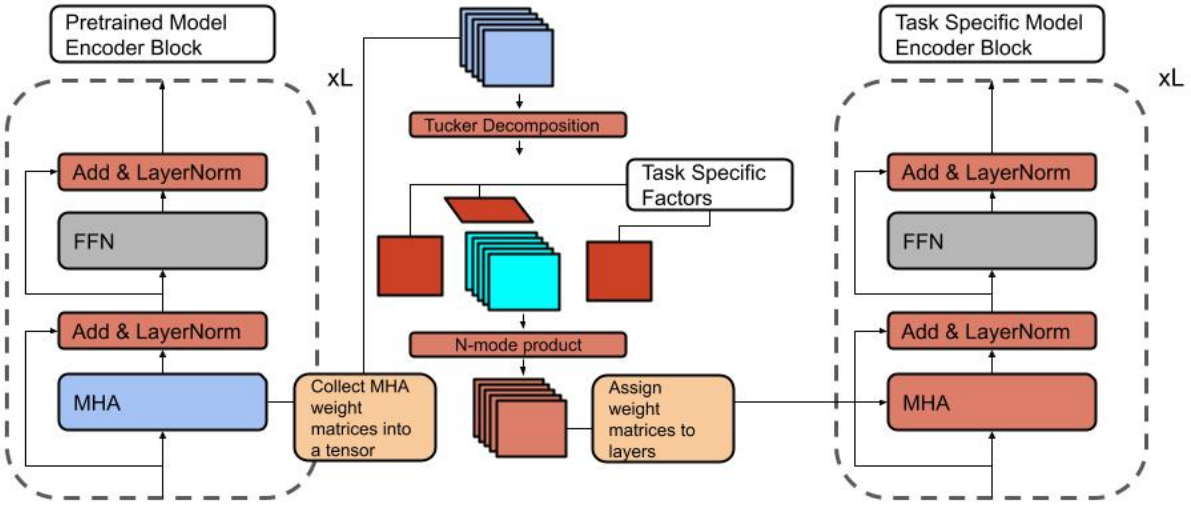

Figure 1: The MHA mechanism of each encoder block L is constructed from a slice of the n-mode product of a core tensor and n factor matrices. The core tensor along with the FFN module of each encoder block are frozen and shared between tasks. Without loss of generality, we depict the stacked weights tensor using three dimensions for easier comprehension.

$W_i^Q, W_i^K \in \mathbb{R}^{h \times d_k}$, $W_i^V \in \mathbb{R}^{h \times d_u}$. The concatenation of the different heads is then projected using a matrix $W^O \in \mathbb{R}^{n_{heads} d_u \times h}$:

$$MHA(h) = Concat(head_1, \ldots, head_{nheads}) W^O,$$
$$head_i = Att(QW_i^Q, KW_i^K, VW_i^V) \quad (4)$$

In this work, following the implementation of the BERT-Large model, we consider a Transformer built by stacking 24 identical encoder blocks, with each block consisting of two residual modules, containing a MHA and a FFN modules followed by a normalization layer (Ba et al., 2016). The MHA mechanism is modeled using four matrices, $W_Q, W_K, W_V, W_O \in \mathbb{R}^{h \times h}$ with the first three being the concatenation of the head matrices defined in (4) with $d_u = d_k = h/nheads$ and with $h = 1024$. The FFN is a 2-layer fully connected network with weight matrices $W_1 \in \mathbb{R}^{h \times 4h}, W_2 \in \mathbb{R}^{4h \times h}$ and a non-linearity between them. Since the MHA matrices are of the same dimensionality, we can construct a high order tensor by grouping each set of weight matrices in each MHA present in every layer of the transformer encoder.

Concretely, we first collect and parameterize the multi-head self-attention weight matrices $W_Q, W_K, W_V, W_O \in \mathbb{R}^{h \times h}$ of a transformer pretrained on large amounts of data into a 4-way tensor $\mathcal{W} \in \mathbb{R}^{I_0 \times I_1 \times I_2 \times I_3}$. Mode I_0 points to layers in the network, I_1 to the index of each of the 4 matrices in the MHA mechanism, and I_2, I_3 to the dimension of input and output features of the layers respectively. Specifically, for the BERT-Large model which is the basis of our experiments, $\mathcal{W} \in \mathbb{R}^{24 \times 4 \times 1024 \times 1024}$. We can then express the above tensor in Tucker form by applying a full rank Tucker decomposition to it:

$$\mathcal{W} = \mathcal{G} \times_1 F^{(1)} \times_2 F^{(2)} \times_3 F^{(3)} \times_4 F^{(4)}. \quad (5)$$

The core tensor $\mathcal{G}$ can be thought of as the collection of the principal components of the task agnostic model's weights, and it is shared among tasks. Since the decomposition is full rank, i.e., $F^{(n)} \in \mathbb{R}^{R_k \times I_k}$ and $R_k = I_k$, the factor matrices act as weight-modulating components and transform the core tensor $\mathcal{G}$ along its modes. Intuitively, the first two factors capture layer-wise and component-wise interactions between weight matrix elements with the same position. $F^{(3)}, F^{(4)}$ function as shared linear projections for all the W_i^Q, W_i^K, W_i^V attention head weight matrices. The factors scale with the hidden size h of the pretrained model since the total number of parameters for the Tucker tensor is given by:

$$N_t = \prod_{k=0}^{4} R_k + \sum_{k=0}^{4} R_k \times I_k \quad (6)$$

where $R_3, R_4 = h$. During training, we adapt the pretrained core tensor $\mathcal{G}$ to each new task S by freezing the core and training the factor matrices with the task-specific data. We do not train any other part except the final classification head of the network leading to extensive parameter savings.

Inference. Our method can be viewed as a special form of a tensor contraction layer (Kossaifi et al., 2020) where the resulting output tensor is of the same dimensions as the input. During the forward pass the core tensor G is multiplied along each mode with the corresponding task factor matrix $F^{(N)}$. The resulting weight tensor holds the weight matrices of each MHA layer in the form of slices $\mathcal{W}_{i_0,i_1,:,:}$ obtained by fixing all but the two last indices of the tensor. Once the weight tensor is formed, the slices are passed to their respective layers and inference is performed as in the standard version of the model.

4 Experimental Evaluation

In this section, we provide implementation details and discuss our experiments. Our models are based on the existing open source PyTorch implementation of a well-known Transformer model[1], while TensorLy (Kossaifi et al., 2019) is used for the tensor operations.

4.1 Implementation Details.

We construct a high-order tensor by collecting the MHA weight matrices of a pretrained BERT-Large model with $h = 1024$ and 24 layers, leading to a 4-way tensor as described in Section 3. We then decompose this tensor and use it to initialize training on each task. We then freeze all the layers except the biases, the normalization layers and the final classification layer which consists of a fully connected layer and a classification head. Classification is done as in (Devlin et al., 2019) by passing a special [CLS] token from the encoder output sequence to the final classification head.

4.2 Experiments with GLUE.

We experimented with GLUE (Wang et al., 2019) since it is a well-known benchmark utilized in the majority of previous works on multitask and incremental learning in order to assess performance. The dataset itself consists of 9 NLP text-classification tasks: CoLA (Warstadt et al., 2018) and SST2 (Socher et al., 2013) are single sentence classification tasks focusing on linguistic acceptability and binary sentiment classification respectively. MRPC (Dolan and Brockett, 2005), STS-B (Cer et al., 2017) and QQP (Quora Question Pairs) are semantic similarity tasks (STS-B is treated as a regression task with scores 1-5). The remaining tasks are all related to inference. MNLI (Williams et al., 2018) and RTE are hypothesis entailment classification tasks. QNLI (Rajpurkar et al., 2016) requires the model to predict whether the piece of text following a given question contains the answer to that question. Finally, WNLI (Levesque et al., 2012) is a reading comprehension task where the task is to predict the reference of a pronoun appearing in a sentence. Following other works (Devlin et al., 2019; Houlsby et al., 2019; Stickland and Murray, 2019), we exclude this task from our results and calculate the test set using a naive majority class label classification.

We performed a hyperparameter search to select the best learning rate, among $\{10^{-5}, 2 \cdot 10^{-5}, 3 \cdot 10^{-5}, 2 \cdot 10^{-4}, 3 \cdot 10^{-4}\}$, and batch size 16 or 32 for each task. We train for 10 epochs. Contrary to adapter-based approaches, our method is straightforward and does not require further architecture search to decide on the best size-performance trade-off and optimal adapter placement within the network. We use the Adam optimizer (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9, \beta_2 = 0.999$ and a weight decay of 0.01. Finally, we linearly increased the learning rate to the specified value for the first 10% of the training steps. All of our training runs were done using a single computing node with four NVIDIA V100 GPUs.

4.3 Collective FFN factorization.

We are interested in investigating the performance of our method when applied using the second FFN weight matrix of each encoder block and comparing it to the factorized multi-head attention variant that we presented in Sec. 3. It is noted that other works (Lu et al., 2021; Houlsby et al., 2019; Tay et al., 2020b) have suggested that fine-tuning or applying adaptation methods only to the second fully connected layer yields better results than applying the same methods to the MHA modules. In order to test this hypothesis in an multi-task adaptation setting, we utilise the strategy of folding layer weight matrices and decomposing them using tensor decompositions (Sec. 3). Using that same strategy, we can incorporate collective FFN factorization

[1] https://huggingface.co/bert-large-uncased

Model	Total Parameters	Trainable Parameters	CoLA	SST	MRPC	STS-B	QQP	MNLI_m	MNLI_{mm}	QNLI	RTE	GLUE Score
(Devlin et al., 2019)	9x	100%	60.5	**94.9**	**89.3/85.4**	**87.6/86.5**	**72.1/89.3**	**86.7**	**85.9**	**92.7**	70.1	**80.5**
(Houlsby et al., 2019)	1.3x	3.6%	59.2	94.3	88.7/84.3	87.3/86.1	71.5/89.4	85.4	85.0	92.4	**71.6**	80.2
Ours-MHA	**1.075**x	**1%**	61.6	93.6	88.6/84.5	87.1/85.8	72.1/89.1	85.7	85.4	92.3	70.9	80.3
Ours-FFN	**1.075**x	**1%**	**61.9**	93.6	86.8/82.5	84.9/83.8	71.7/89.0	85.6	84.7	92.1	67.1	79.4

Table 1: GLUE test-set results reported from the official GLUE website. We report Mathew's Correlation for CoLA, Pearson/Spearman Correlation for STS-B, F1/Accuracy for MRPC and QQP and Accuracy for all other tasks.

Task	Factorized Attention	Factorized Linear
CoLA	**64.3**	60.4
SST-2	92.2	**93.0**
MRPC	87.6	**87.8**
STS-B	**90.0**	87.6
QQP	**89.4**	88.9
MNLI	**85.8**	85.5
QNLI	91.7	**91.8**
RTE	**72.6**	70.0
Average	**84.2**	83.1

Table 2: Comparison of factorized MHA and factorized FFN methods. For tasks with two metrics we report the unweighted average. The average score is reported without including the WNLI task.

into our framework by folding the second FFN weight matrix $W_2 \in \mathbb{R}^{4h \times h}$ of each encoder block into a 3-way tensor $\mathcal{W} \in \mathbb{R}^{4 \times h \times h}$. We then collectively factorize the tensor of all the layers, similarly to the MHA case (Sec. 3). Note that even though the weight matrices of the layers were reshaped to tensors, our decomposition factors $F^{(n)}$ operate in much the same way as in the MHA attention case. We can then employ a similar training process as the one described in Sec. 4.2.

4.4 Rank Ablation.

As mentioned in Sec. 3 we opt for a full-rank decomposition rather than a low-rank one. Our hypothesis is that a low-rank decomposition would lead to the loss of useful information, hindering the effective adaptation to different tasks given our low parameter budget. We have empirically verified this hypothesis by reducing the rank of our decomposition during the training procedure initialization. Concretely, even slight dimensionality reduction (e.g. from 1024 to 768) leads to a significant decrease in performance (e.g. from 61% to 35% in the CoLA dataset/task), with performance becoming gradually worse when further reducing dimensionality.

4.5 Results and Discussion

For the first set of experiments, our results in table 1 show that our MHA variant yields competitive results on the GLUE dataset, being comparable or better than the adapter-based alternative of (Houlsby et al., 2019), albeit requiring *significantly* less total and trainable parameters. In fact, given that the large version of BERT consists of 330M parameters, our approach requires four times less total parameters for all of the GLUE tasks and, for each task, we train at most 1% of the initial model's parameters. Compared to adapters, this leads to an almost four times reduction in the number of training parameters - without relying on extensive parameter search, on which adapters are heavily reliant (Stickland and Murray, 2019; Houlsby et al., 2019). Our method does not introduce extra hyperparameters since the decomposition is always full rank, thus it is also simpler to implement and optimize.

Our second set of experiments is related to recent findings that the fully connected modules of the transformer architecture tend to be more important for a model's performance (Kitaev et al., 2020; Tay et al., 2020a). A number of works have confirmed this claim under different settings (Houlsby et al., 2019; Tay et al., 2020b; Lu et al., 2021), where either the network is trained by remaining frozen except for the FFN module or the second fully connected layer of the FFN is chosen for some weight modulation method. We also selected the second fully connected layer of the FFN module and applied our collective factorization scheme to it. However, when comparing the two variants of our method, the MHA module variant seems to consistently be the better method when it comes to both the Glue test set (table 1) and the dev set (table 2).

5 Conclusions and Future Work

We introduced a collective weight factorization method for adapting a pre-trained transformer to

multiple NLP tasks in an efficient manner. The proposed method is more efficient in terms of total parameters needed and percentage of model parameters trained and stored for each new task. Furthermore, we investigated the effect of our method when applied to different components of the Transformer. The results indicate that the multi-head attention mechanism can effectively adapt to different tasks and achieve competitive results when it is the only trained component, leading to the conclusion that more research needs to be done on the contribution of the different parts of the Transformer to its performance, and the correlation of the performance to the data used for training.

A limitation of out work is that the proposed factorization scheme operates on a single type of module, i.e. requiring same number of weight matrices and matrix dimensions. We plan to extend our method in order to be able to incorporate different types of modules (such as the FFN and MHA) in a joint factorization scheme. Taking into account recent advances in the extension of pretrained transformers to different modalities (Lu et al., 2021), one can also investigate the extension of the proposed method to adapt pretrained Transformer networks to new modalities, incorporating diverse tasks while maintaining a minimum overhead in terms of additional trainable and total parameters.

Acknowledgments

The research work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant" (Project Number: HFRI-FM17-2351). This work was also supported by a grant from The Cyprus Institute on Cyclone under project ID p055.

References

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33*.

Adrian Bulat, Jean Kossaifi, Georgios Tzimiropoulos, and Maja Pantic. 2020. Incremental multi-domain learning with network latent tensor factorization. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. 2000. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*.

Jacob Devlin, 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*.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing*.

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hypernetworks. In *5th International Conference on Learning Representations*.

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. In *Proceedings of the 36th International Conference on Machine Learning*.

Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan V. Oseledets. 2019. Tensorized embedding layers for efficient model compression. *CoRR*, abs/1901.10787.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *8th International Conference on Learning Representations*.

Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review*.

Jean Kossaifi, Zachary C Lipton, Arinbjörn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. 2020. Tensor regression networks. *Journal of Machine Learning Research*.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. 2019. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations*.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. 2021. Pretrained transformers as universal computation engines. *CoRR*, abs/2103.05247.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. 2015. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*.

Ivan V Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing*.

Yannis Panagakis, Jean Kossaifi, Grigorios G Chrysos, James Oldfield, Mihalis A Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou. 2021. Tensor methods in computer vision and deep learning. *Proceedings of the IEEE*, 109(5):863–890.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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*.

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

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020b. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Amir Rosenfeld and John K. Tsotsos. 2020. Incremental learning through deep adaptation. *IEEE*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y 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*.

Asa Cooper Stickland 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*.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking self-attention in transformer models. *CoRR*, abs/2005.00743.

Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. 2020b. Hypergrid: Efficient multi-task transformers with grid-wise decomposable hyper projections. *CoRR*, abs/2007.05891.

Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. 2017. Compressing recurrent neural network with tensor train. In *2017 International Joint Conference on Neural Networks*.

L. R. Tucker. 1963. Implications of factor analysis of three-way matrices for measurement of change. In C. W. Harris, editor, *Problems in measuring change.*, pages 122–137. University of Wisconsin Press, Madison WI.

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

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.