How to Insert Additional Layers Between the Middle Layers of the Pre-trained Model

Anonymous ACL submission

Abstract

In many deep-learning tasks, performance improvements have been achieved through the full fine-tuning of pre-trained models for downstream tasks. Numerous studies have insert an additional layer to a pre-trained model when designing a model for fine-tuning. This additional layer helps optimize the pre-trained model for downstream tasks. In some cases, this additional layer may need to be inserted between the existing middle layers of the pre-trained model . However, most studies have added an additional layer outside the pre-trained model. This is because inserting an additional layer between the pre-trained layers of a pre-trained model can cause performance degradation. In this study, we assume the following reason for the performance degradation: Initializing the additional layer using the existing initialization method with random characteristics and using the activation function changes the output value. We experimentally verified our assumptions by varying the number of additional layers and activation functions. To address this problem, we propose a methodology that initializes by unit tensor and modifies the application of the activation function. The methodology does not modify the output vector during the initial stage of full fine-tuning. We conducted experiments on the various NLP and CV datasets to verify whether the proposed methodology could solve this problem.The code used for the experiments is available on GitHub.[1]

1 Introduction

With the rapid development of deep learning technology, people have become more interested in artificial intelligence (AI) applications. AI application services have been developed in various fields, such as voice assistants, AI-based QA systems, and autonomous driving. Some of the technologies that have led to the rapid development of deep learning include transformer (Vaswani et al., 2017) and transfer learning(He et al., 2019). A transformer is a language model structure based on multi-head attention. Deep learning models such as BERT(Devlin et al., 2019), RoBERTa (Liu et al., 2019), and SwinTransformer(Liu et al., 2021) are large language models (LLMs) based on transformer structures. Numerous recent studies have used pre-trained LLMs with transfer learning.

Many of these studies performed full fine-tuning of pre-trained large language models to perform downstream tasks. As shown in model editing of (Zheng et al., 2023), an additional layer is added outside the pre-trained model to enable the deep learning model to perform the downstream task. There are various methods, such as adding a classification head to change the size of the output tensor (Devlin et al., 2019) or structuring an additional layer that is specialized for the objective (Kim and Kang, 2022). Thus, many studies using full fine-tuning have inserted additional layers outside the pre-trained model to handle downstream tasks. However, there are many possible structures of additional layers to better perform the downstream task, which may require the insertion of additional layers between the middle layers inside the pre-trained model. But inserting an additional layer between the middle layers inside a pre-trained model can cause performance degradation.

Generally, when inserting a fully connected neural network layer as an additional layer, the weights must be initialized because the additional layer is newly created. There are many different weight initialization methods. Many of them assign initialization values probabilistically according to a specific distribution. The weights initialized using these methods have randomness. Thus, before training, if weights initialized with randomness, than the results of an additional layer can also be random. Even if a well-trained input is given to the initialized additional layer, the result will have random

[1] https://anonymous.4open.science/r/Unit_Initalize_and_Weight_Activation-1FE9

characteristics that are likely to be untrained vectors in the pre-training process. Therefore, inserting an additional layer between the middle layers inside a well-trained pre-trained model can act as noise at the pre-trained model.

For example, suppose there is a transformer-based pre-trained language model P. Let P_n be each pre-trained transformer block in the pre-trained model. If we insert a randomly initialized additional layer between P_3 and P_4, even if P_3 results in a well-trained vector, P_4 will get a vector with randomness due to the additional layer. This implies that P_4 will get vectors that are unrelated to the pre-trained information that was learned during the pre-training process. Therefore, during the full fine-tuning of a pre-trained model, inserting an additional layer between the middle layers inside the pre-trained model can noise in the pre-trained model. Moreover, if this noise accumulates in the early stages of full fine-tuning, it may cause the pre-trained information to be lost during full fine-tuning. Experiments and results that verify these issues are presented in Section 4.3.

Additionally, the activation functions applied to the additional layers can act as noise in the pre-trained model. Typically, an activation function is applied to the output of a fully connected neural network. The activation function causes a change in the vector value; therefore, even if the additional layer is well-trained and contains valuable pre-trained information, the activation function can cause a change the value. This can be a problem in full fine-tuning if the newly inserted activation function changes the vector values between the middle layers of the pre-trained model.

This study introduces a methodology to prevent problems caused by inserting additional layers between the middle layers of a pre-trained model. To address this issue, we propose a methodology that inserts additional layers in the early stages of full fine-tuning, where the initialized additional layers do not act as noise to the pre-trained information. To the best of our knowledge, this is the first study to reliably train an additional layer inserted between the middle layers of a pre-trained model in full fine-tuning.

2 Related Work

2.1 Transfer Learning

Transfer Learning. Transfer learning uses the knowledge and information from a model trained for specific purposes to solve problems in other tasks. (He et al., 2019) experimentally demonstrated that the knowledge learned by a deep learning model for a specific task can be transferred to other tasks. In the early days, transfer learning was primarily used in computer vision. Utilizing models pre-trained on large datasets for tasks such as fine-grained image classification (Yuan et al., 2020), segmentation (Long et al., 2015), and detection (Girshick et al., 2014) has shown notable efficacy in various computer vision applications.

Pre-trained Language Model. Many recent studies in the field of natural language processing have used pre-trained language models. Pre-trained language models train a text representation with general characteristics during the pre-training process, and it is common to perform self-supervised learning using a large number of unlabeled corpora to train the text representation. For the training objective of dictionary learning, masked language modeling is used, which masks some tokens in the text and then reconstructs the masked token(Devlin et al., 2019).

Fine-tuning of Pre-trained Language Model. A pre-trained language model trained with general information through pre-training is then fine-tuned using the dataset from a downstream task. Several methods have been considered to effectively fine-tune the models. First, a separate layer was added to the pre-trained language model. For example, SEQSON(Wang et al., 2022) added a single-layer linear classifier on top of the encoder of the pre-trained model to estimate the important parts of the original document in an abstract summarization task. DyLex(Wang et al., 2021) uses word-agnostic tag embedding and contextual embedding of pre-trained models as an additional layer in the sequence labeling task. There are also studies that change the fine-tuning process.

2.2 Weight Initialization

Setting the initial weights is critical when training a deep-learning model. Incorrect initial weights can cause various problems, such as gradient vanishing, which increases the likelihood of convergence to a local minimum. Therefore, considerable research has been conducted on weight initialization methods. In general, the most popular initialization methods used in deep learning models are Xavier initialization (Glorot and Bengio, 2010) and He initialization (He et al., 2015). Xavier initialization (Glorot and Bengio, 2010) initializes the

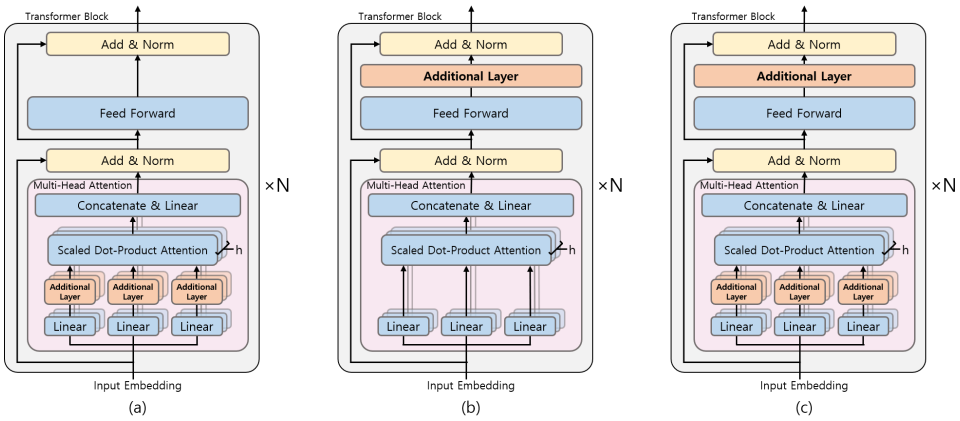

Figure 1: Model structure with additional layer. (A) is model structure of **In multi-head self-attention**. (B) is model structure of **After feed-forward network**. (C) is model structure of **Both**.

weights to follow a normal distribution, with the standard deviation determined by the number of nodes in the previous and next layers. He initialization (He et al., 2015) addresses the limitations of Xavier initialization, offering enhanced efficiency for ReLU(Agarap, 2018) activations.

3 Proposed Method

The proposed method aims to insert an additional layer between the middle layers inside the pre-trained model. When inserting an additional layer, we construct an additional layer such that the inserted layer does not act as noise to the pre-trained information. To insert an additional layer into a traditional deep learning methodology, the additional layer and activation function are structured and weight initialized. For example, if the traditional method uses a fully connected neural network as an additional layer, the structure of the additional layer is as follows: The additional layer operates on learnable parameters W, b $(WX + b)$. The activation function is applied to the output of layer $WX + b$ $(\alpha(WX + b))$. Here, W and b denote the learnable parameter weight and bias, respectively, X is the input, and α is the activation function. In addition, W and b are initialized by an initialization methods such as the Xavier Initialization and He Initialization.

In this study, we assume that the initialization method with randomness and the activation function applied to the output cause the loss of pre-trained information. To solve this problem, we construct an additional layer in the early stage of full fine-tuning step that does not harm the existing pre-trained information. The method to prevent the problem is described in detail in Section 3.2.

3.1 Position of Additional Layers

In this section, we describe the position of the additional layer inserted for the experiment before explaining the proposed methodology. The model structure used for the experiments is illustrated in Figure 1. An additional layer is inserted between the middle layers of the transformer blocks. In this study, we used three model structures, and additional layers were inserted at the following positions: 1. only in multi-head self-attention and 2. only after the feed-forward network, and 3. both.

In Multi-head Self-attention. The model structure with an additional layer inserted at the position in the multi-head self-attention is shown in Fig. 1-A. An additional layer A is positioned after the feed-forward network of generate Q_i,K_i, and V_i in multi-head self-attention. where i denotes the number of heads. We input Q_i, K_i, and V_i into an additional layer A to generate new Q'_i, K'_i, V'_i. Subsequently, Q'_i, K'_i, and V'_i are used to perform multi-head self-attention. The equation for this process is as follows:

$$Q_i = W_i^Q E \tag{1}$$
$$K_i = W_i^K E \tag{2}$$
$$V_i = W_i^V E \tag{3}$$

$$Q'_i = A_i^{Q'}(Q_i) \tag{4}$$
$$K'_i = A_i^{K'}(K_i) \tag{5}$$
$$V'_i = A_i^{V'}(V_i) \tag{6}$$

$$Attention(Q', K', V') = Softmax(\frac{Q'K'^T}{\sqrt{d_{K'}}})V' \tag{7}$$

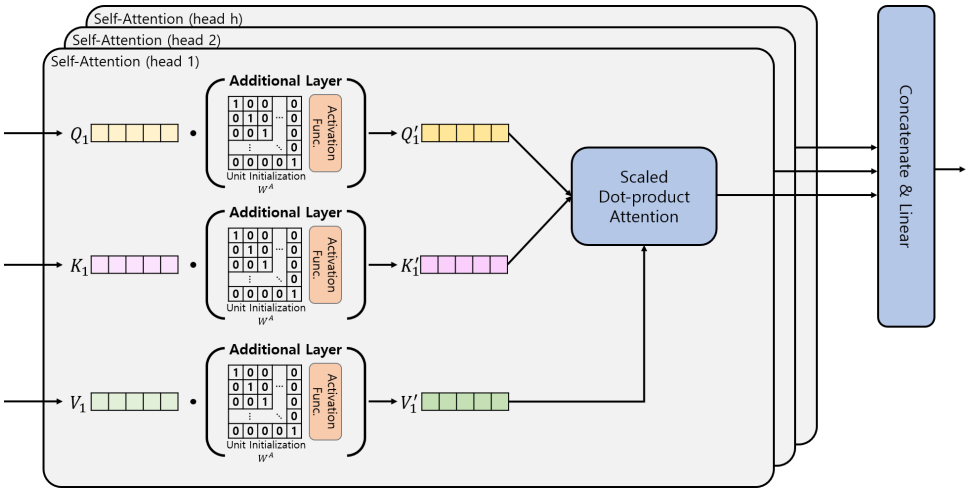

Figure 2: Additional layer structure when layer position is In multi-head self-attention.

$$head_i = Attention(Q'_i, K'_i, V'_i) \quad (8)$$

$$MultiHead(Q', K', V') = Concat(head_1, head_2, ..., head_h)W^O \quad (9)$$

These equations are based on the multi-head attention equations of the transformer(Vaswani et al., 2017). In the equation, E denotes the input embedding and W_i^Q, W_i^K, W_i^V, and W^O are the learnable parameters of multi-head self-attention in the pre-trained model. i is the head index and h is the number of heads. Additionally, $A(\cdot)$ denotes an additional layer, and the structure of A is described in Section 3.2.

After Feed-forward Network. The position of the additional layer in After feed-forward network is shown in Fig 1-B. Additional layers are inserted after the feed-forward network in the transformer blockThe equation for this is as follows:

$$FFN'(x) = A(max(0, xW_1 + b_1)W_2 + b_2) \quad (10)$$

This equation is based on the transformer (Vaswani et al., 2017): The feed-forward network of transformer block experimented with $FFN'(x)$, which is an additional layer A added to the formula $FFN(x)$ in the Transformer paper(Vaswani et al., 2017). W_1, W_2, b_1, and b_2 are learnable parameters of the pre-trained model. $A(\cdot)$ denotes the additional layer.

Both refers to inserting an additional layer in both the multi-head self-attention and after feed-forward neural network. The corresponding structure is shown in Fig 1-C.

3.2 Insert Additional Layer

The proposed method involves inserting an additional layer without harming the pre-trained information, that is, the additional layer does not act as noise in the initial stage of full fine-tuning. We assumed that the transformations of the output vector by the additional layer in the early stages of full fine-tuning were acted as noise in the pre-trained model. In existing pre-trained models, each pre-trained layer receives a well-trained input and passes the well-trained output to the next pre-trained layer. However, if an additional layer is inserted during this process, the next layer receives the value of the transformed output, rather than the well-trained output of the previous pre-trained layer.

The proposed method focuses on two reasons why the output vector changes when an additional layer is inserted: the initialization method of randomly initializing the weights, and the activation function applied to the output. An initialization method with random characteristics may change a well-trained output vector to an output vector with random characteristics. Applying an activation function to the output of an additional layer can change the value of the well-trained output vector. If the output vector between each pre-trained layer is modified in the early stages of full fine-tuning, the pre-trained information can be loss. If more additional layers are inserted, the problem of not using the pre-trained information accumulates, and the pre-trained information may be lost during the fine-tuning process. This was experimentally verified in Section 4.3. Therefore, the proposed method prevents the output of the pre-trained layer

from having random characteristics and prevents the well-trained output from being modified by the activation function. The overall structure of the proposed methodology is illustrated in Figure 2. Figure 2 illustrates the case where an additional layer is added to the in multi-head attention position.

Unit Initialization. In this study, the learnable parameters of the additional layer are initialized using a unit tensor. A unit tensor (matrix) has the property of preserving the value of the input in thea dot product. If the dot product a tensor Y and the unit tensor U, the output is Y ($Y \cdot U = Y$). Therefore, we initialize the learnable parameter of the additional layer W^A as a unit tensor. If the additional layer is initialized as a unit tensor, the next pre-trained layer will not receive a vector of untrained random characteristics as input during the early stages of full fine-tuning. This allows the pre-trained information to be used in the early stages of full fine-tuning. Maintaining the pre-trained information in the early stages of full fine-tuning prevents the pre-trained information from being lost during training. The weight W_0^A of the additional layer is initialized as follows:

$$W_0^A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{11}$$

Weight Activation. The additional layer in the proposed method does not apply an activation function to the output. The proposed methodology applies an activation function to the learnable parameter W^A in an additional layer. GeLU(Hendrycks and Gimpel, 2023) was used as an activation function . When the GeLU activation function is applied to the unit tensor, a tensor similar to the unit tensor is obtained. Similarly, even if the activation function is applied in the initialization phase of the additional layer, the value changes slightly. In the initial stage of full fine-tuning, it can mitigate the additional layer acting as noise in the pre-trained information. It also prevents the loss of pre-trained information after full fine-tuning is completed. The structure of additional layer A is as follows:

$$A(x) = \alpha(W^A)x \tag{12}$$

$x \in R^{d^x}$ denotes the input vector for the additional layer. d^x is the dimension of input vector x. $W^A \in R^{d^x \times d^x}$ is a learnable parameter of the additional layers. $\alpha(\cdot)$ denotes the activation function.

4 Experiment

4.1 Experiment Setup

In this paper, we conduct experiments on various NLP and CV datasets to verify the performance of the proposed methodology. The experimental datasets are GLUE benchmark (Wang et al., 2018), CNN/DM (See et al., 2017), WMT16 (Bojar et al., 2016), ImageNet(Deng et al., 2009), and CIFAR100(Krizhevsky, 2009). A detailed description of each dataset is given in Appendix A.

In this paper, we use the pre-trained model without additional layers as a baseline. We also conduct a comparison experiment with a pre-trained model with Insert Additional Layer without Proposed Method. This comparison experiment verifies that the proposed methodology can solve the issues in this paper. We also conduct comparative experiments with the freeze pre-trained model, which is one of the ways to adapt the additional layer in fine-tuning. Freeze Pre-trained Model only learns the additional layer without the parameters of the pre-trained model. The models used in each experiment are described in Appendix B.

4.2 Experiment Result

table 1 is the result of GLUE benchmark(Wang et al., 2018) experiment on DeBERTaV3$_{large}$(He et al., 2021). The performance without Proposed Method is considerably low compared to the baseline. For CoLA dataset, the performance was as low as 12.89. In addition, the performance of STS-B's In multi-head attention was 47, which is about half of the baseline. Most of the results demonstrate extremely low performance, confirming that full fine-tuning is difficult in this case. This trend is also seen in other datasets that use accuracy as an evaluation metric. The subtasks SST-2, MRPC, QNLI, RTE, and WNLI iare binary classifications. Therefore, an accuracy of 50% is obtained in the untrained case. On many case, the performance was close to 50%, confirming the difficulty of training when without proposed methods.

Freeze Pre-trained Model is an result of fine-tuning only additional layer without pre-trained model parameter. For most datasets, there was no improvement in performance. However, some datasets show a clear performance improvement compared with the insertion of an additional layer

Model	COLA Mcc	SST-2 Acc	MRPC Acc	STS-B Corr	QNLI Acc	RTE Acc	WNLI Acc
DeBERTaV3$_{large}$	75.87	95.87	90.93	92.97	93.61	91.34	91.54
Insert Additional Layer without Proposed Method							
DeBERTaV3$_I$	0	50.92	70.10	47.07	50.54	52.70	56.34
DeBERTaV3$_A$	0	50.92	68.38	34.13	50.54	52.70	57.75
DeBERTaV3$_B$	12.89	82.56	71.08	31.30	50.54	56.68	56.34
Freeze Pre-trained Model							
Adapter$_I$	0	89.44	70.83	22.00	64.34	57.76	56.34
Adapter$_A$	5.29	87.50	72.30	80.86	83.31	57.40	60.56
Adapter$_B$	14.23	85.78	71.32	24.74	63.26	55.60	56.34
with Proposed Method							
Proposed$_I$	75.85	95.99	91.67	93.28	93.78	91.70	92.95
Proposed$_A$	73.03	96.10	91.91	92.84	93.74	91.70	95.77
Proposed$_B$	74.28	95.53	92.16	92.94	93.65	91.70	94.36

Table 1: GLUE benchmank performances with DeBERTaV3$_{large}$. I,A and B is position of additional layer. I is In multi-head attention, A is After feed forward network and B is Both

Model	CNN/DM ROUGE-1	CNN/DM ROUGE-2	CNN/DM ROUGE-N	WMT16 en-ro BLEU
T5$^*_{base}$	42.05	20.34	39.40	28.0
T5$_{base}$	42.60	20.15	39.58	26.45
Insert Additional Layer without Proposed Method				
T5$_I$	39.72	18.30	36.74	24.41
T5$_A$	41.66	19.39	38.53	24.88
T5$_B$	36.84	16.24	33.91	18.30
with Proposed Method				
Proposed$_I$	42.44	20.15	39.37	26.69
Proposed$_A$	43.20	20.70	40.15	26.79
Proposed$_B$	42.21	19.85	39.23	26.72

Table 2: Text Generation performances with T5$_{base}$. * is the performance in the T5 paper(Raffel et al., 2020).

without Proposed Method like SST-2. However, the performance was lower than that of the baseline, confirming that the issue could not be solved.

The performance when using the proposed method is with Proposed Method. The performance of with Proposed Method is similar to or higher than baseline. For the COLA, the performance was lower than that of the baseline in all cases with Proposed Method. However, in contrast to the case without Proposed Method, the performance was similar to that of the baseline. In most cases, the performance was higher than the baseline. We suspect that the performance improvement is due to the success of full fine-tuning of the larger model by inserting an additional layer. We also performed experiments on T5$_{base}$ and the results are in appendix C.

table 2 is the performance evaluation of CNN/DM(See et al., 2017), WMT16 (Bojar et al., 2016) on T5$_{base}$(Raffel et al., 2020). The results show that T5$_{base}$ is relatively robust to additional layers. However, there is a clear performance degradation on CNN/DM and WMT16 without the proposal methodology. When using the proposal methodology, the performance was higher than the baseline. This confirms that the proposal methodology can prevent the performance degradation caused by the additional layer in text generation.

The proposed method was experimented using a pre-trained foundation model for the CV task. Table 3 presents the experimental results on ViT$_{base}$(Dosovitskiy et al., 2021). The performance of Insert Additional Layer without Proposed Method was approximately 60% accurate. So, In

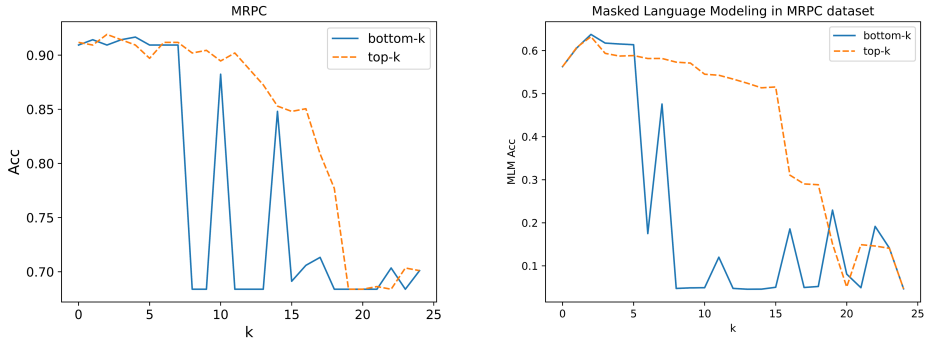

Figure 3: Performance graph of number of Additional layer in MRPC.

Model	ImageNet Acc	CIFAR100 Acc
ViT^{*}_{base}	83.97	91.67
ViT_{base}	83.99	92.81
Insert Additional Layer without Proposed Method		
ViT_{I}	66.85	61.55
ViT_{A}	81.01	82.18
ViT_{B}	53.72	54.69
with Proposed Method		
Proposed_{I}	84.28	92.62
Proposed_{A}	84.09	92.54
Proposed_{B}	84.22	92.77

Table 3: Image Classification performances with ViT_{base} . * is the performance in the ViT paper(Dosovitskiy et al., 2021).

the case of ViT_{base}, we found that training was some possible even if an additional layer was included. However, the performance was considerably poor compared to that of the baseline. In most cases, the proposed method outperformed the baseline method. This confirms that the proposed methodology can improve the loss of pre-trained information even in a CV's pre-tarined foundation model.

4.3 Performance Degradation by Additional Layer

This study assumes that the additional layer inserted into the middle layer inside the pre-trained model can be noise to the pre-trained information. So, we experimentally verified that the performance degradation is caused by the additional layers. We experimented with the performance of the downstream task as a function of the number of additional layers without using the proposed methodology. In this experiment, each additional layer used the Xavier initialization, and GELU was applied to the output. We ran two experiments: one with the top-k additional layers of the total N transformer blocks, and one with the bottom-k additional layers. In this experiment, we used DeBERTaV3_{large}, thus N was 24.

Fig 3 visualizes the downstream task performance according to the number of additional layers in MRPC. From the graph, it is the performance decreases proportionally to the number of additional layers. This result shows that the additional layers without using the proposed method can act as noise in the pre-trained model, and the performance decreases as the noise increases. Additionally, the performance at bottom-k is lower than the performance at top-k when k is the same. We suspect that noise causes more performance degradation in the early layers of the pre-trained model.

We also experimented with the same task as in pre-training. We evaluated the performance of the masked language model(MLM) after train MLM of MRPC dataset. The results showed that MLM accuracy decreased as k increased. This confirms that the additional layer can cause the loss of pre-trained information. To clarify, we conduct the same experiment on CoLA dataset, which is described in the appendix D.

4.4 Ablation Study

The performance was compared with and without the proposed methodology according to the number of additional layers. fig 4 show the performance when an additional layer is inserted into the bottom-k transformer blocks. When full fine-tuning of the pre-trained model, the performance degrades with the number of additional layers without using the proposed method. However, when the proposed method was used, the performance did not decrease as the number of additional layers increased. Suc-

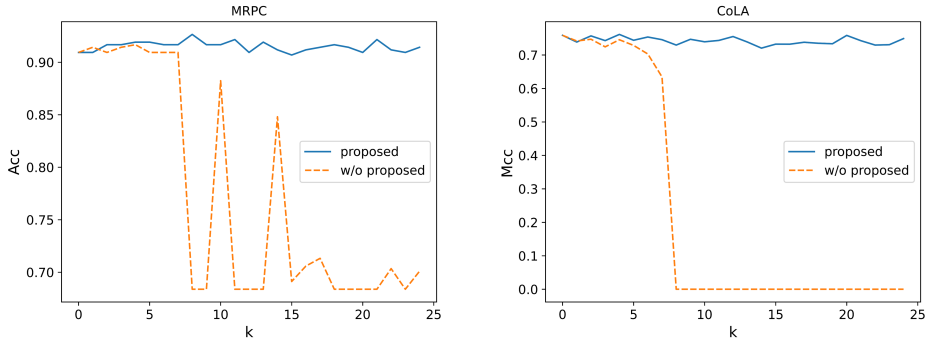

Figure 4: Performance graph of number of additional layer about with or without proposed method. In this graph, the additional layer was inserted into the bottom-k transformer block of the pre-trained model.

Activiation Position	MRPC Acc	CoLA Mcc
No Activation	91.67	73.33
Output Activation	72.06	65.24
Weight Activation	91.67	75.85

Table 4: Performance according to the location of the activation function.

Activiation Position	MRPC Acc	CoLA Mcc
Xavier Initialization	72.55	12.71
Unit Initialization	91.67	85.85

Table 5: Performance according to the Initialization method.

cessful full fine-tuning was achieved regardless of the number of additional layers, with some instances even showing performance improvements. This demonstrates that the proposed methodology effectively prevents performance degradation caused by additional layers.

Table 4 compares the performance of the position of the activation function in the additional layer. In this experiment, an additional layer was inserted in the In multi-head attention position, and the weight was initialized as a unit tensor. The performance degradation occurred when the activation function was applied to the output. The additional layer without activation function showed similar or lower performance than the proposed methodology. For the proposed method, it was possible to prevent the performance degradation caused by output activation.

Table 5 compares the performance of different initialization methods for the additional layer. In this experiment, an GeLU activation is applied to the weight. Existing initialization methods, which use values with random characteristics, lead to performance degradation during full fine-tuning. The proposed methodology successfully prevents this degradation.

5 Conclusion

In this study, we propose a methodology for the full fine-tuning of pre-trained models with additional layers. We experimentally verified that the pre-trained information can be lost when inserting additional layer between middle of pre-trained layer. We assume that the reason for this problem is that the values change between the pre-trained layers by randomly initializing the weights and applying an activation function to the output. To address this issue, we initialized the weight with a unit tensor and applied an activation function to the weight. To verify the performance of the proposed method, we performed experiments on the GLUE benchmark, text generation and image classification. The results show that the performance of the proposed method improved in most cases, and even in cases where the performance decreased, it was similar to that of the baseline.

There are several possible models for processing the different downstream tasks. Some of these cases may require inserting an additional layer in the middle of a pre-trained model. In this study, we verified that the proposed methodology can achieve full fine-tuning when an additional layer is inserted in the middle of a pre-trained model. Therefore, we expect to develop more diverse fine-tuned models using the proposed methodology.

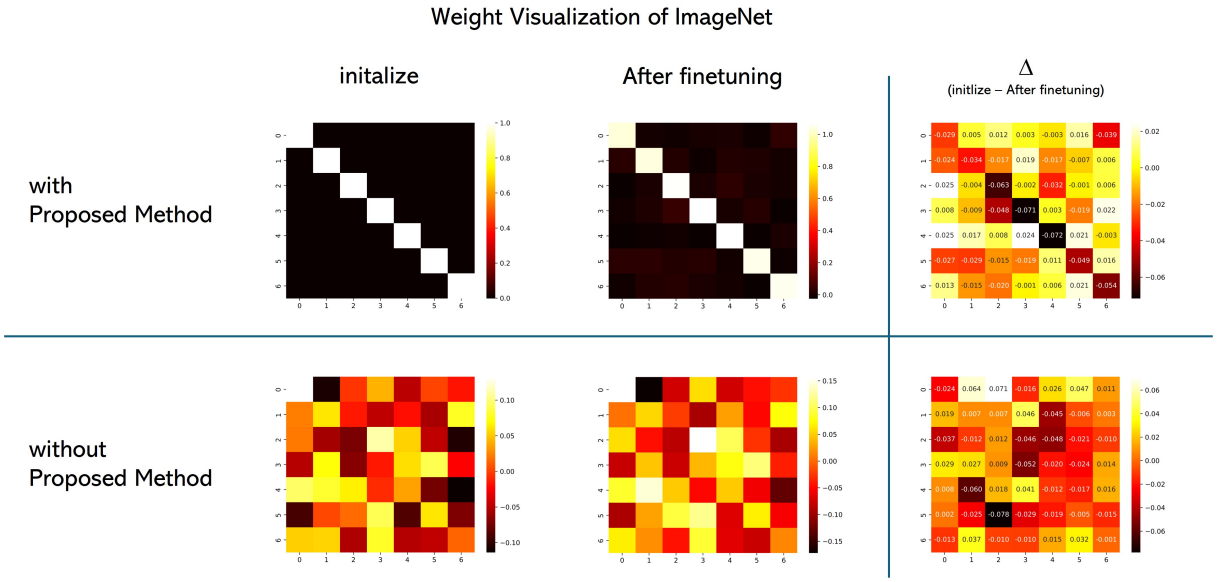

Figure 5: Visualization of additional layer weight of ImageNet. For the visualization, we only used weights up to 7*7 tensor.

Limitation

In this study, we propose a method in which the additional layer inserted in the middle layer of the pre-trained model is not applied as noise in full fine-tuning. Therefore , we prevented the loss of pre-trained information by applying it as noise in the full fine-tuning process. However, the proposed method has certain limitations.

First, the tendency of the unit tensor even after training is shown in Figure 5. For initialization methods with random characteristics, there was no specific tendency in the trained layer. However, in the proposed methodology, the tendency of the unit tensor is visible even after learning because it is initialized with a unit tensor. The Δ of Figure 5 shows that both methods can be trained. Also, the amount of trained value is similar. However, even after training with proposed method, the value for the diagonal line is relatively high, similar to the unit tensor. This may be undesirable for deep learning. In the future, it will be necessary to study initialization methods to reduce this tendency.

In addition, the proposed method uses the property of a unit tensor of value does not change during the dot product. A fully connected neural network dot product an input tensor and weight for the result. Therefore, the proposed methodology initializes the weight of a fully connected neural network as a unit tensor. However, a CNN performs an elementwise product of the kernel. Therefore, this methodology cannot be applied to CNN kernels. The proposed methodology cannot be used in deep learning models that do not use dot products. Therefore, future research needs to investigate methodologies that can be applied to different deep learning models.

References

Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

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 (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Kaiming He, Ross Girshick, and Piotr Dollar. 2019. Rethinking imagenet pre-training. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4917–4926.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing.

Dan Hendrycks and Kevin Gimpel. 2023. Gaussian error linear units (gelus).

N. Houlsby, A. Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and S. Gelly. 2019. Parameter-efficient transfer learning for nlp.

Edward J Hu, 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*.

Gyunyeop Kim and Sangwoo Kang. 2022. Effective transfer learning with label-based discriminative feature learning. *Sensors*, 22(5).

Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. pages 32–33.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47.

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

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022.

J. Long, E. Shelhamer, and T. Darrell. 2015. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, Los Alamitos, CA, USA. IEEE Computer Society.

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

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*, 21(140):1–67.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Ashish Vaswani, 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.

Alex Wang, 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.

Baojun Wang, Zhao Zhang, Kun Xu, Guang-Yuan Hao, Yuyang Zhang, Lifeng Shang, Linlin Li, Xiao Chen, Xin Jiang, and Qun Liu. 2021. DyLex: Incorporating dynamic lexicons into BERT for sequence labeling. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2679–2693, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Fei Wang, Kaiqiang Song, Hongming Zhang, Lifeng Jin, Sangwoo Cho, Wenlin Yao, Xiaoyang Wang, Muhao Chen, and Dong Yu. 2022. Salience allocation as guidance for abstractive summarization. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6094–6106, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing.

Y. Yuan, W. Chen, Y. Yang, and Z. Wang. 2020. In defense of the triplet loss again: Learning robust person re-identification with fast approximated triplet loss and label distillation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1454–1463, Los Alamitos, CA, USA. IEEE Computer Society.

Hongling Zheng, Li Shen, Anke Tang, Yong Luo, Han Hu, Bo Du, and Dacheng Tao. 2023. Learn from model beyond fine-tuning: A survey.

A Dataset

We experimented with NLP and CV to evaluate the performance of the proposed methodology. The proposed methodology inserts an additional layer into the pre-trained model; therefore, we used the dataset to evaluate the performance of the pre-trained model. In NLP, we used the GLUE benchmark (Wang et al., 2018) to evaluate the performance of the proposed methodology in a transformer-based language model. The GLUE benchmark was used to test the natural language understanding ability of a language model. Used dataset of GLUE benchmark are listed in Table 6.

To evaluate the generation performance of the proposed methodology on NLP transformer encoder-decoder-based language models, we used CNN/DM (See et al., 2017) and WMT16 (Bojar et al., 2016). CNN/DM measures the performance of abstract text summarization using a pre-trained language model. WMT16 evaluated the translation performance of English to Romanian using a machine translation dataset.

CV uses image classification datasets to measure the performance of a pre-trained foundation model. The ImageNet(Deng et al., 2009), and CIFAR100(Krizhevsky, 2009) datasets were used. The information for each dataset is table 6.

B Comparison Method

Baseline. In this study, we propose a methodology that inserts an additional layer for the pre-trained model to learn without losing pre-trained information. Experiments of the proposed methodology were conducted using NLP and CV pre-trained models. For NLP, we experimented with DeBERTaV3$_{large}$(He et al., 2021) and T5$_{base}$(Raffel et al., 2020). We also compared the results with those of ViT$_{base}$(Dosovitskiy et al., 2021) for the CV experiments. To evaluate the performance of the proposed methodology, we compare performance with that of the pre-trained model without the additional layer. For a fair evaluation, we experimented with the baseline ourselves.

We also compared the performance of the pre-trained model with an additional layer without unit initialization and weight activation. A comparison with and without the proposed methodology confirmed that it can solve the problem of noise in pre-trained information. The additional layer without the proposed methodology initializes the weights using the Xaver initialization (Glorot and Bengio, 2010) and applies an GELU(Hendrycks and Gimpel, 2023) activation function to the output of the layer. The additional layer is added to the transformer block of the bottom k (k is more than 80% of the total number of transformer blocks) of the

Corpus	Task	Train	Dev	Label	Matrics
GLUE					
COLA(Warstadt et al., 2018)	Sentence acceptability judgment	8.5k	1k	-	Mattews corr
SST-2(Socher et al., 2013)	Sentiment analysis	67.3k	0.9k	2	Accuracy
MRPC(Dolan and Brockett, 2005)	Paraphrasing/sentence similarity	3.7k	0.4k	2	Accuracy
STS-B(Cer et al., 2017)	Paraphrasing/sentence similarity	5.6k	1.5k	-	Pearson corr
QNLI(Levesque et al., 2011)	Natural language inference	105k	5.5k	2	Accuracy
RTE(Wang et al., 2018)	Natural language inference	2.5k	0.3k	2	Accuracy
WNLI(Levesque et al., 2011)	Natural language inference	0.6k	0.2k	2	Accuracy
Text Generation					
CNN/DM	Text summarization	287k	11.5k	-	ROUGE
WMT16 en-ro	Translation	610k	2k	-	BLEU
Image Classification					
ImageNet	Image classification	1.2m	50k	1000	Accuracy
CIFAR100	Image classification	50k	10k	100	Accuracy

Table 6: Summary information about experimental dataset

Hyper-parameter	GLUE	Text Generation	Image Classification
Warmup Steps	{50, 500}	-	-
Learning Rate	{1.5e-5, 3e-5, 5e-5, 1e-4}	{1e-3, 1e-4}	{3e-2, 1e-2}
Batch size	16 or 8	128	32
Epochs/Steps	15	15000 steps	ImageNet:5 / CIFAR100:30
Optimizer	AdamW	AdamW	SGD
Weight Decay	0.01	0.01	0
AdamW ϵ	1e-6	1e-6	-
AdamW β_1, β_2	0.9,0.999	0.9,0.999	-
SGD momentum	-	-	0.9
Image Resolution	-	-	384
Gradient Cliping	1.0	-	1.0
Scheduler	linear	-	cosine
Pre-trained model	DeBERTaV3$_{large}$,T5$_{base}$	T5$_{base}$	ViT$_{base}$
# added transformer block	DeBERTaV3$_{large}$:{20, 24} T5$_{base}$:{10, 12}	{10, 12}	12

Table 7: Hyperparameter for fine-tuning DeBERTaV3$_{large}$, T5$_{base}$, ViT$_{base}$. GLUE benchmacks were fine-tuning on DeBERTaV3$_{large}$ and T5$_{base}$. Text Generation datasets were fine-tuned on T5$_{base}$. Image Classification datasets were fine-tuned on ViT$_{base}$.

Additional Layer Position	DeBERTaV3$_{large}$	T5$_{base}$	ViT$_{base}$
No Additional Layer(baseline)	435M	223M	86M
+ In Multi-head self-Attention	440M	229M	88M
+ After Feed-forward Network	460M	238M	93M
+ Both	465M	243M	95M

Table 8: The number of parameters after inserting an additional layer in the pre-trained model

pre-trained model. The hyperparameters used in the experiments are listed in Table 7. To determine the best performance, we conducted a grid search on the hyperparameters listed in Table 7. All experiments were run on RTX4090 x1 or RTX3090 x1. Also, the number of parameters after inserting an additional layer in the pre-trained model is shown in Table 8. The models and metrics for the experiments used or modified code from Huggingface(Wolf et al., 2020).

Freeze Pre-trained Model. The structure involved in inserting an additional layer into the middle layer of the pre-trained model in the proposed methodology is similar to the adapter mechanism. The adapter mechanism (Houlsby et al., 2019) is one of the parameter-efficient fine-tuning (PEFT) methods. The Adapter mechanism inserts an additional layer, such as a pfeiffer Adapter(Pfeiffer et al., 2021) or LoRA(Hu et al., 2022), into the middle layer of a pre-trained model. The adapter mechanism does not perform full fine-tuning of all parameters of the model like traditional fine-tuning. It freezes the parameters of the pre-trained model and learns only the parameters of the additional layer. The additional layer learns to adapt to the pre-trained model. This increases the training speed

Model	COLA Mcc	SST-2 Acc	MRPC Acc	STS-B Corr	QNLI Acc	RTE Acc
$T5^{*}_{base}$	51.1	95.2	87.5	89.4	93.7	80.1
$T5_{base}$	61.57	94.95	91.18	89.94	93.01	78.70
Insert Additional Layer without Proposed Method						
$T5_I$	19.51	88.07	73.28	80.73	67.38	57.04
$T5_A$	14.58	88.76	83.82	84.21	85.45	58.12
$T5_B$	16.54	87.96	73.28	39.23	67.23	58.84
Freeze Pre-trained Model						
$Adapter_I$	9.69	88.99	71.57	22.75	63.39	56.32
$Adapter_A$	17.51	88.99	82.84	83.33	84.44	58.12
$Adapter_B$	13.46	86.58	71.57	21.81	64.59	54.51
with Proposed Method						
$Proposed_I$	60.06	94.61	90.93	90.76	92.84	84.12
$Proposed_A$	58.57	94.61	91.67	90.40	92.99	81.23
$Proposed_B$	61.59	94.72	92.89	91.07	92.84	83.03

Table 9: GLUE benchmank performances with $T5_{base}$. I,A and B is position of additional layer. I is In multi-head attention, A is After feed forward network and B is Both.

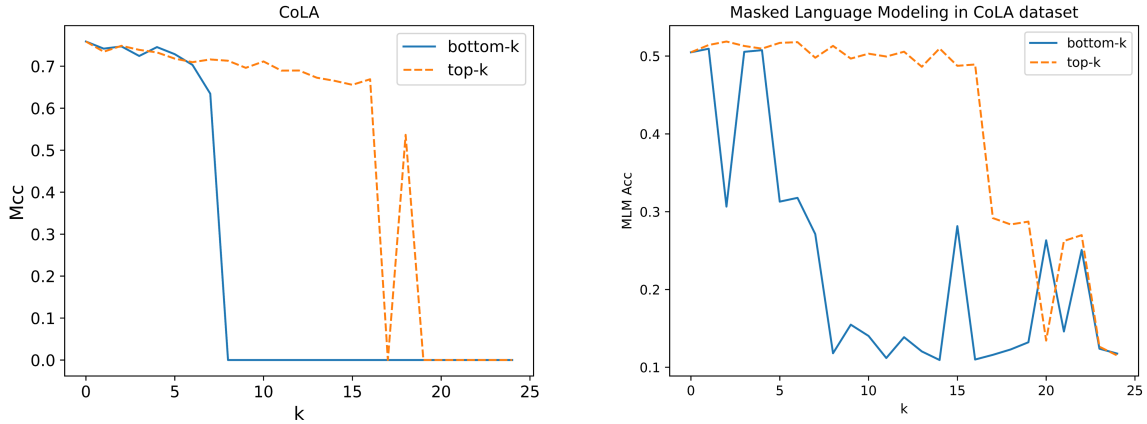

Figure 6: Performance graph of number of Additional layer in CoLA. In this graph, additional layer is initialize according to Xavier initialization and GeLU activation was applied to the output. top-k inserts an additional layer at the back k transformer blocks of the transformer layer (24-k $\sim$ 24). bottom-k inserts an additional layer at the front k transformer blocks of the transformer layer (1 $\sim$ k).

in fine-tuning by learning only a few parameters. In the experiments of this study, we compare an environment similar to the adapter mechanism. In this experiment, an additional layer applied the Xavier initialization and ReLU(Agarap, 2018) activation to the output. We then measured the performance of training only the additional layer while freezing the parameters of the pre-trained model. In this comparison, we verified whether the additional layer could adapt to the frozen model. We did not use any additional method for adaptation in the experiment.

C Experiment with t5 in GLUE benchmark

Table 9 is an evaluation of the GLUE benchmark on $T5_{base}$. Overall, it shows a similar trend to the performance on $DeBERTaV3_{large}$. However, $T5_{base}$ is relatively more robust to the noise caused by the additional layer. In Insert Additional Layer without Proposed Method, the performance degradation is relatively small than $DeBERTaV3_{large}$. However, there is a clear performance degradation, with SST-2, STS-B, and QNLI showing a performance degradation of 10%, and COLA and RTE showing extremely poor performance. STS-B had a perfor-

mance degradation of 20 points. Freeze Pre-trained Model did not demonstrate any performance improvement. Overall, the performance of the After feed forward network position seems to be relatively high. We suspect that the After feed forward network position is less noisy due to fewer layers being inserted compared to the other positions. When using the proposed method, the performance is similar to or higher than the baseline. As with DeBERTaV3$_{large}$, COLA showed a slight performance degradation from baseline except for both positions. For other datasets, most of the performance generally matched or exceeded the baseline.

D performance degradation by number of additional layer in CoLA

Figure 6 shows a graph of performance as a function of the number of additional layers in CoLA. The performance metrics are similar to Figure 3. As the number of additional layers increases, the matthews correlation decreases. Also, for most k, the performance in the bottom-K is lower than the performance in the top-K. This is the same result for masked language modeling.