NTK-approximating MLP Fusion for Efficient Language Model Fine-tuning

Tianxin Wei [*1] **Zeming Guo** [*1] **Yifan Chen** [*2] **Jingrui He** [1]

Abstract

Fine-tuning a pre-trained language model (PLM) emerges as the predominant strategy in many natural language processing applications. However, even fine-tuning the PLMs and doing inference are expensive, especially on edge devices with low computing power. Some general approaches (e.g. quantization and distillation) have been widely studied to reduce the compute/memory of PLM fine-tuning, while very few one-shot compression techniques are explored. In this paper, we investigate the neural tangent kernel (NTK)–which reveals the gradient descent dynamics of neural networks–of the multilayer perceptrons (MLP) modules in a PLM and propose to coin a lightweight PLM through NTK-approximating *MLP fusion*. To achieve this, we reconsider the MLP as a bundle of sub-MLPs, and cluster them into a given number of centroids, which can then be restored as a compressed MLP and surprisingly shown to well approximate the NTK of the original PLM. Extensive experiments of PLM fine-tuning on both natural language understanding (NLU) and generation (NLG) tasks are provided to verify the effectiveness of the proposed method MLP fusion. Our code is available at `https://github.com/weitianxin/MLP_Fusion`.

1. Introduction

Fine-tuning a large pre-trained language models (PLMs) has been the most common method to tackle downstream natural language processing (NLP) tasks (Howard & Ruder, 2018; Kale & Rastogi, 2020). However, despite high prediction accuracy and scalability of fine-tuning (Peters et al., 2018), the users have to suffer from a large computational cost, both on time and space, due to the huge size of a pre-trained language model: the sizes of popular PLMs recently increase from hundreds of millions (Brown et al., 2020) to trillion (Fedus et al., 2021) parameters. Even for the smallest BERT model (Devlin et al., 2018), there are over 110 million parameters.

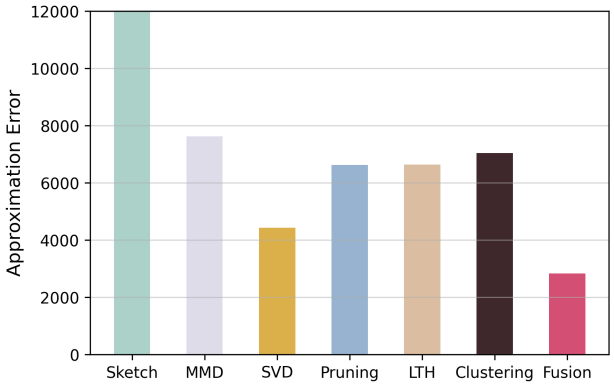

Figure 1: NTK matrix approximation error of compression methods on the validation set of SST2.

To harness the large-scale PLMs, efforts have been made in various fields. A popular techniques in *model compression* is knowledge distillation (Hinton et al., 2015, KD), which train a smaller student network with the information obtained from the original teacher model. KD serves to compress the pre-trained model and eases the subsequent fine-tuning on downstream tasks, while the distillation procedure requires comparable computational resources to pre-training. The burden is sometimes unbearable for users who cannot afford the cost of regular fine-tuning.

In addition to the directions above, there were some previous attempts to establish one-shot model compression methods. Single-shot *pruning* methods (Han et al., 2015; Lee et al., 2018; Wang et al., 2020a; Tanaka et al., 2020, sparsification) identify sub-networks at initialization concerning certain criteria of weight magnitude or gradient flow. However, most works on (entry-wise) pruning focuses on reducing the conceptual training or inference FLOPs, while sparse matrix multiplication is not well supported on modern hardware (e.g. GPUs) and even slows down the training in wall-clock time (Dao et al., 2022). Meanwhile, truncated singular value decomposition (SVD) on the weight matrices (Denton et al.,

[*]Equal contribution [1]University of Illinois Urbana-Champaign [2]Hong Kong Baptist University. Correspondence to: Yifan Chen <yifanc@comp.hkbu.edu.hk>, Jingrui He <jingrui@illinois.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

2014) is applied to speed up large CNNs by exploiting linear structure in the network to remove its redundancy; however, truncated SVD has limited representation power and suffers from poor performance as it greatly reduces the dimensions of the linear transforms within a network.

On a separate note, although efficient attention mechanisms (Kitaev et al., 2020; Choromanski et al., 2020; Chen et al., 2021) have become the mainstream methodology to accelerate the pre-training of language models, we instead focus on the feed-forward neural network (FFN) sub-layers of a pre-tained transformer, composed of two-layer multilayer perceptrons (MLP) with a large intermediate dimension size that is usually four times larger than the hidden input size (Liu et al., 2019; Radford et al., 2019; Devlin et al., 2018). Given the ever-increasing hidden input size of large language models, which now exceeds ten thousand (Touvron et al., 2023), the significance of FFN sub-layers in terms of computation cost has grown substantially. We observe for regular natural language processing (NLP) tasks where the token sequence length is no longer than 512, the computational cost of the MLP modules is heavier than the attention module even though the attention module has a squared complexity (detailed computation and comparison of the computational cost of the two modules are provided in Appendix A.1).

The current limitations of general model compression methods and the realistic need to reduce the compute in MLP modules motivate us to develop a MLP compression technique for efficient language model fine-tuning. To attain competitive fine-tuning performance, we propose a novel perspective on PLM compression, that the compressed model is supposed to approximate not only the model output, but also the training dynamics of original fine-tuning. We turn to neural tangent kernel (Jacot et al., 2018; Arora et al., 2019, NTK) as a proxy of the fine-tuning dynamics and manage to enable the compressed model to approximate the original NTK; specifically, we dissect the structure of MLP in a PLM, connect it with model fusion (Singh & Jaggi, 2020, which layer-wisely fuse multiple MLPs into one), and propose a novel compression method, *MLP fusion*, specific to PLM. As shown in Figure 1, our method "fusion" provably attains the smallest NTK matrix approximation error on a real-world dataset SST2 (Socher et al., 2013).

In summary, the contribution of this work is three-fold:

- We introduce the concept of NTK approximation to PLM compression, in the hope that the compressed model can preserve the training dynamics of the original model.
- We dissect the MLP modules in a PLM and propose a new task-agnostic technique MLP fusion, which utilizes the character of clustering to approximate the NTK.
- We provide extensive experimental results of PLM fine-tuning on both natural language understanding (NLU) and generation (NLG) tasks, verifying the effectiveness and rationality of the proposed fusion method.

2. Related work

There are abundant model compression methods available for compressing the MLPs in a PLM. The first line of research, knowledge distillation (Sanh et al., 2019b; Jiao et al., 2019; Wang et al., 2020b), turns to directly compress the pre-trained model and then fine-tune the yet-compressed model on the downstream tasks. Mean squared error (Hinton et al., 2015), optimal transport (Lohit & Jones, 2022), and maximum mean discrepancy (Huang & Wang, 2017, MMD) are usually adopted as the knowledge distillation loss term. However, in distillation, we still need to load and execute the large teacher PLM, which thus requires a lot of computational resources even before the start of fine-tuning.

Secondly, in certain tasks the fine-tuned model can be compressed for faster inference. For example, FastBert (Liu et al., 2020) proposed a sample-wise adaptive mechanism to flexibly adjusted inference time; DeeBERT (Xin et al., 2020) allowed samples to exit earlier without passing through the entire model to accelerate the inference process. Moefication (Zhang et al., 2021) speeds up the model by splitting the MLP modules of a PLM into several sub-networks, which requires an additional router to decide the corresponding sub-network for each input. These inference-accelerating methods still depend on regular fine-tuning and, most importantly, do not fully utilize the knowledge within PLMs.

In addition to the directions above, a more lightweight efficient fine-tuning paradigm is one-shot model compression. As a representative, single-shot pruning methods (Han et al., 2015; Liu et al., 2017; Lee et al., 2018; Wang et al., 2020a; Tanaka et al., 2020) identify sub-networks at initialization concerning a user-specified criteria (e.g. weight magnitude or gradient flow) and attain sparsity in model weights. A special case of pruning, Lottery tickets hypothesis (Frankle & Carbin, 2018, LTH), demonstrates the existence of sparse subnetworks in deep neural networks. LTH was also successfully verified and adopted in PLM (Chen et al., 2020). However, pruning focuses on reducing the conceptual training or inference FLOPs, while sparse matrix multiplication is not well supported on modern hardware (e.g. GPUs) and even slows down the training in wall-clock time. Furthermore, some classical computation techniques, such as truncated SVD (Denton et al., 2014) and randomized sketching (Woodruff et al., 2014; Chen et al., 2022c) can intuitively be translated to one-shot PLM compression as well; in Section 6 we implement those methods as baselines for a comprehensive comparison.

3. Preliminaries and notations

The notations for MLP are introduced in Section 3.1. We also provide brief preliminaries to NTK in Section 3.2.

3.1. Problem setup

Across this paper, we denote the input sequence as a feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, where n is the sequence length and p is the dimension of the MLP input/output (the dimensions of MLP input and output agree with the embedding dimension in most PLMs). The neural architecture of interest (a pre-trained transformer) is denoted as f, which is different from the scalar loss function ℓ. For the simplicity of notation, the output of $f(\boldsymbol{X})$ is assumed to be a scalar in this paper, which is the case in regression and binary classification tasks. Our derivation however still holds for vector/matrix output if we analyze the output element-wise.

Specifically, an MLP in the FFN sub-layer of a transformer is expressed as

$$\boldsymbol{H} = \sigma(\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}\boldsymbol{b}_1^T)\boldsymbol{W}_2 + \mathbf{1}\boldsymbol{b}_2^T. \tag{1}$$

In the equation above, $\boldsymbol{W}_1 \in \mathbb{R}^{p \times p_I}, \boldsymbol{b}_1 \in \mathbb{R}^{p_I}$ (resp. $\boldsymbol{W}_2 \in \mathbb{R}^{p_I \times p}, \boldsymbol{b}_2 \in \mathbb{R}^{p}$) are the weight matrix and the bias term of the first (resp. second) linear transform within the FFN sub-layer, and $\sigma(\cdot)$ is the element-wise activation function. Some other constantly used notations involve the MLP intermediate dimension p_I (the subscript I is short for "intermediate").

3.2. Neural tangent kernel

NTK is a powerful theoretical technique to study the gradient descent dynamics of neural networks (Jacot et al., 2018). It originated from the research on infinitely wide or ultra-wide neural networks. In applying NTK to convolutional neural networks for computer vision tasks, Arora et al. (2019) noted that NTK can be extended to arbitrary neural architecture f and initialization θ_0, which induces the (SGD) NTK as

$$\langle \nabla_{\theta_0} f(x;\theta_0),\ \nabla_{\theta_0} f(z;\theta_0) \rangle, \tag{2}$$

We note that the NTK above is specific to the stochastic gradient descent (SGD) (Robbins & Monro, 1951) optimizer. As for Adam (Kingma & Ba, 2015), the most common optimizer for language model fine-tuning, its corresponding NTK in the early stage of training (which is believed to match the nature of the short-period fine-tuning) can be approximated by the so-called Asymmetric SignGD Kernel (Malladi et al., 2022)

$$K^{(\mathrm{AS})}(x,z) := \langle \nabla_{\theta_0} f(x;\theta_0), \mathrm{sign}(\nabla_{\theta_0} f(z;\theta_0)) \rangle, \tag{3}$$

we will refer to this kernel by Adam NTK and focus on the analysis thereof along this paper, considering Adam is the dominant optimizer in language model fine-tuning.

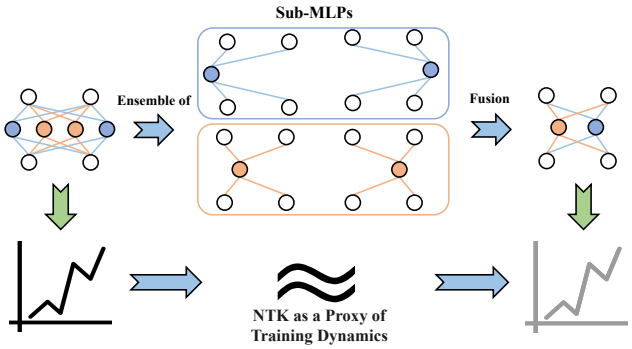

Figure 2: An overview of MLP Fusion. The original MLP is decomposed as an ensemble of p_I sub-MLPs; through MLP fusio, we cluster the sub-MLPs and re-construct a smaller MLP, which is shown to approximate the NTK of the original MLP and is thus expected to enjoy a similar training dynamics to the full-size PLM.

Recent works show directly using the NTK Equation (2) extracted from a pre-trained model f can obtain decent performance on computer vision tasks (Wei et al., 2022a), and in some cases can capture the training dynamics of language model fine-tuning (Malladi et al., 2022). There has already been some theoretical discussion on compressing a trained (fine-tuned) network with NTK preserved through pruning and quantization (Gu et al., 2022). We will shortly leverage the useful tool to guide the design of our proposed models and serve as a sanity check as well.

4. MLP fusion with approximate NTK

For the reader's convenience, we first derive the concrete form of NTK for MLP in Section 4.1; Section 4.2 then provides the exact form of the proposed clustering-based efficient method and then verify in Section 4.3 that the proposed method meets the expectations raised before.

4.1. Preparation: NTK for MLP

As the gradients w.r.t. the model weights are the building blocks of NTK, we provide the expressions of the gradients (based on $\nabla_{\boldsymbol{H}} f \in \mathbb{R}^{n \times p}$) thanks to the chain rule:

$$\begin{aligned} \nabla_{\boldsymbol{W}_2} f &= \boldsymbol{\sigma}^T \nabla_{\boldsymbol{H}} f, \quad \nabla_{\boldsymbol{b}_2} f = (\nabla_{\boldsymbol{H}} f)^T \mathbf{1} \\ \nabla_{\boldsymbol{W}_1} f &= \boldsymbol{X}^T \left[(\nabla_{\boldsymbol{H}} f\, \boldsymbol{W}_2^T) \odot \boldsymbol{\sigma}^0 \right] \\ \nabla_{\boldsymbol{b}_1} f &= \left[(\nabla_{\boldsymbol{H}} f\, \boldsymbol{W}_2^T) \odot \boldsymbol{\sigma}^0 \right]^T \mathbf{1}, \end{aligned} \tag{4}$$

where $\odot$ represents Hadamard product and σ^0 is the derivative (used in mainstream automatic differentiation software) of the activation function σ, and we abuse the boldfaced notation $\boldsymbol{\sigma}, \boldsymbol{\sigma}^0$ as shorthand notation for $\sigma\left(\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}\boldsymbol{b}_1^T\right)$ and $\sigma^0\left(\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}\boldsymbol{b}_1^T\right)$ respectively. The computation of NTK would then be boiled down to the proper inner products of the aforementioned gradient terms.

It is worth mentioning that the classical model compression technique, pruning, can be expected to well approximate the NTK. Assuming the output of a pruned MLP is close to the original one, the gradient terms will be roughly approximated by the Hadamard product of the mask matrix and the original gradient terms in Equation (4).

4.2. MLP Fusion with Clustering

We mainly develop the proposed method in this subsection for the purpose that the new output $\widetilde{\boldsymbol{H}}_C$ can approximate the original one $\boldsymbol{H}$. We turn to MLP fusion for this intuitive purpose, through a view that an MLP can be taken as the ensemble of multiple bottleneck-1 sub-MLPs. We rewrite the MLP output as the following form:

$$\begin{aligned}\boldsymbol{H} &= \sigma(\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}\boldsymbol{b}_1^T)\boldsymbol{W}_2 + \mathbf{1}\boldsymbol{b}_2^T \\ &= \sum_{i=1}^{p_I} [\sigma(\boldsymbol{X}\boldsymbol{W}_{1,:,i} + \boldsymbol{b}_{1,i}\mathbf{1})\boldsymbol{W}_{2,i,:}] + \mathbf{1}\boldsymbol{b}_2^T,\end{aligned}$$

where by convention we represent the i-th column (resp. row) in the weight matrix $\boldsymbol{W}_1$ (resp. $\boldsymbol{W}_2$) as $\boldsymbol{W}_{1,:,i}$ (resp. $\boldsymbol{W}_{2,i,:}$). The summation implies that it is feasible to approximate MLP via a few bottleneck-1 sub-MLPs (the summands on the right-hand-side above), in a similar way to numerical methods such as importance sampling (Hammersley & Morton, 1954) and sketching (Woodruff et al., 2014; Chen et al., 2022c). Considering the existence of the nonlinear activation function σ, we turn to clustering and will show as follows how the classical machine learning technique can serve to approximate the above sub-MLP summation.

To obtain the "embedding" of the sub-MLPs for clustering, we consider the original MLP as a bundle of supporting points $\boldsymbol{W} = \left[\boldsymbol{W}_1^T, \boldsymbol{b}_1, \boldsymbol{W}_2\right] \in \mathbb{R}^{p_I \times (2p+1)}$. An intuitive idea to compress an MLP is therefore representing the empirical distribution (MLP) by the output centroids of k clusters. We suggest to use $\boldsymbol{w}_i = [\boldsymbol{W}_{1,:,i}^T, \boldsymbol{b}_{1,i}, \boldsymbol{W}_{2,i,:}]^T$ as the embedding vector for the i-th sub-MLP, as $\boldsymbol{w}_i$ can uniquely decide the i-th sub-MLP. Upon the embeddings, Lloyd's algorithm (Lloyd, 1982) can be directly applied to solve the k-means clustering problem, obtain k clusters $\{P_j\}_{j=1}^k$, and return a one-hot clustering matrix $\boldsymbol{C} \in \mathbb{R}^{k \times p_I}$ with elements $\boldsymbol{C}_{ji} = \mathbb{1}_{\{\boldsymbol{w}_i \in P_j\}}$. Normalizing $\boldsymbol{C}$ and making the rows sum to 1, we can accordingly construct an averaging matrix $\boldsymbol{C}$ so that $\boldsymbol{C}\boldsymbol{W}$ will return the desired centroid matrix $\widetilde{\boldsymbol{W}} = \left[\widetilde{\boldsymbol{W}}_1^T, \tilde{\boldsymbol{b}}_1, \widetilde{\boldsymbol{W}}_2\right] \in R^{k \times (2p+1)}$. In general, conducting clustering is minimizing the distance from a point to its closest centroid, which partially explains our intuition that replacing the original sub-MLPs with their corresponding centroids can benefit the compression of MLP.

Relation with model fusion (Singh & Jaggi, 2020). In principle, we consider the clustering of sub-MLPs share the same spirit as model fusion, which takes a single layer of MLPs as an empirical distribution of the corresponding weights (either $\boldsymbol{W}_{1,:,i}$'s or $\boldsymbol{W}_{2,i,:}$'s in our context) and then fuses multiple MLPs into a new one through solving a Wasserstein barycenter problem (Peyré et al., 2019). The procedure of clustering itself is actually deeply connected to the problem above in the sense that the output centroids are the optimal barycenters when the number of points $\boldsymbol{w}_i$s assigned to each cluster is fixed. Due to the connection, we similarly name the clustering procedure as MLP fusion in this paper.

The expression of the compressed MLP. We replace each sub-MLP parameter vector $\boldsymbol{w}_i$ with the corresponding centroid (equivalently, we replace $\boldsymbol{W}$ with $\boldsymbol{C}^T\widetilde{\boldsymbol{W}}$). The new output can be naturally simplified as

$$\begin{aligned}&\sigma\left(\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}(\tilde{\boldsymbol{b}}_1)^T\right)\boldsymbol{C}\right)\boldsymbol{C}^T\widetilde{\boldsymbol{W}}_2 + \mathbf{1}\boldsymbol{b}_2^T \\ =&\sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}(\tilde{\boldsymbol{b}}_1)^T\right)\left(\boldsymbol{C}\boldsymbol{C}^T\right)\widetilde{\boldsymbol{W}}_2 + \mathbf{1}\boldsymbol{b}_2^T,\end{aligned}$$

where the above equation holds because $\boldsymbol{C}$ simply "copies" the centroids and thus can be taken out of the activation function. We will shortly show in Section 4.3 the computational properties of the one-hot clustering matrix $\boldsymbol{C}$ is indeed critical for Adam NTK approximation.

After being taken out of the activation function, $\boldsymbol{C}$ is then allowed to be combined with $\boldsymbol{C}^T$ to form the scaling matrix referred to as $\boldsymbol{P} = \boldsymbol{C}\boldsymbol{C}^T$, which is a $k \times k$ diagonal fixed matrix that greatly reduces the computation compared with the original equation. Note the architecture of the final MLP has not yet been specified, since there are different ways to address the scaling matrix $\boldsymbol{P}$: it can

- either be incorporated into $\widetilde{\boldsymbol{W}}_2$, or
- stand alone as a constant scaling matrix.

It is worth mentioning that these two strategies perform the same during forward propagation, but during backward propagation, the gradient of the second method is multiplied by the scaling matrix $\boldsymbol{P}$.

4.3. NTK Approximation

We analyze that when $\boldsymbol{P}$ is designed to stand alone, the specific MLP named MLP Fusion can serve to approximate the NTK. It is formally given as follows:

$$\widetilde{\boldsymbol{H}}_C := \sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\boldsymbol{P}\widetilde{\boldsymbol{W}}_2 + \mathbf{1}\boldsymbol{b}_2^T \tag{5}$$

where we choose $\widetilde{\boldsymbol{W}}_1 := \boldsymbol{W}_1\boldsymbol{C}^T$, $\tilde{\boldsymbol{b}}_1 := \boldsymbol{C}\boldsymbol{b}_1$, $\widetilde{\boldsymbol{W}}_2 := \boldsymbol{C}\boldsymbol{W}_2$ as the new parameters for the compressed MLP, and $\boldsymbol{P}$ is designed to stand alone as a constant scaling matrix. It is important to highlight that $\mathbf{P}_{ii} = \mathbf{C}\mathbf{C}_{ii}^T = \sum_q \mathbf{C}_{iq}^2 =$

$\sum_q \mathbf{C}_{iq}$ ($\mathbf{P}_{ij} = \mathbf{C}\mathbf{C}^T_{ij} = 0$ for $i \neq j$) represents the number of points in cluster i. From an intuitive perspective, the backward process of multiplying the gradient by the scaling matrix $\mathbf{P}$ can be interpreted as assigning different learning rates to different clusters, which means that larger clusters will be assigned a larger learning rate. To perform a more detailed analysis of the problem, we will then demonstrate how the specific MLP (5) can serve to approximate the NTK.

To exploit the potentials of the pre-trained models, in addition to retaining the output, we also expect the efficient MLP modules can induce an Adam NTK similar to the original one. This expectation implies the following requirements: first, the new output $\widetilde{\boldsymbol{H}}$ is supposed to approximate the original output $\boldsymbol{H}$, which we have heuristically show in the previous discussion. We also provide a standard analysis of MLP output $\widetilde{\boldsymbol{H}}$ in Appendix C; second, the hidden representation $\boldsymbol{\sigma}$ and the related composition term $\left(\nabla_{\boldsymbol{H}} f\ \boldsymbol{W}_2^T\right) \quad \boldsymbol{\sigma}^0$ should also be preserved.

This subsection will thus be devoted to verify that the proposed method can induce an Adam NTK close to the original one. The key step is to show the inner product of the four gradient terms in Equation (4) will approximately remain. We prepare some additional notations to ease the following discussions, and let the compressed neural model equipped with the compressed MLP module as f_c. The input token sequence is denoted as $\boldsymbol{X}$ or $\boldsymbol{Z}$, respectively.

We first make an assumption that the clustering can capture the MLP empirical distribution, so that, to some sense, $\boldsymbol{C}\widetilde{\boldsymbol{W}} \quad \boldsymbol{W}$ and $\widetilde{\boldsymbol{H}}_C \quad \boldsymbol{H}$ as derived in Section 4.2. This assumption implies that $\nabla_{\boldsymbol{H}} f$ can be preserved by $\nabla_{\widetilde{\boldsymbol{H}}_C} f_c$, since they depend on $\boldsymbol{H}/\widetilde{\boldsymbol{H}}_C$ in the same way. We can automatically obtain $\langle \nabla_{\boldsymbol{b}_2} f(\boldsymbol{X}), \mathrm{sign}(\nabla_{\boldsymbol{b}_2} f(\boldsymbol{Z}))\rangle$ $\langle \nabla_{\boldsymbol{b}_2} f_c(\boldsymbol{X}), \mathrm{sign}(\nabla_{\boldsymbol{b}_2} f_c(\boldsymbol{Z}))\rangle$.

We then analyze the term $\langle \nabla_{\boldsymbol{W}_2} f(\boldsymbol{X}), \mathrm{sign}(\nabla_{\boldsymbol{W}_2} f(\boldsymbol{Z}))\rangle$, where the notation $\langle\ ,\ \rangle$ is reloaded as the matrix inner product $\langle \boldsymbol{X}, \boldsymbol{Z}\rangle := \mathrm{Tr}\left(\boldsymbol{X}^T\boldsymbol{Z}\right)$. The term equals[1]

$$\mathrm{Tr}\left[(\nabla_{\boldsymbol{H}} f(\boldsymbol{X}))^T \boldsymbol{\sigma}_x \quad \mathrm{sign}\left(\boldsymbol{\sigma}_z^T \nabla_{\boldsymbol{H}} f(\boldsymbol{Z})\right)\right],$$

and can be shown to approach

$$\left\langle \nabla_{\widetilde{\boldsymbol{W}}_2} f_c(\boldsymbol{X}), \mathrm{sign}\left(\nabla_{\widetilde{\boldsymbol{W}}_2} f_c(\boldsymbol{Z})\right)\right\rangle.$$

Concretely, we re-utilize the deduction $\nabla_{\boldsymbol{H}} f \quad \nabla_{\widetilde{\boldsymbol{H}}_C} f_c$ to make it sufficient to study whether $(\tilde{\boldsymbol{\sigma}}_x \boldsymbol{P}) \quad \mathrm{sign}\left(\boldsymbol{P}\tilde{\boldsymbol{\sigma}}_z^T\right)$ can approximate its counterpart $\boldsymbol{\sigma}_x \quad \mathrm{sign}\left(\boldsymbol{\sigma}_z^T\right)$.

[1] $\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_z$ are the shorthand for $\sigma\left(\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}\boldsymbol{b}_1^T\right)$ and $\sigma\left(\boldsymbol{Z}\boldsymbol{W}_1 + \mathbf{1}\boldsymbol{b}_1^T\right)$ respectively. Similarly, we define $\tilde{\boldsymbol{\sigma}}_x := \sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)$ and $\tilde{\boldsymbol{\sigma}}_z := \sigma\left(\boldsymbol{Z}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)$ for f_c.

Analogous analyses of the matrix product

$$\left[(\nabla_{\boldsymbol{H}} f(\boldsymbol{X})\boldsymbol{W}_2^T) \quad \boldsymbol{\sigma}_x^0\right] \quad \left[(\nabla_{\boldsymbol{H}} f(\boldsymbol{Z})\boldsymbol{W}_2^T) \quad \boldsymbol{\sigma}_z^0\right]^T$$

can also be performed for the rest two terms

$$\langle \nabla_{\boldsymbol{W}_1} f(\boldsymbol{X}), \mathrm{sign}(\nabla_{\boldsymbol{W}_1} f(\boldsymbol{Z}))\rangle \quad \text{and}$$
$$\langle \nabla_{\boldsymbol{b}_1} f(\boldsymbol{X}), \mathrm{sign}(\nabla_{\boldsymbol{b}_1} f(\boldsymbol{Z}))\rangle.$$

Due to the limited space, we defer both derivations to Appendix A.3, and remark the element-wise sign function associated with the Adam optimizer plays an important role in NTK preservation. For the regular SGD, however, some additional conditions and modifications are needed. We refer the readers interested to Appendix A.4 for more details.

4.4. Layer-wise task-specific tuning

In the previous section, our proposed approach MLP Fusion manages to exploit the potentials of the pre-trained models in a one-shot and task-agnostic manner without the need for any data, where we retain the training dynamics of neural networks through NTK preservation. To more effectively acquire the knowledge within each task, we can leverage the idea from distillation and intuitively design a layer-wise (and thus lightweight) task-specific tuning module, which further tune the fused MLP with task-specific unsupervised training data. Compared to classical distillation, the layer-wise tuning lasts a shorter period (e.g. only 1 epoch in our experiments in Section 6) and only the weights in the fused MLP will be updated.

To be specific, we set the tuning loss as the mean squared error (MSE) between the layer output $\boldsymbol{H}_t^l$ in the teacher model and the layer output $\boldsymbol{H}^l$ in the student model for layer l of the PLM. The tuning loss is then computed as:

$$\ell^{\text{tune}} = \sum_{l=1}^{L} \mathrm{MSE}(\boldsymbol{H}_t^l, \boldsymbol{H}^l) \tag{6}$$

where L is the number of layers in the PLM and MSE denotes the mean squared error.

5. Experimental Setup

We evaluate the proposed MLP fusion on various downstream NLP tasks, and provide a sketch of these tasks in this section. In addition, we succinctly introduce two intuitive while non-trivial baseline methods, "Sketching" and "MMD", in Section 2. Part of the experiment implementations are borrowed from Chen et al. (2022a;b). The code for our algorithms will be publicly available on GitHub.

5.1. Dataset Details

- The Stanford Sentiment Treebank (Socher et al., 2013, **SST2**) is a corpus of movie reviews and human annotations of their sentiment. The corpus is part of the GLUE

benchmark (Wang et al., 2019). In SST2, the sequence lengths are on average 13.3 and max 66. 67k sentences are incorporated into the training set and 0.9k into the dev set. Following the GLUE benchmark, we report the accuracy metric on whether the sentiment of a review is positive or negative.

- The Multi-Genre Natural Language Inference Corpus (Williams et al., 2018, **MNLI**) consists of enormous sentence pairs of premises and hypotheses, labeled with human textual entailment annotations. The premise sentences include ten distinct genres, and there are 393k pairs in the training set, 10k in the dev set. (Only the dev set is used for evaluation in Table 2.) The metric of this task is mismatched (cross-domain) accuracy following (Wang et al., 2019).

- **WebNLG** dataset is composed of data/text pairs, where "data" is in a format of *(subject, property, object)* triple. For the train and the validation set, there are nine categories extracted from DBpedia; while in the test set, there are five extra unseen categories, which can partially reflect the generalizability of the methods. The input sequences in the training set contain 1 to 7 triples, and the lengths of most sequences are bounded by 50 (as each triple only includes three short phrases). The official evaluation script is used in our experiments, and we report BLEU (Papineni et al., 2002), METEOR (Banerjee & Lavie, 2005), (Banerjee & Lavie, 2005) and TER (Snover et al., 2006) as the metrics.

5.2. Hyperparameter Setup

All the models in this work are implemented by PyTorch. The experiments are all conducted on one Tesla V100 32 GB GPU. For NLU tasks, We fine-tune RoBERTa (Liu et al., 2019) with an AdamW (Loshchilov & Hutter, 2018) optimizer and use a polynomial learning rate scheduler to make the learning rate linearly decay; concretely, the learning rate is linearly warmed up from 0 for the first 0.06 epoch. The learning rate is searched in the range of f1e-5, 2e-5,4e-5, 6e-5, 8e-5g, and the batch size is fixed as 32. For NLG tasks, we keep using AdamW optimizer to fine-tune GPT-2 (Radford et al., 2019), and a linear learning rate scheduler with a 500-step warmup duration is used. The learning rate is tuned in the same range as above while the batch size is fixed to 8. By default, all the compared methods reduce the MLP intermediate size to 768 or a comparable number of parameters from 3076. The reduction/sketching is performed on the last 8 layers of the PLM by default. For Clustering, we adopt the K-Means algorithm due to its simplicity and effectiveness. To reduce the random variability in the results, the experiments are all averaged over three runs.

5.3. Compared methods

We mainly compare our method with several other one-shot compression methods: regular fine-tuning (Howard & Ruder, 2018), truncated SVD (Denton et al., 2014), Pruning (single-shot unstructured pruning) (Han et al., 2015; Lee et al., 2018; Wang et al., 2020a; Tanaka et al., 2020), LTH (Lottery Ticket Hypothesis) (Frankle & Carbin, 2018; Chen et al., 2020), GEM-MINER (Sreenivasan et al.), Moefication (Zhang et al., 2021). We also compare the structured pruning method FLOP(Factorized Low-rank Pruning) (Wang et al., 2020c) with our method. By default, all the compared methods reduce the MLP intermediate size to 768 or a comparable number of parameters from 3076. Specifically, truncated SVD linearly compressed the weight matrix in the MLP. The SVD is defined as $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\boldsymbol{T}}$ where $\boldsymbol{S}$ is a diagonal matrix with the singular values on the diagonal. As the singular values of $\boldsymbol{W}$ decay rapidly, $\boldsymbol{W}$ can be approximated by keeping only the t largest entries of $\boldsymbol{S}$ as $\tilde{\boldsymbol{W}} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{S}}\tilde{\boldsymbol{V}}^{\boldsymbol{T}}$. Here we extract the top 768 entries for compression. Unstructured pruning globally removes a certain ratio of connections by exploring the weight magnitude and gradient. Concretely, we mask 75% of the connections to match the compression rate (25%). As a special case of pruning, Lottery tickets hypothesis (LTH) (Frankle & Carbin, 2018; Chen et al., 2020), demonstrates the existence of sparse subnetworks in DNNs, which performs iterative sparsification during tuning to find the matching network. We iteratively prune the MLP for one epoch to make it computationally equivalent to the Layer-wise task-specific tuning module. The network mask ratio is also 75% as pruning. Moefication splits the MLP modules of a PLM into several sub-networks, and designs the additional route mechanism to decide the corresponding sub-network for each input. Here we split the original MLP into 4 sub-networks to match the compressed network size.

Furthermore, we demonstrate as follows how two machine learning techniques, Randomized Sketching and MMD, can be directly applied to MLP compression.

Sketching the weight matrices. The idea of Sketching is to reduce the size of $\boldsymbol{W}_1, \boldsymbol{W}_2$ by multiplying a matrix $\boldsymbol{S}$:

$$\widetilde{\boldsymbol{H}}_S = \sigma(\boldsymbol{X}\boldsymbol{W}_1\boldsymbol{S} + \mathbf{1}\boldsymbol{b}_1^T\boldsymbol{S})\boldsymbol{S}^T\boldsymbol{W}_2 + \mathbf{1}\boldsymbol{b}_2^T,$$

where $\boldsymbol{S}$ can be a Gaussian Sketching Matrix, which applies Johnson–Lindenstrauss transform (Ailon & Chazelle, 2009) to the weight matrices $\boldsymbol{W}_1, \boldsymbol{W}_2$. We expect Sketching can more or less preserve the information within the PLM.

Compression via minimizing MMD. Following the empirical distribution perspective introduced in Section 4.2, we can naturally compress the MLP (distribution of sub-MLPs) through finding a new empirical distribution with smaller support points. The problem can be reduced to minimizing the MMD distance (for the reader's convenience we pro-

Table 1: Approximation error of each baseline method on SST2 validation set with RoBERTa as the backone. Specifically, SVD is a deterministic method and therefore gives 0 standard deviation. The results are averaged over 3 runs.

	Approximation Error			
	Output		NTK	
Sketch	24.48	0.61	242757.23	42629.38
MMD	8.92	0.22	7620.49	527.86
SVD	5.89	0.00	4423.38	108.89
Pruning	5.18	0.23	6623.20	463.72
LTH	5.10	0.18	6628.73	462.03
Clustering	**4.83**	**0.02**	7030.91	561.52
MLP Fusion (Ours)	7.26	0.14	**2826.59**	**155.06**

vide a preliminary introduction to MMD in Appendix A.2) between the original MLP and the compressed MLP.

In addition, we list the performance of DistilRoBERTa (Sanh et al., 2019a) as a reference in the two GLUE tasks. To clearly ablate the effect of NTK approximation, we implement a clustering-based method while the fused weights $\widetilde{\boldsymbol{W}}_1, \tilde{\boldsymbol{b}}_1, \widetilde{\boldsymbol{W}}_2$ will be replaced by $\boldsymbol{W}_1\boldsymbol{C}^T\boldsymbol{P}^{\frac{1}{2}}$, $\boldsymbol{C}\boldsymbol{b}_1\boldsymbol{P}^{\frac{1}{2}}$, $\boldsymbol{P}^{\frac{1}{2}}\boldsymbol{C}\boldsymbol{W}_2$ respectively (recall $\boldsymbol{P} = \boldsymbol{C}\boldsymbol{C}^T$ is a diagonal matrix); the corresponding MLP is then specified by

$$\sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\widetilde{\boldsymbol{W}}_2 + \mathbf{1}\boldsymbol{b}_2^T,$$

which enjoys the same architecture as "Sketching" and "MMD". For the layer-wise task-specific tuning, we adopt the original RoBERTa model (Liu et al., 2019) (GPT-2 (Radford et al., 2019) for language generation) as the teacher and the language model with fused MLP as the student, on the two NLU tasks. The tuning only lasts for 1 epoch so as to match the computational cost of the pre-processing in LTH.

6. Numerical results

In the section, we show the numerical results of our proposed MLP Fusion compared with representative baselines on both the NLU and NLG tasks.

In the main text we focus our attention on the prediction accuracy. As for the runtime, we compare our proposed MLP fusion to two representative methods, regular fine-tuning and pruning, in Appendix B.

6.1. Preliminary Evaluation of Approximation Error

As a sanity check, we first perform the preliminary evaluation on NTK approximation. We examine the output and the induced NTK of the first-layer MLP on the validation set of SST2 with RoBERTa-base (Liu et al., 2019) compressed by different baseline approaches. The results are summarized in Table 1. We come up with the following observations: i Most of the listed methods can well approximate the MLP output with a small output distance between the original RoBERTa and the compressed model. ii Our proposed MLP Fusion achieves the minimum NTK distance among all approaches, which maximizes the retention of the training dynamics of the original PLM. The distance is defined as the l_2 difference of the NTK kernel calculated on the evaluation samples before and after compression. The preliminary experiments successfully verify our assumption about NTK approximation.

Table 2: Accuracy of each baseline method on SST2 and MNLI validation sets with RoBERTa as the PLM.

	SST2		MNLI	
RoBERTa	94.61	0.09	87.34	0.28
DistilRoBERTa	92.50	0.12	84.03	0.18
Sketch	91.90	0.14	83.30	0.11
MMD	92.54	0.41	84.20	0.24
SVD	92.55	0.24	85.23	0.04
FLOP	92.12	0.19	84.05	0.21
Pruning	92.78	0.17	85.82	0.12
LTH	92.91	0.15	85.96	0.10
GEM-MINER	92.89	0.16	85.51	0.11
Moefication	92.19	0.20	84.83	0.27
Clustering	93.01	0.17	85.75	0.04
MLP Fusion (Ours)	93.23	0.23	86.10	0.06
+Task-specific Tuning	**93.79**	**0.07**	**86.32**	**0.06**

6.2. Evaluation of Natural Language Understanding

We provide extensive experimental comparisons based on RoBERTa as the PLM with a set of representative baselines on four natural language understanding benchmarks SST2, MNLI, STS-B, and QNLI. The test results on SST2 and MNLI can be found in Table 2, while the test results on the rest of the benchmarks are provided in Appendix F. Furthermore, we present performance comparisons among various methods after task-specific fine-tuning in Appendix E and two additional baselines that try to maintain MLP output and NTK in Appendix G. Note that DistilRoBERTa is the distilled version of RoBERTa with the same training process but a lightweight network architecture. For all the compressed methods, we reduce the intermediate size to 25% (3072 ! 768), which is the same as the MLP input size. For Pruning/LTH, we mask 75% of connections in the MLP. In Moefication, the MLP is divided by 4 experts to reduce the model size. For a fair comparison, all the reduction in each method is conducted on the last 8 layers of PLM.

According to the table, we can have the following findings: (i) MLP Fusion outperforms all the baselines, which demonstrates the effectiveness of our proposed approach. (ii) Without the property of NTK approximation, there is an obvious reduction in the performance of Clustering. which verifies

Table 3: Performance(%) of each baseline method on WebNLG with GPT-2 as the PLM. **Bold** results are the best scores under each metric. The down-arrow notation indicates that a lower metric represents better performance.

	WebNLG								
	BLEU			MET			TER #		
	S	U	A	S	U	A	S	U	A
Fine-tuning	57.93	22.55	42.17	0.42	0.25	0.34	0.39	0.76	0.56
Sketch	43.16	8.07	27.28	0.32	0.13	0.23	0.54	0.95	0.73
MMD	56.73	19.90	40.17	0.41	0.23	0.33	0.41	0.79	0.59
Pruning	55.21	**21.80**	40.55	0.40	0.25	0.33	0.42	**0.76**	0.57
Clustering	54.75	20.63	39.81	0.40	0.25	0.33	0.43	0.77	0.59
MLP Fusion (Ours)	**57.12**	21.03	40.79	0.41	0.25	0.33	0.41	0.78	0.58
+Task-specific Tuning	56.75	21.41	**41.04**	**0.42**	**0.25**	**0.33**	**0.40**	0.77	**0.57**

[a] The letters S, U, and A in the WebNLG metric denote SEEN, UNSEEN, and ALL; instances under the SEEN categories are used for training; instances under the UNSEEN categories are used for testing; ALL has all the instances in it.

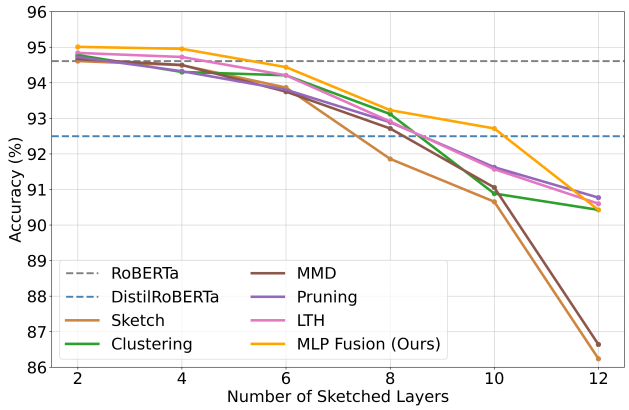

Figure 3: Accuracy of each baseline method with respect to various numbers of sketched layers on the SST2 data set.

the necessity of NTK approximation. (iii) Layer-wise task-specific tuning further enhances the performance of MLP Fusion by incorporating task-specific knowledge. Meanwhile, the constrained structured pruning method FLOP tends to yield lower performance. It is worth noting that LTH and GEM-MINER also require pre-processing when masking connections. By making them computationally comparable, the superior performance further validates the advantageousness of our approach.

6.3. Evaluation of Natural Language Generation

In this part, we investigate the effectiveness of the proposed MLP Fusion by evaluating on the natural language generation benchmark WebNLG with a set of one-shot comparable baselines. The results are reported in Table 3. The compressing/pruning setup is as the same as the NLU evaluation shown in Section 6.2. Our proposed method generally retains the performance of naive fine-tuning method best. The average accuracy of MLP Fusion is increased by about 1% compared to baselines. Among all the baselines, the pruning method stands out, which can be attributed to the preservation of the original MLP weight matrix size. However, sparse matrix multiplication is not well supported on modern GPU hardware.

Impact of sketch layers In the previous sections, we perform experiments with the sketch of the last 8 layers of PLM on the SST2 data set. Here we explore the impact of the sketch layers in Figure 3 to offer more insights. The dashed lines denote the performances of RoBERTa and DistilRoBERTa. The number 2 on the horizontal axis indicates the setting of Sketching the last 2 layers of PLM. From the figure, we can find that MLP Fusion consistently exhibits comparable or better performance than the baselines. To our surprise, MLP Fusion can even outperforms the raw RoBERTa when the number of sketched layers is less than 6, which can be regarded as a free lunch for the PLM fine-tuning. It also reveals the potential of our method for removing redundancy in neural networks. In addition, we find that MLP Fusion always exhibits superior performance than DistilRoBERTa as long as the number of sketched layers is less than 10. However, when we sketch all the layers (12) in the PLM, the prediction performance will drop substantially. This suggests that the bottom layers of the PLM contain a wealth of semantic information that should not be replaced, which is consistent with the finding in (Zhang et al., 2020).

6.4. Ablation Studies

The choice of the intermediate dimension in MLP fusion. In the previous experiments, we set the reduced intermediate dimension to 768, which is the same as the MLP input size. In this section, we test the training dynamics and testing performance of our proposed MLP Fusion with various intermediate sizes. From the results in Figure 4, we can observe the training loss is stable when the intermediate size is larger than 768. However, when the dimension is smaller

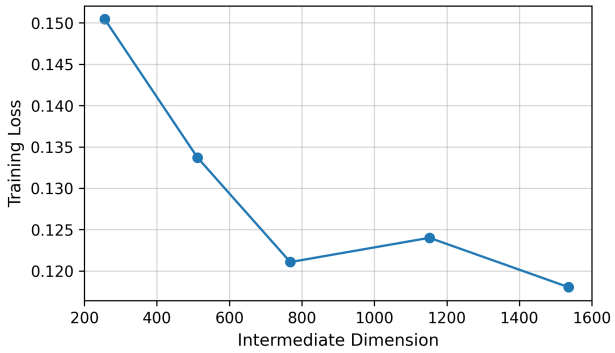

Figure 4: Training loss (dynamic) of our proposed MLP Fusion with respect to various intermediate dimensions on the SST2 data set.

Table 4: Performance of Proposed Method and The Baseline Method at Different Levels of Compression

Intermediate Dimension	Sketch	MLP Fusion
256	88.19	90.71
768	91.90	93.23
1536	92.09	93.46

than 768, there is a dramatic increase in the curve. The phenomenon is mainly because the intermediate dimension is smaller than the input dimension, resulting in the MLP which is not full rank and consequently leads to poor results. We further test the performance of our MLP Fusion and Sketch baseline with different intermediate sizes in Table 4, which has a similar trend as training dynamics.

7. Conclusion

We propose MLP fusion, a novel one-shot model compression method that heavily utilizes the properties of clustering to approximate the NTK of the original PLM. It turns out that the fused MLP can both well approximate the output and attain the closest NTK to the original one compared to other one-shot compression methods. Therefore, we believe MLP fusion sheds some light on the the new paradigm for efficient language model fine-tuning.

One direct extension of our work is using MLP fusion as an initialization method for distillation. Compared to re-training from scratch, we expect the information preserved in the fused MLP can ease the following distillation and speed up the model convergence.

Acknowledgements

This work is supported by National Science Foundation under Award No. IIS-1947203, IIS-2117902, IIS-2137468, and Agriculture and Food Research Initiative (AFRI) grant no. 2020-67021-32799/project accession no.1024178 from the USDA National Institute of Food and Agriculture. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies or the government.

References

Ailon, N. and Chazelle, B. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on computing*, 39(1):302–322, 2009.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019.

Balles, L. and Hennig, P. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In *International Conference on Machine Learning*, pp. 404–413. PMLR, 2018.

Banerjee, S. and Lavie, A. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pp. 65–72, 2005.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

Chen, Y., Zeng, Q., Ji, H., and Yang, Y. Skyformer: Remodel self-attention with gaussian kernel and nystr*n*" om method. *Advances in Neural Information Processing Systems*, 34:2122–2135, 2021.

Chen, Y., Hazarika, D., Namazifar, M., Liu, Y., Jin, D., and Hakkani-Tur, D. Empowering parameter-efficient transfer learning by recognizing the kernel structure in self-attention. In *Findings of the Association for Computational Linguistics: NAACL 2022*. Association for Computational Linguistics, 2022a.

Chen, Y., Hazarika, D., Namazifar, M., Liu, Y., Jin, D., and Hakkani-Tur, D. Inducer-tuning: Connecting prefix-tuning and adapter-tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2022b.

Chen, Y., Zeng, Q., Hakkani-Tur, D., Jin, D., Ji, H., and Yang, Y. Sketching as a tool for understanding and accelerating self-attention for long sequences. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5187–5199, Seattle, United States, July 2022c. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.381. URL https://aclanthology.org/2022.naacl-main.381.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., and Weller, A. Rethinking attention with performers. *CoRR*, abs/2009.14794, 2020. URL https://arxiv.org/abs/2009.14794.

Dao, T., Chen, B., Sohoni, N. S., Desai, A., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Ré, C. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dong, L., Xu, S., and Xu, B. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5884–5888. IEEE, 2018.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Fukushima, K. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3):121–136, 1975.

Geng, S., Liu, S., Fu, Z., Ge, Y., and Zhang, Y. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 299–315, 2022.

Gu, L., Du, Y., Zhang, Y., Xie, D., Pu, S., Qiu, R. C., and Liao, Z. "lossless" compression of deep neural networks: A high-dimensional neural tangent kernel approach. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=NaW6T93F34m.

Hammersley, J. M. and Morton, K. W. Poor man's monte carlo. *Journal of the Royal Statistical Society: Series B (Methodological)*, 16(1):23–38, 1954.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network, 2015. URL http://arxiv.org/abs/1503.02531. cite arxiv:1503.02531Comment: NIPS 2014 Deep Learning Workshop.

Howard, J. and Ruder, S. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 328–339, 2018.

Huang, Z. and Wang, N. Like what you like: Knowledge distill via neuron selectivity transfer. *arXiv preprint arXiv:1707.01219*, 2017.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Kale, M. and Rastogi, A. Text-to-text pre-training for data-to-text tasks. In *Proceedings of the 13th International Conference on Natural Language Generation*, pp. 97–102, Dublin, Ireland, December 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.inlg-1.14.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Liu, W., Zhou, P., Zhao, Z., Wang, Z., Deng, H., and Ju, Q. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*, 2020.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pp. 2736–2744, 2017.

Lloyd, S. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Lohit, S. and Jones, M. Model compression using optimal transport. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2764–2773, 2022.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Malladi, S., Wettig, A., Yu, D., Chen, D., and Arora, S. A kernel-based view of language model fine-tuning. *arXiv preprint arXiv:2210.05643*, 2022.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pp. 2227–2237, 2018.

Peyré, G., Cuturi, M., et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019a.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019b.

Singh, S. P. and Jaggi, M. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33:22045–22055, 2020.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pp. 223–231, 2006.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Sreenivasan, K., Sohn, J.-y., Yang, L., Grinde, M., Nagle, A., Wang, H., Xing, E., Lee, K., and Papailiopoulos, D. Rare gems: Finding lottery tickets at initialization. *Advances in neural information processing systems*. URL https://par.nsf.gov/biblio/10395564.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020a.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33: 5776–5788, 2020b.

Wang, Z., Wohlwend, J., and Lei, T. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6151–6162, Online, November 2020c. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.496. URL https://aclanthology.org/2020.emnlp-main.496.

Wei, A., Hu, W., and Steinhardt, J. More than a toy: Random matrix models predict how real-world neural representations generalize. *arXiv preprint arXiv:2203.06176*, 2022a.

Wei, T. and He, J. Comprehensive fair meta-learned recommender system. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1989–1999, 2022.

Wei, T., You, Y., Chen, T., Shen, Y., He, J., and Wang, Z. Augmentations in hypergraph contrastive learning: Fabricated and generative. In *Advances in Neural Information Processing Systems*, 2022b.

Williams, A., Nangia, N., and Bowman, S. 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, Volume 1 (Long Papers)*, pp. 1112–1122, 2018.

Woodruff, D. P. et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, 2020.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Zhang, T., Wu, F., Katiyar, A., Weinberger, K. Q., and Artzi, Y. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*, 2020.

Zhang, Z., Lin, Y., Liu, Z., Li, P., Sun, M., and Zhou, J. Moefication: Conditional computation of transformer models for efficient inference. *arXiv preprint arXiv:2110.01786*, 2021.

A. Derivations omitted in the main text

A.1. Computational costs of attention and MLP moduels

Condensing FFN sub-layers is critical to obtaining a lightweight pre-trained model. Besides self-attention sub-layers, FFN sub-layers also take a lot of computation time, and even become the actual bottleneck when the input sequence length is short. We will verify this claim through the following derivation.

We first recall the most common setting of a MLP in PLMs. Taking RoBERTa-base as an example, the hidden dimension is $p = 768$ and there are $h = 12$ heads in each self-attention module; the intermediate dimension in MLP is $p_I = 4p = 3072$. In the self-attention sub-layer, given the length-n input $\boldsymbol{X}$ we need to first compute the query, key, and value matrix $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$, which takes $3 \ \ np^2$ operations to perform the linear transform (omitting the bias). For the core self-attention module, we will at least need $h \ \ 2n^2(p/h)$ multiplication operations; the final linear transform will again take np^2 cost. The total FLOPs of a self-attention sub-layer are around $4np^2 + 2n^2p$.

As for the FFN sub-layer, the computational cost is clear: $2npp_I = 8np^2$. We can check for regular nlp tasks in which the input length n is bounded by 512, $8np^2$ proves to be larger than $4np^2 + 2n^2p$, when $p = 768$. More specifically, when input length $n < 2p$, the computation cost of FFN layers becomes the primary bottleneck. This condition is particularly applicable to modern foundational language models (Touvron et al., 2023), which often possess a massive hidden size even exceeding ten thousand.

A.2. Maximum mean discrepancies (MMD)

We start with a brief introduction to MMD. The expression of MMD between two distributions P and Q is given as

$$\begin{aligned}\mathrm{MMD}(P,Q) &= \sup_{\|f\|_{\mathcal{H}} \ 1} \mathsf{E}_{X \ P}[f(X)] \quad \mathsf{E}_{Y \ Q}[f(Y)] \\ &= \|\mathsf{E}_{X \ P}[\varphi(X)] \quad \mathsf{E}_{Y \ Q}[\varphi(Y)]\|_H, \end{aligned} \tag{7}$$

where $\varphi(\ ) : X \ ! \ H$ is the feature map inducing the kernel function $k(x,y) = h\varphi(x), \varphi(y)i_H$ associated with a reproducing kernel Hilbert space (RKHS) H. Through the strong reproducing property of the map φ, we can rewrite the the squared MMD as

$$\begin{aligned}\mathrm{MMD}^2(P,Q) &= \|\mathsf{E}_{X \ P}\varphi(X) \quad \mathsf{E}_{Y \ Q}\varphi(Y)\|_H^2 \\ &= h\mathsf{E}_{X \ P}\varphi(X), \mathsf{E}_{X' \ P}\varphi(X')i_H + h\mathsf{E}_{Y \ Q}\varphi(Y), \mathsf{E}_{Y' \ Q}\varphi(Y')i_H \quad 2h\mathsf{E}_{X \ P}\varphi(X), \mathsf{E}_{Y \ Q}\varphi(Y)i_H \\ &= \mathsf{E}_{X,X' \ P}k(X,X') + \mathsf{E}_{Y,Y' \ Q}k(Y,Y') \quad 2\mathsf{E}_{X \ P,Y \ Q}k(X,Y), \end{aligned} \tag{8}$$

which is easier to optimize using back-propagation.

Following the empirical distribution view of MLP, we denote the original MLP as μ_w, a uniform discrete distribution over the rows of the embedding matrix $\boldsymbol{W}$, and the compressed MLP similarly as μ_m, an empirical distribution evenly distributed over the rows in matrix $\boldsymbol{W}^{(m)} = [\boldsymbol{W}_1^{(m)}, \boldsymbol{b}^{(m)}, \boldsymbol{W}_2^{(m)}] \ 2 \ R^{k \ (2p+1)}$. We can then optimize the following problem

$$\min_{\boldsymbol{W}^{(m)}} \mathrm{MMD}^2\left(\mu_w, \mu_m\left(\boldsymbol{W}^{(m)}\right)\right), \tag{9}$$

from which we can obtain $\boldsymbol{W}^{(m)}$. As in Section 4.2, we can construct the condensed MLP with $\boldsymbol{W}^{(m)}$ as

$$\widetilde{\boldsymbol{H}}_M = \frac{p_I}{k}\left[\sigma\left(\boldsymbol{X}(\boldsymbol{W}_1^{(m)})^T + \mathbf{1}(\boldsymbol{b}^{(m)})^T\right)\boldsymbol{W}_2^{(m)} + \mathbf{1}\boldsymbol{b}_2^T\right], \tag{10}$$

which additionally introduces a factor p_I/k since expectations rather than sums are involved in MMD.

A.3. NTK preservation

In the main text, we have made the assumption that $\boldsymbol{C}\widetilde{\boldsymbol{W}} \quad \boldsymbol{W}$ and $\widetilde{\boldsymbol{H}}_C \quad \boldsymbol{H}$. This assumption implies that $r_{\boldsymbol{H}} f$ can be preserved by $r_{\widetilde{\boldsymbol{H}}_C} f_c$, which helps obtain $h\, r_{\boldsymbol{b}_2} f(\boldsymbol{X}), \mathrm{sign}(r_{\boldsymbol{b}_2} f(\boldsymbol{Z}))i \quad \left\langle r_{\tilde{\boldsymbol{b}}_2} f_c(\boldsymbol{X}), \mathrm{sign}\left(r_{\tilde{\boldsymbol{b}}_2} f_c(\boldsymbol{Z})\right)\right\rangle$.

For the remaining three term, we first address ① := $\left\langle \nabla_{\widetilde{\boldsymbol{W}}_2} f_c(\boldsymbol{X}), \operatorname{sign}\left(\nabla_{\widetilde{\boldsymbol{W}}_2} f_c(\boldsymbol{Z})\right)\right\rangle$:

$$
\begin{aligned}
① &= \operatorname{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \tilde{\boldsymbol{\sigma}}_x \boldsymbol{P} \ \operatorname{sign}\left(\boldsymbol{P}\tilde{\boldsymbol{\sigma}}_z^T \nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right)\right] \\
&= \operatorname{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\boldsymbol{P} \ \operatorname{sign}\left(\boldsymbol{P}^T\sigma\left(\widetilde{\boldsymbol{W}}_1^T\boldsymbol{Z}^T + \tilde{\boldsymbol{b}}_1\mathbf{1}^T\right)\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right)\right] \\
&\overset{(i)}{=} \operatorname{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\boldsymbol{C}\boldsymbol{C}^T \operatorname{sign}\left(\sigma\left(\widetilde{\boldsymbol{W}}_1^T\boldsymbol{Z}^T + \tilde{\boldsymbol{b}}_1\mathbf{1}^T\right)\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right)\right] \\
&\overset{(ii)}{=} \operatorname{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1\boldsymbol{C} + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\boldsymbol{C}\right)\operatorname{sign}\left(\sigma\left(\boldsymbol{C}^T\widetilde{\boldsymbol{W}}_1^T\boldsymbol{Z}^T + \boldsymbol{C}^T\tilde{\boldsymbol{b}}_1\mathbf{1}^T\right)\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right)\right] \\
&\quad \operatorname{Tr}\left[\left(\nabla_{\boldsymbol{H}} f(\boldsymbol{X})\right)^T \sigma\left(\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}\boldsymbol{b}_1^T\right)\operatorname{sign}\left(\sigma\left(\boldsymbol{W}_1^T\boldsymbol{Z}^T + \boldsymbol{b}_1\mathbf{1}^T\right)\nabla_{\boldsymbol{H}} f(\boldsymbol{Z})\right)\right] \\
&= \left\langle \nabla_{\boldsymbol{W}_2} f(\boldsymbol{X}), \operatorname{sign}\left(\nabla_{\boldsymbol{W}_2} f(\boldsymbol{Z})\right)\right\rangle,
\end{aligned}
$$

where equation (i) above holds since $\boldsymbol{P} = \boldsymbol{C}\boldsymbol{C}^T$ and the positive diagonal matrix $\boldsymbol{P}$ will not impact the sign of the matrix elements; as for equation (ii), the "copy" matrix $\boldsymbol{C}$, as discussed in Section 4.2, is free to be brought inside both the sign function and the activation function.

For $\left\langle \nabla_{\boldsymbol{W}_1} f(\boldsymbol{X}), \operatorname{sign}\left(\nabla_{\boldsymbol{W}_1} f(\boldsymbol{Z})\right)\right\rangle$, we need to verify the product $\boldsymbol{X}^T\left[\left(\nabla_{\boldsymbol{H}} f(\boldsymbol{X})\boldsymbol{W}_2^T\right) \quad \boldsymbol{\sigma}_x^{\prime}\right] \operatorname{sign}\left(\left[\left(\nabla_{\boldsymbol{H}} f(\boldsymbol{Z})\boldsymbol{W}_2^T\right) \quad \boldsymbol{\sigma}_z^{\prime}\right]^T \boldsymbol{Z}\right)$ can be approximated by ② := $\boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\boldsymbol{P}\right) \quad \tilde{\boldsymbol{\sigma}}_x^{\prime}\right] \operatorname{sign}\left(\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\widetilde{\boldsymbol{W}}_2^T\boldsymbol{P}\right) \quad \tilde{\boldsymbol{\sigma}}_z^{\prime}\right]^T \boldsymbol{Z}\right)$, where $\tilde{\boldsymbol{\sigma}}_x^{\prime} := \sigma^{\prime}\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)$, $\tilde{\boldsymbol{\sigma}}_z := \sigma^{\prime}\left(\boldsymbol{Z}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)$, and $\sigma^{\prime}(\,)$ is the derivative of the activation function $\sigma(\,)$ [2]. We show the derivation as follows:

$$
\begin{aligned}
② &\overset{(i)}{=} \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \quad \tilde{\boldsymbol{\sigma}}_x^{\prime}\right]\boldsymbol{P} \ \operatorname{sign}\left(\boldsymbol{P}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\widetilde{\boldsymbol{W}}_2^T\right) \quad \tilde{\boldsymbol{\sigma}}_z^{\prime}\right]^T \boldsymbol{X}\right) \\
&= \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \quad \tilde{\boldsymbol{\sigma}}_x^{\prime}\right]\boldsymbol{C}\boldsymbol{C}^T \ \operatorname{sign}\left(\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\widetilde{\boldsymbol{W}}_2^T\right) \quad \tilde{\boldsymbol{\sigma}}_z^{\prime}\right]^T \boldsymbol{X}\right) \\
&= \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\boldsymbol{C}\right) \quad (\tilde{\boldsymbol{\sigma}}_x^{\prime}\boldsymbol{C})\right] \ \operatorname{sign}\left(\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\widetilde{\boldsymbol{W}}_2^T\boldsymbol{C}\right) \quad (\tilde{\boldsymbol{\sigma}}_z^{\prime}\boldsymbol{C})\right]^T \boldsymbol{Z}\right) \\
&\quad \boldsymbol{X}^T\left[\left(\nabla_{\boldsymbol{H}} f(\boldsymbol{X})\boldsymbol{W}_2^T\right) \quad \boldsymbol{\sigma}_x^{\prime}\right] \ \left[\left(\nabla_{\boldsymbol{H}} f(\boldsymbol{Z})\boldsymbol{W}_2^T\right) \quad \boldsymbol{\sigma}_z^{\prime}\right]^T \boldsymbol{Z},
\end{aligned}
$$

in which we obtain equation (i) because $\boldsymbol{P}$ as a diagonal matrix has the same scaling effect on the Hadamard product $\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \quad \tilde{\boldsymbol{\sigma}}_x^{\prime}\right]$ as on one of its component $\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T$; the rest equations simply follow the previous derivations.

For the last term $\left\langle \nabla_{\boldsymbol{b}_1} f(\boldsymbol{X}), \operatorname{sign}\left(\nabla_{\boldsymbol{b}_1} f(\boldsymbol{Z})\right)\right\rangle$, we solely need to replace the above input matrix $\boldsymbol{X}, \boldsymbol{Z}$ with $\mathbf{1}^T$, and all the derivation steps will follow.

A.4. Model requirements for SGD NTK

To preserve the regular SGD NTK, the scale of the weight parameters needs to be adjusted. We re-define the efficient MLP model as ($f_c, \boldsymbol{\sigma}_x, \boldsymbol{\sigma}_z, \tilde{\boldsymbol{\sigma}}_x, \tilde{\boldsymbol{\sigma}}_z$ will also be accordingly re-defined):

$$
\widetilde{\boldsymbol{H}}_C := \sigma\left(\boldsymbol{X}\boldsymbol{W}_1^{(c)} + \mathbf{1}\left(\boldsymbol{b}_1^{(c)}\right)^T\right)\boldsymbol{W}_2^{(c)}, \tag{11}
$$

where $\boldsymbol{W}_1^{(c)} := \widetilde{\boldsymbol{W}}_1\boldsymbol{P}^{\frac{1}{2}}, \boldsymbol{b}_1^{(c)} := \boldsymbol{P}^{\frac{1}{2}}\tilde{\boldsymbol{b}}_1$ and $\boldsymbol{W}_2^{(c)} := \boldsymbol{P}^{\frac{1}{2}}\widetilde{\boldsymbol{W}}_1$ incorporate the diagonal scaling matrix $\boldsymbol{P}$ in Equation (5).

[2] For simplicity we assume the activation function $\sigma(\cdot)$ is differentiable everywhere.

We also require the activation function to have the following property:

$$\sigma(\boldsymbol{AP}) = \sigma(\boldsymbol{A})\boldsymbol{P},$$

for arbitrary non-negative diagonal matrix $\boldsymbol{P}$, which implies $\sigma(0) = 0$, $\sigma(\ )$ is piece-wise linear on R^+, R , and $\sigma^0(x)$ is piece-wise constant ($\sigma(1)$ on R^+ and $\sigma(\ \ 1)$ on R); as an instance, the commonly used Rectified Linear Units (ReLU) function (Fukushima, 1975) $\sigma_r(x) = \max\{0, x\}$ can satisfy this requirement.

Equation (11) can keep maintaining the approximation for $\boldsymbol{H}$ and still $\langle \nabla_{\boldsymbol{b}_2} f(\boldsymbol{X}), \nabla_{\boldsymbol{b}_2} f(\boldsymbol{Z})\rangle \approx \left\langle \nabla_{\boldsymbol{b}_2^{(c)}} f_c(\boldsymbol{X}), \nabla_{\boldsymbol{b}_2^{(c)}} f_c(\boldsymbol{Z})\right\rangle$, as we only modify the scale of the weight matrices. We then follow the derivation in the previous subsection and similar results are obtained

For the remaining three term, again we first address ① $:= \left\langle \nabla_{\boldsymbol{W}_2^{(c)}} f_c(\boldsymbol{X}), \nabla_{\boldsymbol{W}_2^{(c)}} f_c(\boldsymbol{Z})\right\rangle$:

$$\begin{aligned}
① &= \mathsf{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \tilde{\boldsymbol{\sigma}}_x \ \ \tilde{\boldsymbol{\sigma}}_z^T \nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right] \\
&= \mathsf{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \sigma\left(\boldsymbol{X}\boldsymbol{W}_1^{(c)} + \mathbf{1}\left(\boldsymbol{b}_1^{(c)}\right)^T\right) \ \ \sigma\left(\left(\boldsymbol{W}_1^{(c)}\right)^T \boldsymbol{Z}^T + \boldsymbol{b}_1^{(c)}\mathbf{1}^T\right) \nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right] \\
&\overset{(i)}{=} \mathsf{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\boldsymbol{P}^{\frac{1}{2}}\boldsymbol{P}^{\frac{1}{2}}\sigma\left(\widetilde{\boldsymbol{W}}_1^T\boldsymbol{Z}^T + \tilde{\boldsymbol{b}}_1\mathbf{1}^T\right) \nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right] \\
&= \mathsf{Tr}\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\right)^T \sigma\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\boldsymbol{C}\boldsymbol{C}^T\sigma\left(\widetilde{\boldsymbol{W}}_1^T\boldsymbol{Z}^T + \tilde{\boldsymbol{b}}_1\mathbf{1}^T\right) \nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\right] \\
&\quad \langle \nabla_{\boldsymbol{W}_2} f(\boldsymbol{X}), \nabla_{\boldsymbol{W}_2} f(\boldsymbol{Z})\rangle,
\end{aligned}$$

where equation (i) holds since $\boldsymbol{P}^{\frac{1}{2}}$, as we require, is free to be brought outside the activation function; the rest derivation simply follows the counterpart in Appendix A.3.

For $\langle \nabla_{\boldsymbol{W}_1} f(\boldsymbol{X}), \nabla_{\boldsymbol{W}_1} f(\boldsymbol{Z})\rangle$, we similarly need to verify the product $\boldsymbol{X}^T\left[\left(\nabla_{\boldsymbol{H}} f(\boldsymbol{X})\boldsymbol{W}_2^T\right) \ \ \boldsymbol{\sigma}^0\right]\left[\left(\nabla_{\boldsymbol{H}} f(\boldsymbol{Z})\boldsymbol{W}_2^T\right) \ \ \boldsymbol{\sigma}^0\right]^T\boldsymbol{X}$ can be approximated by ② $:= \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\left(\boldsymbol{W}_2^{(c)}\right)^T\right) \ \ \tilde{\boldsymbol{\sigma}}_x^0\right]\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\left(\boldsymbol{W}_2^{(c)}\right)^T\right) \ \ \tilde{\boldsymbol{\sigma}}_z^0\right]^T\boldsymbol{X}$.

$\tilde{\boldsymbol{\sigma}}_x^0 := \sigma^0\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right), \tilde{\boldsymbol{\sigma}}_z := \sigma^0\left(\boldsymbol{Z}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)$

We then show the derivation as follows:

$$\begin{aligned}
② &= \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\left(\boldsymbol{W}_2^{(c)}\right)^T\right) \ \ \tilde{\boldsymbol{\sigma}}_x^0\right] \ \left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\left(\boldsymbol{W}_2^{(c)}\right)^T\right) \ \ \tilde{\boldsymbol{\sigma}}_z^0\right]^T\boldsymbol{X} \\
&\overset{(i)}{=} \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \tilde{\boldsymbol{\sigma}}_x^0\right]\boldsymbol{P}^{\frac{1}{2}}\boldsymbol{P}^{\frac{1}{2}} \ \left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \tilde{\boldsymbol{\sigma}}_z^0\right]^T\boldsymbol{X} \\
&= \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \tilde{\boldsymbol{\sigma}}_x^0\right]\boldsymbol{C}\boldsymbol{C}^T \ \left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{Z})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \tilde{\boldsymbol{\sigma}}_z^0\right]^T\boldsymbol{X} \\
&= \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \sigma^0\left(\boldsymbol{X}\boldsymbol{W}_1^{(c)} + \mathbf{1}\left(\boldsymbol{b}_1^{(c)}\right)^T\right)\right]\boldsymbol{C}\boldsymbol{C}^T \\
&\qquad \left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \sigma^0\left(\boldsymbol{X}\boldsymbol{W}_1^{(c)} + \mathbf{1}\left(\boldsymbol{b}_1^{(c)}\right)^T\right)\right]^T\boldsymbol{X} \\
&\overset{(ii)}{=} \boldsymbol{X}^T\left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \sigma^0\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\right]\boldsymbol{C}\boldsymbol{C}^T \\
&\qquad \left[\left(\nabla_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X})\widetilde{\boldsymbol{W}}_2^T\right) \ \ \sigma^0\left(\boldsymbol{X}\widetilde{\boldsymbol{W}}_1 + \mathbf{1}\tilde{\boldsymbol{b}}_1^T\right)\right]^T\boldsymbol{X},
\end{aligned}$$

in which we obtain equation (i) because $\boldsymbol{P}^{\frac{1}{2}}$ as a diagonal matrix has the same scaling effect on the Hadamard product

$\left[\left(r_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X}) \left(\boldsymbol{W}_2^{(c)}\right)^T\right) \quad \tilde{\boldsymbol{\sigma}}_x^0\right]$ as on one of its component $r_{\widetilde{\boldsymbol{H}}_C} f_c(\boldsymbol{X}) \left(\boldsymbol{W}_2^{(c)}\right)^T$; equation (ii) holds because σ^0 is piece-wise constant and the scaling matrix $\boldsymbol{P}^{\frac{1}{2}}$ will not change the signs of the elements within. Then, following the same derivation as in the previous derivations, we have ② $\boldsymbol{X}^T \left[\left(r_{\boldsymbol{H}} f(\boldsymbol{X}) \boldsymbol{W}_2^T\right) \quad \boldsymbol{\sigma}^0\right] \left[\left(r_{\boldsymbol{H}} f(\boldsymbol{Z}) \boldsymbol{W}_2^T\right) \quad \boldsymbol{\sigma}^0\right]^T \boldsymbol{X}$

For the last term $h\, r_{b_1} f(\boldsymbol{X}),\ r_{b_1} f(\boldsymbol{Z}) i$, we can again replace the above input matrix $\boldsymbol{X}$ with $\mathbf{1}^T$, and all the derivation steps will follow.

B. Runtime of fine-tuning after PLM compression

Since our proposed MLP fusion only differs from the sketching and mmd baselines in initialization, we focus on the runtime evaluation of MLP fusion along with two representative methods, regular fine-tuning and pruning.

For a fair comparison, we intentionally run the two NLU tasks on a cluster server (so that no other processes will compete with the model fine-tuning) with one core of a server CPU (Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz) on Ubuntu 18.04. In this setting, we train the RoBERTa model for 100 steps with batch size 32.

Specifically, on SST2, it will take the model with MLP fusion, pruning, and regular fine-tuning around 6746, 18066, 9342 seconds to finish the training, respectively; on MNLI, the time cost is around 6956, 17060, 18966 seconds for the training. We remark the architecture of MLP fusion can accelerate the regular fine-tuning by 30% on SST2, and is even 2.7 times faster in MNLI, which has longer average sequence length. As for pruning, although it has a comparable prediction performance in the two tasks, its time cost is no less than regular fine-tuning and is much higher in the more lightweight task SST2, due to some overhead cost from its implementation.

C. Bounding the error of MLP output

Recall the object of clustering is:

$$\min_C k\mathbf{W} \quad \mathbf{C}^T\widetilde{\mathbf{W}}k_F^2$$

The assumption can thus be rewritten in a mathematical manner, which is $k\mathbf{W} \quad \mathbf{C}^T\widetilde{\mathbf{W}}k_F \quad \varepsilon$ with small ε. Assuming $f(\mathbf{W}, \mathbf{C}^\mathbf{T}\widetilde{\mathbf{W}}) = k\mathbf{W} \quad \mathbf{C}^\mathbf{T}\widetilde{\mathbf{W}}k_F \quad \varepsilon$, we can provide a standard analysis of MLP output (ignoring b_1 for simplicity) as follows.

Denoting $:= \sigma(\mathbf{X}\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C}) \quad \sigma(\mathbf{X}\mathbf{W}_\mathbf{1})$ and following the technical assumptions in [4] that $k\mathbf{W}_\mathbf{1}k_2 \quad C_1, k\mathbf{W}_\mathbf{2}k_2 \quad C_2, k\mathbf{X}k_F \quad C_X$ and the activation function $\sigma(\ )$ is L-Lipschitz continuous. Further assuming $\sigma(0) = 0$ (the assumptions holds for commonly used activation functions in PLMs, e.g., ReLU and GELU), we first have

$$k \quad k_F \quad Lk\mathbf{X}\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C} \quad \mathbf{X}\mathbf{W}_\mathbf{1}k_F \quad Lk\mathbf{X}k_Fk\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C} \quad \mathbf{W}_\mathbf{1}k_F \quad LC_X\varepsilon.$$

We can then bound $k\mathbf{H} \quad \widetilde{\mathbf{H}}_\mathbf{C}k_F$ as

$$\begin{aligned}
k\sigma(\mathbf{X}\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C})\mathbf{C}^\mathbf{T}\widetilde{\mathbf{W}}_\mathbf{2} \quad \sigma(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2k_F \quad & k\sigma(\mathbf{X}\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C})(\mathbf{C}^\mathbf{T}\widetilde{\mathbf{W}}_\mathbf{2} \quad \mathbf{W}_\mathbf{2})k_F + k(\sigma(\mathbf{X}\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C}) \quad \sigma(\mathbf{X}\mathbf{W}_1))\mathbf{W}_2k_F \\
& k\sigma(\mathbf{X}\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C})k_Fk\mathbf{C}^\mathbf{T}\widetilde{\mathbf{W}}_\mathbf{2} \quad \mathbf{W}_\mathbf{2}k_F + k\sigma(\mathbf{X}\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C}) \quad \sigma(\mathbf{X}\mathbf{W}_\mathbf{1})k_Fk\mathbf{W}_2k_2 \\
=& k \quad + \sigma(\mathbf{X}\mathbf{W}_\mathbf{1})k_Fk\mathbf{C}^\mathbf{T}\widetilde{\mathbf{W}}_\mathbf{2} \quad \mathbf{W}_\mathbf{2}k_F + k \quad k_Fk\mathbf{W}_2k_2 \\
& (k \quad k_F + k\sigma(\mathbf{X}\mathbf{W}_\mathbf{1})k_F) \quad \varepsilon + k \quad k_F \quad C_2 \\
& L(C_2C_X + C_1C_X) \quad \varepsilon + LC_X \quad \varepsilon^2,
\end{aligned}$$

where we utilize

$$\begin{aligned}
k\widetilde{\mathbf{W}}_\mathbf{1}\mathbf{C} \quad \mathbf{W}_\mathbf{1}k_F, k\mathbf{C}^\mathbf{T}\widetilde{\mathbf{W}}_\mathbf{2} \quad \mathbf{W}_\mathbf{2}k \quad k\mathbf{W} \quad \mathbf{C}^T\widetilde{\mathbf{W}}k_F \quad \varepsilon \\
k\sigma(\mathbf{X}\mathbf{W}_\mathbf{1})k_F = k\sigma(\mathbf{X}\mathbf{W}_\mathbf{1}) \quad \sigma(\mathbf{0})k_F \quad Lk\mathbf{X}\mathbf{W}_\mathbf{1}k_F \quad Lk\mathbf{X}k_Fk\mathbf{W}_\mathbf{1}k_2 = LC_1C_X,
\end{aligned}$$

for the derivation. From the bound, we can observe with the well-learned $\mathbf{C}^T\widetilde{\mathbf{W}}$ from clustering (small ε), the output error ($k\mathbf{H} \quad \widetilde{\mathbf{H}}_\mathbf{C}k_F$) will also be small.

Moreover, a similar error analysis can also be applied to Adam NTK. The error analysis $kK \quad \widetilde{K}_{\mathbf{C}}k_F$ of NTK kernel K can be simplified as analyzing $\mathsf{Tr}[\mathbf{A}^{\mathbf{T}}\mathsf{sign}(\mathbf{B}) \quad \widetilde{\mathbf{A}}^{\mathbf{T}}\mathsf{sign}(\widetilde{\mathbf{B}})]$ where $\mathbf{A}$, $\mathbf{B}$ represent arbitrary matrices. The precise derivation of the approximation error bound on Adam NTK, considering the additional assumption on $j\mathsf{sign}(\mathbf{B}) \quad \mathsf{sign}(\widetilde{\mathbf{B}})j_F$ as described in (Balles & Hennig, 2018), is left as future work.

D. Discussions

We are not aware of any potential negative societal impacts regarding our work to the best of our knowledge. For all the used data sets, there is no private personally identifiable information or offensive content.

Regarding future work, beyond the combination with distillation, we also plan to explore practical compression methods in various domains, including speech processing (Dong et al., 2018), recommender system (Geng et al., 2022; Wei & He, 2022), and graph mining (Ying et al., 2021; Wei et al., 2022b). The derivation of a more precise error analysis with regard to the pre-trained model is also a challenging and promising direction.

E. Performance Comparison of Representative Methods after Task-specific Fine-tuning

Table 5: Accuracy of Representative Methods After Task-specific Fine-tuning on SST2 Validation Set

Method+Task-specific Fine-tuning	Accuracy
Sketch	91.86
Clustering	93.35
MMD	92.43
SVD	93.01
LTH	93.42
MLP Fusion(Ours)	**93.79**

From Table 5 and Table 2 in the paper, we can observe that not all methods can benefit from the layer-wise task-specific tuning module. For example, the accuracy of the Sketch method drops from 91.90 to 91.86. Meanwhile, our proposed MLP Fusion provides a promising starting point for subsequent optimization through NTK approximation. By incorporating a layer-wise task-specific tuning module, we can further enhance its performance and still achieve the best results compared to all other baseline methods.

F. Experiment Results on More Benchmark Datasets

Table 6: Accuracy of Each Baseline Method on STS-B And QNLI Validation Sets with RoBERTa as The PLM

Method	STS-B(7k)	QNLI(105k)
Sketch	86.99	89.84
Clustering	88.12	90.63
LTH	87.37	90.87
RoBERTa	91.20	92.80
DistilRoBERTa	88.30	90.80
MLP Fusion(Ours)	**89.37**	**91.03**

Table 6 shows the experimental results on two additional data sets QNLI and STS-B within the GLUE benchmark. We can see the proposed method is still able to achieve the best performance over strong baselines.

G. Performance Comparison between Proposed Method and Maintaining Output/NTK of The MLP Model

Table 7: Performance of Proposed Method and Maintaining Output/NTK of The MLP Model

Methods	Accuracy
Maintain NTK Gradient	91.74
Maintain Output	92.35
MLP Fusion (Ours)	93.23

In Table 7, we compare two additional baselines that try to maintain the MLP output and NTK of the pre-trained language model. Our NTK approximation method MLP Fusion still achieves the best performance. The loss that attempts to maintain the output with unsupervised data ranked second. The method that tries to maintain the NTK of the MLP model with gradient has the lowest accuracy. This is mainly because the gradient difference in the loss is difficult to minimize since it requires operating second-order derivatives, which can also be time-consuming. Additionally, gathering labeled data for the loss calculation can also be burdensome.