Appendix

A Trainability and Generalization

A.1 Trainability

Following the previous work (Jacot et al., 2018) training neural networks in function space instead of parameter space, we analyze the trainability of an over-parameterized model by investigating the evolution of its predictions. Specifically, after a step of gradient descent

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}, \tag{7}$$

the updated predictions can be approximated using the first-order Taylor expansion as

$$\begin{aligned} f(\mathcal{X}; \boldsymbol{\theta}_{t+1}) &= f(\mathcal{X}; \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}) \\ &\approx f(\mathcal{X}; \boldsymbol{\theta}_t) - \eta \nabla_{\boldsymbol{\theta}_t} f(\mathcal{X}; \boldsymbol{\theta}_t)^\top \nabla_{\boldsymbol{\theta}_t} \mathcal{L} \\ &= f(\mathcal{X}; \boldsymbol{\theta}_t) - \eta \nabla_{\boldsymbol{\theta}_t} f(\mathcal{X}; \boldsymbol{\theta}_t)^\top \nabla_{\boldsymbol{\theta}_t} f(\mathcal{X}; \boldsymbol{\theta}_t) \nabla_{\mathcal{Z}} \mathcal{L} \\ &= f(\mathcal{X}; \boldsymbol{\theta}_t) - \eta \boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) \nabla_{\mathcal{Z}} \mathcal{L}. \end{aligned} \tag{8}$$

Motivated by Arora et al. (2019), we rewrite the equation into the form of norm and eigenpairs:

$$\begin{aligned} \|f(\mathcal{X}; \boldsymbol{\theta}_{t+1}) - f(\mathcal{X}; \boldsymbol{\theta}_t)\|_2^2 &\approx \|\eta \boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) \nabla_{\mathcal{Z}} \mathcal{L}\|_2^2 \\ &= \eta^2 (\boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) \nabla_{\mathcal{Z}} \mathcal{L})^\top (\boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) \nabla_{\mathcal{Z}} \mathcal{L}) \\ &= \eta^2 \nabla_{\mathcal{Z}} \mathcal{L}^\top \boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) \boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) \nabla_{\mathcal{Z}} \mathcal{L} \\ &= \eta^2 \nabla_{\mathcal{Z}} \mathcal{L}^\top (\sum_i^{nk} \lambda_i \mathbf{u}_i \mathbf{u}_i{}^\top)(\sum_i^{nk} \lambda_i \mathbf{u}_i \mathbf{u}_i{}^\top) \nabla_{\mathcal{Z}} \mathcal{L} \\ &\overset{(i)}{=} \eta^2 \nabla_{\mathcal{Z}} \mathcal{L}^\top (\sum_i^{nk} \lambda_i^2 \mathbf{u}_i \mathbf{u}_i{}^\top \mathbf{u}_i \mathbf{u}_i{}^\top) \nabla_{\mathcal{Z}} \mathcal{L} \\ &\overset{(ii)}{=} \eta^2 \nabla_{\mathcal{Z}} \mathcal{L}^\top \sum_i^{nk} \lambda_i^2 \mathbf{u}_i \mathbf{u}_i{}^\top \nabla_{\mathcal{Z}} \mathcal{L} \\ &= \eta^2 \sum_i^{nk} \lambda_i^2 (\mathbf{u}_i{}^\top \nabla_{\mathcal{Z}} \mathcal{L})^\top \mathbf{u}_i{}^\top \nabla_{\mathcal{Z}} \mathcal{L} \\ &= \eta^2 \sum_i^{nk} \lambda_i^2 (\mathbf{u}_i{}^\top \nabla_{\mathcal{Z}} \mathcal{L})^2, \end{aligned} \tag{9}$$

where (i) follows $\mathbf{u}_i{}^\top \mathbf{u}_j = 0, \forall i \neq j$ and (ii) follows $\mathbf{u}_i{}^\top \mathbf{u}_i = \|\mathbf{u}_i\|^2 = 1$. Then we can derive Eq. 9 into:

$$\|f(\mathcal{X}; \boldsymbol{\theta}_{t+1}) - f(\mathcal{X}; \boldsymbol{\theta}_t)\|_2 = \sqrt{\sum_{i=1}^{nk} (\eta \lambda_i \mathbf{u}_i{}^\top \nabla_{\mathcal{Z}} \mathcal{L})^2}. \tag{10}$$

As we can see, at every step of the gradient descent, the model learns the target function faster along the eigen-directions corresponding to the larger eigenvalues.

Further, for the loss function assumed by squared error loss, we characterize the loss reduction by the following directional derivative (Wang et al., 2020):

$$\Delta \mathcal{L}(\boldsymbol{\theta}) = \lim_{\epsilon \to 0} \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})) - \mathcal{L}(\boldsymbol{\theta})}{\epsilon}, \tag{11}$$

using Taylor's first-order approximation $\mathcal{L}(\boldsymbol{\theta} + \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})) \approx \mathcal{L}(\boldsymbol{\theta}) + \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$,

$$\Delta \mathcal{L}(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{12}$$

expanded by chain rule $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta})$ into

$$\begin{aligned} \Delta \mathcal{L}(\boldsymbol{\theta}) &= (\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta}))^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta}) \\ &= \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta})^\top (\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})) \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta}), \end{aligned} \tag{13}$$

then following Eq. 1 and matrix decomposition of mNTK,

$$\begin{aligned} \Delta \mathcal{L}(\boldsymbol{\theta}) &= \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta})^\top (\boldsymbol{\Theta}) \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta}) \\ &= \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta})^\top (\sum_{l=1}^{L} \boldsymbol{\Theta}^l) \nabla_{\mathcal{Z}} \mathcal{L}(\boldsymbol{\theta}) \\ &= \mathcal{Y}^\top (\sum_{l=1}^{L} \sum_{i=1}^{nk} \lambda_i^l \mathbf{u}_i^l \mathbf{u}_i^{l\top}) \mathcal{Y} \\ &= \sum_{l=1}^{L} \sum_{i=1}^{nk} \lambda_i^l (\mathbf{u}_i^{l\top} \mathcal{Y})^\top (\mathbf{u}_i^{l\top} \mathcal{Y}) \\ &= \sum_{l=1}^{L} \sum_{i=1}^{nk} \lambda_i^l \left(\mathbf{u}_i^{l\top} \mathcal{Y} \right)^2. \end{aligned} \tag{14}$$

The directional derivative of the loss function is closely related to the eigenspectrum of mNTKs. As for the latter projection item $\left(\mathbf{u}_i^{l\top} \mathcal{Y} \right)^2$, Arora et al. (2019) have studied the relationship between the projection norm and labels, and they demonstrate that true labels generate better alignment with top eigenvectors, especially for the maximum one. Therefore, we can assume that $\left(\mathbf{u}_i^{l\top} \mathcal{Y} \right)^2 \propto \lambda_i^l$ in a real dataset.

A.2 Generalization

We denote the set of n training instances as $\mathcal{X} = (\mathbf{x}, \ldots, \mathbf{x}_n)$ and the target set as $\mathcal{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)^\top$. A two-layer neural network with the ReLU activation function can be written as

$$f_{\boldsymbol{\theta}, \boldsymbol{a}}(\mathcal{X}) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma \left(\boldsymbol{\theta}_r^\top \mathcal{X} \right) \tag{15}$$

where $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_m$ are the weight vectors in the first layer, $a_1, a_2, ..., a_m$ are weights in the second layer. For simplicity, we denote $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m)$ and $\boldsymbol{a} = (a_1, \ldots, a_m)^\top$. Following (Arora et al., 2019), we assume the network is over-parameterized and is trained by gradient descent on the quadratic loss over dataset $(\mathcal{X}, \mathcal{Y})$. We freeze the second layer $\boldsymbol{a}$ and optimize the first layer $\boldsymbol{\theta}$ through gradient descent. Let $\boldsymbol{\theta}(0)$ and $\boldsymbol{\theta}(t)$ denote the weights initialized from scratch and the weights after t iterations, respectively.

Under these settings, according to the Lemma 5.3 of (Arora et al., 2019), the embedding function $f_{\boldsymbol{\theta}, \boldsymbol{a}}$ learned from GD is in a restricted class of neural networks with weights close to initialization $\boldsymbol{\theta}(0)$. Therefore, the Rademacher complexity of the two-layer function can be bounded as follows:

Lemma 2 *Given $R > 0$, with probability at least $1 - \delta$ over the random initialization $(\boldsymbol{\theta}(0), \boldsymbol{a})$, simultaneously for every $B > 0$, the following function class*

$$\begin{aligned} \mathcal{F}_{R,B}^{\boldsymbol{\theta}(0), \boldsymbol{a}} = \{ f_{\boldsymbol{\theta}, \boldsymbol{a}} : &\| \theta_r - \theta_r(0) \|_2 \leq R \ (\forall r \in [m]), \\ &\| \boldsymbol{\theta} - \boldsymbol{\theta}(0) \|_F \leq B \} \end{aligned} \tag{16}$$

has empirical Rademacher complexity bounded as:

$$\begin{aligned}\mathcal{R}_S\left(\mathcal{F}_{R,B}^{\boldsymbol{\theta}(0),\boldsymbol{a}}\right) &= \frac{1}{n}\mathbb{E}_{\boldsymbol{\varepsilon}\in\{\pm 1\}^n}\left[\sup_{f\in\mathcal{F}_{R,B}^{\boldsymbol{\theta}(0),\boldsymbol{a}}}\sum_{i=1}^{n}\varepsilon_i f\left(\mathbf{x}_i\right)\right] \\ \leq &\frac{B}{\sqrt{2n}}\left(1+\left(\frac{2\log\frac{2}{\delta}}{m}\right)^{1/4}\right)+\frac{2R^2\sqrt{m}}{\kappa}+R\sqrt{2\log\frac{2}{\delta}}.\end{aligned} \tag{17}$$

Lemma 2 indicates that the Rademacher complexity is proportional to the weight distance from its initialization. Specifically, a greater weight distance implies a more significant amount of Rademacher complexity and is thus associated with weaker generalization ability. Following those intuitions, we extend the connection between weight distance and Rademacher complexity, initially portrayed in a two-layer network, into more generalized deep neural networks.

For deep models, as mentioned in (Hoffer et al., 2017), the weight distance from its initialization grows logarithmically with the number of iterations, which can be described as

$$\|\boldsymbol{\theta}(t)-\boldsymbol{\theta}(0)\|_F \propto \log t. \tag{18}$$

Combining Lemma 2 and Eq. 18, we can discover that as training iterations increase, the model's Rademacher complexity also grows with its weights more deviated from initializations, which subsequently impairs the model's generalization ability. However, solutions remain. As trainability varies across various modules, we can prevent the significant growth of Rademacher complexity and thus retain a satisfying model generalization by ignoring the updates of those modules with undesirable mNTK eigenvalues.

B Experiments

B.1 Experimental Settings

BERT and Switch-Transformer. We generally follow the settings of Liu et al. (2019) to train BERT and Switch-Transformer from scratch using self-supervised Masked Language Modeling (MLM). In detail, we uniformly sample approximately 15% input tokens to replace them with a mask, and the learning objective is to predict those masked tokens accurately with a cross-entropy loss.

All experiments are conducted on 8 $\times$ NVIDIA GeForce RTX 3090 GPUs, and we list those key hyperparameters in Table 4.

Table 4: Hyperparameters configuration in BERT and Switch-Transformer

Hyperparameter	BERT	Switch-Transformer
Number of layers	12	12
Attention heads	12	12
Hidden dimension size	768	768
Dropout	0.1	0.1
Attention dropout	0.1	0.1
Sequence length	512	512
Batch size	8	8
Warmup steps	0	12k
Max steps	12k	300k
Weight decay	0	0.01
Peak learning rate	2e-4	1e-4
Learning rate decay	Linear	Cosine
Adam $[\epsilon, \beta_1, \beta_2]$	[1e-6, 0, 0]	[1e-6, 0.9, 0.99]
Number of experts		4
Capacity factor		1.5

The baseline methods consist of: 1) the vanilla model, 2) the Multirate (Vlaar & Leimkuhler, 2022) algorithm, and 3) the adaptive training with a random selection of the updated modules (abbreviated as Rand).

For BERT, Multirate randomly partitions the heads into fast and slow groups into a $50\%/50\%$ split, with fast heads updated every step of stepsize η and slow ones updated every $k = 5$ step of stepsize $k\eta$.

BERT-Rand randomly selects and updates 50% heads in every epoch. MAT first randomly samples 64 data samples to approximate 64 eigenvalues for each head-based mNTK. The modular policy and temporal policy are set with $\alpha = 0.1, \beta = 10^{-3}$. The adaptive training algorithm starts from the 5^{th} epoch and performs every 8 epochs.

For Switch-Transformer, Multirate considers each head/expert as a single module and partitions heads/experts into fast and slow sets at $50\%/50\%$ ratios. The fast heads and experts are updated every step with current stepsize η, and the slow ones are updated every $k = 5$ steps with stepsize $k\eta$. Switch-Rand randomly selects 50% heads or experts for updating at the beginning of each epoch. MAT first randomly samples 64 data samples and separately approximate 64 eigenvalues for each head-based mNTK and expert-based mNTK. The modular policy and temporal policy are set with $\alpha_h = 0.1, \beta_h = 10^{-3}$ and $\alpha_e = 0.2, \beta_e = 10^{-3}$ for head and expert, respectively. The adaptive training algorithm performs every 2 epochs after warming up.

To avoid the case that no heads or experts are updated in any layer, we utilize a protecting strategy that maintains the gradient of at least one head or expert in each layer for all baselines and our model.

VGG. All baselines of VGG are initialized with Kaiming initialization (He et al., 2015) and are trained with SGD for 200 epochs with an initial learning rate of 0.1 and batch size of 128. We decay the learning rate by 0.1 at $1/2$ and $3/4$ of the total number of epochs. Specifically, Multirate randomly partitions the filters into fast and slow parts by $50\%/50\%$ in each layer. VGG-Rand randomly selects 50% filters to update filters in each epoch. MAT approximates eigenvalues for each filter-based mNTK using random 64 data samples. The modular policy and temporal policy are set with $\alpha = 0.2, \beta = 10^{-6}$. All experiments are repeated by three runs and the final computation costs are calculated on average.

B.2 MAT and Network Pruning

Network pruning (Frankle & Carbin, 2018; Sanh et al., 2020; Liu et al., 2021) applies various criteria to determine the importance of different components and prunes those that are most redundant to compress model size, which often results in slight performance drops. The proposed MAT distinguishes from network pruning in that MAT only sparsifies modules during backward propagation, while pruning methods eliminate the forward and backward propagations of pruned modules. Besides, MAT is the first work to employ the principal eigenvalue of mNTK as the module selection criterion.

Our empirical study reveals some modules that are never or rarely selected by the proposed adaptive training method (MAT), showing potential for being pruned to achieve further computation savings. However, due to the complete removal of modules, existing network pruning methods may negatively impact the model's performance and generalizations. Based on those observations, we have explored whether we can apply MAT to network pruning methods to accelerate the training process or to improve performance.

We employ a BERT model for this experiment, and we prune 50% attention heads according to the ranking of $\lambda_{\max}$ of head-based mNTKs at the 15^{th} epoch. Across the training session, we use MAT with $\alpha = 0.2$ to introduce sparsity in the backward pass, resulting in approximately 25% sparsity of weight gradients.

Table 5: Results of BERT on WikiText-2 by pruning methods.

Method	Sparsity (Pruning Ratio)		Test Loss (Log PPL)	Computation (PFLOPs)
	Forward Pass	Backward Pass	@ Convergence	
Vanilla	0%	0%	4.41	27.80
SNIP (50%)	50%	50%	4.39	19.02
SNIP (75%)	75%	75%	4.70	15.22
MAT	50%	$\sim$75%	**4.32**	**12.03**

Table 5 compares the extended MAT, the vanilla BERT model, and SNIP (Lee et al., 2018b) in terms of forward and backward sparsity.

SNIP is a widely used pruning method that operates based on connection sensitivity, enabling sparsity in over-parameterized models. In our implementation, we apply SNIP in a modular manner by calculating the connection sensitivity of each module. As shown in the Table, SNIP achieves a 31.6% reduction in computation when pruning 50% of the attention heads without any performance degradation. However, when the pruning ratio (sparsity) is increased to 75%, SNIP fails to achieve comparable performance with the vanilla model. This suggests that a large ratio of sparsity can have a negative impact on model performance.

In contrast, using the criteria of MAT, we prune 50% of the attention heads while training the remaining ones by MAT. This approach leads to a further acceleration of computations by 56.7% while slightly improving the overall performance. This experiment serves as an example highlighting the potential of MAT in network pruning, showcasing the trade-off between computation savings and performance maintenance.

B.3 Computational Complexity and Overhead of MAT

The proposed MAT can introduce additional computational and memory overheads as it involves the calculation of mNTKs and their principal eigenvalues. However, we demonstrate in this subsection that MAT only yields a negligible proportion of extra computations, and we also report the numeric overhead results from experiments to support this claim.

To clarify, we have employed two strategies to accelerate the computation of mNTKs significantly: (1) sampling a subset of size $S(\ll n)$ for the NTK approximation instead of using the entire set, and (2) computing the gradient using the sum-of-logits approach instead of considering all output units. With those strategies, we approximate the Jacobian matrix $J \in \mathbb{R}^{nk\times m}$ with the approximated Jacobian matrix $\widetilde{J} \in \mathbb{R}^{S\times m}$, and we only need to perform S additional gradient propagations and concatenate the gradients together. mNTKs are then computed with the approximated Jacobian matrix, and we perform the eigen-decomposition to mNTKs to obtain the principal eigenvalue.

Apart from those techniques mentioned above, the module division strategy also accelerates the matrix multiplication of the Jacobian. Unlike the integral NTK, MAT applies a lightweight NTK estimation by modular NTK that significantly reduces the computation time required, and this estimation can be scalable to deeper structured networks. The complexity of computing integral NTK is $O(Sm^2)$; however, the overall time complexity reduces to $O(LS(m/L)^2) = O(Sm^2/L)$ in MAT, assuming we are computing L mNTKs. As for the singular value decomposition (SVD), since $S \ll n \ll m$, its complexity, $O(LS^3)$, can be far lower than others. Table 6 illustrates the comparison of computational complexities, showcasing MAT's significant computational advantages for NTK approximation. In short, **the overhead produced by MAT is negligible**.

Table 6: Complexity comparison, where n denotes the number of training data, m the number of parameters, k the output dimension, L the number of components, and S the sample number.

Complexity	Full	Our Approximation
NTK computaion	$O(nkm^2)$	$O(Sm^2/L)$
SVD computaion	$O(n^3k^3)$	$O(LS^3)$

We also measure the actual computation overhead by MAT. Following the experimental setting of Turc et al. (2019), we apply the proposed MAT to BERT models with different network scales, namely BERT-Mini (L=4, H=256), BERT-Small (L=4, H=512), BERT-Medium (L=8, H=512), BERT-Base (L=12, H=768), and BERT-Large (L=24, H=1024). Table 7 demonstrates the computational costs across varying multi-scale BERT models, and we can see that MAT can save 26.4%, 33.4%, 39.4%, 40.6%, and 50.9% computations for BERT-Mini, BERT-Small, BERT-Medium, BERT-Base, and BERT-Large, respectively. These observations indicate that applying MAT to larger models can better improve training efficiency. Notably, when compared to the overall computational costs, the overheads introduced by MAT are extremely small and can be arguably ignored.

Table 7: The computation (PFLOPs) required for training to convergence of models of different scale on WikiText-2.

Model	BERT-Mini	BERT-Small	BERT-Meduim	BERT-Base	BERT-Large
Vanilla computation	1.44	4.28	9.52	27.80	61.25
MAT computation	1.06	2.85	5.77	16.50	30.09
MAT overhead	0.01	0.04	0.08	0.24	0.45