Applications of Neural Tangent Kernel Theory in Deep Learning

by Xupeng Shi

B.S. in Sciences, Zhejiang University
M.S. in Sciences, Chinese Academy of Sciences

A dissertation submitted to

The Faculty of
the College of Science of
Northeastern University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

April 8, 2022

Dissertation Committee

Prof. Aidong Adam Ding, Chair

Prof. Paul Hand

Prof. Christopher King

Prof. Luo Luo

Prof. Ting Zhou

Dedication

To My Family

Acknowledgments

I would like first to owe my deepest gratitude to my advisor, Prof. Aidong Adam Ding. I have learned a lot from the numerous discussions with him in these years. His consistent help makes the graduate study not that hard. The occasional discussion of non-academic topics is also quite cheering. And I would also like to give thanks to my PhD dissertation defense committee members, Prof. Paul Hand, Prof. Christopher King, Prof. Luo Luo and Prof. Ting Zhou.

I owe my special thanks to my esteemed friend Dr. Weizhong Zhang. When I was depressed in the summer of 2018 , he introduced me the area of deep learning and we are collaborators since then. This makes the degree completion possible. Also I would like to give thanks to Pengfei Zheng and Zonghao Chen, who helped a lot in the coding, which is not my expertise. Thanks also to my other collaborators, including but not limited to Dr. Jiaxing Wang, Dr. Haoli Bai, etc.

In the meantime, I would like to thank all my colleagues and friends in the department. The department offers nice pizza in the graduate student seminar, where I learned a lot from other PhD students who gave presentations in the seminar. I also heard a lot of interesting stories and tales from Ruosen Xiong.

I also wish to thank Prof. Shiu-Chun Wong for his organization of the undergraduate student seminars in Zhejiang University, where I made a lot of precious friends and learned most of my knowledge in pure math as an undergraduate student. Since I came to USA, I enjoyed a lot from the discussion of various academic and non-academic topics with Prof. Xiping Zhang, Dr. Xuntao Hu and Dr. Gong Cheng.

Personally I would like to thank my other friends: Dr. Rui Zhang for his help for the summer intern and job interviews. Dr. Pengxiang Cui, as my best friend, shows me around the city of Shenzhen and introduces me so many new friends. And Dr. Li Tang for the nice weekends last summer. All these make the graduate life better than expected.

Finally, I am deeply indebted to my parents for their respect of my choices in each stage of my life,

even though sometimes they might not be able to understand why. This dissertation is specially dedicated to my grandfather, who taught me the basics of arithmetic but passed away in a sudden in Spring 2018.

Abstract of Dissertation

Deep neural networks (DNNs) have been shown to succeed at learning complex tasks and are widely used as state-of-the-art machine learning classification systems due to the great performance gains in recent years. The study of the training dynamics of infinitely width neural network shows that the neural network outputs are completely described by the so called *neural tangent kernel* (NTK). Based on recent theoretical works of NTK, we study in this dissertation the applications of NTK in deep learning problems. We first study the regularization by function norm induced by NTK. We prove such regularization induces an equivalence of neural network training and kernel ridge regression under NTK. We further show that our method achieve better performance than other function norm regularization methods on various of tasks. The second application is based on the study of dynamics of adversarial training. We propose a method of finding sparse network structure, which preserves the dynamics of adversarial training hence can be trained to be robust. The search of such robust sparse network structures provides theoretical and empirical evidence for lottery ticket hypothesis in the adversarial context.

Table of Contents

List of Figures

List of Tables

Disclaimer

I hereby declare that the work in this thesis is that of the candidate alone, except where indicated in the text. Further, I have acknowledged all sources used and have cited these in the reference section.

_____-

Introduction

Deep neural networks (DNNs) have been shown to succeed at learning complex tasks when a large number of training data is available and are widely used as the state-of-art machine learning classification systems due to the great performance gains in recent years. However, it is also shown that DNNs can easily overfit [92] and are unstable under small perturbations on inputs [87]. Recent theoretical studies [39, 50] have shown that the training dynamics of neural networks can be described by the so-called *neural tangent kernel* (NTK). Since the training dynamics describes how neural networks evolve as the training goes on, it is possible to analyze the generalizability and adversarial robustness of DNNs with NTK theory. Therefore, in this dissertation, we attempt to discuss two applications of NTK theory in improving the generalizability and adversarial robustness of DNNs.

Firstly, in areas where data is scarce or data labeling is expensive, DNNs may overfit and require effective regularization techniques to systematically control the complexity of the functions encoded by them to improve generalization [92]. Unfortunately, commonly used regularization techniques in deep learning are developed for regularizing less expressive models rather than highly overparameterized DNNs used in modern machine learning. Recent work on double decent in deep learning with over-parameterized DNNs [4] has demonstrated that notions of regularization and generalization developed for smaller models may not transfer to highly expressive over-parameterized DNNs. This raises the question of why parameter space regularization techniques fall short and to what extent over-parameterized neural networks may benefit from alternative regularization techniques that do not explicitly regularize the network parameters but rather regularize a principled measure of network complexity.

Recent work has shown that weight penalization cannot regularize the complexity of functions [90] and that network parameters are a poor proxy for the functions induced by the DNN [5]. It is also demonstrated that regularizing the parameter norm is unnecessary for better generalization [102, 40].

Theoretical work by Bietti and Mairal [6] has proved that functions encoded by a DNN lie in a *reproducing kernel Hilbert space* (RKHS) under smooth approximation for ReLU activations, therefore a natural candidate for regularization technique is the RKHS norm.

Unfortunately, computing the RKHS norm is challenging since the characteristics of the RKHS defined by the DNN is not well-understood. Prior work on function space regularization in DNNs has explored using the L_2 norm [5] or using the Jacobian norm as a lower bound of RKHS norm [7]. However, both methods fall short, as L_2 space is a trivial approximation and loses many topological information of the space of functions and regularizing the lower bound has no direct relation to regularizing RKHS norm.

Drawing on a result in Arora et al. [2] that a fully-trained wide multi-layer perceptron is equivalent to a kernel regression predictor under the NTK, we argue that the functions encoded by a given DNN can be well approximated by functions in a RKHS associated with the network's NTK. Under this assumption, we derive the RKHS norm associated with NTK from the perspective of kernel regression and use it as a function-space regularizer. Interestingly, we can further validate our approach by deriving the same regularizer from the perspective of function-space maximum a posteriori inference (FS-MAP). The leads to the first work in this dissertation, which is to directly regularize the norm of function mappings induced by the neural network to more effectively control the complexity of the learned function and improve generalization in over-parameterized models.

Secondly, as pointed out in Szegedy et al. [87], state-of-the-art DNN are usually vulnerable to attacks by *adversarial examples*, inputs that are distinguishable to human eyes but can fool classifiers to make arbitrary predictions. Such undesirable property may prohibit DNNs from being applied to security-sensitive applications. Various of adversarial defense methods [29, 68, 75, 79, 84] were then proposed to prevent adversarial examples attack. However, most of the defense methods were quickly broken by new adversarial attack methods. *Adversarial training* (AT), proposed in Madry et al. [57], is one among the few that remains resistant to adversarial attacks.

Despite the success in training robust network, the min-max nature of adversarial training makes

the computation costly for overparameterized neural networks. In the literature, network pruning is shown to be an outstanding method which can significantly reduce the model size while keep the network performance. Sparsity hence can potentially help speed up adversarial training. Moreover, theoretical works by Guo et al. [34] have shown that sparsity can also improve robustness. It is then natural to develop algorithm to find sparse subnetworks which can be trained to be robust.

However, adversarial test accuracy is significantly different from the usual test accuracy. There is no evidence to believe pruning methods for natural training can be directly applied in adversarial context to obtain a robust subnetwork structure. Also, typical pruning algorithms follow the three-stage 'training-pruning-fine-tuning' pipelines, where 'unimportant' weights are pruned according to certain pruning strategies, such as magnitudes of weights. But as observed in Liu et al. [55], fine-tuning a pruned model with inherited weights only gives comparable or worse performance than training that model with randomly initialized weights, which suggests that the inherited 'important' weights are not necessarily useful for fine-tuning. We empirically found that the change of model outputs dynamics is a potential reason for this phenomenon.

Note that the convergence and performance of AT have been theoretically justified by recent works [26, 91, 103]. Hence the robustness of sparse network can be achieved if we can control its dynamics to be the same as dense AT, which is the exactly the second topic in this dissertation. The search of such subnetwork structures requires the study of training dynamics of AT, which no surprisingly is closely related to the theory of NTK. And also, Frankle and Carbin [23] proposed the so-called *Lottery Ticket Hypothesis* (LTH), stating that there exists subnetwork structure, which can achieve comparable performance as the dense model when trained in isolation. The search of such dynamics preserving subnetwork can also provide evidence for the verification of LTH in adversarial setting.

Chapter 1

Preliminary

In this chapter, we briefly discuss the necessary background materials. This includes an introduction of deep neural networks (DNNs), theory of kernel methods, the recent works of neural tangent kernel (NTK) and basics of adversarial example.

1.1 Deep Neural Networks

An L-hidden-layer (bias free) fully-connected neural network $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}^k$ is defined recursively as follows:

$$\begin{aligned} f^{(h)}(\mathbf{x}) &= \mathbf{W}^{(h)} g^{(h-1)}(\mathbf{x}) \in \mathbb{R}^{d_h} \quad g^{(0)}(\mathbf{x}) = \mathbf{x} \\ g^{(h)}(\mathbf{x}) &= \sqrt{\frac{c_\sigma}{d_h}} \sigma\big(f^{(h)}(\mathbf{x})\big) \quad \forall h = 1, 2, \cdots, L \end{aligned} \tag{1.1}$$

where $\mathcal{X}$ is the space of inputs[1], $\mathbf{W}^{(h)} \in \mathbb{R}^{d_h \times d_{h-1}}$ is the weight matrix in the h-th layer, σ is the activation function and $c_\sigma^{-1} = \mathbb{E}_{z\sim\mathcal{N}(0,1)}[\sigma(z)^2]$ is a scaling factor. The last layer is given by

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = f^{(L+1)}(\mathbf{x}) = \mathbf{W}^{(L+1)} g^{(L)}(\mathbf{x}) \tag{1.2}$$

where $\boldsymbol{\theta} = (\mathbf{W}^{(1)}, \cdots, \mathbf{W}^{(L+1)})$ represents all the parameters in the network.

Training a neural network is the same as solving the following optimization problem:

$$\min_{\boldsymbol{\theta}} \mathcal{L} = \frac{1}{N} \sum_{j=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_j)) \tag{1.3}$$

where ℓ is the cost function. Typically the optimization problem is solved by stochastic gradient descent (SGD) on mini-batches. To be precise, for any mini-batch $\mathcal{B} \subset \mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, the

[1] In this dissertation, bold small letter such as $\mathbf{x}$ will represents data points, bold capital letter represents data set, and calligraphic letter will represent vector spaces.

parameter is updated as follows:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \frac{\eta}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} \ell(f_{\boldsymbol{\theta}}(\mathbf{x})) \tag{1.4}$$

where η is the learning rate.

The performance of neural network can be trained to be state-of-the-art due to the overparameterization. For many cases, the loss can even be trained to be zero. This might somehow make the network overfit to the training data, hence potentially hurts the generalizability of the neural network. A typical method to improve the network generalization is adding regularizer in the loss function. A commonly used regularization method is weight decay, i.e., by controlling the ℓ_2 norm of the parameter vector. To be precise, instead of optimizing the loss alone, we optimize the following objective function:

$$\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{reg}} = \frac{1}{N} \sum_{j=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_j)) + \zeta \left\| \boldsymbol{\theta} \right\|^2 \tag{1.5}$$

where ζ is some regularizing constant.

1.2 Kernel Methods

Consider a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is symmetric and positive definite. That is to say $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x})$ for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ and

$$\int_{\mathcal{X} \times \mathcal{X}} f(\mathbf{x}) K(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \mathrm{d}\mu(\mathbf{x}) \mathrm{d}\mu(\mathbf{x}') > 0 \tag{1.6}$$

for any $f \in L_2(\mathcal{X}, \mu)$, where L_2 is the squared integrable function space over $\mathcal{X}$ with respect to a probability measure μ on $\mathcal{X}$. According to Moore–Aronszajn theorem [1], for any symmetric, positive definite kernel K on $\mathcal{X}$ we have a unique RKHS $\mathcal{H}_K$, which is the completion of the pre-Hilbert space consisting of all linear span of feature maps $\{K_{\mathbf{x}}(\cdot) : \mathbf{x} \in \mathcal{X}\}$, where $K_{\mathbf{x}}(\cdot) = K(\cdot, \mathbf{x})$. To be precise, we have

$$\mathcal{H}_K = \left\{ f(\mathbf{x}) = \sum_{j=1}^{\infty} \alpha_j K(\mathbf{x}^{(j)}, \mathbf{x}) : (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots) \subset \mathcal{X}, \|f\|_{\mathcal{H}_K}^2 = \sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) < \infty \right\} \tag{1.7}$$

The kernel function K is called *reproducing kernel* since for any $f \in \mathcal{H}_K$, we have the following reproducing property:

$$f(\mathbf{x}) = \langle f, K_{\mathbf{x}} \rangle_{\mathcal{H}_K} \tag{1.8}$$

Classical kernel regression considers the following non-parametric optimization problem:

$$f^* = \arg\min_{f \in \mathcal{H}_K} \frac{1}{N} \sum_{j=1}^{N} \ell(f(\mathbf{x}^{(j)})) \tag{1.9}$$

A popular example for kernel method is the *support vector machine* (SVM). Also, linear regression can be viewed as a special case of kernel regression, where the reproducing kernel function is the inner product of two vectors.

The regularized version is kernel ridge regression, which optimizes the following objective:

$$f^* = \arg\min_{f \in \mathcal{H}_K} \frac{1}{N} \sum_{j=1}^{N} \ell(f(\mathbf{x}^{(j)})) + \zeta \left\| f \right\|_{\mathcal{H}_K}^2 \tag{1.10}$$

The Representer Theorem [77] implies the optimal solution of (1.10) must be a finite linear combination of feature maps $K_{\mathbf{x}^{(1)}}, \cdots, K_{\mathbf{x}^{(N)}}$ evaluated on the training set, i.e.,

$$f^*(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j K(\mathbf{x}^{(j)}, \mathbf{x}) \qquad \forall\, \mathbf{x} \in \mathcal{X} \tag{1.11}$$

By passing to Equation (1.7), we see that the RKHS norm of the optimal solution is:

$$\left\| f^* \right\|_{\mathcal{H}_K}^2 = \sum_{1 \le i,j \le N} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \boldsymbol{\alpha}^\top K(\mathbf{X}, \mathbf{X}) \boldsymbol{\alpha} \tag{1.12}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_N]^\top$ represents the coordinate of f^* in $\mathcal{H}_K$ with respect to the basis $\Phi = [K_{\mathbf{x}^{(1)}}, \cdots, K_{\mathbf{x}^{(N)}}]^\top$. $K(\mathbf{X}, \mathbf{X}) = [K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})]_{1 \le i,j \le N}$ is the kernel matrix evaluated on training set, hence is symmetric and positive definite. In particular, $K(\mathbf{X}, \mathbf{X})$ is invertible.

For squared loss, by combining Equation (1.11) and (1.12), the objective in Equation (1.10) becomes:

$$\begin{aligned} f^*(\mathbf{x}) &= \arg\min_{f \in \mathcal{H}_K} \frac{1}{N} \sum_{j=1}^{N} \ell(f(\mathbf{x}^{(j)})) + \zeta \left\| f \right\|_{\mathcal{H}_K}^2 \\ &= \arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^N} \frac{1}{N} \left\| K(\mathbf{X}, \mathbf{X}) \boldsymbol{\alpha} - \mathbf{Y} \right\|^2 + \zeta \boldsymbol{\alpha}^\top K(\mathbf{X}, \mathbf{X}) \boldsymbol{\alpha} \end{aligned} \tag{1.13}$$

The first order condition is

$$\frac{2}{N}K(\mathbf{X},\mathbf{X})\big(K(\mathbf{X},\mathbf{X})\boldsymbol{\alpha}-\mathbf{Y}\big)+2\zeta K(\mathbf{X},\mathbf{X})\boldsymbol{\alpha}=0 \tag{1.14}$$

This implies the optimal coordinate is:

$$\boldsymbol{\alpha}=\big(K(\mathbf{X},\mathbf{X})+\zeta N\mathbf{I}\big)^{-1}\mathbf{Y} \tag{1.15}$$

Therefore the optimal solution for kernel ridge regression problem is:

$$f(\mathbf{x})=K(\mathbf{x},\mathbf{X})\big(K(\mathbf{X},\mathbf{X})+\zeta N\mathbf{I}\big)^{-1}\mathbf{Y} \tag{1.16}$$

By letting $\zeta\to 0$, we see the kernel regression solution is:

$$f(\mathbf{x})=K(\mathbf{x},\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}\mathbf{Y} \tag{1.17}$$

If we take $K(\mathbf{x},\mathbf{x}')=\mathbf{x}^\top\mathbf{x}'$, then the result reduces to regularized linear regression, and we have:

$$\begin{aligned} f(\mathbf{x}) =&\mathbf{x}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X}+\zeta N\boldsymbol{I})^{-1}\mathbf{Y}\\ =&\mathbf{x}^\top(\mathbf{X}\mathbf{X}^\top+\zeta N\mathbf{I})^{-1}\mathbf{X}\mathbf{Y}\end{aligned} \tag{1.18}$$

Kernel methods have been shown to generalize well. The following generalization bounds are classical results.

Theorem 1.2.1. *Let $K:\mathcal{X}\times\mathcal{X}\to\mathbb{R}$ be a positive definite symmetric kernel, $\Phi:\mathcal{X}\to\mathcal{H}_K$ a feature mapping associated to K and $H=\{x\mapsto\boldsymbol{w}\cdot\Phi(\mathbf{x}):\|\boldsymbol{w}\|_{\mathcal{H}_K}\le\Lambda\}$. Assume there exists $r>0$ such that $K(\mathbf{x},\mathbf{x})\le r^2$ and $|f(\mathbf{x})|\le\Lambda r$ for all $\mathbf{x}\in\mathcal{X}$. Then for any $\delta>0$, with probability at least $1-\delta$, each of the following inequalities holds for all $h\in H$:*

$$R(h)\le\hat{R}(h)+\frac{8r^2\Lambda^2}{\sqrt{N}}\left(1+\frac{1}{2}\sqrt{\frac{\log\frac{1}{\delta}}{2}}\right) \tag{1.19}$$

$$R(h)\le\hat{R}(h)+\frac{8r^2\Lambda^2}{\sqrt{N}}\left(\sqrt{\frac{\mathrm{Tr}(K)}{Nr^2}}+\frac{3}{4}\sqrt{\frac{\log\frac{2}{\delta}}{2}}\right) \tag{1.20}$$

where $R(h)$ and $\hat{R}(h)$ are generalization error and empirical error respectively, N is the sample size. Therefore, as the sampel size $N\to\infty$, we see that any hypothesis $h\in H$ generalizes. The proof can be found in Theorem 10.7 [61].

1.3 Neural Tangent Kernel

Neural tangent kernel (NTK) is proposed by Jacot et al. [39] in the study of training dynamics of infinite width neural networks. For simplification, we assume gradient descent for full batch. Therefore, a continuous gradient descent can be described as:

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} &= -\eta\nabla_{\boldsymbol{\theta}}\mathcal{L} \\ &= -\eta\nabla_{\boldsymbol{\theta}} f_t(\mathbf{X})^\top \nabla_{f_t(\mathbf{X})}\mathcal{L}\end{aligned} \tag{1.21}$$

where we use $f_t = f_{\boldsymbol{\theta}_t}$ to denote the network function at time t and $f_t(\mathbf{X}) = \mathrm{vec}\big([f_t(x)_{x\in\mathbf{X}}]\big)$ is the model output on a dataset $\mathbf{X}$. Then the evolution of f_t can be computed by chain rule as follows:

$$\begin{aligned}\frac{\mathrm{d}f_t}{\mathrm{d}t}(\mathbf{x}) =& \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x})\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} \\ =& -\eta\nabla_{\boldsymbol{\theta}} f_t(\mathbf{x})\nabla_{\boldsymbol{\theta}} f_t(\mathbf{X})^\top \nabla_{f_t(\mathbf{X})}\mathcal{L}\end{aligned} \tag{1.22}$$

We see that the dynamics is completely described by the matrix $\nabla_{\boldsymbol{\theta}} f_t(\mathbf{x})\nabla_{\boldsymbol{\theta}} f_t(\mathbf{X})^\top$. Hence accordingly we can define the neural tangent kernel at time t to be:

$$\Theta_t(\mathbf{x}, \mathbf{x}') = \langle \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}), \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}')\rangle \tag{1.23}$$

Currently, NTK is not yet a kernel function in the sense of classical kernel theory because it varies for different time t and hence depends on the particular neural network function, and also the value of parameters. The only exception is when f is a linear function, i.e. if $f(\mathbf{x}) = \boldsymbol{\theta}\cdot\mathbf{x}$, then $\nabla_{\boldsymbol{\theta}} f(\mathbf{x}) = \mathbf{x}$ so NTK reduces to the inner product kernel. However, surprisingly Jacot et al. [39] proved that when the number of neurons in each layer goes to infinity, the kernel function tends to a deterministic function $\mathcal{K}$, which is usually referred as the analytic kernel function. To be precise,

the limiting kernel $\mathcal{K}$ can be computed iteratively by[2]:

$$\begin{aligned}
\Sigma^{(0)}(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^\top \mathbf{x}' \\
\Lambda^{(h)}(\mathbf{x}, \mathbf{x}') &= \begin{bmatrix} \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}') \\ \Sigma^{(h-1)}(\mathbf{x}', \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{x}', \mathbf{x}') \end{bmatrix} \\
\Sigma^{(h)}(\mathbf{x}, \mathbf{x}') &= c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}(0,\Lambda^{(h)})}[\sigma(u)\sigma(v)] \\
\dot{\Sigma}^{(h)}(\mathbf{x}, \mathbf{x}') &= c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}(0,\Lambda^{(h)})}[\dot{\sigma}(u)\dot{\sigma}(v)] \\
\mathcal{K}(\mathbf{x}, \mathbf{x}') &= \sum_{h=1}^{L+1} \left(\Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}') \cdot \prod_{h'=h}^{L+1} \dot{\Sigma}^{(h')}(\mathbf{x}, \mathbf{x}') \right)
\end{aligned} \tag{1.24}$$

as long as the activation function σ is Lipschitz[3]. Lee et al. [50] (Theorem 2.1) further proved that if the number of neurons in each layer is $d_1 = \cdots = d_L = m$, under mild conditions, we have

$$\sup_{t\geq 0} \|\Theta_t - \Theta_0\|_F = \mathcal{O}(m^{-\frac{1}{2}}) \quad \text{as } m \to \infty \tag{1.25}$$

This implies the kernel function is stable during training if the network is super wide. And in this case, the initial kernel Θ_0 is a good approximation of NTK.

If we go back to the training dynamics in Equation (1.22), we now can replace the non-constant matrix $\Theta_t(\mathbf{X}, \mathbf{X}) = \nabla_{\boldsymbol{\theta}} f_t(\mathbf{X}) \nabla_{\boldsymbol{\theta}} f_t(\mathbf{X})^\top$ by the constant kernel matrix $\Theta_0(\mathbf{X}, \mathbf{X}) \approx \mathcal{K}(\mathbf{X}, \mathbf{X})$ and expect the training dynamics remains to be similar. In fact, Lee et al. [50] studied the linearized network:

$$f_t^{\text{lin}}(\mathbf{x}) := f_0(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} f_0(\mathbf{x})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0) \tag{1.26}$$

where the corresponding training dynamics is given as we expected by

$$\frac{\mathrm{d} f_t^{\text{lin}}}{\mathrm{d}t}(\mathbf{x}) = -\eta \Theta_0(\mathbf{x}, \mathbf{X}) \nabla_{f_t^{\text{lin}}(\mathbf{X})} \mathcal{L} \tag{1.27}$$

They proved that such linearized network is a good approximation of the neural network output $f_{\boldsymbol{\theta}}(\mathbf{x})$, in the sense that

$$\sup_{t\geq 0} \left\| f_t(\mathbf{x}) - f_t^{\text{lin}}(\mathbf{x}) \right\| = \mathcal{O}(m^{-\frac{1}{2}}) \quad \text{as } m \to \infty \tag{1.28}$$

[2]For biased case, adding a bias term for Σ.

[3]This assumption is generally satisfied for common choices of σ such as ReLU.

For squared loss $\ell(f_{\boldsymbol{\theta}}(\mathbf{x}), \boldsymbol{y}) = \frac{1}{2}(f_{\boldsymbol{\theta}}(\mathbf{x}) - \boldsymbol{y})^2$, the differential equation (1.27) has a closed form solution:

$$f_t^{\text{lin}}(\mathbf{x}) = f_0(\mathbf{x}) - \Theta_0(\mathbf{x}, \mathbf{X})\Theta_0^{-1}\big(\mathbf{I} - e^{-\eta\Theta_0 t}\big)(f_0(\mathbf{X}) - \mathbf{Y}) \tag{1.29}$$

Then for completely trained network ($t \to \infty$) which initialized to be zero, the linearized network is the kernel regression solution under NTK. This equivalence is also clarified by Arora et al. [2] for more loose conditions. To be precise, let $f_{nn}(\boldsymbol{\theta}, \mathbf{x}) = \kappa f_{\boldsymbol{\theta}}(\mathbf{x})$ be the scaled network output where κ is a small constant so the initial network output is close to zero, and

$$f_{nn}(\mathbf{x}) = \lim_{t\to\infty} f_{nn}(\boldsymbol{\theta}_t, \mathbf{x}) \tag{1.30}$$

be the fully trained network output. Also let f_{ntk} be the kernel regression solution under NTK as indicated in Equation (1.17) and λ_0 the smallest eigenvalue of NTK, Arora et al. [2] proved the following equivalence theorem:

Theorem 1.3.1. *Suppose $\sigma(z) = \max(0, z)$ be the ReLU activation, $1/\kappa = \text{poly}(1/\varepsilon, \log(N/\delta))$ and $d_1 = \cdots = d_L = m$ with $m \geq \text{poly}(1/\kappa, L, 1/\lambda_0, N, \log(1/\delta))$. Then for any $\mathbf{x}$ with $\|\mathbf{x}\| = 1$, with probability at least $1 - \delta$ over the random initialization, we have*

$$|f_{nn}(\mathbf{x}) - f_{ntk}(\mathbf{x})| \leq \varepsilon \tag{1.31}$$

1.4 Adversarial Examples

Although DNNs can achieve state-of-the-art performance, their stability with respect to small perturbations on the inputs was found intriguing [87]. The most typical example is the following *adversarial example*:

As we see in Figure 1.1, the clean input is correctly classified as panda by the classifier, while adding a particular designed noise, the corrupted image is indistinguishable to human eyes but the classifier returns the label as gibbon in a very high confidence. Such intriguing property certainly rises potential security concerns in deep learning areas.

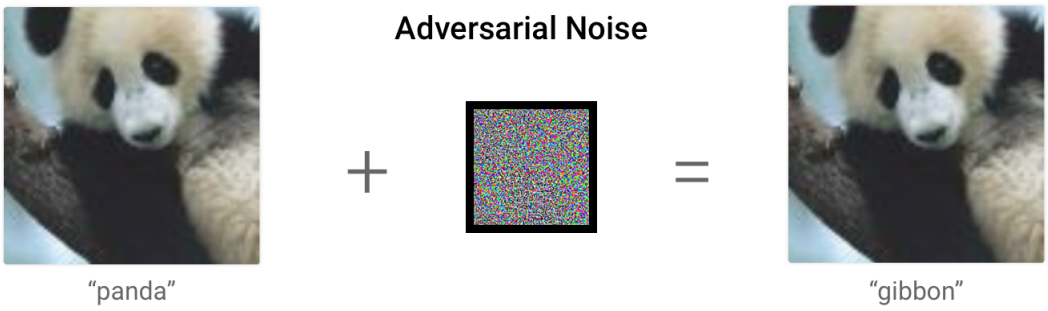


Figure 1.1: Adversarial example: Credits to Goodfellow et al. [29].

Mathematically, given an clean input $\mathbf{x}$, an adversarial example[4] is usually modelled as a perturbed data $\mathbf{x}'$ such that

$$f(\mathbf{x}') \neq f(\mathbf{x}) = \mathbf{y} \qquad \text{but} \quad d(\mathbf{x}, \mathbf{x}') < \varepsilon \tag{1.32}$$

where ε is the allowed adversarial noise level and d is certain distance function.

Given any neural network function, adversarial examples are usually generated by adversarial attack algorithms. The first effective attack algorithm is the *fast gradient sign method* (FGSM), which generates the adversarial noise according to the loss gradient. To be precise, we have:

$$\mathbf{x}' = \mathbf{x} + \varepsilon \cdot \operatorname{sign}\big(\nabla_{\mathbf{x}} \ell(f(\mathbf{x}))\big) \tag{1.33}$$

This corresponds to the one step update of the optimization problem:

$$\max_{\mathbf{x}'} \ \ell\big(f(\mathbf{x}')\big) \qquad \text{s.t. } \|\mathbf{x}' - \mathbf{x}\|_\infty \leq \varepsilon \tag{1.34}$$

We can also use any $\ell_p, p \geq 1$ distance instead of ℓ_∞ distance, then the sign of gradient will be replaced by the corresponding q-normalized gradient.

Instead of one step gradient update, we can use k steps gradient update, which leads to the stronger *projected gradient descent* (PGD) attack. To be precise, the adversarial example is generated as

[4]Such type of adversarial example is called *untargeted*, while *targeted* adversarial example requires $f(\mathbf{x}') = t$ for any given targeted label t.

follows:

$$\begin{aligned}\mathbf{x}_0 =& \mathbf{x} \\ \mathbf{x}_t =& \mathbf{x}_{t-1} + \alpha \cdot \operatorname{sign}\big(\nabla_{\mathbf{x}} \ell(f(\mathbf{x}_{t-1}))\big) \quad 1 \le t \le k \\ \mathbf{x}' =& \mathbf{x}_k\end{aligned} \tag{1.35}$$

where α is the step size for each intermediate step, and certain clip operation applies in order to make sure each $\mathbf{x}_t$ is within the range of input domain. Details can be found in Madry et al. [57]. While attack algorithms aim to effectively generate adversarial examples to fool the classifier, certain defense methods are developed to train a robust network which can prevent such attacks. Most of the defense methods proposed in the literature quickly break down by new attack algorithms, among which *adversarial training* proposed in Madry et al. [57] remains resistant to most of the attack algorithms.

The idea of adversarial training is iteratively generating adversarial examples in the intermediate steps and updating the parameters by gradients evaluated on these corrupted inputs. Mathematically, adversarial training optimizes the following objective function:

$$\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{adv}} = \frac{1}{N} \sum_{j=1}^{N} \max_{\|\mathbf{x}'_j - \mathbf{x}\| \le \varepsilon} \ell\big(f_{\boldsymbol{\theta}}(\mathbf{x}'_j)\big) \tag{1.36}$$

Therefore, different from the natural training (1.3), adversarial training consists of the following two steps:

$$\begin{aligned}\mathbf{x}' &\leftarrow \text{Attack}(\mathbf{x}) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \eta \mathop{\mathbb{E}}_{\mathbf{x} \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} \ell\big(f_{\boldsymbol{\theta}}(\mathbf{x}')\big)\end{aligned} \tag{1.37}$$

The first step generates intermediate adversarial examples by some strong attack algorithm such as PGD attack, then the parameters are updated according to the gradients evaluated on the generated adversarial examples. The min-max nature of adversarial training makes the training more expensive than natural training. However, the fact that adversarial training can obtain robust solution has been justified both theoretically and empirically.

Chapter 2

Function Norm Regularizer Induced by Neural Tangent Kernel

2.1 Overview

In this chapter, we will discuss the regularization of deep neural networks (DNNs) using reproducing kernel Hilbert space (RKHS) norm induced by neural tangent kernel (NTK). As we have already discussed in the introduction, regularizing by function norm is better than just regularizing the parameters. Although the natural candidate is the L_2 norm, its triviality ignores the correlation between different function values. From a point view of the RKHS theory, L_2 norm regularization corresponding to find an optimal solution in the squared integrable function space, which is too large for neural network training. Since any RKHS is subspace of L_2 space weighted by eigenvalues of the reproducing kernel, the search of the proper RKHS is the same as finding the reproducing kernel corresponds to neural network training. In the following sections we will show both theoretically and empirically the correct candidate is the NTK.

This chapter is oragnized as follows: In Section 2.2 we derive the regularizer induced by NTK from kernel ridge regression perspective and function space maximum a posteriori inference perspective. Then in Section 2.3 we briefly discuss the related works. In Section 2.4 we present the detailed implementation of our method in practice and in Section 2.5 we present the experimental results to check the empirical performance of our method. The proof details are presented in Section 2.6 and finally we discuss the theoretical details of construction of function priors in Section 2.7.

2.2 Theoretical Results

In this section, we will introduce the theoretical details about regularizing DNNs by RKHS norm induced by NTK. The theory can be derived from a kernel ridge regression perspective, as well as from a function-space maximum a posteriori inference (FS-MAP) perspective.

2.2.1 A Kernel Ridge Regression Perspective

Recall that the RKHS norm of the kernel ridge regression solution can be calculated as follows:

$$\|f\|_{\mathcal{H}_K}^2 = \sum_{1\leq i,j\leq N} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \boldsymbol{\alpha}^\top K(\mathbf{X}, \mathbf{X})\boldsymbol{\alpha} \tag{2.1}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_N]^\top$ represents the coordinate of f in $\mathcal{H}_K$ with respect to the basis of feature functions. Equation (2.1) implies that $\mathcal{H}_K$ can be viewed as a finite dimensional realization of the dual space of $\mathcal{X}$ equipped with the metric induced by K so we refer to Equation (2.1) as the dual form of $\|f\|_{\mathcal{H}_K}$.

If we let $f(\mathbf{X}) = [f(\mathbf{x}^{(1)}), \cdots, f(\mathbf{x}^N)]^\top$, then $f(\mathbf{X}) = K(\mathbf{X}, \mathbf{X})\boldsymbol{\alpha}$, hence we can alternatively calculate the RKHS norm in Equation (2.1) as follows:

$$\begin{aligned} &\|f\|_{\mathcal{H}_K}^2 = \boldsymbol{\alpha}^\top K(\mathbf{X}, \mathbf{X})\boldsymbol{\alpha} \\ =&f(\mathbf{X})^\top K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} f(\mathbf{X}) \\ =&f(\mathbf{X})^\top K(\mathbf{X}, \mathbf{X})^{-1} f(\mathbf{X}) \end{aligned} \tag{2.2}$$

The advantage of Equation (2.2) is the RKHS norm is completely determined by function values and the inverse of kernel matrix, hence can be calculated directly. While Equation (2.1) requires the knowledge of α, which can only be known after solving the kernel ridge regression problem. We will refer to Equation (2.2) as the primal form of $\|f\|_{\mathcal{H}_K}$.

Also recall that the (empirical) NTK function can be defined as:

$$\Theta(\mathbf{x}, \mathbf{x}') = \langle \nabla_{\boldsymbol{\theta}} f(\mathbf{x}), \nabla_{\boldsymbol{\theta}} f(\mathbf{x}') \rangle \tag{2.3}$$

As we discussed in the previous chapter, NTK is stable in the training as long as the network is super wide and is a good approximation of the analytic kernel $\mathcal{K}$. Moreover, we know from Theorem 1.31 that a fully-trained sufficiently wide neural net is equivalent to the kernel regression solution under analytic NTK $\mathcal{K}$. It is then natural to consider the regularized version of this equivalence. Note that during the neural network training, we have no knowledge about the coordinate α of f, therefore, Equation (2.2) becomes the natural choice for computing the RKHS norm. In particular, we propose the following optimization problem:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{j=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}^{(j)})) + \zeta f_{\boldsymbol{\theta}}(\mathbf{X})^{\top} \mathcal{K}(\mathbf{X}, \mathbf{X})^{-1} f_{\boldsymbol{\theta}}(\mathbf{X}) \tag{2.4}$$

Note that under squared loss, the solution of kernel ridge regression problem under NTK can be explictly found as

$$f_{ntr}(\mathbf{x}) = \mathcal{K}(\mathbf{x}, \mathbf{X}) \Big(\mathcal{K}(\mathbf{X}, \mathbf{X}) + \zeta N \mathbf{I} \Big)^{-1} \mathbf{Y} \tag{2.5}$$

where $\mathbf{I}$ is the identity matrix. Similarly, let $f_{nr}(\mathbf{x}) = \kappa \lim_{t \to \infty} f_{\boldsymbol{\theta}_t}(\mathbf{x})$ be the scaled neural network prediction for (2.4) fully trained by SGD, and let λ_0 be the smallest eigenvalue of NTK $\mathcal{K}$. Then we can prove the following equivalence theorem.

Theorem 2.2.1. *Suppose $\sigma(z)$ is the ReLU activation function, $1/\kappa = \mathrm{poly}(1/\varepsilon, \log(N/\delta))$, $d_1 = d_2 = \cdots = d_L = m$ with $m \geq \mathrm{poly}(1/\kappa, L, 1/(\lambda_0 + \zeta N))$. Then for any $\mathbf{x}$ such that $\|\mathbf{x}\| = 1$, with probability at least $1 - \delta$ over the random initialization, we have*

$$|f_{nr}(\mathbf{x}) - f_{ntr}(\mathbf{x})| \leq \varepsilon \tag{2.6}$$

Theorem 2.2.1 shows the equivalence between the neural network training and kernel ridge regression when RKHS norm regularizer is applied. This result extends the original result by Arora et al. [2]. Using this fact, we can derive the generalization bound for the learning problem (2.4).

Theorem 2.2.2. *Let f_{nr} be defined as the optimal solution of* (2.4) *as above. Let $R(f_{nr}) = \mathbb{E}[(f_{nr}(\mathbf{x}) - \mathbf{y}(\mathbf{x}))^2]$ be the generalization error, and $\hat{R}(f_{nr})$ the corresponding empirical error. Assume f_{nr} has bounded function value and first order derivative, i.e., $|f_{nr}(\mathbf{x})| \leq C_0 \kappa$ and*

$\|\partial_{\boldsymbol{\theta}} f_{nr}(\mathbf{x})\| \leq C_1$ *for all* $\|\mathbf{x}\| \leq 1$*, assume* $\mathbb{E}|\mathbf{y}| = c$*. Then for any* $\delta > 0$*, with probability at least* $1 - \delta$*, we have the following bound:*

$$R(f_{nr}) - \hat{R}(f_{nr}) \leq 4(C_0\kappa + c)\varepsilon + \frac{8C_1^2\Lambda^2}{\sqrt{N}}\left(\sqrt{\frac{C_K}{NC_1^2}} + \frac{3}{4}\sqrt{\frac{\log\frac{2}{\delta}}{2}}\right) \tag{2.7}$$

where N *is the number of training points and* Λ *is a constant only depends on the regularization constant* ζ*.* C_K *is the trace of NTK.* ε *is the gap of outputs in Theorem 2.2.2.*

It has been proven that the eigenvalues of NTK decays polynomially in Geifman et al. [27], hence $C_K = \mathrm{Tr}(\mathcal{K})$ is a bounded constant. This bounds implies that as the sample size $N \to \infty$, our method generalizes. Detailed proofs of the Theorems will be given in Section 2.6.

Algorithm 1 Function-space regularization

Initialize $\boldsymbol{\theta}$
for training epoch $m = 1, 2 \ldots M$ **do**
  for each training iteration **do**
    Sample a mini batch of data $\mathcal{B} = \left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \ldots, \left(\mathbf{x}^{(B)}, \mathbf{y}^{(B)}\right)\right\}$
    Sample a sub batch of data $\mathcal{I} \subset \mathcal{B}$
    Compute NTK $\Theta(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}) \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top$
    Compute Loss $\mathcal{L}_\mathcal{B} = -\sum_{i=1}^{B} \log p(\mathbf{y}^{(i)} | f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + \tau f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top \Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})$
    Update $\boldsymbol{\theta}$: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_\mathcal{B}$
  end for
end for
return $\boldsymbol{\theta}$

2.2.2 A Maximum A Posteriori Perspective

The function-space regularizer in Equation (2.2) can also be derived from the perspective of function-space maximum a posteriori inference (FS-MAP). First we briefly revisit parameter-space maximum a posteriori inference (MAP).

MAP. Assume $\boldsymbol{\theta}$ follow some prior distribution $p(\boldsymbol{\theta})$, then the rule of maximum a posteriori states that the optimal $\boldsymbol{\theta}$ is the best guess that maximizes the value of posteriori distribution. According to the Bayes's theorem, the posteriori distribution is proportional to the product of prior and likelihood, so we have the following formula:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}}\{\log p(\boldsymbol{\theta}|\mathcal{D})\} \\ &= \arg\max_{\boldsymbol{\theta}}\{\log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\} \\ &= \arg\max_{\boldsymbol{\theta}}\{p(\mathcal{D}|\boldsymbol{\theta}) + \tau||\boldsymbol{\theta}||\},\end{aligned}$$

where the norm depends on the choice of prior distribution over parameters and τ is a positive scaling factor.

FS-MAP. Instead of expressing the optimization objective *directly* in terms of the model parameters $\boldsymbol{\theta}$, we suggest a function-space objective, which is expressed in terms of the functions $f_{\boldsymbol{\theta}}(\cdot)$ induced by the parameters $\boldsymbol{\theta}$. To keep notation tight in this section, we restrict $f_{\boldsymbol{\theta}}(\cdot)$ to be a scalar function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}$. Similar to MAP, if we assume $f_{\boldsymbol{\theta}}(\cdot)$ follows some prior distribution $p(f_{\boldsymbol{\theta}}(\cdot))$, then we have:

$$f_{\boldsymbol{\theta}}{}^* = \arg\max_{\boldsymbol{\theta}}\{\log p(f_{\boldsymbol{\theta}}(\mathbf{X})|\mathcal{D}) + \log p(f_{\boldsymbol{\theta}}(\cdot))\} \tag{2.8}$$

Remark. Although we hope to find the optimal function that maximizes this objective, the optimization is performed on parameters given that the function is parametrized by $\boldsymbol{\theta}$. Note that there is an abuse of notation in the above formula. The function-space prior distribution $p(f_{\boldsymbol{\theta}}(\cdot))$ is a distribution over functions, i.e stochastic process [46]. The density function $p(f_{\boldsymbol{\theta}}(\cdot))$ is ill-defined and does not even exist because there is no infinite-dimensional Lebesgue measure [21]. For a more rigorous definition of Equation (2.8) from the perspective of measure theory, please refer to Section 2.7.

Following prior work [86, 74], in order to avoid the pathology in infinite-dimensional probability density, we only evaluate the log density of function-space prior distribution on finite number of

points $\mathbf{X}_\mathcal{I}$ sampled from some distribution $q(\mathbf{x})$. Under this assumption, Equation (2.8) becomes:

$$f^*_{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \{\log p(f_{\boldsymbol{\theta}}(\mathbf{X})|\mathcal{D}) + \mathop{\mathbb{E}}_{\mathbf{X}_\mathcal{I} \sim q(\mathbf{x})} \log p(f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}))\} \tag{2.9}$$

Function-space prior. Now we consider the choice of function-space prior distribution evaluated at a finite number of samples $p(f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}))$. We propose to use the function-space prior distribution induced by the prior distribution on network's parameters. Although many other function-space priors are also widely used such as RBF Gaussian prior [71], our choice of function-space prior distribution best captures the correlations between function values induced by the neural network. A similar function-space prior has been used in Gaussian process [42].

We assume all the network's parameters follow isotropic Gaussian prior distribution $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_P)$. The same assumption has been used in mean-field variational inference in Bayesian neural networks [10]. However, the derivation of $p(f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}))$ from $p(\boldsymbol{\theta})$ is intractable and is usually estimated via Monte Carlo sampling. To solve the problem, we propose to use a local linearization of the function mapping $f_{\boldsymbol{\theta}}(\cdot)$ about the mean of the prior distribution over parameters.

Proposition 2.2.3 (Rudner et al. [74])**.** *Suppose we have a neural network whose parameters $\boldsymbol{\theta}$ follow a prior isotropic Gaussian distribution $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_P)$. Under local linearization around the prior mean, the prior distribution over function values evaluated at a finite number of samples $p(f_{\boldsymbol{\theta}}(X_\mathcal{I}))$ can be approximated by a multivariate Gaussian distribution $\tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(X_\mathcal{I}))$:*

$$p(f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})) \approx \tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})) = \mathcal{N}\left(\mathbf{0}, \sigma^2 \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}) \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}\left(\mathbf{X}_\mathcal{I}\right)^\top\right).$$

The intuition behind Proposition 2.2.3 is that linearization turns $f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})$ into a linear function $\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})$ with respect to $\boldsymbol{\theta}$ and linear transformation of Gaussian distribution is still Gaussian. Full proof can be found in Section 2.6.3.

So the log prior term of FS-MAP in Equation (2.9) evaluated at a finite number of samples $X_\mathcal{I}$ can be approximate by

$$\log \tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})) = c\, f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top \Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}),$$

for $\Theta^{-1}(\mathbf{X}_{\mathcal{I}}, \mathbf{X}_{\mathcal{I}})$ as defined in Equation (2.3), which agrees with Equation (2.2) up to a constant c. The expectation in Equation (2.9) can be estimated by sampling $\mathbf{X}_{\mathcal{I}}$ uniformly from $\mathbf{X}$ at every iteration. Therefore, we arrive at the same regularization technique using a different approach.

2.3 Related Works

In this section, we briefly discuss the related works of function norm regularization, neural network generalizability and kernel methods.

Function-space regularization. Bietti and Mairal [6] has proved that functions encoded by deep neural networks lie in a RKHS under smooth approximations for ReLU activation, but the RKHS norm is difficult to compute. Benjamin et al. [5] uses function L_2 norm as a trivial approximation to the RKHS norm and Bietti et al. [7] gives a lower bound of the RKHS norm with Jacobian norm.

The concept of function space regularization has also been employed in other areas like variational inference [86, 74, 12, 42, 9] and continual learning [88, 67]. These function-space regularization techniques directly constrain the space of functions and as such allow for a more explicit incorporation of prior information about the functions to be learned.

For variational inference, function-space approaches avoid the limitations in parameter-space variational inference [10, 22] and achieve better uncertainty quantification as well as calibration, but function-space approaches is usually not scalable. Sun et al. [86] uses an expensive gradient estimator which limits the application to low-dimensional data, and Rudner et al. [74] requires backpropagation through the NTK-induced covariance matrix which limits its application under ResNet18 [37].

For continual learning, function-space approaches usually adopt a variational inference framework. The regularization term is KL divergence from the variational posterior to the posterior of the previous task which is used as prior in the current task [88]. Benjamin et al. [5] uses function L_2 norm as a function-space regularizer; FROMP [67] first constructs a GP based on DNN2GP [42]

and then uses the GP posterior in KL divergence, which deviates from the original network. We adapt our function-space regularizer to continual learning and propose to regularize the RKHS norm of the relative function change from the previous task to the current task.

Generalization in neural networks. Classical learning theory attributes generalization ability to low-capacity class of hypotheses space [92, 61, 3], but neural networks have contradicted traditional concept by demonstrating that good generalization can be achieved despite over-parameterization [102, 40]. Keskar et al. [41] argues that flat minima lead to better generalization, but Dinh et al. [20] shows that sharp minima can also generalize well. Although it remains an open problem why over-parameterized DNNs can generalize well, DNNs will definitely overfit when the number of training samples is small and we propose to improve DNN generalization under limited number of samples.

Kernel methods. Kernel methods, especially support vector machines [78], are popular statistical learning methods. Classical choices of kernels include Gaussian kernel, polynomial kernels etc. Some of the previous works also consider families of Gaussian kernels [60], hyperkernels [66]. The generalization performance of kernel methods have been verified in theory [47, 15]. Despite the theoretical success of kernel methods, its high computational costs limits its application, so many approximation methods have been proposed to scale up the kernel machines, such as random Fourier features [70] and Nyström method [96].

2.4 Method

In this section, we formalize our approach with some practical techniques and extend it to continual learning.

2.4.1 Regularization in Neural Networks

Based on the discussion in Section 2.2, we compute the RKHS norm induced by NTK and use it as a function-space regularizer. Unfortunately, the naive way to compute the NTK $\Theta(\mathbf{X}, \mathbf{X})$ has $\mathcal{O}(P^2)$ space complexity and $\mathcal{O}(N^2P)$ time complexity, which is too expensive for large networks. So we propose to use an implicit way to compute NTK provided by the *neural tangents library* [65] available in JAX [11] and we also propose to use batch estimation to avoid using full training set $\mathbf{X}$ to evaluate NTK.

Approximation 1 (Batch estimation)**.** *Suppose $\mathbf{X}_\mathcal{I}$ is sampled from the training set $\mathbf{X}$ under some distribution $q(\mathbf{x})$, then we have:*

$$\begin{aligned}\|f\|^2_{\mathcal{H}_\Theta} &= f_{\boldsymbol{\theta}}(\mathbf{X})^\top \Theta^{-1}(\mathbf{X}, \mathbf{X}) f_{\boldsymbol{\theta}}(\mathbf{X}) \\ &\approx \frac{|\mathbf{X}|}{|\mathbf{X}_\mathcal{I}|} \mathop{\mathbb{E}}_{\mathbf{X}_\mathcal{I} \sim q(\mathbf{x})} \left[f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top \Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}) \right]\end{aligned}$$

At every gradient step, we sample $\mathbf{X}_\mathcal{I}$ from $\mathbf{X}$ to compute the RKHS norm and scale it up accordingly. Note that the inverse batch NTK matrix $\Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I})$ is only a block inversion of the full NTK matrix, so it is a biased estimation of the whole inverse NTK matrix $\Theta^{-1}(\mathbf{X}, \mathbf{X})$.

The objective can be formalized as:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)} | f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + \tau \mathop{\mathbb{E}}_{\mathbf{X}_\mathcal{I} \sim q(\mathbf{x})} \left[f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top \Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}) \right], \tag{2.10}$$

where τ is a positive scaling factor[1]. The expectation in Equation (2.10) can estimated by iteratively sampling $\mathbf{X}_\mathcal{I}$ at every gradient step (See algorithm 1).

For infinitely wide neural networks under least squared error, our method using full $\Theta(\mathbf{X}, \mathbf{X})$ would be equivalent to kernel ridge regression under Θ_{ntk}. Therefore, our method can be viewed as parameterizing the solutions of kernel ridge regression with a neural network so that we can use stochastic gradient descent and thus avoid the expensive computation required in kernel regression. Given that we combine the expressiveness of neural networks along with the regularization technique

[1] τ is a hyperparameter.

in kernel ridge regression, it is not surprising that we achieve better generalization performance in Section 2.5.

2.4.2 Continual Learning

Continual learning is the setting where a model receives K distinct datasets sequentially one at a time and the model is required to memorize previously trained tasks when learning a new one [44]. We denote S sequential datasets as $\mathcal{D}_s = \{\mathbf{x}_s^{(n)}, \mathbf{y}_s^{(n)}\}_{n=1}^{N_s}, \quad s = 0, 1, \ldots, S-1$. When learning the current task, the model has no access to previous datasets except for a small number of samples contained in the coreset[2] C_s. We use subscript s to indicate the parameters $\boldsymbol{\theta}_s$ and the corresponding functions $f_{\boldsymbol{\theta}_s}(\cdot)$ at task s. We also denote the NTK obtained at task s as Θ_s.

At task 0, we use maximum log-likelihood as optimization objective without any regularization. When learning task s, in order to maintain the memory on the previous task, we penalize the changes of the input-output function encoded by the neural networks. Therefore, we propose to regularize the distance between the function learned at task $s-1$ and the function to be learned at task s. In Benjamin et al. [5], this distance is trivially measured by L_2 norm as $\|f_s - f_{s-1}\|_{L_2}$. Based on the discussion in Section 2.2 where we argue that the function norm should consider the Hilbert structure of the space of functions induced by the neural network, we diverge from Benjamin et al. [5] and use the norm of $f_s - f_{s-1}$ in the RKHS $\mathcal{H}_{\Theta_{s-1}}$ after task $s-1$ has been learned, i.e. $\|f_s - f_{s-1}\|_{\mathcal{H}_{\Theta_{s-1}}}$.

In continual learning, we do not need to explicitly sample $\mathbf{X}_\mathcal{I}$ from $\mathbf{X}$ as we can directly evaluate function values on coresets C_s. So we write the objective for continual learning at task s as follows:

$$\mathcal{L}_s(\boldsymbol{\theta}_s) = \sum_{i=1}^{N_s} \log p(\mathbf{y}^{(i)} | f_{\boldsymbol{\theta}_s}(\mathbf{x}^{(i)})) + \tau \Delta f(C_s)^\top \Theta_{s-1}^{-1}(C_s, C_s) \Delta f(C_s) \tag{2.11}$$

where $\Delta f(C_s) = f_{\boldsymbol{\theta}_s}(C_s) - f_{\boldsymbol{\theta}_{s-1}}(C_s)$ denotes the relative change of function values evaluated on coreset C_s from task $s-1$ to task s. Intuitively, the objective trades off fitting the data at task s

[2]Coreset is a commonly-used concept in continual learning and is usually a small set of samples (several hundred) randomly selected at every task.

while also maintaining the previous memory at task $s-1$. And note that our approach only needs to keep a working memory of function values at coresets and the NTK matrix, which is more efficient than storing a snapshot of all parameters.

2.5 Experimental Results

We argue in Section 2.4 that we propose a better regularization technique for neural networks, so we evaluate the performance of our method in three different scenarios where regularization is particularly important.

Firstly, we follow Bietti et al. [7] to evaluate generalization performance of our approach on MNIST and CIFAR-10 by training with a small number of samples where regularization is crucial to prevent the model from overfitting.

Secondly, MLP-Mixer [89] has recently drawn much attention as it has demonstrated competitive performance on large-scale datasets with much higher throughput compared to convolution networks or self-attention models. However, the application of MLP-Mixer is limited because MLP-Mixer is very easy to overfit [89]. We propose to use our function-space regularizer on MLP-Mixer and show that our regularization technique alleviates overfitting and consequently leads to higher accuracy as well as lower test negative log-likelihood.

Then, based on the discussion in Section 2.4.2, we evaluate our method in continual learning tasks where regularization is particularly effective in keeping memory on previously learned tasks [63, 44, 5, 88, 67]. Finally, we show the ablation study results on batch estimation and the computational overhead of our method.

Note that we are not doing few-shot learning (FSL) experiments although FSL is proposed to tackle the problem of generalizing from a few examples. The reason is that existing studies [95] show that FSL requires specific techniques such as extracting prior knowledge from other related tasks, instead of simply introducing a general regularizer into training.

Notation. In this section, we refer to our method as $\mathcal{K}^{\Theta}$, function L_2 norm in Benjamin et al. [5] as $\mathcal{K}^{L_2}$, Jacobian norm in Bietti et al. [7] as $\mathcal{K}^{\mathcal{J}}$. We also refer to weight decay as MAP and kernel regression under NTK as $\mathcal{K}_{ntk}$.

Configuration For MNIST, we use a LeNet style network with three convolutional layers of 6, 16 and 120 5×5 filters and a fully-connected final layer of 84 hidden units. An average pooling operation is placed after each convolutional layer and ReLU activation is used. For CIFAR-10, we use ResNet-18. For both MNIST and CIFAR-10, we use SGD optimizer with a learning rate of 0.03 and momentum of 0.9. The learning rate would decay by a rate of 0.3 at every 10 epochs. During training, batch size is set to 128 for MNIST and 256 for CIFAR-10. At every iteration, 10 input points are randomly sampled from the current batch as $\mathbf{X}_{\mathcal{I}}$ to compute the RKHS norm.
When we compare our method against $\mathcal{K}^{L_2}$ and $\mathcal{K}^{\mathcal{J}}$, we copy the same experimental set-up as in the original paper. For MAP and Dropout, we use a outstanding validation set to select the best hyper-parameter. In the end, we find that for MAP, the best τ is $5e^{-4}$, and for Dropout, the best dropout rate is 0.15.

2.5.1 Generalization on Small Datasets

Following Bietti et al. [7], we train with a small number of samples on MNIST and CIFAR-10 to demonstrate generalization performance. We also gradually increase the number of training samples to show that our method does not lead to underfitting when the number of samples is large. We use a LeNet-style [48] network for MNIST and ResNet18 [37] for CIFAR-10. We compare our method with MAP, dropout [85], $\mathcal{K}^{L_2}$ and $\mathcal{K}^{\mathcal{J}}$. The comparison against $\mathcal{K}_{ntk}$ is only implemented on MNIST because the kernel regression under Θ_{ntk} is too expensive on ResNet18 even with Nyström approximation. Actually, there are many methods that improve generalization by regularizing implicitly, such as batch normalization and SGD [8, 104], and we are not comparing against these methods mainly because these implicit methods can be easily combined with explicit regularization

techniques like ours, so that we already use them throughout our experiments.

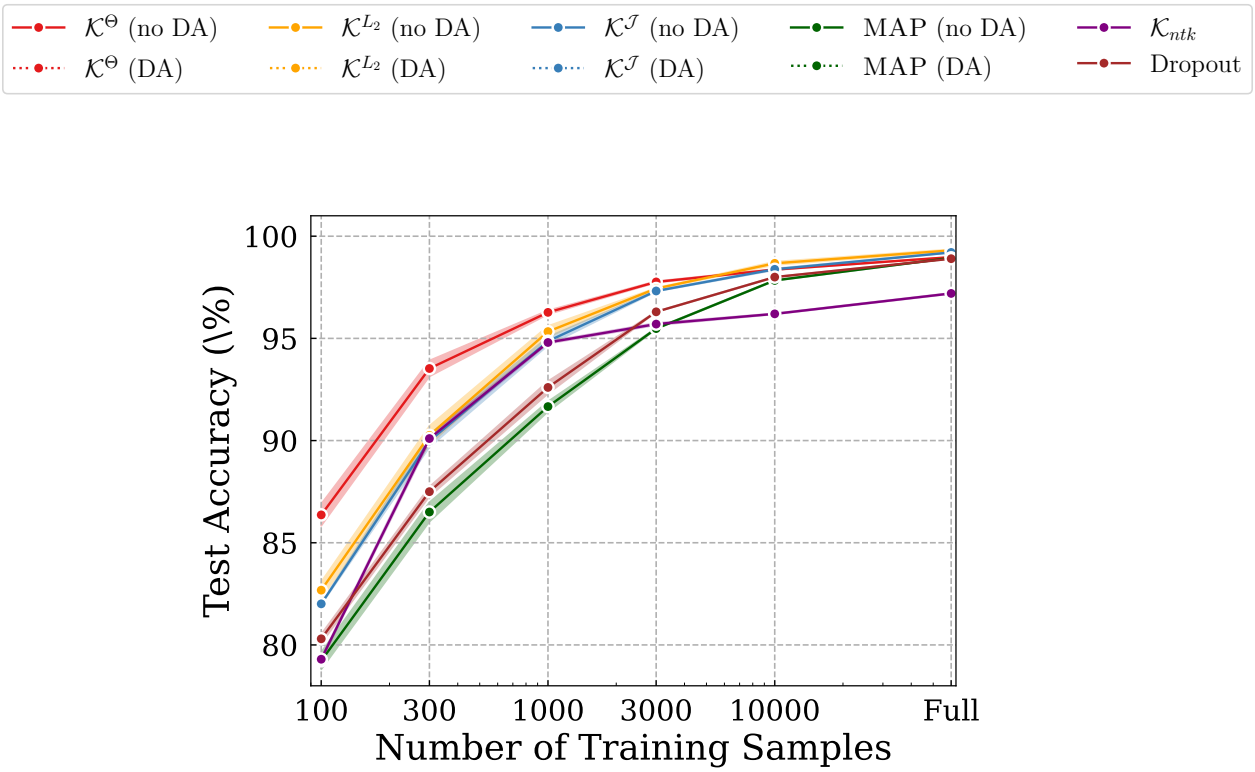


Figure 2.1: NTK regularization performance on MNIST.

Figure 2.1 and 2.2 show the test accuracy on MNIST and CIFAR-10 as we increase the number of training samples. On small datasets like MNIST or CIFAR-10, our networks can easily reach 100% accuracy on training set, therefore higher test accuracy indicates better generalization performance. We can see that $\mathcal{K}^{\Theta}$ has the best generalization performance when the number of training samples are small, but MAP gradually catches up as the number of training samples grows.

We believe that higher test accuracy on a small number of training samples is caused by stronger regularization in that the RKHS norm $\|f_\theta\|_{\mathcal{H}_\Theta}$ enforces the neural network to find the smoothest function among all the functions in $\mathcal{H}_\Theta$ that can perfectly fit the training data, and hence achieves the best generalization performance. Moreover, our method also outperforms $\mathcal{K}_{ntk}$ due to the expressiveness of neural networks. However, as the number of training samples grow, $\mathcal{K}^{\Theta}$ loses its advantage because the large number of training samples makes the networks harder to overfit. We

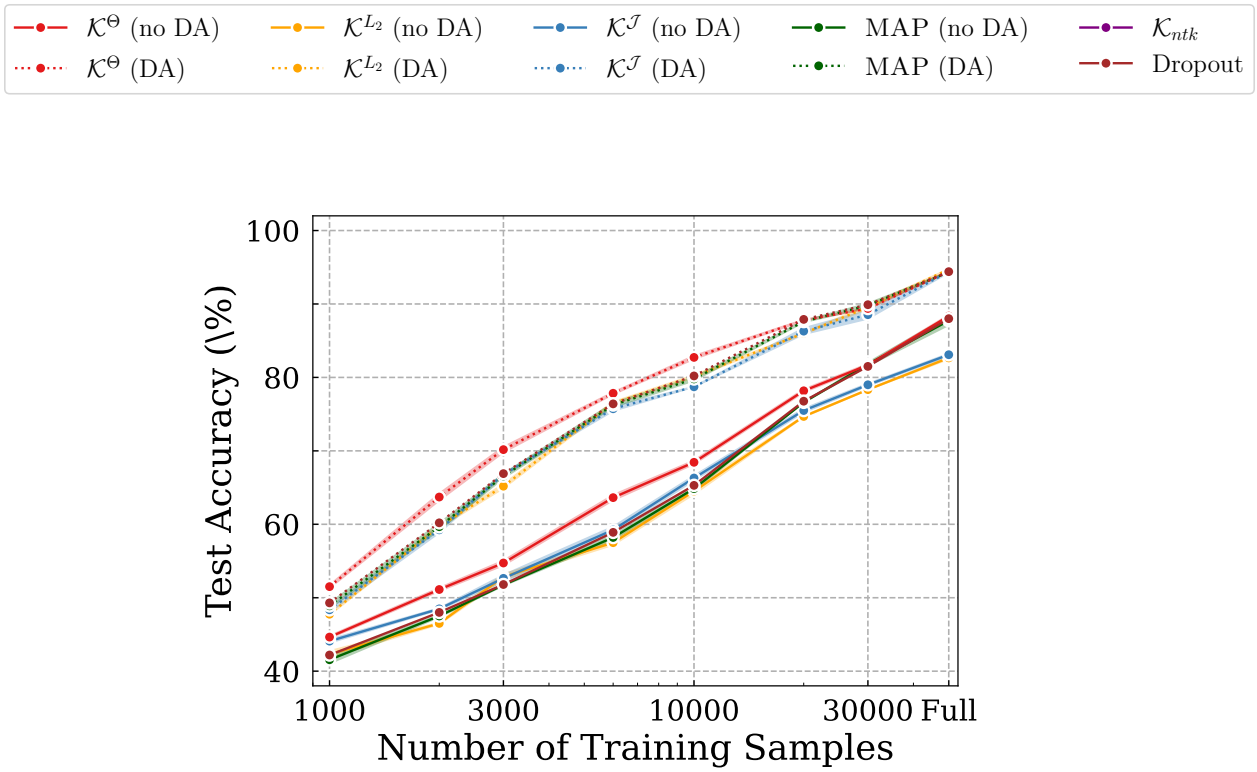


Figure 2.2: NTK regularization performance on CIFAR-10.

also note that in Figure 2.2, $\mathcal{K}^{\Theta}$ is more effective when there is no data augmentation, because data augmentation is also a strong regularization technique that offsets the regularization effect of $\mathcal{K}^{\Theta}$.

2.5.2 Regularization on MLP-Mixer

Based on the pratical techniques discussed in Section 2.4.1, we are able to train an MLP-Mixer of B/16 architecture on CIFAR-10 and CIFAR-100 with objective (2.10). We load MLP-Mixer B/16 parameters pretrained on ImageNet [19] at initialization from public resource online.

We use 10% of the training set as the validation set to conduct a hyperparameter search over the scaling factor τ and learning rate η for all methods. We choose the set of hyperparameter that yielded the lowest validation negative log-likelihood for all experiments. All experiments are implemented in JAX [11].

We use MLP-Mixer of B/16 architecture [89] and load the pretrained parameters on ImageNet from

public source online. The training of MLP-Mixer on CIFAR-100 requires 8 GTX 3090 GPUs. We use the SGD optimizer with learning rate 0.01 and momentum 0.9.

Method	CIFAR-10		CIFAR-100	
	NLL[2]	Acc (%)	NLL	Acc (%)
ML[1]	$0.10_{\pm 0.01}$	$96.5_{\pm 0.1}$	$0.56_{\pm 0.01}$	$84.5_{\pm 0.1}$
ours	$0.10_{\pm 0.00}$	$\mathbf{97.0}_{\pm 0.1}$	$\mathbf{0.54}_{\pm 0.00}$	$\mathbf{85.2}_{\pm 0.1}$

Table 2.1: MLP-Mixer performance on CIFAR-10 and CIFAR-100. [1]Maximum log-likelihood. [2]Negative log likelihood.

We can see that $\mathcal{K}^{\Theta}$ has higher accuracy than maximum log-likelihood without regularization. Although the improvement is quite small, the reported values of standard error show that the improvement is statistically significant so that our regularization technique is effective in reducing overfitting for MLP-Mixer. Note that the accuracy on CIFAR-10 almost reaches the upper limit so even 0.5% increase is a huge improvement.

2.5.3 Continual Learning

We evaluate our approach discussed in Section 2.4.2 on single-head permuted-MNIST and multi-head split-MNIST, which are standard continual learning benchmarks. Following prior works, we use two-layer fully-connected neural network with 100 hidden units per layer for permuted MNIST and two-layer fully-connected neural network with 256 hidden units per layer for split-MNIST. We use 200 coreset points for permuted-MNIST and 40 coreset points for split-MNIST, in accordance with prior work.

For all continual learning experiments, 60,000 data samples are used for training and 10,000 data samples are used for testing. The input are converted to float values in the range of [0, 1].

In the multi-head setup, a model receives 5 sequential datasets and uses a different output head for each task. The original 10 classes in MNIST is split into 5 different binary classification tasks. The

network is a multilayer perceptron with two layers and 100 hidden units for each layer. 40 coreset points are randomly chosen at each task[3].

In the single-head setup, a model receives 10 sequential datasets and uses a same output head for each task. At every task, the MNIST images undergo a random permutation of pixels. The network is a multilayer perceptron with two layers and 256 hidden units for each layer. 200 coreset points are randomly chosen at each task.

For both multi-head split MNIST and single-head permuted MNIST, we use the Adam optimizer of learning rate 10^{-3} with default settings of $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$ and we use batch size of 128 in all experiments.

Method	Permuted MNIST	Split MNIST
EWC	84.0%$\pm$0.3	63.1%$\pm$0.2
SI	86.0%	98.9%
VCL	92.5%$\pm$0.1	97.8%$\pm$0.0
FROMP	94.9%$\pm$0.0	99.0%$\pm$0.0
FRCL	94.3%$\pm$0.0	97.8%$\pm$0.2
L_2	93.5%$\pm$0.0	98.4%$\pm$0.1
ours	**95.2%**$\pm$0.0	**99.3%**$\pm$0.0

Table 2.2: Comparisons of predictive average accuracy computed across all S tasks against a selection of popular continual learning methods.

[3]There are other ways to select coreset such as k-centering, but this is not a continual learning paper so we only choose coreset randomly.

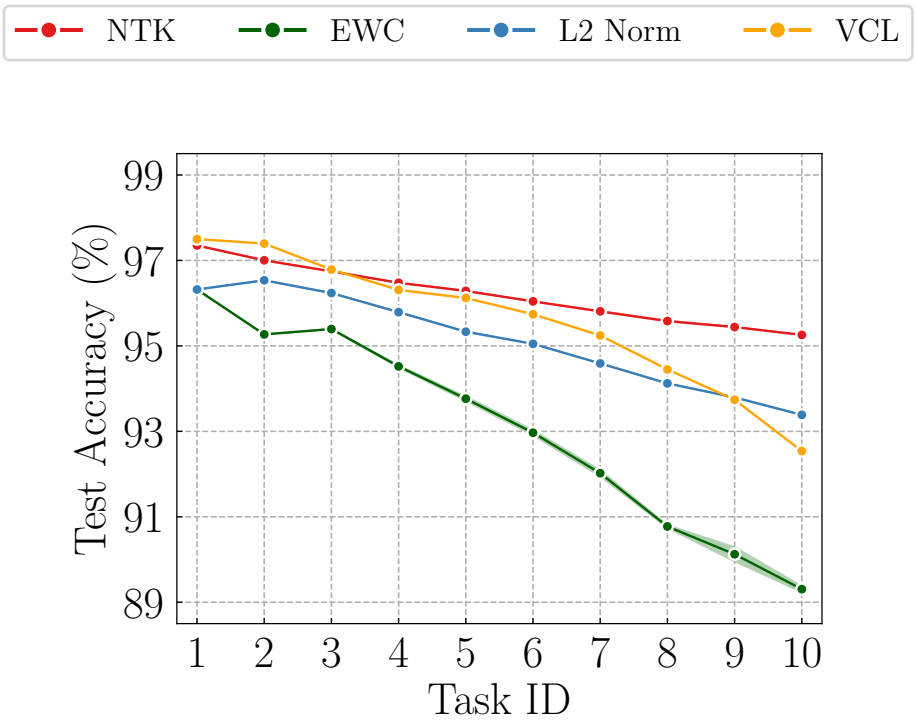


Figure 2.3: NTK regularization performance on permuted MNIST.

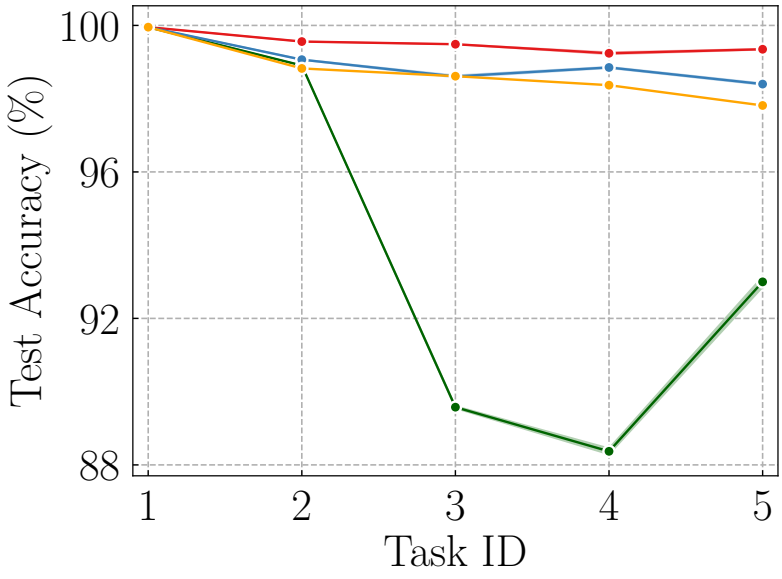


Figure 2.4: NTK regularization performance on split MNIST.

We can see from Figure 2.3 and 2.4 and Table 2.2 that function-space methods provide higher

average accuracy across all tasks than parameter-space methods, which shows that function space regularization results in better memorization of past abilities. Among all function space approaches, our method provides the highest average accuracy. We attribute our performance to effective regularization on the relative function change in the RKHS $\mathcal{H}_{\Theta_s}$, which ensures that the function to be learned at the next task $s+1$ still lies in $\mathcal{H}_{\Theta_s}$ and consequently avoids forgetting of the previous tasks.

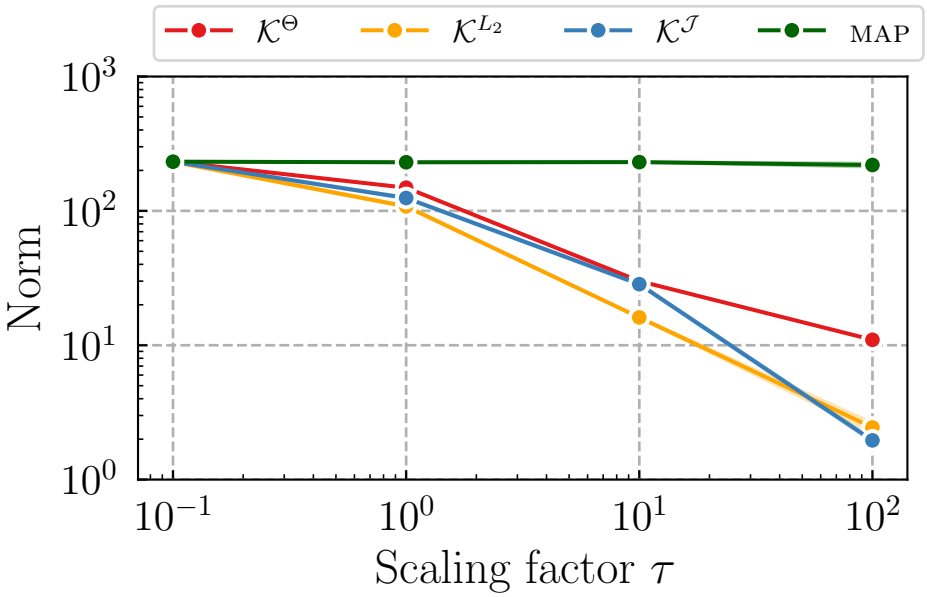


Figure 2.5: Ablation study of the dependence on scaling factor τ.

2.5.4 Ablation study

We see from Figure 2.5 that the $\mathcal{K}^{\Theta}$ norm decreases as scaling factor τ grows, showing that the regularization term in Eqn.(2.10) is taking effect. We observe a discrepancy between the MAP and $\mathcal{K}^{\Theta}$ norms, which shows that weight penalty cannot limit the complexity of functions. This agrees with findings in Benjamin et al. [5] and Zhang et al. [102]. Furthermore, we notice that the $\mathcal{K}^{\mathcal{J}}$ and $\mathcal{K}^{L_2}$ norms also decrease even if we are not explicitly regularizing them. This phenomenon can be

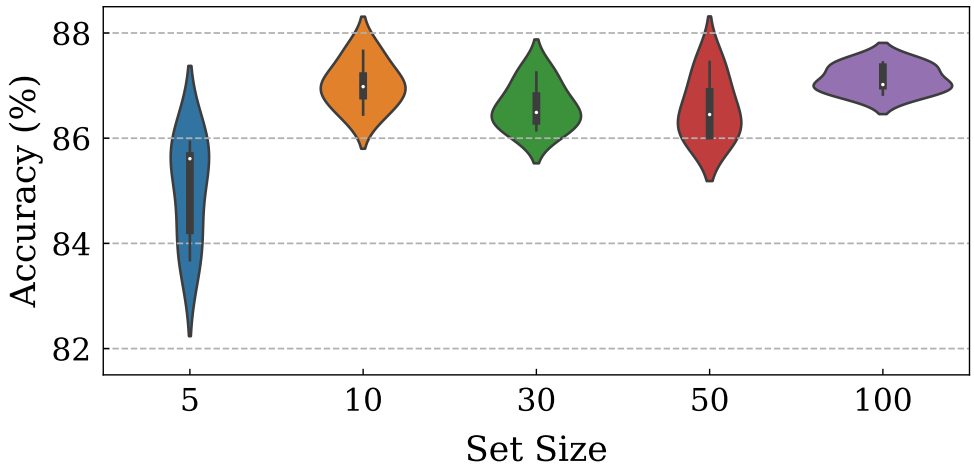


Figure 2.6: Ablation on batch estimation

Method	$\mathcal{K}^{\Theta}$	$\mathcal{K}^{\mathcal{J}}$	Dropout	MAP
Time	170	141	26	25

Table 2.3: Wall-clock-time (s)

explained in that although L_2 norm ignores the Hilbert structure of the function space, it is a better measure of function complexity than MAP, and the $\mathcal{K}^{\mathcal{J}}$ norm is upper bounded by $\mathcal{K}^{\Theta}$ norm [7].

2.5.5 Computational Overhead

Although our approach requires the inversion of a kernel matrix of size $|\mathbf{X}_{\mathcal{I}}| \times |\mathbf{X}_{\mathcal{I}}|$, we can subjectively choose the size of $\mathbf{X}_{\mathcal{I}}$ to effectively control the computational cost, because we used batch estimation (Approximation 1) to approximate the computation of function norm. And our ablation study in Figure 2.6 shows that, as long as $|\mathbf{X}_{\mathcal{I}}|$ is chosen to be not too small, the performance will not significantly degrade.

We also provide a comparison on the time of every epoch under ResNet-18 using a single RTX 2080 Ti in Table 3. $\mathcal{K}^{\Theta}$ has similar computational cost as $\mathcal{K}^{\mathcal{J}}$, but demonstrates much stronger regularization effect in practice.

2.6 Proofs

In this section, we present the proof details of Theorem 2.2.1, Theorem 2.2.2 and Proposition 2.2.3.

2.6.1 Proof of Equivalence

Recall the definition of fully-connected neural networks:

$$\begin{aligned} f^{(h)}(\mathbf{x}) &= \mathbf{W}^{(h)} g^{(h-1)}(\mathbf{x}) \in \mathbb{R}^{d_h} \quad g^{(0)}(\mathbf{x}) = \mathbf{x} \\ g^{(h)}(\mathbf{x}) &= \sqrt{\frac{c_\sigma}{d_h}} \sigma\big(f^{(h)}(\mathbf{x})\big) \quad \forall h = 1, 2, \cdots, L \end{aligned} \tag{2.12}$$

And the last layer is given by

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = f^{(L+1)}(\mathbf{x}) = \mathbf{W}^{(L+1)} g^{(L)}(\mathbf{x}) \tag{2.13}$$

The optimization problem regularizing the RKHS norm induced by NTK is

$$\min_{\boldsymbol{\theta}} \mathcal{L} = \frac{1}{N} \sum_{j=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}^{(j)})) + \zeta f_{\boldsymbol{\theta}}(\mathbf{X})^\top \mathcal{K}(\mathbf{X}, \mathbf{X})^{-1} f_{\boldsymbol{\theta}}(\mathbf{X}) \tag{2.14}$$

We first study the dynamics of problem (2.14). Suppose the loss function is squared loss $\ell(f) = (f(\mathbf{x}) - \mathbf{y})^2$, the continuous SGD is

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = -\eta \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = -\eta \bigg(\frac{2}{N} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{X})^\top (f_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) + 2\zeta \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{X})^\top \mathcal{K}(\mathbf{X}, \mathbf{X})^{-1} f_{\boldsymbol{\theta}}(\mathbf{X}) \bigg) \tag{2.15}$$

Therefore, the training dynamics is

$$\frac{\mathrm{d}f_{\boldsymbol{\theta}}}{\mathrm{d}t}(\mathbf{X}) = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{X}) \frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = -\eta \bigg(\frac{2}{N} \Theta(\mathbf{X}, \mathbf{X})(f_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) + 2\zeta \Theta(\mathbf{X}, \mathbf{X}) \mathcal{K}(\mathbf{X}, \mathbf{X})^{-1} f_{\boldsymbol{\theta}}(\mathbf{X}) \bigg) \tag{2.16}$$

Note that in practice, we use the empirical kernel Θ in the training, and the inverse kernel matrix is not involved in auto-differentiation in each gradient update step. This gives another training dynamics:

$$\begin{aligned} \frac{\mathrm{d}f_{\boldsymbol{\theta}}}{\mathrm{d}t}(\mathbf{X}) &= -\eta \bigg(\frac{2}{N} \Theta(\mathbf{X}, \mathbf{X})(f_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) + 2\zeta \Theta(\mathbf{X}, \mathbf{X}) \Theta(\mathbf{X}, \mathbf{X})^{-1} f_{\boldsymbol{\theta}}(\mathbf{X}) \bigg) \\ &= -\eta \bigg(\frac{2}{N} \Theta(\mathbf{X}, \mathbf{X})(f_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) + 2\zeta f_{\boldsymbol{\theta}}(\mathbf{X}) \bigg) \end{aligned} \tag{2.17}$$

First, we prove these two dynamics are close, hence our practical algorithm can approximate the true solution very well.

Lemma 2.6.1. *Let f_{ntk} and f_{entk} be the solution of* (2.16) *and* (2.17) *with the same initial condition. Assume f is bounded, let $d_1 = d_2 = \cdots = d_L = m$, then*

$$\sup_{t\geq 0} \|f_{ntk}(\mathbf{X}) - f_{entk}(\mathbf{X})\| = \mathcal{O}(m^{-1/2}) \quad \text{as } m \to \infty \tag{2.18}$$

Proof. By comparing the dynamics in (2.16) and (2.17), we have

$$\begin{aligned}
&-\frac{1}{\eta}\left[\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}} - \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{\theta}}\right] \\
=&\frac{2}{N}\Theta_f(f_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) - \frac{2}{N}\Theta_g(g_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) + 2\zeta\Theta_f\mathcal{K}^{-1}f_{\boldsymbol{\theta}}(\mathbf{X}) - 2\zeta g_{\boldsymbol{\theta}}(\mathbf{X}) \\
=&\frac{1}{N}\big(2\Theta_f + 2\zeta\Theta_f\mathcal{K}^{-1}\big)(f_{\boldsymbol{\theta}}(\mathbf{X}) - g_{\boldsymbol{\theta}}(\mathbf{X})) + \big(\Theta_f - \Theta_g\big)\left(\frac{2}{N}(g_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) + 2\zeta\mathcal{K}^{-1}g_{\boldsymbol{\theta}}(\mathbf{X})\right)
\end{aligned} \tag{2.19}$$

Here for short, we use $\Theta_f = \Theta_f(\mathbf{X}, \mathbf{X})$ for empirical kernel matrix with respect to f (and g similarly). Let $\mathbf{A} = \frac{1}{N}\big(2\Theta_f + 2\zeta\Theta_f\mathcal{K}^{-1}\big)$, $\boldsymbol{v} = \big(\Theta_f - \Theta_g\big)\big(\frac{2}{N}(g_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}) + 2\zeta\mathcal{K}^{-1}g_{\boldsymbol{\theta}}(\mathbf{X})\big)$ and $h = f - g$. Also let λ_0 be the small eigenvalue of $\mathcal{K}$. Since $h(0) = 0$, we have

$$f_{ntk}(\mathbf{X}) - f_{entk}(\mathbf{X}) = h(t) = -\eta e^{-\eta\int_0^t \mathbf{A}\mathrm{d}t}\int_0^t e^{\int_0^s \eta\mathbf{A}\mathrm{d}t}\boldsymbol{v}(s)\mathrm{d}s \tag{2.20}$$

Note that for ultra-wide neural networks, we have $\|\Theta - \mathcal{K}\|_F = \mathcal{O}(m^{-1/2})$ (Theorem 2.1 [50]), hence for $\varepsilon \leq cm^{-1/2}$, we have

$$\|\Theta_f - \Theta_g\| \leq \|\Theta_f - \Theta_g\|_F \leq \|\Theta_f - \mathcal{K}\|_F + \|\Theta_g - \mathcal{K}\|_F \leq 2\varepsilon \tag{2.21}$$

So we have

$$\|\boldsymbol{v}\| \leq \|\Theta_f - \Theta_g\|\left(\frac{2}{N}\|g_{\boldsymbol{\theta}}(\mathbf{X}) - \mathbf{Y}\| + 2\zeta\left\|\mathcal{K}^{-1}g_{\boldsymbol{\theta}}(\mathbf{X})\right\|\right) \leq 4\varepsilon\left(C_1 + \zeta\frac{NC_0}{\lambda_0}\right) \tag{2.22}$$

Let $\mathbf{A}_0 = \frac{1}{N}(2\mathcal{K} + 2\zeta I)$, then

$$\|\mathbf{A} - \mathbf{A}_0\| \leq \frac{1}{N}(2\|\Theta_f - \mathcal{K}\| + 2\zeta\left\|\mathcal{K}^{-1}\right\|\|\Theta_f - \mathcal{K}\|) \leq \frac{2\varepsilon}{N}\left(1 + \frac{\zeta}{\lambda_0}\right) \tag{2.23}$$

Hence by Lemma 2.6.2, we have

$$\begin{aligned}\lambda_{\min}(\int_s^t \mathbf{A}\mathrm{d}s) \geq & \lambda_{\min}(\int_s^t \mathbf{A}_0\mathrm{d}s) - \left\|\int_s^t (\mathbf{A}-\mathbf{A}_0)\mathrm{d}s\right\| \\ \geq & \left[\lambda_0 - \frac{2\varepsilon}{N}\left(1+\frac{\zeta}{\lambda_0}\right)\right](t-s)\end{aligned} \tag{2.24}$$

Hence we see that

$$\begin{aligned}\|h(t)\| &\leq \eta\int_0^t \left\|e^{\int_t^s \eta\mathbf{A}\mathrm{d}t}\right\| \|\boldsymbol{v}\|\,\mathrm{d}s \\ &\leq 4\eta\varepsilon\left(C_1+\zeta\frac{NC_0}{\lambda_0}\right)\int_0^t e^{-\eta\lambda_{\min}(\int_s^t \mathbf{A}\mathrm{d}t)}\mathrm{d}s \\ &\leq 4\eta\varepsilon\left(C_1+\zeta\frac{NC_0}{\lambda_0}\right)\int_0^t e^{\eta C'(s-t)}\mathrm{d}s \qquad C' = \lambda_0 - \frac{2\varepsilon}{N}\left(1+\frac{\zeta}{\lambda_0}\right) \\ &\leq 4\eta\varepsilon\left(C_1+\zeta\frac{NC_0}{\lambda_0}\right)\frac{1}{\eta C'}(1-e^{-\eta C' t}) \\ &\leq \frac{4\varepsilon}{C'}\left(C_1+\zeta\frac{NC_0}{\lambda_0}\right)\end{aligned} \tag{2.25}$$

Which proves the lemma. □

Lemma 2.6.2. *Let A, B be $n\times n$ Hermitian matrices, arrange the eigenvalues in descending order as $\lambda_1(A)\geq\lambda_2(A)\geq\cdots\geq\lambda_n(A)$ (and for B as well). Then for all $1\leq i\leq n$, we have*

$$|\lambda_i(A+B)-\lambda_i(A)|\leq\|B\| \tag{2.26}$$

Proof. Courant-Fischer min-max theorem (Reed and Simon [72]) implies for any Hermitian matrix, we have

$$\lambda_i(A) = \sup_{\dim V=i}\inf_{v\in V,\|v\|=1} v^*Av \tag{2.27}$$

$$\lambda_i(A) = \inf_{\dim V=n-i+1}\sup_{v\in V,\|v\|=1} v^*Av \tag{2.28}$$

According to Equation (2.27), we can find a subspace U with $\dim U=i$, such that $\lambda_i(A)\geq u^*Au$ for all unit vector $u\in U$. And similarly using Equation (2.28), we can find a subspace W with $\dim W=n-i+1$, we have $\lambda_i(A+B)\leq w^*(A+B)w$. Since $\dim U\cap W=\dim U+\dim W-\dim U\cup W\geq i+n-i+1-n=1$, we find a non-trivial solution v such that

$$\lambda_i(A+B)-\lambda_i(A)\leq v^*(A+B)v-v^*Av=v^*Bv\leq\|B\| \tag{2.29}$$

Similarly we can prove $\lambda_i(A+B) - \lambda_i(A) \geq -\|B\|$ using the same argument. These two inequalities prove the claim of the lemma. □

Theorem 2.6.3. *Suppose* $\sigma(z) = \max(0, z)$ *be the ReLU activation function,* $1/\kappa = \mathrm{poly}(1/\varepsilon, \log(N/\delta))$, $d_1 = d_2 = \cdots = d_L = m$ *with* $m \geq \mathrm{poly}(1/\kappa, L, 1/(\lambda_0 + \zeta N))$. *Then for any* $\mathbf{x}$ *such that* $\|\mathbf{x}\| = 1$, *with probability at least* $1 - \delta$ *over the random initialization, we have*

$$|f_{nr}(\mathbf{x}) - f_{ntr}(\mathbf{x})| \leq \varepsilon \tag{2.30}$$

Proof. Note that the training dynamics 2.17 can be written as

$$\frac{\mathrm{d}f_{\boldsymbol{\theta}}}{\mathrm{d}t} + \frac{2\eta}{N}(\Theta + \zeta N\mathbf{I})f_{\boldsymbol{\theta}} = \frac{2\eta}{N}\Theta\mathbf{Y} \tag{2.31}$$

if we replace Θ by the analytic kernel $\mathcal{K}$, the linear system above becomes

$$\frac{\mathrm{d}f_{\boldsymbol{\theta}}}{\mathrm{d}t} + \frac{2\eta}{N}(\mathcal{K} + \zeta N\mathbf{I})f_{\boldsymbol{\theta}} = \frac{2\eta}{N}\mathcal{K}\mathbf{Y} \tag{2.32}$$

Since $\mathcal{K}$ is constant, we have a closed form solution

$$f_t(\mathbf{x}) = \mathcal{K}(\mathbf{x}, \mathbf{X})\big(\mathcal{K}(\mathbf{X}, \mathbf{X}) + \zeta N\mathbf{I}\big)^{-1}\left(I - e^{-\frac{2\eta}{N}t\mathcal{K}(\mathbf{X},\mathbf{X})}\right)\mathbf{Y} \tag{2.33}$$

This is exactly the solution of kernel ridge regression under NTK when $t \to \infty$, i.e. $f_t \to f_{ntr}$. Taking the difference of the two linear systems, we have

$$\frac{\mathrm{d}h}{\mathrm{d}t} + \frac{2\eta}{N}(\mathcal{K} + \zeta N\mathbf{I})h = \frac{2\eta}{N}(\Theta - \mathcal{K})(\mathbf{Y} - f) \tag{2.34}$$

Let $\mathbf{B} = \frac{2\eta}{N}(\mathcal{K} + \zeta N\mathbf{I})$ and $\mathbf{u} = \frac{2\eta}{N}(\Theta - \mathcal{K})(\mathbf{Y} - f)$, we see that

$$f_{entk}(\mathbf{X}) - f_{ntr}(\mathbf{X}) = h(t) = e^{-\int_0^t \mathbf{B}\mathrm{d}t}\int_0^t e^{\int_0^s \mathbf{B}\mathrm{d}t}\mathbf{u}(s)\mathrm{d}s \tag{2.35}$$

This implies

$$\begin{aligned} |h(t)| &\leq \int_0^t \left\| e^{\int_s^t \mathbf{B}\mathrm{d}t} \right\| \|\mathbf{u}\| \,\mathrm{d}s \\ &\leq \frac{2\eta}{N}\|\Theta - \mathcal{K}\|\,\|\mathbf{Y} - f(\mathbf{X})\|\,\frac{1}{\|\mathbf{B}\|}\big(1 - e^{-t\|\mathbf{B}\|}\big) \leq \frac{NC_0\varepsilon}{\lambda_0 + \zeta N} \end{aligned} \tag{2.36}$$

where C_0 is constant bounds the function value and λ_0 is the smallest eigenvalue of NTK. Equation (2.36) together with Theorem 2.6.1 prove the Theorem on training set.

To complete the proof for any test point $\mathbf{x}$, note that we only need to mimic the proof of Lemma F.1 in [2]. All the bounds remain the same, except now $\mathcal{K}$ is replaced by $\mathcal{K} + \zeta N\mathbf{I}$, so λ_0 is replaced by $\lambda_0 + \zeta N$. □

2.6.2 Proof of Generalization Bound

We now give a proof of the generalization bound.

Theorem 2.6.4. *Let f_{nr} be defined as the optimal solution of* (2.14) *as above. Let $R(f_{nr}) = \mathbb{E}[(f_{nr}(\mathbf{x}) - \mathbf{Y}(\mathbf{x}))^2]$ be the generalization error, and $\hat{R}(f_{nr})$ the corresponding empirical error. Assume f_{nr} has bounded function value and first order derivative, i.e., $|f_{nr}(\mathbf{x})| \leq C_0\kappa$ and $\|\partial_{\boldsymbol{\theta}} f_{nr}(\mathbf{x})\| \leq C_1$ for all $\|\mathbf{x}\| \leq 1$, assume $\mathbb{E}|\mathbf{Y}| = c$. Then for any $\delta > 0$, with probability at least $1 - \delta$, we have the following bound:*

$$R(f_{nr}) - \hat{R}(f_{nr}) \leq 4(C_0\kappa + c)\varepsilon + \frac{8C_1^2\Lambda^2}{\sqrt{N}}\left(\sqrt{\frac{C_K}{NC_1^2}} + \frac{3}{4}\sqrt{\frac{\log\frac{2}{\delta}}{2}}\right) \tag{2.37}$$

where N is the number of training points and Λ is a constant only depends on the regularization constant ζ. C_K is the trace of NTK.

Proof. First note that the definition of neural network function in the very beginning is homogeneous since it is bias free, and ReLU is also homogeneous. Hence for any t, we have $f(t\mathbf{x}) = tf(\mathbf{x})$. Therefore, it is sufficient to assume $\|\mathbf{x}\| \leq 1$ for all $\mathbf{x} \in \mathcal{X}$.

Since we have proven the equivalence between neural network optimization and kernel ridge regression, we have

$$R(f_{nr}) - R(f_{ntr}) = \mathbb{E}\big[\big(f_{nr}(\mathbf{x}) - f_{ntr}(\mathbf{x})\big)\big(f_{nr}(\mathbf{x}) + f_{ntr}(\mathbf{x}) - 2\mathbf{Y}(\mathbf{x})\big)\big] \leq 2\varepsilon(C_0\kappa + c) \tag{2.38}$$

where C_e is the supremum of the error term. Similarly, we have

$$\hat{R}(f_{ntr}) - \hat{R}(f_{nr}) \leq 2\varepsilon(C_0\kappa + c) \tag{2.39}$$

Therefore,

$$R(f_{nr}) - \hat{R}(f_{nr}) \leq R(f_{ntr}) - \hat{R}(f_{ntr}) + 4\varepsilon(C_0\kappa + c) \tag{2.40}$$

The classical generalization bound in Theorem 1.2.1 implies

$$R(f_{ntr}) - \hat{R}(f_{ntr}) \leq \frac{8r^2\Lambda^2}{\sqrt{N}}\left(\sqrt{\frac{\mathrm{Tr}(\mathcal{K})}{Nr^2}} + \frac{3}{4}\sqrt{\frac{\log\frac{2}{\delta}}{2}}\right) \tag{2.41}$$

where r and Λ are constants such that $\mathcal{K}(\mathbf{x}, \mathbf{x}) \leq r^2$ and $\|f\|_{\mathcal{H}_K} \leq \Lambda$. Note that the condition $\|f\|_{\mathcal{H}_K} \leq \Lambda$ corresponds to the regularizer, hence Λ is uniquely determined by ζ. We can take $r = C_1$ since $\mathcal{K}(\mathbf{x}, \mathbf{x}) \approx \langle \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}), \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}) \rangle \leq C_1^2$ under the ultra-wide assumption.

It has been proven in [27] that the eigenvalues of NTK satisfy $ck^{-d} \leq \lambda_k \leq Ck^{-d}$ for all $k > k_0$, where d is the dimension of input data points. Therefore, $\mathrm{Tr}(\mathcal{K}) \leq \lambda_1 + \cdots + \lambda_{k_0} + C\sum_{k>k_0} k^{-d} := C_K < \infty$. Combining these facts completes the proof of the theorem. □

2.6.3 Proof of Proposition

This proof follows the steps in [74]. We denote $\boldsymbol{\theta} \in \mathbb{R}^P$ as parameters of a stochastic neural networks that follows a Gaussian distribution $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 I_P)$. Unlike in the main text, here we denote functions encoded by the neural network as $f(\cdot; \boldsymbol{\theta})$ to explicitly show that f is parameterized by $\boldsymbol{\theta}$. In proposition 2, the function f is evaluated on finite number of samples $\mathbf{X}_{\mathcal{I}}$, here we get rid of the subscript $\mathcal{I}$ and use $\mathbf{X}$ directly as there is no confusion here.

Note that $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \sigma^2 I_P)$, so we can linearize the function f around the parameter mean $\boldsymbol{\mu}$. Let $\mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}) := \nabla_{\boldsymbol{\mu}} f(\mathbf{x}; \boldsymbol{\mu})$, then the function evaluted at finite number of samples $f(\mathbf{X}; \boldsymbol{\theta})$ can be approximated as $f(\mathbf{X}; \boldsymbol{\theta}) \approx \tilde{f}(\mathbf{X}; \boldsymbol{\theta}) = f(\mathbf{X}; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{X})(\boldsymbol{\theta} - \boldsymbol{\mu})$. Therefore, the distribution of $\tilde{f}(\mathbf{X}; \boldsymbol{\theta})$ is a Gaussian multivariate distribution fully defined by some predictive mean $m(\mathbf{x})$ and predictive covariance $S(\mathbf{x}, \mathbf{x}')$, for $\forall \mathbf{x}, \mathbf{x}' \in \mathbf{X}$:

$$m(\mathbf{x}) = \mathbb{E}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta})] \tag{2.42}$$

and

$$S(\mathbf{x}, \mathbf{x}') = \mathrm{Cov}(\tilde{f}(\mathbf{x}; \boldsymbol{\theta}), \tilde{f}(\mathbf{x}'; \boldsymbol{\theta})) = \mathbb{E}[(\tilde{f}(\mathbf{x}; \boldsymbol{\theta}) - \mathbb{E}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta})])(\tilde{f}(\mathbf{x}'; \boldsymbol{\theta}) - \mathbb{E}[\tilde{f}(\mathbf{x}'; \boldsymbol{\theta})])^\top] \tag{2.43}$$

To see that $m(\mathbf{x}) = \mathbb{E}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta})] = 0$, note that, by linearity of expectation, we have

$$\begin{aligned} m(\mathbf{x}) = \mathbb{E}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta})] &= \mathbb{E}[f(\mathbf{x}; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x})(\boldsymbol{\theta} - \boldsymbol{\mu})] \\ &= f(\mathbf{x}; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x})(\mathbb{E}[\boldsymbol{\theta}] - \boldsymbol{\mu}) = f(\mathbf{x}; \boldsymbol{\mu}) \end{aligned} \tag{2.44}$$

To see that $S(\mathbf{x}, \mathbf{x}') = \mathrm{Cov}(\tilde{f}(\mathbf{x}; \boldsymbol{\theta}), \tilde{f}(\mathbf{x}'; \boldsymbol{\theta})) = \sigma^2 \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}) \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')^\top$, note that in general, $\mathrm{Cov}(\mathbf{x}, \mathbf{x}) = \mathbb{E}[\mathbf{x}\mathbf{x}^\top] + \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{x}]^\top$, and hence,

$$\mathrm{Cov}(\tilde{f}(\mathbf{x}; \boldsymbol{\theta}), \tilde{f}(\mathbf{x}'; \boldsymbol{\theta})) = \mathbb{E}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta}) \tilde{f}(\mathbf{x}'; \boldsymbol{\theta})^\top] - \mathbb{E}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta})]\mathbb{E}[\tilde{f}(\mathbf{x}'; \boldsymbol{\theta})]^\top \tag{2.45}$$

We already know that $\mathbb{E}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta})] = f(\mathbf{x}; \boldsymbol{\mu})$, so we only need to find $\mathbb{E}[f(\mathbf{x}; \boldsymbol{\theta}) f(\mathbf{x}'; \boldsymbol{\theta})^\top]$:

$$\begin{aligned} &\mathbb{E}_{p(\boldsymbol{\theta})}[\tilde{f}(\mathbf{x}; \boldsymbol{\theta}) \tilde{f}(\mathbf{x}'; \boldsymbol{\theta})^\top] \\ =&\mathbb{E}_{p(\boldsymbol{\theta})}[(f(\mathbf{x}; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x})(\boldsymbol{\theta} - \boldsymbol{\mu}))(f(\mathbf{x}'; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')(\boldsymbol{\theta} - \boldsymbol{\mu}))^\top] \\ =&\mathbb{E}_{p(\boldsymbol{\theta})}[f(\mathbf{x}; \boldsymbol{\mu}) f(\mathbf{x}'; \boldsymbol{\mu})^\top + (\mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x})(\boldsymbol{\theta} - \boldsymbol{\mu}))(\mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')(\boldsymbol{\theta} - \boldsymbol{\mu}))^\top \\ &+ f(\mathbf{x}; \boldsymbol{\mu})(\mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')(\boldsymbol{\theta} - \boldsymbol{\mu}))^\top + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x})(\boldsymbol{\theta} - \boldsymbol{\mu}) f(\mathbf{x}'; \boldsymbol{\mu})^\top] \\ =&f(\mathbf{x}; \boldsymbol{\mu}) f(\mathbf{x}'; \boldsymbol{\mu})^\top + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}) \mathbb{E}_{p(\boldsymbol{\theta})}[(\boldsymbol{\theta} - \boldsymbol{\mu})(\boldsymbol{\theta} - \boldsymbol{\mu})^\top] \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')^\top \\ &+ f(\mathbf{x}; \boldsymbol{\mu})(\mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')(\underbrace{\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}] - \boldsymbol{\mu}}_{=0}))^\top + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x})(\underbrace{\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}] - \boldsymbol{\mu}}_{=0}) f(\mathbf{x}'; \boldsymbol{\mu})^\top \\ =&f(\mathbf{x}; \boldsymbol{\mu}) f(\mathbf{x}; \boldsymbol{\mu})^\top + \sigma^2 \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}) \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')^\top \end{aligned} \tag{2.46}$$

Therefore, we obtain the covariance function

$$S(\mathbf{x}, \mathbf{x}') = \sigma^2 \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}) \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')^\top \tag{2.47}$$

So far we have proved a more general form than proposition 2. In deep neural networks, the parameters $\boldsymbol{\theta}$ is usually initialized around $\mathbf{0}$ [36], which indicates that mean $m(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\mu}) = 0$. However, the Jacobian matrix $\mathcal{J}_{\mathbf{0}}$ would also become $\mathbf{0}$, which makes the covariance matrix

meaningless. So we compute the Jacobian with respect to an actual realization $\boldsymbol{\theta}$ instead of the mean. So now we have the following:

$$p(f_{\boldsymbol{\theta}}(\mathbf{X})) \approx \tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X})) = \mathcal{N}(\mathbf{0}, \sigma^2 \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{X}) \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{X})^\top) \tag{2.48}$$

which concludes the proof.

The linearization might not be accurate because the prior distribution $\mathcal{N}(\mathbf{0}, \sigma^2 I_P)$ usually has large σ and parameters for deep neural networks are high-dimensional so a sample drawn from the prior distribution might be far from the mean. Although it means that $\tilde{p}(\tilde{f}(\mathbf{X}; \boldsymbol{\theta}))$ is only a rough approximation of $p(f)$, $\tilde{p}(\tilde{f})$ still effectively incorporates the information of prior distribution of parameters and the neural network architecture into the prior distribution of functions. Moreover, using other priors over functions does not violate the general structure of FS-MAPand different priors would lead to different forms of function-space regularization for neural networks, which would be an interesting direction for future work.

2.7 Measure theoretical description of FS-MAP

de Garis Matthews [18] has provided a rigorous definition of FS-MAP in the form of a measure theoretic version of Bayes' theorem. However, the measure theoretic version of FS-MAPdoes not give a definition for the log prior term in Equation (2.8). So in this section, we discuss how to define a prior over function spaces and then compute the log prior term in Equation (2.8).

Notations and Background Materials A measure space is described by a triple $(\mathcal{X}, \sigma_{\mathcal{X}}, \mu)$, where $\mathcal{X}$ is the underlying space, $\sigma_{\mathcal{X}}$ is the σ-algebra and μ is the measure. μ is a probability measure if $\mu(\mathcal{X}) = 1$, and we will denote $p(\mathbf{x})$ as the probability density function of μ if applicable. Let $f : \mathcal{X} \to \mathbb{R}$ be any function on $\mathcal{X}$ and $\mathbb{R}^{\mathcal{X}} = \prod_{\mathbf{x} \in \mathcal{X}} \mathbb{R}^{\mathbf{x}}$ be the direct product of $\mathbb{R}$. Note that any function f can be identified by the evaluation on $\mathcal{X}$ via $f \mapsto [f(\mathbf{x})]_{\mathbf{x} \in \mathcal{X}} := f(\mathcal{X})$, hence f can be viewed as a vector of infinite length in $\mathbb{R}^{\mathcal{X}}$.

A Note On Indexing by Uncountably Infinite Set If $\mathcal{X}$ is finite or countable, there is no ambiguity in the construction of $\mathbb{R}^{\mathcal{X}}$ and the evaluation map $ev_{\mathcal{X}}(f) = [f(\mathbf{x})]_{\mathbf{x}\in\mathcal{X}}$ is well-defined if we arrange the points in $\mathcal{X}$ with a given order. The axiom of choice ensures we can extend this construction to the case of uncountable $\mathcal{X}$. Roughly speaking, we are able to choose a proper 'order' of uncountable index set so that the target $f(\mathcal{X})$ is determined for any given function f.

Definition 2.7.1 (Pushforward measure)**.** *Given measure spaces $(\mathcal{X}, \sigma_{\mathcal{X}}, \mu)$ and $(\mathcal{Y}, \sigma_{\mathcal{Y}}, \nu)$, let $g : \mathcal{X} \to \mathcal{Y}$ be a measurable function. We say the measure ν is pushforward of μ through g if $\nu(A) = \mu(g^{-1}(A))$ for any measurable set $A \in \sigma_{\mathcal{Y}}$.*

Definition 2.7.2 (Cylindrical σ-algebra)**.** *Let*

$$\mathbb{R}^{\mathcal{X}} = \{f : f(\mathbf{x}) \in \mathbb{R}, \mathbf{x} \in \mathcal{X}\} \tag{2.49}$$

be the set of real valued functions as above. A cylinder subset is a finitely restricted set defined as

$$C_{\mathbf{x}_1,\cdots,\mathbf{x}_n}(A_1, \cdots, A_n) = \{f \in \mathbb{R}^{\mathcal{X}} : f(\mathbf{x}_i) \in A_i, \forall 1 \le i \le n)\} \tag{2.50}$$

The cylindrical σ-algebra is the smallest σ-algebra generated by all cylinder subsets. To be precise, let

$$\mathcal{G}_{\mathbf{x}_1,\cdots,\mathbf{x}_n} = \{C_{\mathbf{x}_1,\cdots,\mathbf{x}_n}(A_1, \cdots, A_n) : A_i \in \sigma_{\mathbb{R}}, \forall 1 \le i \le n\} \tag{2.51}$$

$$\mathcal{G}_{\mathbb{R}^{\mathcal{X}}} = \bigcup_{n=1}^{\infty} \bigcup_{\mathbf{x}_i\in\mathcal{X}, i\le n} \mathcal{G}_{\mathbf{x}_1,\cdots,\mathbf{x}_n} \tag{2.52}$$

Then the cylindrical σ-algebra of $\mathbb{R}^{\mathcal{X}}$ is the smallest σ-algebra containing $\mathcal{G}_{\mathbb{R}^{\mathcal{X}}}$. Note $\sigma_{\mathbb{R}}$ is commonly chosen to be the σ-algebra of Borel sets.

Note that, if $\mathcal{X}$ is infinite, the cylindrical σ-algebra is in general not the same as product σ-algebra, which is too large to be considered. Intuitively, the generators C of cylindrical σ-algebra define a product σ-algebra on finite dimensional subspaces, which means when considering elements in $\mathbb{R}^{\mathcal{X}}$, we are actually considering the finite realizations.

Definition 2.7.3 (Canonical projection)**.** *Suppose $\mathbf{X}_\mathcal{I}$ and $\mathbf{X}_\mathcal{J}$ are two subsets of $\mathcal{X}$ such that $\mathbf{X}_\mathcal{J} \subseteq \mathbf{X}_\mathcal{I}$. We define the canonical projection map as*

$$\begin{aligned} \pi_{\mathcal{I}\to\mathcal{J}}: \qquad \mathbb{R}^{\mathbf{X}_\mathcal{I}} \quad &\longrightarrow \qquad \mathbb{R}^{\mathbf{X}_\mathcal{J}} \\ w = [w_\mathbf{x}]_{\mathbf{x}\in\mathbf{X}_\mathcal{I}} &\mapsto w|J = [w_\mathbf{x}]_{\mathbf{x}\in\mathbf{X}_\mathcal{J}} \end{aligned}$$

Note that $\pi_{\mathcal{I}\to\mathcal{J}}$ is actually induced by the natural projection $\mathbf{X}_\mathcal{I} \to \mathbf{X}_\mathcal{J}$ with respect to the inclusion $\mathbf{X}_\mathcal{J} \subseteq \mathbf{X}_\mathcal{I}$. We consider $\mathbf{X}_\mathcal{I}$ and $\mathbf{X}_\mathcal{J}$ are further indexed by $\mathcal{I}$ and $\mathcal{J}$, hence $\pi_{\mathcal{I}\to\mathcal{J}}$ is induced by $\pi : \mathcal{I} \to \mathcal{J}$ with respect to the inclusion $\mathcal{J} \subseteq \mathcal{I}$. Therefore, we will abuse the notations for objects indexed by $\mathbf{X}_\mathcal{I}$ or $\mathcal{I}$ in the following text.

Theorem 2.7.4 (Kolmogorov extension theorem))**.** *Let $\mathbb{R}^\mathcal{X}$ be the measurable space with cylindrical σ-algebra defined as above. Suppose for each finite subset $\mathbf{X}_\mathcal{I} \subseteq \mathcal{X}$, we have a probability measure $\mu_\mathcal{I}$ on $\mathbb{R}^{\mathbf{X}_\mathcal{I}}$, which satisfy the following compatibility relationship: for each subset $\mathbf{X}_\mathcal{J} \subset \mathbf{X}_\mathcal{I}$, we have*

$$\mu_\mathcal{J} = \mu_\mathcal{I} \circ \pi^{-1}_{\mathcal{I}\to\mathcal{J}} \tag{2.53}$$

Then there exists a unique probability measure μ on $\mathbb{R}^\mathcal{X}$ such that for all finite subsets $\mathbf{X}_\mathcal{I} \subset X$,

$$\mu_\mathcal{I} = \mu \circ \pi^{-1}_{\mathcal{X}\to\mathbf{X}_\mathcal{I}} \tag{2.54}$$

Kolmogorov extension theorem allows us to glue probability measures $\mu_\mathcal{I}$ defined on finitely dimensional subspaces together to get probability measure μ on the whole space, which is infinitely dimensional, as long as these $\mu_\mathcal{I}$s are consistent. Some presentation of the extension theorem may require $\mu_\mathcal{I}$s are invariant under permutations of $\mathbf{X}_\mathcal{I}$. Since we are considering elements in $\mathbb{R}^\mathcal{X}$, we have already assigned a particular 'order'[4] of $\mathcal{X}$. Therefore, permutation on $\mathcal{I}$ will give the same representation of $\mathbf{X}_\mathcal{I}$.

[4]For more concrete description of the order, one may refer any book about set theory. The extension theorem can be viewed as a special case of direct limit, if the reader is familiar with category theory.

Construction of Function Prior Now consider the function $f_{\boldsymbol{\theta}}$ described by a given neural network at initialization. Let $\mathbf{X}_{\mathcal{I}}$ be an i.i.d. sample from $\mathcal{X}$ sampled according to some probability density $p(x)$. Note that $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ is a random vector, so we can form the joint probability distribution of $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ and obtain a probability measure $\mu_{\mathcal{I}}$ on $\mathbb{R}^{\mathbf{X}_{\mathcal{I}}}$. If $\mu_{\mathcal{I}}$ satisfy the compatibility condition (2.53), then Kolmogorov extension theorem implies there is a probability measure μ on $\mathbb{R}^{\mathcal{X}}$.

Note that the density of a probability measure μ is the Radon-Nikodym derivative of μ with respect to the Lebesgue measure m, i.e $p(\mathbf{x}) = \frac{\mathrm{d}\mu}{\mathrm{d}m}$. However, there is no Lebesgue measure on infinitely dimensional space since any translation invariant measure is either zero or infinite. To tackle this issue, notice that for any measurable subset A of $\mathbb{R}^{\mathcal{X}}$, A can be decomposed into a countable union of cylinder subsets. That is to say,

$$P(f \in A) = \int_A d\mu(f) = \sum_{j=1}^{\infty} \int_{A_j} \mathrm{d}\mu(f) \tag{2.55}$$

where $A_j \in C_{\mathbf{X}_{\mathcal{I}_j}}$. This infinite sum can be well approximated by the partial sum $P_n = \sum_{j=1}^{n} \int_{A_j} \mathrm{d}\mu(f)$ in the sense that if n is sufficiently large, $|P - P_n| \leq \varepsilon$ for any precision ε. Therefore, we are able to find a finite index $\mathcal{D} = \bigcup_{j=1}^{n} \mathcal{I}_j$ such that $\pi^{-1}_{\mathcal{D}\to\mathcal{I}_j}(A_j) \subseteq \mathbf{X}_{\mathcal{D}}$ for all $1 \leq j \leq n$. So calculation in Equation (2.55) can be replaced by calculation in $\mathbb{R}^{\mathbf{X}_{\mathcal{D}}}$. To be precise, we have

$$P(f \in A) \approx \sum_{j=1}^{n} \int_{A_j} \mathrm{d}\mu(f) = \sum_{j=1}^{n} \int_{\pi_{\mathcal{D}\to\mathcal{I}_j}(A_j)} \mathrm{d}\mu_{\mathcal{D}}(f(\mathbf{X}_{\mathcal{D}})) = P_{\mathcal{D}}(\pi_{\mathcal{X}\to\mathbf{X}_{\mathcal{D}}}(A)) \tag{2.56}$$

In this case, we can naturally define

$$p(f) = p(f(\mathbf{X}_{\mathcal{D}})) \tag{2.57}$$

One may argue that the above choice of $\mathcal{D}$ depends on A. Note that in statistical learning theory, we often make the assumption that we have enough samples to recover the probability distribution to be learned. Hence a natural choice of a uniform $\mathbf{X}_{\mathcal{D}}$ is just the sample, in which case we consider A to be $\mathcal{X}$.

Now we check the compatibility condition. Fortunately, we have proved in Section 2.6.3 that $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ follows a Gaussian distribution under linearization. Moreover, for deep neural networks under the

infinite width approximation, if $\boldsymbol{\theta}$ is initialized to be standard normal, $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ follows a Gaussian distribution[5] with mean zero and a deterministic covariance matrix. Therefore, the compatibility condition is satisfied and the log prior term can be computed according to Equation (2.57).

[5] See [99, 69, 76, 98] for more details, and [59, 64] for a formal treatment.

Chapter 3

Adversarial Winning Tickets

3.1 Overview

As we have pointed out in Chapter 1 that state-of-the-art DNNs are vulnerable to attacks by *adversarial examples*. Such undesirable property may prohibit DNNs from being applied to security-sensitive applications. Although various of adversarial defense methods [29, 68, 75, 79, 84] were then proposed to prevent adversarial examples attack. Most of the defense methods were quickly broken by new adversarial attack methods, except *adversarial training*.

The min-max nature of adversarial training makes the computation costly for overparameterized neural networks. In the literature, network pruning is shown to be an outstanding method which can significantly reduce the model size while keep the network performance. Sparsity hence can potentially help speed up adversarial training. Moreover, theoretical works by Guo et al. [34] have shown that sparsity can also improve robustness. And surprisingly, Frankle and Carbin [23] proposed the so-called *Lottery Ticket Hypothesis* (LTH), stating that there exists subnetwork structure, which can achieve comparable performance as the dense model when trained in isolation. Based on these facts, we intend to find trainable sparse network which is also adversarial robust, which also provides evidence for the verification of LTH in adversarial setting.

However, since adversarial test accuracy is significantly different from the usual test accuracy. There is no evidence to believe pruning methods for natural training can be directly applied in adversarial context to obtain a robust subnetwork structure. To overcome this difficulty, in this part, we aim to control the training dynamics of the desired sparse network. On other words, if the sparse network has similar dynamics as the dense one, then such subnetwork can be adversarially trained to be robust by the robustness of the dense network.

The rest of this paper is organized as follows: We first present the main results in Section 3.2, then in Section 3.3, we briefly discuss the related works. After that we show in Section 3.4 the experimental results on real datasets to test the performance of AWT. Proof details are given in Section 3.5, and finally we discuss the possible extensions to more general case in Section 3.6 and present additional experimental results in Section 3.7. This chapter is based on the work in Shi et al. [83].

3.2 Dynamics Preserving Sub-Networks

In this section, we propose a way of finding trainable sparse network structure based on the study of adversarial training dynamics. Theoretically we can prove an error bound between the dense and the sparse model outputs, which implies AWT can be trained to be robust, hence verifies LTH in adversarial context.

3.2.1 Dynamics of Adversarial Training

Let $\mathcal{D} = \mathbf{X} \times \mathbf{Y} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$ be the empirical data distribution, $f_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^{k\times 1}$ the network function defined by a fully-connected network, and $f_{\boldsymbol{\theta}}(\mathbf{X}) = \mathrm{vec}\big([f_{\boldsymbol{\theta}}(\mathbf{x})]_{\mathbf{x}\in\mathbf{X}}\big) \in \mathbb{R}^{k|\mathcal{D}|\times 1}$ be the model outputs on training data.

Recall that adversarial training solves the following optimization problem:

$$\begin{aligned}\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{adv}} &= \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}} \max_{\mathbf{r}\in S_\varepsilon(\mathbf{x})} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}+\mathbf{r}), \mathbf{y}) \\ &= \frac{1}{N}\sum_{i=1}^{N} \max_{\mathbf{r}_i\in S_\varepsilon(\mathbf{x}_i)} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i+\mathbf{r}_i), \mathbf{y}_i)\end{aligned} \tag{3.1}$$

The inner sub-problem of this min-max optimization problem is usually solved by an effective attack algorithm. If we use $\tilde{\mathbf{x}}_j$ denote the adversarial example of $\mathbf{x}$ obtained at j-th step, then any k steps ℓ_p ($1 \le p \le \infty$) iterative attack algorithm with allowed perturbation strength ε can be formulated as follows:

$$\begin{aligned}&\tilde{\mathbf{x}}_0 = \mathbf{x}, \quad \tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_{t-1} + \mathbf{r}_t \quad \tilde{\mathbf{x}} = \tilde{\mathbf{x}}_k \\ &\text{s.t.} \left\|\mathbf{r}_i\right\|_p \le \delta \quad \left\|\sum \mathbf{r}_i\right\|_p \le \varepsilon \quad \forall 1 \le t \le k\end{aligned} \tag{3.2}$$

In practice, PGD attack as proposed in Madry et al. [57] is a common choice. Also, for bounded domains, clip operation need to be considered so that each $\tilde{\mathbf{X}}_t$ still belongs to the domain. However, such restriction is impossible to be analyzed in general. So we remove the restriction by assuming the sample space is unbounded. In this case, adversarial training algorithm updates the parameters by stochastic gradient descent on adversarial example batches. To be precise, we have the following discrete parameter updates:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}}(\tilde{\mathbf{X}}_t) \tag{3.3}$$

For an infinitesimal time $\mathrm{d}t$ with learning rate $\eta_t = \eta \mathrm{d}t$, one can obtain the continuous gradient descent by chain rules as follows:

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = \frac{\boldsymbol{\theta}_{t+\mathrm{d}t} - \boldsymbol{\theta}_t}{\mathrm{d}t} = -\eta \nabla_{\boldsymbol{\theta}}^T f_t(\tilde{\mathbf{X}}_t) \nabla_{f_t} \mathcal{L}(\tilde{\mathbf{X}}_t) \tag{3.4}$$

where we use the short notation $f_t(\mathbf{x}) = f_{\boldsymbol{\theta}_t}(\mathbf{x})$ and the following notation for convenience[1]:

$$\nabla_f \mathcal{L}(\mathbf{X}) = \begin{bmatrix} | \\ \nabla_f \ell(f(\mathbf{x}_i), \mathbf{y}_i) \\ | \end{bmatrix} \tag{3.5}$$

Accordingly, we can obtain the following theorem relating to dynamics of adversarial training.

Theorem 3.2.1. *Let $f_t(\mathbf{x})$ be the timely dependent network function describing adversarial training process and $\tilde{\mathbf{X}}_t$ the adversarial examples generated at time t by any chosen attack algorithm. Then f_t satisfies the following differential equation:*

$$\begin{aligned} \frac{\mathrm{d}f_t}{\mathrm{d}t}(\mathbf{X}) &= \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}) \frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} \\ &= -\frac{\eta}{N} \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}) \nabla_{\boldsymbol{\theta}}^T f_t(\tilde{\mathbf{X}}_t) \nabla_{f_t} \mathcal{L}(\tilde{\mathbf{X}}_t) \end{aligned} \tag{3.6}$$

Equation (3.6) is referred as the dynamics of adversarial training because it describes how the adversarially trained network function f_t evolves along time t. On the other hand, training a adversarial robust network $f_{\boldsymbol{\theta}}$ is the same as solving Equation (3.6) for given certain initial conditions.

[1] We drop the labels Y here since in adversarial training, the labels assigned to adversarial examples are the same as the clean ones.

Let $\Theta_t(\mathbf{x}, \mathbf{x}') = \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}) \nabla_{\boldsymbol{\theta}}^T f_t(\mathbf{x}')$ be the empirical *neural tangent kernel* (NTK), and let f^{nat} be the model function of natural training, then the dynamics of natural training are given by

$$\frac{\mathrm{d}\boldsymbol{\theta}_t^{nat}}{\mathrm{d}t} = -\eta \nabla_{\boldsymbol{\theta}}^T f_t^{nat}(\mathbf{X}) \nabla_{f_t^{nat}} \mathcal{L}_{nat}(\mathbf{X}) \tag{3.7}$$

$$\frac{\mathrm{d}f_t^{nat}}{\mathrm{d}t}(\mathbf{X}) = -\eta \Theta_t(\mathbf{X}, \mathbf{X}) \nabla_{f_t^{nat}} \mathcal{L}_{nat}(\mathbf{X}) \tag{3.8}$$

If we compare Equation (3.4) with Equation (3.7), we see that the gradient descent of adversarial training can be viewed as natural training with clean images $\mathbf{X}$ replaced by adversarial images $\tilde{\mathbf{X}}_t$ at each step. This matches our intuition because in practice, the parameter update is based on the adversarial examples as we discussed above, so adversarial training is closely related to natural training on adversarial images.

However, if we compare Equation (3.6) and Equation (3.8), we can see from the evolution of the model outputs that the usual NTK is now replaced by $\Theta_t(\mathbf{x}, \tilde{\mathbf{x}}_t) = \nabla_{\boldsymbol{\theta}} f_t(\mathbf{x}) \nabla_{\boldsymbol{\theta}}^T f_t(\tilde{\mathbf{x}}_t)$, which we call *Mixed Tangent Kernel* (MTK). Unlike NTK, MTK is not symmetric in general. It involves both clean images $\mathbf{X}$ and adversarial images $\tilde{\mathbf{X}}_t$. This indicates adversarial training is not simply a naturally model training on adversarial images, but some more complicated training method continuously couples clean images and adversarial images during training procedure. This coupling of clean images and adversarial images gives an intuition why adversarial training can achieve both good model accuracy and adversarial robustness.

3.2.2 Finding Adversarial Winning Ticket

To verify the LTH in adversarial setting, we need to find out a trainable sparse sub-network which has similar training dynamics as the dense network. We are then aiming to find a mask m with given sparsity density p such that the sparse classifier $f^s(\mathbf{x}) = f_{m \odot \boldsymbol{\theta}}(\mathbf{x})$ can be trained to be adversarial robust from scratch[2]. For simplicity, we assume, as in Lee et al. [50] and Liu and Zenke [54],

[2]Without loss of generality, we use superscript s to mean items correspond to sparse networks, while items without superscript correspond to dense networks.

Algorithm 2 Finding Adversarial Winning Ticket

Input: clean images $\mathbf{X}$, labels $\mathbf{Y}$, model structure f, dense initialization $\boldsymbol{\theta}_0$, learning rate η, adversarial perturbation strength ε, sparsity level k, mask update frequency T_m.

Initialize: initial weight $\mathbf{w}_0 = \boldsymbol{\theta}_0$, initial binary mask m based on $\mathbf{w}_0$, adversarial noises $\mathbf{R}_0$, $t = 1$.

for $t = 1$ to N **do**

  Sample a mini batch S and calculate the gradient $\nabla_{\mathbf{w}} \mathcal{L}_{awt}$ on S.

  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}_{awt} - \beta m \odot \mathbf{w}$

  if $t \,\%\, T_m = 0$ **then**

    update m according to magnitudes of current w

  end if

end for

Return: m.

the cost function to be squared loss[3] $\ell(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) = \frac{1}{2} \|f_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}\|^2$. A discussion of other loss functions, such as cross-entropy, is given in Section 3.6. Let $\tilde{\mathbf{X}}$ be the collection of adversarial examples as above, then

$$\nabla_{f_t} \mathcal{L}(\tilde{\mathbf{X}}_t) = f_t(\tilde{\mathbf{X}}_t) - \mathbf{Y} \tag{3.9}$$

And therefore, the dynamics of model outputs of dense network in Equation (3.6) becomes

$$\frac{\mathrm{d}f_t}{\mathrm{d}t}(\mathbf{X}) = -\frac{\eta}{N} \Theta_t(\mathbf{X}, \tilde{\mathbf{X}}_t)(f_t(\tilde{\mathbf{X}}_t) - Y) \tag{3.10}$$

To achieve our goal, note that the dense classifier $f_t(\mathbf{x})$ in Theorem 3.2.1 converges eventually to the solution of adversarial training, so it is supposed to be adversarial robust if fully-trained. On the other hand, the dynamics of the sparse adversarial training $f_t^s(\mathbf{x})$ can be described similarly as:

$$\frac{\mathrm{d}f_t^s}{\mathrm{d}t}(\mathbf{X}) = -\frac{\eta}{N} \Theta_t^s(\mathbf{X}, \tilde{\mathbf{X}}_t^s)(f_t^s(\tilde{\mathbf{X}}_t^s) - \mathbf{Y}) \tag{3.11}$$

[3]Norms without subscript will denote ℓ_2 norm.

where $\tilde{\mathbf{X}}_t^s$ is the collection of adversarial images corresponding to sparse network and $\Theta_t^s(\mathbf{X}, \tilde{\mathbf{X}}_t^s)$ is the MTK of sparse classifier. Therefore, to get the desired mask m, it is sufficient to make $f_t^s(\mathbf{x}) \approx f_t(\mathbf{x})$ for all t. According to Equation (3.10) and Equation (3.11), this can be achieved by making the MTK distance and adversarial target distance between dense and sparse networks close enough at any time t in the training. That is to say,

$$\Theta_t(\mathbf{X}, \tilde{\mathbf{X}}_t) \approx \Theta_t^s(\mathbf{X}, \tilde{\mathbf{X}}_t^s) \qquad f_t(\tilde{\mathbf{X}}_t) \approx f_t^s(\tilde{\mathbf{X}}_t^s) \tag{3.12}$$

for all t. Under mild assumptions, we may expect all these items are determined at $t = 0$ because of the continuous dependence of the solution of differential equations on the initial values. This then leads to the consideration of the following optimization problem:

$$\min_m \mathcal{L}_{awt} = \frac{1}{N} \left\| f_{\boldsymbol{\theta}_0}(\tilde{\mathbf{X}}_0) - f^s_{m\odot\boldsymbol{\theta}_0}(\tilde{\mathbf{X}}_0^s) \right\|^2 + \frac{\gamma^2}{N^2} \left\| \Theta_0(\mathbf{X}, \tilde{\mathbf{X}}_0) - \Theta_0^s(\mathbf{X}, \tilde{\mathbf{X}}_0^s) \right\|_F^2 \tag{3.13}$$

where $\|\cdot\|$ is the ℓ_2 norm of vectors and $\|\cdot\|_F$ is the Frobenius norm of matrices. In equation (3.13), the first and second items in the right hand side are referred as *target distance* and *kernel distance*, respectively. We call the resulting mask *Adversarial Winning Ticket* (AWT). Our method is summarized in algorithm 2. Since the binary mask m cannot be optimized directly, instead we train a student network $f_{m\odot\mathbf{w}}(\mathbf{x})$. The mask m is then updated according to the magnitudes of current weights w after several steps, which is specified by the mask update frequency. To get sparse adversarial robust network, the obtained AWT $f_{m\odot\boldsymbol{\theta}_0}(\mathbf{x})$ will be adversarially trained from scratch. This intuition can be further illustrated by Figure 3.1. For each iteration of gradient descent ($t = 1$ to $t = 2$ in the figure), adversarial training contains two steps: adversarial attack (vertical dash line) and parameter update (horizontal dash line). Our goal is to make the blue curve (sparse) close to the green one (dense). This can be done by making the attack and parameter update curves of sparse and dense networks close enough for each time t. However, as we can see from the figure, f_t and f_t^s are determined by the initial condition, hence we get the above optimization problem.

Formally we have the following theorem to estimate the error bound between dense and sparse outputs.

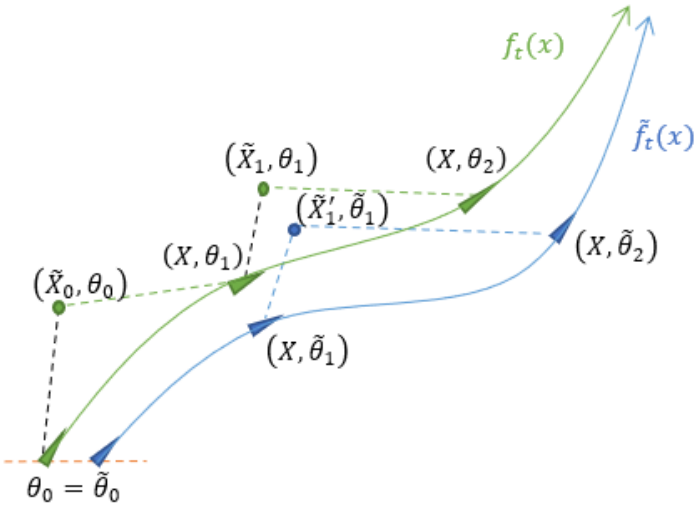


Figure 3.1: Schematic illustration of networks' outputs evolution under adversarial training.

Theorem 3.2.2. *Let $f_{\boldsymbol{\theta}}(\mathbf{x})$ denote the dense network function. Suppose $f_{\boldsymbol{\theta}}$ has identical number of neurons for each layer, i.e. $d_1 = d_2 = \cdots = d_L = n$ and assume n is large enough. Denote $f^s_{m\odot\boldsymbol{\theta}}(\mathbf{x})$ the corresponding sparse network with $1-\beta$ weights being pruned. Assume f and f^s have bounded first and second order derivatives with respect to x, i.e.*

$$\max_{t,x} \left\{ \|\partial_x f_t\|_q , \|\partial_x f_t^s\|_q \right\} \leq C_{1,q}$$
$$\max_{t,x} \left\{ \left\|\partial^2_{xx} f_t\right\|_{p,q} , \left\|\partial^2_{xx} f_t^s\right\|_{p,q} \right\} \leq C_{2,q}$$

where we choose an ℓ_p attack to generate adversarial examples such that q is the conjugate of p in the sense of $1/p + 1/q = 1$.[4] Denote the optimal loss value for AWT optimization problem (3.13) *to be $\mathcal{L}^*_{awt} = \alpha^2$. Then for all $t \leq T$ with T the stop time, with learning rate $\eta = O(T^{-1})$, we have*

$$\mathop{\mathbb{E}}_{x\in\mathcal{D}} \|f_t(\mathbf{x}) - f_t^s(\mathbf{x})\|^2 \leq 4(\alpha + 4C_q\varepsilon)^2 \tag{3.14}$$

where $C_q = C_{1,q} + \varepsilon C_{2,q}$ is a constant.

Note that we put no restriction on any specific attack algorithm, hence we can choose any strong attack algorithm for generating adversarial examples. In practice, PGD attack is commonly chosen

[4] If $p = \infty$, we take $q = 1$.

for adversarial training. Also, our theoretical results consider any ℓ_p attack with $1 \leq p \leq \infty$. The uniform bound assumption of derivatives are reasonable. If we take the Taylor expansion of f_t with respect to f_0, then the derivatives are functions of $\boldsymbol{\theta}_t$. Since we apply weight decay in our training, $\boldsymbol{\theta}_t$ is uniformly bounded for all $t \leq T$, also we only have finite training data, so the derivatives can be assumed to be uniformly bounded. Moreover, C_q can be adjusted by carefully choosing the regularizing constant of weight decay. Proof details and a discussion of the constants are presented in Section 3.5.

Equation (3.14) shows that the expected error between sparse and dense outputs are bounded by the optimal loss value and adversarial perturbation strength. In practice, the optimial loss value α^2 and adversarial perturbation strength ε are small, we may expect the output of AWT is close to dense output, hence is adversarial robust if fully-trained. Therfore Theorem 3.2.2 can be viewed as theoretical justification of the existence of LTH in adversarial setting, and we can find AWT by solving the optimization problem (3.13).

Theorem 3.2.2 reduces to natural training if we take $\varepsilon = 0$. In this case, the AWT found is winning ticket for natural training. Hence Theorem 3.2.2 also verifies the classical LTH as a special case. Meanwhile, our method reduces to *Neural Tangent Transfer* (NTT) in Liu and Zenke [54] and Equation (3.14) gives an error bound of NTT. Furthermore, for ideal case when $\alpha = 0$, Equation (3.14) implies $f_t(\mathbf{x}) = f_t^s(\mathbf{x})$ for all x, hence the dense and sparse networks have identical outputs for all time t, which extends Proposition 1 in Liu and Zenke [54].

3.2.3 Toy Example

We provide here a toy example to illustrate our method. Consider a binary classification problem of mixed Gaussian distribution $p(\mathbf{x}) = 0.5\mathcal{N}(\boldsymbol{\mu}_+, \Sigma_+) + 0.5\mathcal{N}(\boldsymbol{\mu}_-, \sigma_-)$ and any linear classifier $C(\mathbf{x}) = \operatorname{sgn}(f_{\boldsymbol{\theta}}(\mathbf{x})) = \operatorname{sgn}(\boldsymbol{\theta} \cdot \mathbf{x})$. Let $\boldsymbol{\mu} = 0.5(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$ be the mean difference and suppose $\Sigma_+ = \Sigma_- = \sigma^2 \boldsymbol{I}_d$, then the adversarial accuracy with given adversarial perturbation strength ε can

be calculated explicitly using the following result in [82]:

$$\begin{aligned} p_{adv-acc} = 1 - p_{adv} &= p_m + \Phi\left(\frac{\boldsymbol{\theta}\cdot\boldsymbol{\mu}}{\|\boldsymbol{\theta}\|\,\sigma} - \frac{\varepsilon}{\sigma}\right) \\ &= p_m + \Phi\left(\frac{\|\boldsymbol{\mu}\|}{\sigma}\cos\gamma - \frac{\varepsilon}{\sigma}\right) \end{aligned} \tag{3.15}$$

where p_m is the misclassification rate, p_{adv} is the probability of the existence of adversarial examples, and Φ is the cumulative density function of standard normal distribution. This shows the adversarial robustness of $C(\mathbf{x})$ can be measured explicitly by the deflection angle γ between $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$. In particular, $p_{adv-acc}$ attains its maximum at $\boldsymbol{\theta} = \boldsymbol{\mu}$, i.e. Bayes classifier is the best classifier in the sense of adversarial robustness.

To illustrate our idea, we randomly generate 5000 data points from $p(\mathbf{x})$, where $\boldsymbol{\mu}_+ = [3, 0, \cdots, 0] = -\boldsymbol{\mu}_-$, $\Sigma_+ = \Sigma_- = \boldsymbol{I}_d$ and the dimension is $d = 100$. We minimize the loss in equation (3.20) to obtain a sparse robust structure with a sparsity level at 10%, i.e. we only keep 10 nonzero coordinates of $\boldsymbol{\theta}$. Then we adversarially train the AWT to get the sparse robust network. We use SVM as baseline of model accuracy and standard adversarial training as baseline of adversarial accuracy. The result is summarized in table 3.1.

	Bayes	SVM	Adv.Tr.	AWT
Accuracy	0.999	0.995	0.995	0.995
Angle	0.0	0.555	0.202	0.118
Cosine	1.0	0.850	0.980	0.993
Robustness	0.843	0.714	0.823	0.838

Table 3.1: Performance comparison on synthetic data.

Table 3.1 shows that SVM reaches comparable model accuracy as Bayes model but with a large deflection angle ($0.555 \approx 31.8°$), this simulates the case that natural training can reach high accuracy but fails to be adversarial robust. Adversarial training (Adv.Tr.) reaches the same model accuracy but also improved adversarial accuracy (0.823), which is very close to the one of Bayes classifier. This is also reflected by the deflection angle ($0.202 \approx 11.6°$), which shows a significant

decrease when compared with the deflected angle of SVM. Our method (AWT) gets slightly better performance than the dense adversarial training with only 10% of the weights left.

3.3 Related Works

3.3.1 Adversarial Robust Learning

The study of adversarial examples naturally splits into two areas: attack and defense. Adversarial attack methods aim to fool state-of-the-art networks. In general, attack methods consist of white-box attack and black-box attack, depending on how much information about the model we can have. White-box attacks are widely used in generating adversarial examples for training or testing model robustness where we can have all information about the model. This includes *Fast Gradient Sign Method* (FGSM) [29], *Deep Fool* [62], *AutoAttack* [17] and so on. Black-box attacks [13, 58] are usually developed to attack model in physical world, therefore we have very limited information about the model structure or parameters.

Meanwhile, defense methods have been studied to train an adversarial robust network which can prevent attacks from adversarial examples. Augmentation with adversarial examples generated by strong attack algorithms has been popular in the literature. Madry et al. [57] motivates *Projected Gradient Descent* (PGD) as a universal 'first order adversary' and solve a min-max problem by iteratively generating adversarial examples and parameter updating on adversarial examples. Such method is referred as *adversarial training* (AT). The convergence and performance of AT have been theoretically justified by recent works [91, 103, 26]. Also, methods [81, 97] have been developed to speed up the training of AT for large scale datasets such as ImageNet.

3.3.2 Sparse Learning

Pruning Methods Network pruning [35, 33, 101, 53, 56, 38, 105] has been extensively studied in recent years for reducing model size and improve the inference efficiency of deep neural networks.

Since it is a widely-recognized property that modern neural networks are always over-parameterized, pruning methods are developed to remove unimportant parameters in the fully trained dense networks to alleviate such redundancy. According to the granularity of pruning, existing pruning methods can be roughly divided into two categories, i.e., unstructured pruning and structured pruning. The former one is also called weight pruning, which removes the unimportant parameters in an unstructured way, that is, any element in the weight tensor could be removed. The latter one removes all the weights in a certain group together, such as kernel and filter. Since structure is taken into account in pruning, the pruned networks obtained by structured pruning are available for efficient inference on standard computation devices. In both structured and unstructured pruning methods, their key idea is to propose a proper criterion (e.g., magnitude of the weight) to evaluate the importance of the weight, kernel or filter and then remove the unimportant ones. he results in the literature [33, 55, 101, 53] demonstrate that pruning methods can significantly improve the inference efficiency of DNNs with minimal performance degradation, making the deployment of modern neural networks on resource limited devices possible.

Lottery Ticket Hypothesis Frankle and Carbin [23] proposed *Lottery Ticket Hypothesis* (LTH), which states that there exists subnetwork structure, which can achieve comparable performance as the dense model when trained in isolation. The resulting subnetwork structure is called *winning ticket*. Commonly, winning ticket is found by iterative training and pruning [24, 73].

Along the research line of LTH, recent works, e.g., SNIP [51] and GraSP [93], empirically show that it is possible to find a winning ticket at intialization step, without iteratively training and pruning procedure as the classical pruning methods. The key idea is to find a sub-network, which preserves the gradient flow at initialization. NTT [54] utilizes the NTK theory and prune the weights by preserving the training dynamics of model outputs, which is captured by a system of differential equations.

Adversarial Pruning Methods Recent works by Guo et al. [34] have proven sparsity can improve adversarial robustness. A typical way of verifying the *Lottery Ticket Hypothesis* (LTH) is finding the winning ticket by iteratively training and pruning. Such strategy is also considered in adversarial context [16, 94, 52, 28], with natural training replaced by adversarial training. Other score based pruning methods have also been considered [80]. Recent work [25] also considered sub-network structure with inborn robustness without training.

Other works bring in the model compression methods into sparse adversarial training. Gui et al. [31] integrates pruning, low-rank factorization and quantization into a unified flexible structural constraint. Ye et al. [100] proposes concurrent weight pruning to reach robustness. Both works introduce certain sparse constraints and solve the optimization problem under *alternating direction method of multipliers* (ADMM) framework.

3.4 Experiments

We now empirically verify the performance of our method. To be precise, we first show the effectiveness of our method in preserving the dynamics of adversarial training, that is, our method can find a sparse sub-network, whose training dynamics are close to the dense network. Then we evaluate the robustness of the sparse neural networks obtained by our method. At last, we give a preliminary experimental result to show the possibility to extend our method to large-scaled problems.

3.4.1 Implementation

We conduct experiments on standard datasets, including MNIST [49] and CIFAR-10 [45]. All experiments are performed in JAX [11], together with the neural-tangent library [65]. Due to the high computational and storage costs of NTK, following the experimental setting in [54], we mainly evaluate our proposed method on two networks: MLP and 6-layer CNN. The preliminary experiment of scalability in Section 3.4.4 is conducted on VGG-16 with CIFAR-10.

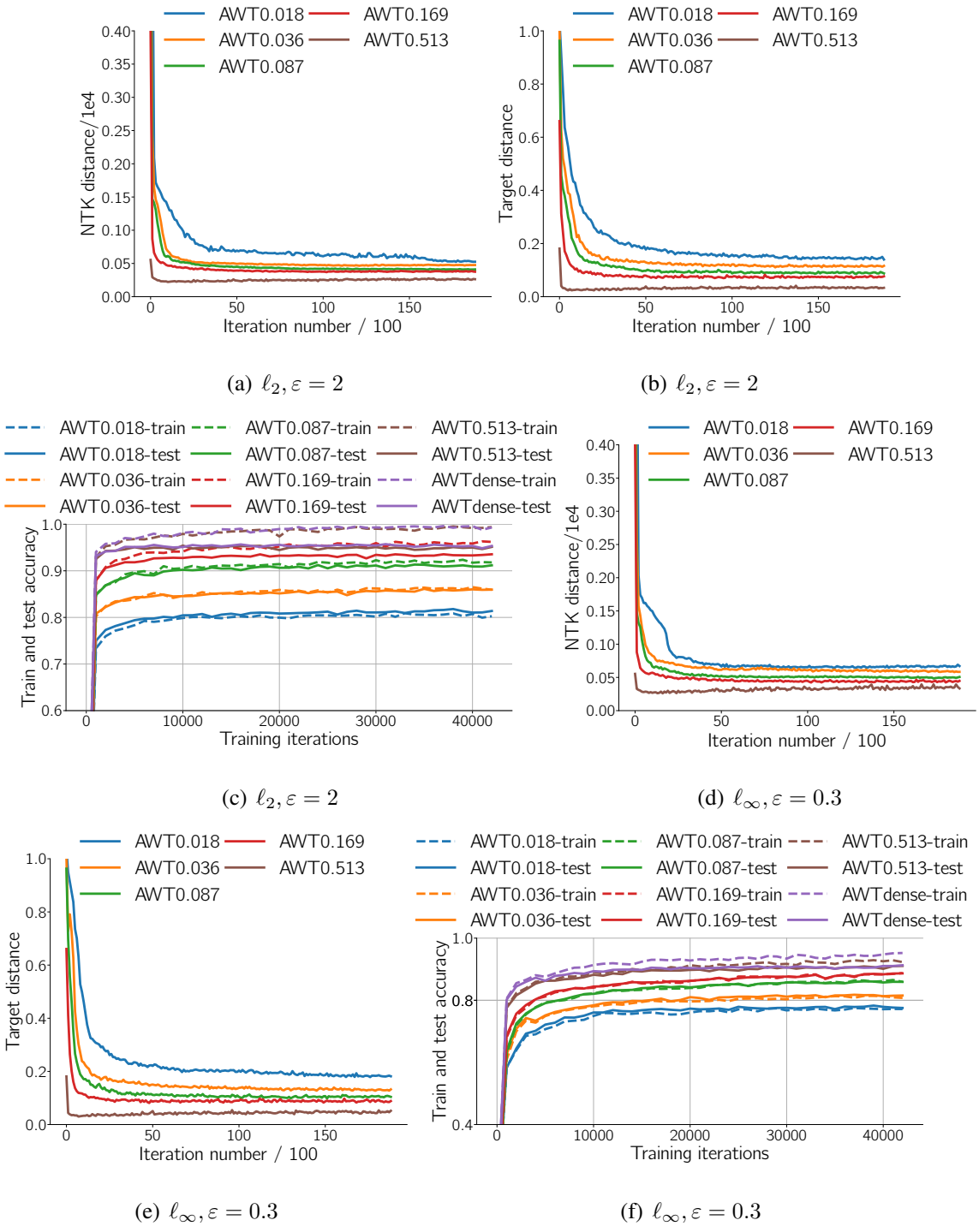
We use PGD attacks for adversarial training and robustness evaluation as suggested in Guo et al. [32] and Wang et al. [94]. In practice, ℓ_∞ attacks are commonly used and we use adversarial strength $\varepsilon = 0.3$ for MNIST and $\varepsilon = 8/255$ for CIFAR-10. We take 100 iterations for robustness evaluation, the step size is taken to be $2.5 \cdot \varepsilon/100$ as suggested by Madry et al. [57]. Other detailed experimental configurations such as the learning rate and batch size can be found in the supplementary materials.

3.4.2 Effectiveness in Preservation of Training Dynamics

In this part, we evaluate the ability of our method in preserving the training dynamics. To be precise, at each density level, we first present the evolution curves of kernel distance and target distance over the whole procedure of finding the adversarial winning ticket. Then we show the adversarial training/testing accuracy during the training process. Since Theorem 3.2.2 is valid for any ℓ_p attack algorithms, we also present experimental results under ℓ_2 attacks as well as ℓ_∞ attacks.

Figure 3.2 (a)/(d) and (b)/(e) show the kernel and target distance curves at different density levels under ℓ_2/ℓ_∞ attack. We can see that as the optimization goes on, both of the kernel and target distances decrease very quickly. As expected, the distance becomes smaller as the density level increases. Figures 3.2 (c)/(f) show the adversarial training/testing accuracy. We can see that when the density becomes larger, the accuracy curve gets closer to the dense one. This indicates that the training dynamics are well preserved.

To verify the quality of our winning ticket, we compare our method with the latest work by Cosentino et al. [16], which finds the winning ticket by iteratively pruning and adversarial training. As indicated by the authors [16], their method is computationally expensive so they only evaluated it on small MLPs. Therefore we only give the comparison result on MNIST with MLPs here. Specifically, we give the test accuracy on natural/adversarial examples in Table 3.2. It shows that our method can outperform the baseline with a large margin. For example, at the density of 1.8%, the adversarial test accuracy of our method is 40% higher than that of Cosentino et al. 16. We can also see that the accuracy of our method can converge to the dense model much more quickly than



(a) $\ell_2, \varepsilon = 2$

(b) $\ell_2, \varepsilon = 2$

(c) $\ell_2, \varepsilon = 2$

(d) $\ell_\infty, \varepsilon = 0.3$

(e) $\ell_\infty, \varepsilon = 0.3$

(f) $\ell_\infty, \varepsilon = 0.3$

Figure 3.2: Statistics of AWT on MNIST with MLP.

the baseline as the density increases. This is benefited from the dynamics preserving property of our sparse sub-network structure.

density	Cosentino et al. 16	AWT
full model	98.96/91.14	
51.3%	98.07/60.14	99.13/91.21
16.9%	97.73/59.91	96.58/89.30
8.7%	97.20/57.60	94.48/87.51
3.6%	95.58/48.81	91.74/83.60
1.8%	92.67/38.23	87.69/78.66

Table 3.2: Test accuracy on natural/adversarial examples over different density levels on MNIST with MLP.

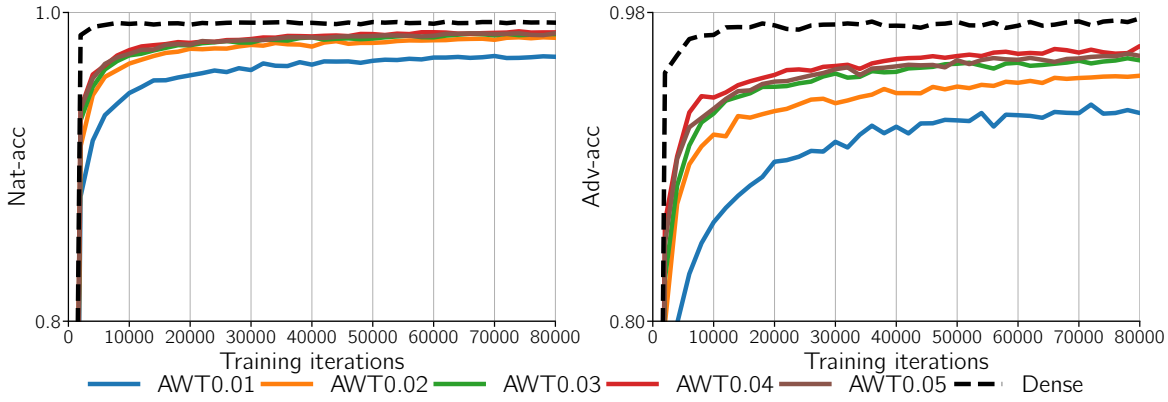


Figure 3.3: Test accuracy on natural and adversarial examples of CNN trained on MNIST.

3.4.3 Robustness of Trained Sparse Networks

In this part, we evaluate the robustness of fully adversarially trained AWT at different density levels. We first present the test accuracy on natural and adversarial examples of the CNN models trained on MNIST and CIFAR10. For CIFAR10, the density varies in $\{0.05, 0.1, 0.2, 0.3, \ldots, 0.9\}$. For MNIST, since it can be classified with much sparser networks compared with CIFAR10, in this section, we only check densities between $\{0.01, \ldots, 0.05\}$ and the results under higher density levels can be found in the Section 3.7. Figure 3.3 and 3.4 give the results on MNIST and CIFAR10,

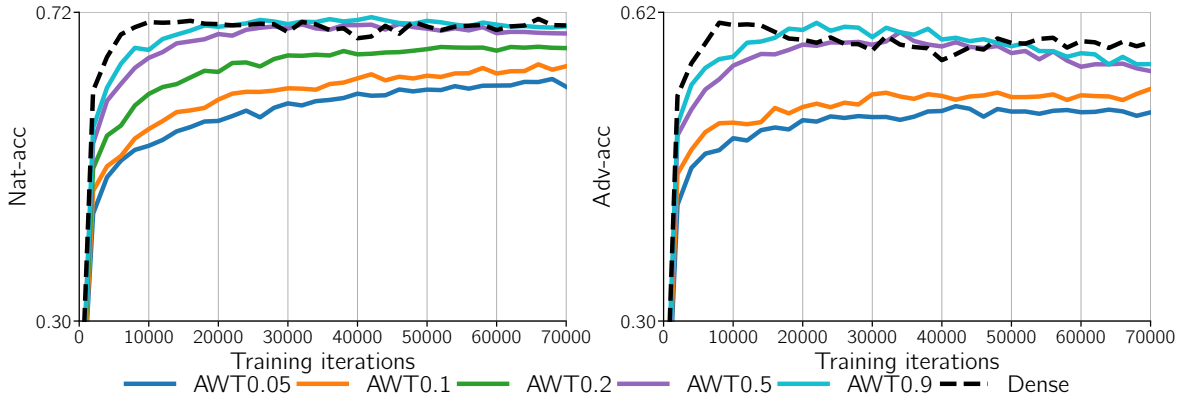


Figure 3.4: Test accuracy on natural and adversarial examples of CNN trained on CIFAR10.

respectively. Both of these two Figures show that the models trained by our method have high natural and adversarial test accuracy even when the model is very sparse. For example, Figure 3.3 shows that at the density of 0.03, the model trained by our method can reach the test accuracy of 0.98 and 0.96 on natural and adversarial examples, which are quite close to the dense model. We can also see that the training dynamic, i.e., the test accuracy curves, can converge to that of the dense model as the density increases. And the sparse models obtained by our method can achieve comparable test accuracy with the dense model after trained with the same number of epochs.

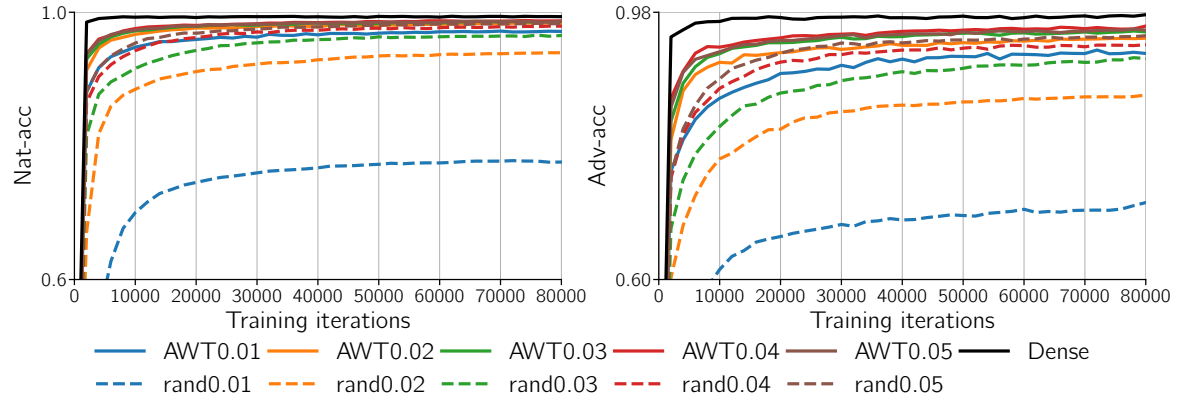


Figure 3.5: Natural and adversarial test accuracy of the models trained from AWT and random structure on MNIST with CNN.

We then compare the performance of models trained from our sparse structure and the random structure at the same density level. We give the results of CNN trained on MNIST with the density

varies in $\{0.01, 0.02, \ldots, 0.05\}$ in Figure 3.5. More results can be found in Section 3.7. From Figure 3.5, our method can achieve much higher natural and adversarial test accuracy than the models trained from the random structure. For example, the network trained by our method at the density of 1% achieves higher adversarial test accuracy than the model trained from random structure at the density of 3%. It also shows that the learning curves of our method are much closer to the dense model than the random structure. This demonstrates that our sparse structures indeed have similar training dynamic with the dense model.

3.4.4 Discussion on Scalability

In the same situation as existing works on NTK, the expensive computational cost of NTK hinders us from conducting large-scale experiments. Fortunately, the following preliminary experiment shows that the methods such as sampling on the NTK matrix could be promising to improve the efficiency of our method.

density	WT	S-AWT	IWI
10%	45.10	46.68	47.53
5%	40.15	45.87	47.59

Table 3.3: Adversarial test accuracy of the VGG16 trained on CIFAR10 at different density levels.

To be specific, inspired from Jacobi preconditioner method [14] in optimization theory, we sample only the diagonal elements in the MTK matrices $\Theta_0(\mathbf{X}, \tilde{\mathbf{X}}_0)$ and $\Theta_0^s(\mathbf{X}, \tilde{\mathbf{X}}_0^s)$ in Equation (3.13) and keep all other settings the same as above. In this way, the computational complexity of training can be significantly reduced. We conduct a preliminary experiment on CIFAR-10 with large-sized model VGG-16. To the best of our knowledge, we are the first to apply NTK into models as large as VGG-16. The result is presented in Table 3.3, where WT represents winning ticket, which applies iteratively pruning and adversarial training method. S-AWT is AWT with sampling. IWI represents Inverse Weight Inheritance. WT and IWI results are copied from Wang et al. [94]. It shows that the

performance degradation caused by the sampling is unnoticeable. We will investigate this kind of approaches to improve the efficiency of NTK based methods in the future.

3.5 Proof of Theorems

Let $\mathcal{D} = \mathbf{X} \times \mathbf{Y} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$ be the empirical data distribution, $f_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^k$ the network function, and $f_{\boldsymbol{\theta}}(\mathbf{X}) = \mathrm{vec}\big([f_{\boldsymbol{\theta}}(\mathbf{x})]_{\mathbf{x}\in\mathbf{X}}\big) \in \mathbb{R}^{k|\mathcal{D}|\times 1}$ be the model outputs as before. Recall that adversarial training optimizes the following objective function

$$\min_{\boldsymbol{\theta}} \mathcal{L} = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}} \max_{\mathbf{r}\in S_\varepsilon(\mathbf{x})} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}+\mathbf{r}), \mathbf{y}) \tag{3.16}$$

Note that for squared loss $\ell(f(\mathbf{x}), \mathbf{y}) = \dfrac{1}{2}\left\|f(\mathbf{x}) - \mathbf{y}\right\|_2^2$, we have the following notation for convenience:

$$\nabla_f \mathcal{L}(\mathbf{X}) = \begin{bmatrix} | \\ \nabla_f \ell(f(\mathbf{x}_i), \mathbf{y}) \\ | \end{bmatrix} = \begin{bmatrix} | \\ f(\mathbf{x}_i) - \mathbf{y}_i \\ | \end{bmatrix} = f(\mathbf{X}) - Y$$

Theorem 3.5.1. *Let $f_t(\mathbf{x}) := f_{\boldsymbol{\theta}_t}(\mathbf{x})$ be the timely dependent network function and $\tilde{\mathbf{X}}_t$ the adversarial examples generated at time t. Then the continuous gradient descent of adversarial training is:*

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = -\frac{\eta}{N}\nabla_{\boldsymbol{\theta}}^T f_t(\tilde{\mathbf{X}}_t)\nabla_f \mathcal{L}(\tilde{\mathbf{X}}_t) \tag{3.17}$$

As a result, f_t satisfies the following differential equation:

$$\begin{aligned} \frac{\mathrm{d}f_t}{\mathrm{d}t}(\mathbf{X}) &= \nabla_{\boldsymbol{\theta}} f_t(\mathbf{X}) \frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} \\ &= -\frac{\eta}{N}\nabla_{\boldsymbol{\theta}} f_t(\mathbf{X})\nabla_{\boldsymbol{\theta}}^T f_t(\tilde{\mathbf{X}}_t)\nabla_f \mathcal{L}(\tilde{\mathbf{X}}_t) \end{aligned} \tag{3.18}$$

Proof. Note that at time t adversarial training consists of an attack step and parameter update step. To be precise, for some chosen strong attack algorithm, we first generate the set of adversarial examples $\tilde{\mathbf{X}}_t$, then update the parameter according to

$$\begin{aligned} \boldsymbol{\theta}_{t+dt} &= \boldsymbol{\theta}_t - \eta_t \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_t}(\tilde{\mathbf{X}}_t) \\ &= \boldsymbol{\theta}_t - \frac{\eta_t}{N}\nabla_{\boldsymbol{\theta}}^T f_t(\tilde{\mathbf{X}}_t)\nabla_f \mathcal{L}(\tilde{\mathbf{X}}_t) \end{aligned} \tag{3.19}$$

If we take the infinitesimal learning rate to be[5] $\eta_t = \eta dt$ and taking the limit $dt \to 0$, we obtain the continuous gradient descent as in Equation (3.17). The evolution of f_t in Equation (3.18) is a direct result by chain rule. □

We consider the following optimization problem to find AWT:

$$\min_m \mathcal{L}_{awt} = \frac{1}{N} \left\| f_0(\tilde{\mathbf{X}}_0) - f_0^s(\tilde{\mathbf{X}}_0^s) \right\|^2 + \frac{\gamma^2}{N^2} \left\| \Theta_0(\mathbf{X}, \tilde{\mathbf{X}}_0) - \Theta_0^s(\mathbf{X}, \tilde{\mathbf{X}}_0^s) \right\|_F^2 \tag{3.20}$$

where $\Theta(\mathbf{x}, \mathbf{x}') = \langle \nabla_{\boldsymbol{\theta}} f(\mathbf{x}), \nabla_{\boldsymbol{\theta}} f(\mathbf{x}') \rangle$ is the empirical neural tangent kernel and sup-script s represents quantities involving sparse structure.

Theorem 3.5.2 (Existence of adversarial winning ticket). *Let $f_{\boldsymbol{\theta}}(\mathbf{x})$ denote the dense network function. Suppose $f_{\boldsymbol{\theta}}$ has identical number of neurons for each layer, i.e. $d_1 = d_2 = \cdots = d_L = n$ and assume n is large enough. Denote $f^s_{m \odot \boldsymbol{\theta}}(\mathbf{x})$ the corresponding sparse network with $1 - \beta$ weights being pruned. Assume f and f^s have bounded first and second order derivatives with respect to $\mathbf{x}$, i.e.*

$$\max_{t,\mathbf{x}} \left\{ \|\partial_{\mathbf{x}} f_t\|_q, \|\partial_{\mathbf{x}} f_t^s\|_q \right\} \le C_{1,q}$$
$$\max_{t,\mathbf{x}} \left\{ \left\|\partial^2_{\mathbf{xx}} f_t\right\|_{p,q}, \left\|\partial^2_{\mathbf{xx}} f_t^s\right\|_{p,q} \right\} \le C_{2,q}$$

where we choose an ℓ_p attack to generate adversarial examples such that q is the conjugate of p in the sense of $1/p + 1/q = 1$.[6] Denote the optimal loss value for AWT optimization problem (3.13) *to be $\mathcal{L}^*_{awt} = \alpha^2$. Then for all $t \le T$ with T the stop time, with learning rate $\eta = O(T^{-1})$, we have*

$$\mathop{\mathbb{E}}_{\mathbf{x} \in \mathcal{D}} \|f_t(\mathbf{x}) - f_t^s(\mathbf{x})\|^2 \le 4(\alpha + 4C_q \varepsilon)^2 \tag{3.21}$$

where $C_q = C_{1,q} + \varepsilon C_{2,q}$ is a constant.

In order to prove the theorem, we need the following lemma of estimation of error bound.

[5] The discrete parameter update corresponds to the case when $dt = 1$.

[6] If $p = \infty$, we take $q = 1$.

Lemma 3.5.3. *For any $1 < p \leq \infty$, assume an k iterative ℓ_p attack algorithm updates as $\tilde{\mathbf{x}}_0 = \mathbf{x},\ \tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_{t-1} + \mathbf{r}_t$ with $\|\mathbf{r}_t\|_p \leq \delta$ for any $1 \leq t \leq k$ and with total allowed perturbation strength $\|\sum \mathbf{r}_j\|_p \leq \varepsilon$. Assume $k\delta \leq 2\varepsilon$. If the neural network function f has bounded first and second order derivative with respect to $\mathbf{x}$, i.e. $\|\partial_{\mathbf{x}} f\|_q \leq C_{1,q}, \|\partial^2_{\mathbf{xx}} f\|_{p,q} \leq C_{2,q}$, where q be the conjugate of p such that $1/p + 1/q = 1$. Then for any adversarial example $\tilde{\mathbf{x}}$ generated by the attack algorithm, we have*

$$|f(\tilde{\mathbf{x}}) - f(\mathbf{x})| \leq 2\varepsilon C_{1,q} + 2\varepsilon^2 C_{2,q} = 2\varepsilon C_q \tag{3.22}$$

Proof of Lemma: Consider the series of second order Taylor expansions for any $1 \leq t \leq k$

$$f(\tilde{\mathbf{x}}_t) - f(\tilde{\mathbf{x}}_{t-1}) = \partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1})\mathbf{r}_t + \frac{1}{2}\mathbf{r}_t^T \partial^2_{\mathbf{xx}} f(\xi_{t-1})\mathbf{r}_t \tag{3.23}$$

We have

$$\begin{aligned} |f(\tilde{\mathbf{x}}_t) - f(\tilde{\mathbf{x}}_{t-1})| \leq & \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1})\mathbf{r}_t\| + \left\| \frac{1}{2}\mathbf{r}_t^T \partial^2_{\mathbf{xx}} f(\xi_{t-1})\mathbf{r}_t \right\| \\ \leq & \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1})\|_q \|\mathbf{r}_t\|_p + \frac{1}{2}\|\mathbf{r}_t\|_p \left\|\partial^2_{\mathbf{xx}} f(\xi_{t-1})\mathbf{r}_t\right\|_q \\ \leq & \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1})\|_q \delta + \frac{1}{2}\delta \left\|\partial^2_{\mathbf{xx}} f(\xi_{t-1})\right\|_{p,q} \|\mathbf{r}_t\|_p \\ \leq & C_{1,q}\delta + \frac{1}{2}C_{2,q}\delta^2 \end{aligned} \tag{3.24}$$

where we use Hölder inequality in the second step and definition of (p, q) norm in the third step. On the other hand, by Mean-value theorem we have

$$\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_t) = \partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1}) + \partial^2_{\mathbf{xx}} f(\eta_{t-1})\mathbf{r}_t \tag{3.25}$$

Hence we have

$$\begin{aligned} \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_t)\|_q \leq & \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1})\|_q + \left\|\partial^2_{\mathbf{xx}} f(\eta_{t-1})\mathbf{r}_t\right\|_q \\ \leq & \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1})\|_q + \left\|\partial^2_{\mathbf{xx}} f(\eta_{t-1})\right\|_{p,q} \|\mathbf{r}_t\|_p \\ \leq & C_{1,q} + C_{2,q}\delta \end{aligned} \tag{3.26}$$

where we use Minkowski inequality in the first step. Together, we have the following estimation:

$$\begin{aligned}
|f(\tilde{\mathbf{x}}) - f(\mathbf{x})| =& |f(\tilde{\mathbf{x}}_k) - f(\tilde{\mathbf{x}}_0)| \\
\leq& \sum_{t=1}^{k} |f(\tilde{\mathbf{x}}_t) - f(\tilde{\mathbf{x}}_{t-1})| \\
\leq& \delta \sum_{t=1}^{k} \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_{t-1})\|_q + \frac{1}{2}\delta^2 \sum_{t=1}^{k} \left\|\partial^2_{\mathbf{xx}} f(\xi_{t-1})\right\|_{p,q} \\
=& \delta \left(\sum_{t=1}^{k} \|\partial_{\mathbf{x}} f(\tilde{\mathbf{x}}_0)\|_q + C_{2,q}\delta(t-1) \right) + \frac{1}{2}\delta^2 k C_{2,q} \\
=& k\delta C_{1,q} + \frac{1}{2}k^2\delta^2 C_{2,q}
\end{aligned} \tag{3.27}$$

Then Equation (3.22) is valid if we plug in the assumption $k\delta \leq 2\varepsilon$. □

Remark 1: We did not specify any particular algorithm in the presentation of our lemma. In practice, k steps PGD attacks is the common choice for inner subproblem of adversarial training. For k steps PGD attack, the number of attack iteration is usually taken to be 7 (ImageNet) or 20 (MNIST/CIFAR-10), so we may assume $k = \Omega(1)$ for future use. The assumption $k\delta \leq 2\varepsilon$ is also for practical consideration, where we usually choose the step size δ approximately to be $2\varepsilon/k$ as suggested in [57].

Remark 2: We might get more accurate bound by looking deeper into the dynamics of continuous first-order attack:

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{x}}(f_t(\mathbf{x}_t)), \qquad \mathbf{x}_0 = \mathbf{x} \tag{3.28}$$

The analysis of the above differential equations would possibly weaken the current assumption on derivatives. We would leave this as a future work.

Now we are ready to prove the main theorem.

Proof of Theorem. Suppose $\mathcal{L}^*_{awt} = \alpha^2$, then we have

$$\left\| f_0(\tilde{\mathbf{X}}_0) - f_0^s(\tilde{\mathbf{X}}_0^s) \right\| \leq \sqrt{N}\alpha, \qquad \left\| \Theta_0(\mathbf{X}, \tilde{\mathbf{X}}_0) - \Theta_0^s(\mathbf{X}, \tilde{\mathbf{X}}_0^s) \right\|_F \leq N\frac{\alpha}{\gamma}$$

To bound the distance between dense and sparse output, we do induction on time t and prove the following stronger estimation

$$\|f_t(\mathbf{X}) - f_t^s(\mathbf{X})\| \leq \left(1 + \frac{t}{T}\right)\sqrt{N}\big(\alpha + 4C_q\varepsilon\big) \tag{3.29}$$

where $C_q = C_{1,q} + \varepsilon C_{2,q}$ and $C_{1,q}$ and $C_{2,q}$ are bounds of first and second order derivative of f. Note that at $t = 0$, we have

$$\begin{aligned}\|f_0(\mathbf{X}) - f_0^s(\mathbf{X})\| \leq& \left\|f_0(\tilde{\mathbf{X}}_0) - f_0^s(\tilde{\mathbf{X}}_0^s)\right\| + \left\|f_0(\tilde{\mathbf{X}}_0) - f_0(\mathbf{X})\right\| + \left\|f_0^s(\tilde{\mathbf{X}}_0^s) - f_0^s(\mathbf{X})\right\| \\ \leq& \sqrt{N}\big(\alpha + 4C_q\varepsilon\big)\end{aligned} \tag{3.30}$$

At time t, assume we have

$$\|f_t(\mathbf{X}) - f_t^s(\mathbf{X})\| \leq \left(1 + \frac{t}{T}\right)\sqrt{N}\big(\alpha + 4C_q\varepsilon\big) \tag{3.31}$$

Then according to the dynamical equation (3.18), we have

$$\begin{aligned}f_{t+1}(\mathbf{X}) &= f_t(\mathbf{X}) - \frac{\eta}{N}\Theta(\mathbf{X}, \tilde{\mathbf{X}}_t)\big(f_t(\tilde{\mathbf{X}}_t) - \mathbf{Y}\big) \\ f_{t+1}^s(\mathbf{X}) &= f_t^s(\mathbf{X}) - \frac{\eta}{N}\Theta^s(\mathbf{X}, \tilde{\mathbf{X}}_t^s)\big(f_t^s(\tilde{\mathbf{X}}_t^s) - \mathbf{Y}\big)\end{aligned} \tag{3.32}$$

Then

$$\begin{aligned}&\left\|f_{t+1}(\mathbf{X}) - f_{t+1}^s(\mathbf{X})\right\| \\ \leq& \|f_t(\mathbf{X}) - f_t^s(\mathbf{X})\| + \frac{\eta}{N}\left\|(\Theta_t - \Theta_t^s)(f_t(\tilde{\mathbf{X}}_t) - \mathbf{Y})\right\| + \frac{\eta}{N}\left\|\Theta_t^s(f_t(\tilde{\mathbf{X}}_t) - f_t^s(\tilde{\mathbf{X}}_t^s))\right\|\end{aligned} \tag{3.33}$$

Let $K_t(\mathbf{X}, \mathbf{X}) = \nabla_{\boldsymbol{\theta}} f_t(\mathbf{X})\nabla_{\boldsymbol{\theta}}^T f_t(\mathbf{X})$ be the empirical neural tangent kernel at time t, then according to Theorem 2.1 in Lee et al. [50], $\|K_t - K_0\|_F = \frac{C_K}{\sqrt{n}}$, where n is the number of neurons in each layer. Therefore we have

$$\begin{aligned}&\left\|\Theta_t(\mathbf{X}, \tilde{\mathbf{X}}_t) - \Theta_t^s(\mathbf{X}, \tilde{\mathbf{X}}_t^s)\right\|_F \\ \leq& \left\|\Theta_t(\mathbf{X}, \tilde{\mathbf{X}}_t) - \Theta_0(\mathbf{X}, \tilde{\mathbf{X}}_0)\right\|_F + \left\|\Theta_t^s(\mathbf{X}, \tilde{\mathbf{X}}_t^s) - \Theta_0^s(\mathbf{X}, \tilde{\mathbf{X}}_0)\right\|_F + \left\|\Theta_0(\mathbf{X}, \tilde{\mathbf{X}}_0) - \Theta_0^s(\mathbf{X}, \tilde{\mathbf{X}}_0^s)\right\|_F \\ \leq& \|K_t(\mathbf{X}, \mathbf{X}) - K_0(\mathbf{X}, \mathbf{X})\|_F + \|K_t^s(\mathbf{X}, \mathbf{X}) - K_0^s(\mathbf{X}, \mathbf{X})\|_F + \left\|\nabla_{\boldsymbol{\theta}} f_t(\mathbf{X})\nabla_{\boldsymbol{\theta},\mathbf{x}}^2 f(\xi)R_t\right\| \\ &+ \left\|\nabla_{\boldsymbol{\theta}} f_t^s(\mathbf{X})\nabla_{\boldsymbol{\theta},\mathbf{x}}^2 f(\tilde{\xi})R_t^s\right\|_F + N\frac{\alpha}{\gamma} \\ \leq& \frac{C_K}{\sqrt{n}}\left(1 + \frac{1}{\sqrt{\beta}}\right) + 2N\varepsilon C_{1,q}C_{2,q} + N\frac{\alpha}{\gamma}\end{aligned} \tag{3.34}$$

And by Lemma (3.5.3), we have

$$\begin{aligned}\left\|f(\tilde{\mathbf{X}}_t)-f^s(\tilde{\mathbf{X}}_t^s)\right\| &\le \|f_t(\mathbf{X})-f_t^s(\mathbf{X})\|+\left\|f(\tilde{\mathbf{X}}_t)-f_t(\mathbf{X})\right\|+\left\|f_t^s(\tilde{\mathbf{X}}_t)-f_t^s(\mathbf{X})\right\| \\ &\le \left(1+\frac{t}{T}\right)\sqrt{N}\big(\alpha+4C_q\varepsilon\big)+4\sqrt{N}C_q\varepsilon\end{aligned} \tag{3.35}$$

Then Equation (3.33) reads

$$\begin{aligned}&\left\|f_{t+1}(\mathbf{X})-f_{t+1}^s(\mathbf{X})\right\| \\ \le&\left(1+\frac{t}{T}\right)\sqrt{N}\big(\alpha+4C_q\varepsilon\big)+\frac{\eta}{N}\left(\frac{C_K}{\sqrt{n}}\left(1+\frac{1}{\sqrt{\beta}}\right)+2N\varepsilon C_{1,q}C_{2,q}+N\frac{\alpha}{\gamma}\right)\sqrt{N}c \\ &+\frac{\eta}{N}NC_{2,q}\left[\left(1+\frac{t}{T}\right)\sqrt{N}\big(\alpha+4C_q\varepsilon\sqrt{N}\big)+4\sqrt{N}C_q\varepsilon\right]\end{aligned} \tag{3.36}$$

where $c=\max\{|f(\mathbf{x})-y|:\mathbf{x}\in\mathbf{X}\}$ is bounded essentially. Take

$$\eta=\min\left\{\frac{1}{T(2+\frac{c}{\gamma})},\quad \frac{4C_q\varepsilon}{T\big(2cC_{1,q}C_{2,q}+8C_q\big)}\right\} \tag{3.37}$$

One can check that if N is sufficiently large such that $\frac{C_K}{\sqrt{Nn}(1+\frac{1}{\sqrt{\beta}})}\to 0$, then Equation (3.36) reads

$$\left\|f_{t+1}(\mathbf{X})-f_{t+1}^s(\mathbf{X})\right\|\le\left(1+\frac{t+1}{T}\right)\sqrt{N}\big(\alpha+4C_q\varepsilon\big) \tag{3.38}$$

which completes the proof. □

3.6 Extensions to Other Loss Functions

We only consider the squared loss for simplicity in our main theorem. However, it is possible to extend our result to the cross-entropy loss case. Actually, we use cross-entropy loss in our experiment, so we have already checked our method empirically.

To see how our method works theoretically for cross-entropy loss, let

$$\ell_{ce}(f(\mathbf{x}),y)=-\log\frac{e^{f_y}}{\sum_i e^{f_i}} \tag{3.39}$$

be the cross-entropy loss, where $f(\mathbf{x})=[\cdots,f_j(\mathbf{x}),\cdots]^T$ is the model output. Note that we have

$$\frac{\partial\ell_{ce}}{\partial f_y}=\frac{e^{f_y}}{\sum_i e^{f_i}}-1,\qquad \frac{\partial\ell_{ce}}{\partial f_j}=\frac{e^{f_j}}{\sum_i e^{f_i}},j\neq y \tag{3.40}$$

Therefore,

$$\|\nabla_f \ell_{ce}(f(\mathbf{x}), y) - \nabla_{f^s} \ell_{ce}(f(\mathbf{x}), y)\|^2 = \sum_j \left(\frac{e^{f_j}}{\sum_i e^{f_i}} - \frac{e^{f_j^s}}{\sum_i e^{f_i^s}} \right)^2 \tag{3.41}$$

In practice, we may expect the change of model outputs is usually larger than the outputs after cross-entropy loss, i.e.

$$\forall j, \left(\frac{e^{f_j}}{\sum_i e^{f_i}} - \frac{e^{f_j^s}}{\sum_i e^{f_i^s}} \right)^2 \leq [f_i(\mathbf{x}) - f_i^s(\mathbf{x})]^2 \tag{3.42}$$

Hence, this implies

$$\|\nabla_f \ell_{ce}(f(\mathbf{x}), y) - \nabla_f \ell_{ce}(f^s(\mathbf{x}), y)\| \leq \|f(\mathbf{x}) - f^s(\mathbf{x})\| \tag{3.43}$$

Recall that the dynamics of adversarial training is

$$\frac{\mathrm{d}f_t}{\mathrm{d}t}(\mathbf{X}) = -\eta \Theta_t(\mathbf{X}, \tilde{\mathbf{X}}_t) \nabla_{f_t} \mathcal{L}(\tilde{\mathbf{X}}_t) \tag{3.44}$$

We may expect the optimization problem of finding adversarial winning ticket for general loss function is

$$\min_m \mathcal{L}'_{awt} = \frac{1}{N} \left\| \nabla_f \mathcal{L}(\tilde{\mathbf{X}}_0) - \nabla_{f^s} \tilde{\mathcal{L}}(\tilde{\mathbf{X}}'_0) \right\| + \frac{1}{N^2} \left\| \Theta_0(\mathbf{X}, \tilde{\mathbf{X}}_0) - \Theta_0^s(\mathbf{X}, \tilde{\mathbf{X}}'_0) \right\|_F \tag{3.45}$$

Now we compare the optimization problem (3.45) with problem (3.20). The first term of problem (3.45), as discussed above, is bounded by $\|f_0(\mathbf{X} + R_0) - f_0^s(\mathbf{X} + R_0^s)\|$. This shows that, if we find adversarial winning ticket according to optimization problem (3.45), the resulting training dynamics is bounded by the one we obtained before, so is close to the dynamics of dense network also. This suggests that the optimization problem (3.20) is general for both squared loss and cross-entropy loss.

Remark: We may expect that for many other loss functions, the following condition is true

$$\|\nabla_f \ell(f(\mathbf{x}), y) - \nabla_{f^s} \ell(f^s(\mathbf{x}), y)\| \leq C \|f(\mathbf{x}) - f^s(\mathbf{x})\| \tag{3.46}$$

Our method generalizes to the case using any loss function satisfying condition (3.46). Actually, for any convex loss function ℓ, if the second order derivative is bounded, then the above condition (3.46) is true by Taylor expansion.

3.7 Additional Experiments

In this section, we present additional experimental results which are not included in the maintext.

3.7.1 Experimental Configuration

We conduct experiments on standard datasets, including MNIST [49], CIFAR-10 and CIFAR-100 [45]. All experiments are performed in JAX [11], together with the neural-tangent library [65].

For the neural networks, in this paper, we evaluate our proposed method on two networks: MLP, 6-layer CNN and VGG16. The MLP is comprised of two hidden layers containing 300 and 100 units with rectified linear unit(ReLUs) followed with a linear output layer. The CNN has two convolutional layers with 32 and 64 channels, respectively. Each convoluational layer is followed by a max-pooling layer and the final two layers are fully connected with 1024 and 10/100 hidden nodes, respectively. For the experiments on VGG16, we explore the possibility of scaling up our method on large-size modern neural networks by using the technique such as sampling on the MTK matrix.

We evaluate the performance of our method on different density levels. To be precise: 1) For the experiments on MNIST with MLP, we vary the density level in $\{0.018, 0.036, 0.087, 0.169, 0.513\}$; For the experiments with CNN, the density level is set to be $[0.05, 0.1, 0.2, 0.3, \ldots, 0.9]$; 3) Since MNIST is much easier to be classified compared with CIFAR10 and CIFAR100, we also evaluate the performance of our method on MNIST with CNN at the density levels $\{0.01, 0.02, \ldots, 0.05\}$. Each experiment is comprised of two phases. For the experiments on MLP and CNN, in phase one, we run Algorithm 1 for 20 epochs to find the adversarial winning ticket. The weight γ of the kernel distance in Eqn.(10) is choosen to be $1e-3$. In phase two, we adversarially train the winning ticket from the original initialization with the cross entropy loss for 100 epochs by using L_∞-PGD attack. The ϵ in PGD attack in both phase one and two is set to be 0.3 and 8/255 in the experiments on MNIST and CIFAR-10/100, respectively. In both of these two phases, we adopt adopt Adam [43] to

solve the corresponding optimization problem. The learning rate is set to be $5e-4$ and $1e-3$ in phase one and phase two, respectively. The batch size is 64. For the experiments on VGG16, we run phase one for 10 epochs and phase two for 20 epochs. Other settings such as ϵ are the same as the experiments on MLP and CNN.

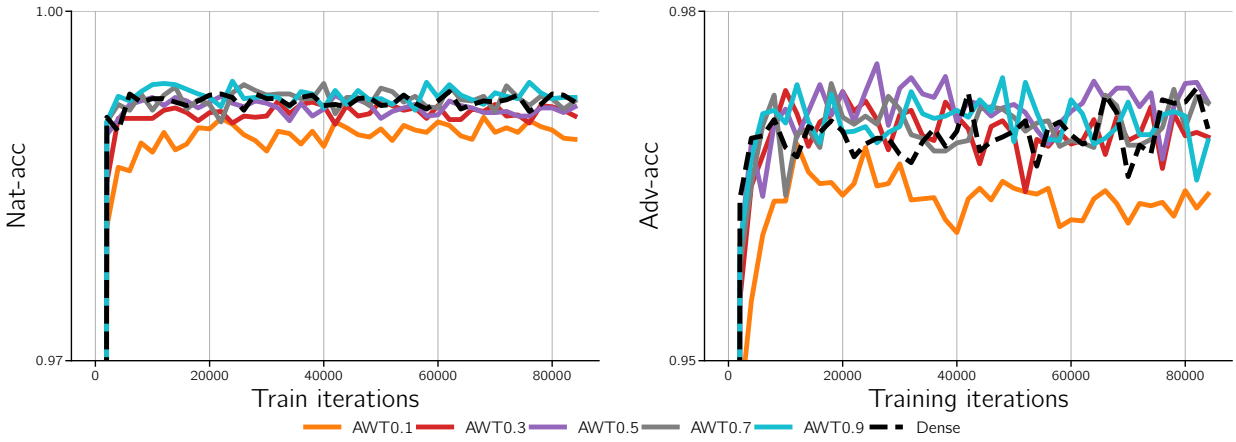


Figure 3.6: Test accuracy on natural and adversarial examples of CNN trained on MNIST.

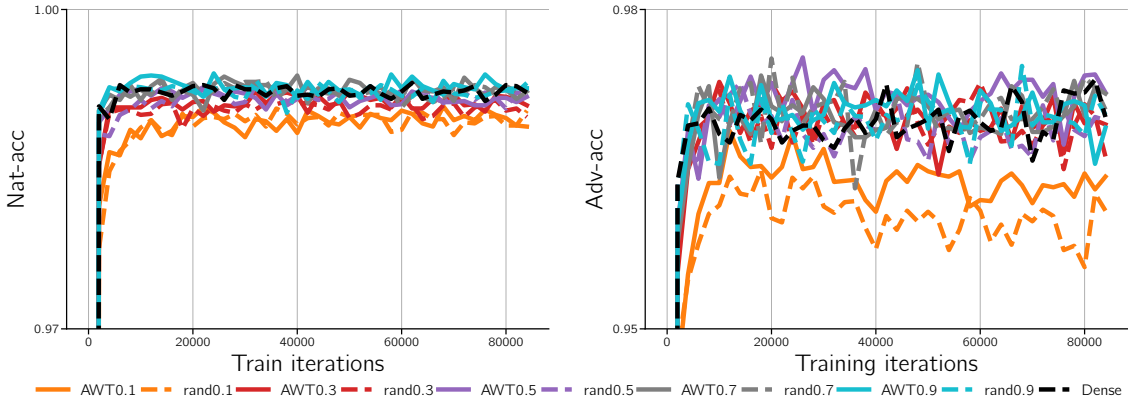


Figure 3.7: Natural and adversarial test accuracy of the models trained from AWT and random structure on MNIST with CNN.

3.7.2 More Results

Figure 3.6 presents the natural and adversarial testing accuracy of CNN trained on MNIST with the density level varies in $\{0.1, 0.2, \ldots, 0.9\}$. We can see that when the density level is larger than

0.2, the accuracy is very close to the dense mode. The reason could be that when the density level is larger than 0.2, the model begins to be overparameterized. This can also be seen in Figure 3.7. That is when the density level is larger than 0.2, there is even no significant difference between the winning ticket an the random structure. That's why we give the results with the density level varying in $\{0.01, 0.02, \ldots, 0.05\}$ in the main text.

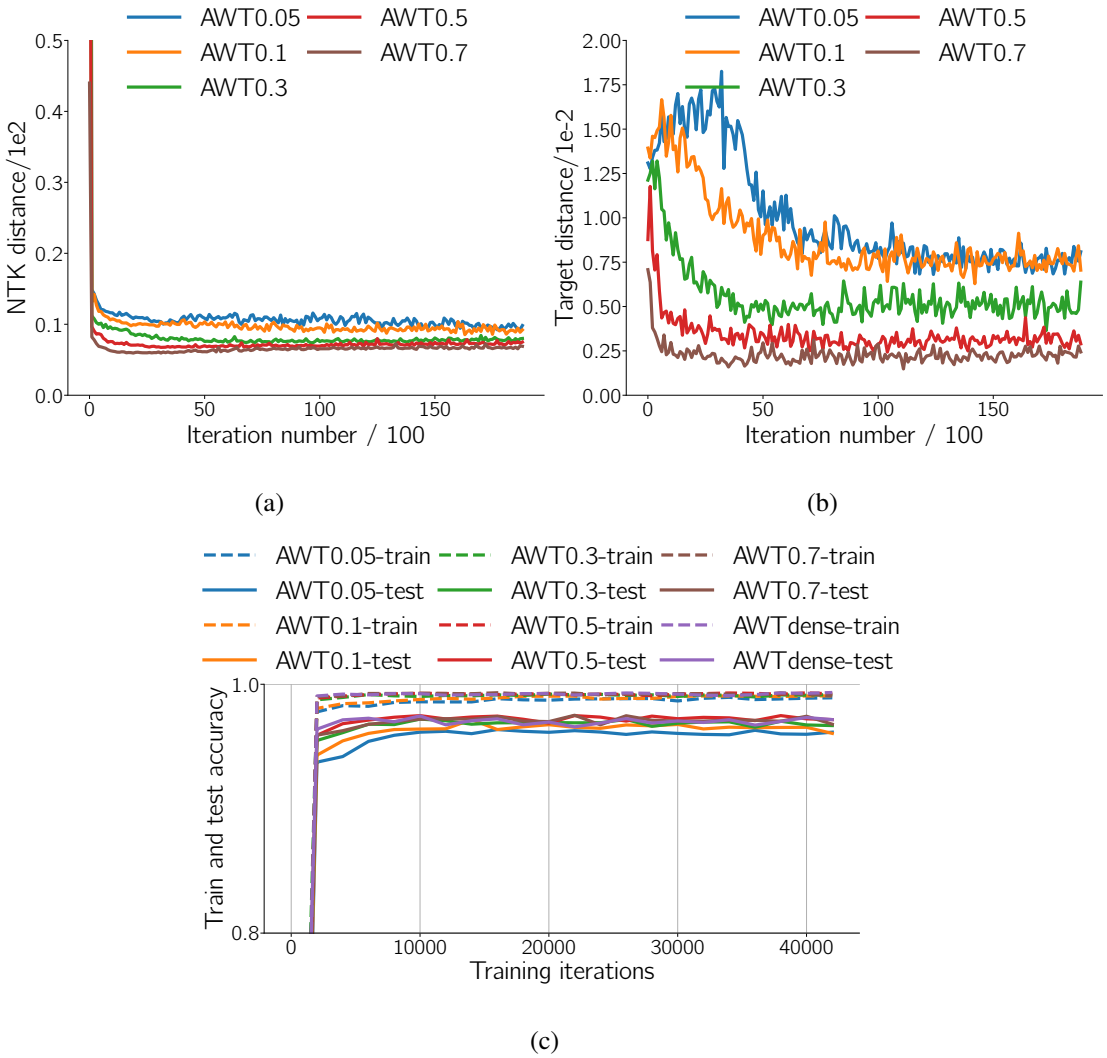


Figure 3.8: Statistics of AWT on MNIST with CNN over different density levels.

Figure 3.8(a) and (b) are the statistics of AWT, i.e, the kernel and target distances, in the procedure of searching masks on MNIST with cnn over density levels of $\{0.05, 0.1, 0.2, 0.3, \ldots, 0.9\}$. Figure 3.8(c) is the adversarial training and test accuracy curves in the training process (phase two). To

make the curves not too crowded, we omit the curves at the density levels of $\{0.2, 0.4, 0.6, 0.8, 0.9\}$. We can see that the target and kernel distances can decrease quickly in phase one and training dynamic of the winning ticket in phase two would become closer to the dense network when the density level increases. This is consistent with our theoretical analysis.

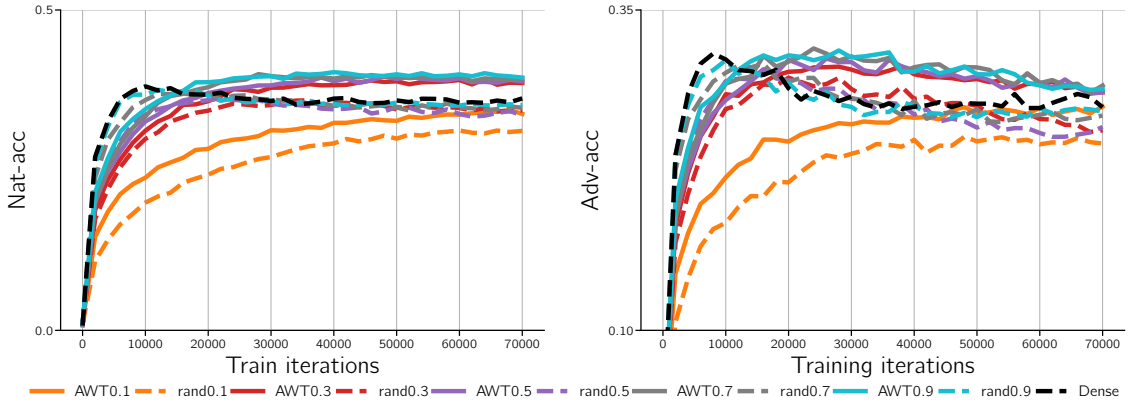


Figure 3.9: Natural and adversarial test accuracy of the models trained from AWT and random structure on CIFAR100 with CNN.

Figure 3.9 shows the performance of the models trained on CIFAR100 from our winning ticket and the random structure. We can see that after training, our winning ticket has significantly higher natural and adversarial test accuracy than that of the random structure. In this experiment, all the models cannot achieve the comparable test accuracy on natural examples as ResNet18 reported in the existing studies. The reason is that our model is a 6-layer CNN, whose capacity is much smaller than ResNet18.

3.7.3 An Ablation Study

Typical pruning algorithms follow the three-stage 'training-pruning-fine-tuning' pipelines, where 'unimportant' weights are pruned according to certain pruning strategies, such as magnitudes of weights. However, as observed in Liu et al. [55], fine-tuning a pruned model with inherited weights only gives comparable or worse performance than training that model with randomly initialized weights, which suggests that the inherited 'important' weights are not necessarily useful for fine-

tuning. We argue below that the change of model outputs dynamics is a potential reason for this phenomenon.

We have already known that the dynamics of model outputs can be completely described by NTK and the initial predictions. Hence the difference of dynamics between two neural networks can be quantified by the difference of their NTKs and initial predictions. Based on this result, Liu and Zenke [54] proposed *Neural Tangent Transfer* (NTT) to find trainable sparse sub-network structure which preserves the dynamics of model outputs by controlling the NTK distance and target distance between dense and sparse networks.

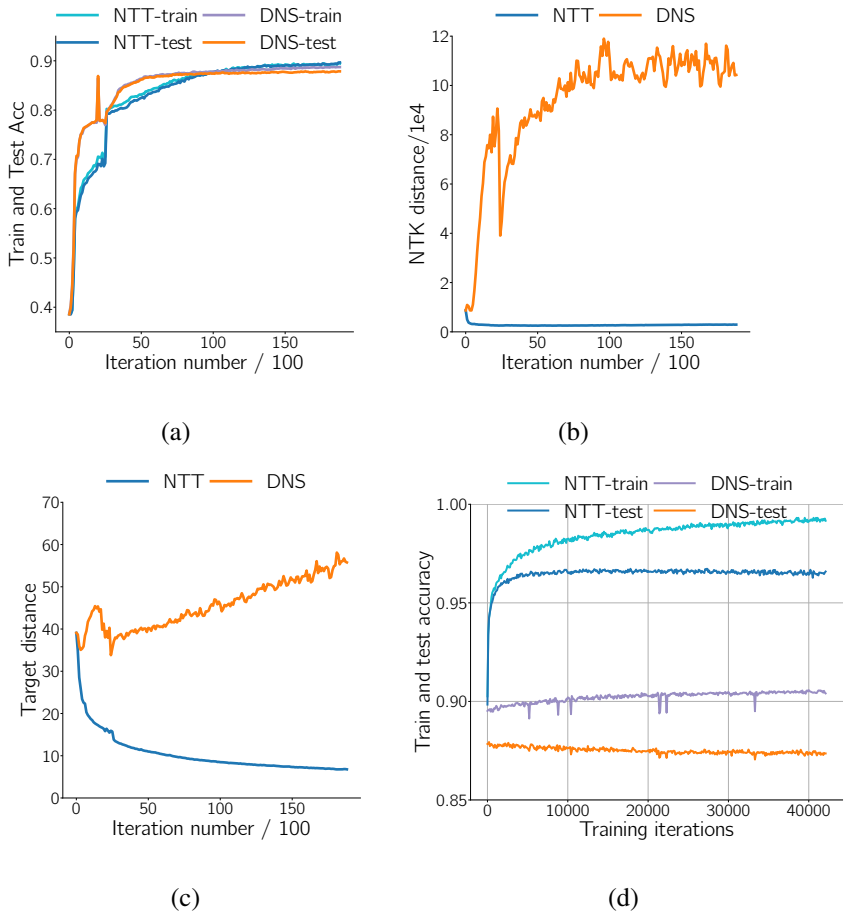


Figure 3.10: Statistics of NTT and DNS during mask searching and during fine-tuning.

We empirically compare various statistics of NTT with the well-known *Dynamics Network Surgery* (DNS) proposed in Guo et al. [33] during mask searching and retraining/fine-tuning procedures with the pruning rate being 0.02. NTT prunes the network at initialization, while DNS prunes the

network during training. In both NTT and DNS, we prune the network in 20 epochs with batch size being 64. And then we fine tune the obtained sparse network for 50 epochs.

In Figure 3.10 (a), train and test accuracy increase during mask search for both NTT and DNS. This indicates that both methods successfully find sparse network with good performance. However, as shown in Figure 3.10 (b) and (c), the NTK distances and target distances between dense and sparse networks obtained by NTT remain in a low scale, while for DNS these two quantities blow up. This indicates that DNS flows in a different way as NTT, which lead to a different dynamics as the original dense network. As a result, we can see in Figure 3.10 (d) that the sparse network found by DNS is harder to be fine-tuned, while we can train the sparse network obtained by NTT from scratch to get a better performance. This observation suggests that preserving the dynamics of outputs does help to find trainable sparse structures.

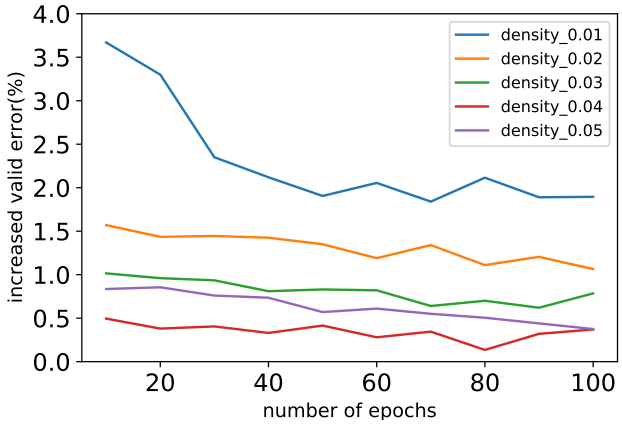


Figure 3.11: Graft results.

To show whether the training dynamics is preserved, instead of only looking at the kernel distance and target distance, we adopt a technique named network grafting [30] to verify whether dynamics are preserved. The idea is if two networks have same dynamics, then one can be grafted/connected onto the other at each layer at any epoch of training without significant error increase. We give the result on MNIST with CNN in Figure 3.11, where the error increases are very small, especially when the densities are low. This verifies the claim.

References

[1] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950. URL http://dx.doi.org/10.2307/1990404.

[2] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang. On exact computation with an infinitely wide neural net. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8141–8150, 2019.

[3] P. L. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482, 2002. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlr3.html#BartlettM02.

[4] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences of the United States of America*, 116(32):15849–15854, 2019.

[5] A. Benjamin, D. Rolnick, and K. Kording. Measuring and regularizing networks in function space. In *International Conference on Learning Representations*, 2018.

[6] A. Bietti and J. Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations. *The Journal of Machine Learning Research*, 20(1):876–924, 2019.

[7] A. Bietti, G. Mialon, D. Chen, and J. Mairal. A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning*, pages 664–674. PMLR, 2019.

[8] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger. Understanding batch normalization. *Advances in neural information processing systems*, 31, 2018.

[9] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Apr 2017. ISSN 1537-274X. doi: 10.1080/01621459.2017.1285773. URL http://dx.doi.org/10.1080/01621459.2017.1285773.

[10] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[11] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. Jax: composable transformations of python+ numpy programs, 2018. *URL http://github. com/google/jax*, 4:16, 2020.

[12] D. R. Burt, S. W. Ober, A. Garriga-Alonso, and M. van der Wilk. Understanding variational inference in function-space. *arXiv preprint arXiv:2011.09421*, 2020.

[13] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26, 2017.

[14] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In *Sparse matrix computations*, pages 309–332. Elsevier, 1976.

[15] C. Cortes, M. Mohri, and A. Rostamizadeh. Generalization bounds for learning kernels. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 247–254, 2010.

[16] J. Cosentino, F. Zaiter, D. Pei, and J. Zhu. The search for sparse, robust neural networks. *arXiv preprint arXiv:1912.02386*, 2019.

[17] F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020.

[18] A. G. de Garis Matthews. *Scalable Gaussian process inference using variational methods*. PhD thesis, University of Cambridge, 2017.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[20] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. PMLR, 2017.

[21] N. Eldredge. Analysis and probability on infinite-dimensional spaces. *arXiv preprint arXiv:1607.03591*, 2016.

[22] S. Farquhar, M. A. Osborne, and Y. Gal. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1352–1362. PMLR, 2020.

[23] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

[24] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

[25] Y. Fu, Q. Yu, Y. Zhang, S. Wu, X. Ouyang, D. Cox, and Y. Lin. Drawing robust scratch tickets: Subnetworks with inborn robustness are found within randomly initialized networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[26] R. Gao, T. Cai, H. Li, C.-J. Hsieh, L. Wang, and J. D. Lee. Convergence of adversarial

training in overparametrized neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[27] A. Geifman, A. Yadav, Y. Kasten, M. Galun, D. Jacobs, and B. Ronen. On the similarity between the laplace and neural tangent kernels. In *Advances in Neural Information Processing Systems*, volume 33, pages 1451–1461. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1006ff12c465532f8c574aeaa4461b16-Paper.pdf.

[28] J. Gilles. *The lottery ticket hypothesis in an adversarial setting*. PhD thesis, Massachusetts Institute of Technology, 2020.

[29] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

[30] Y. Gu, W. Zhang, C. Fang, J. D. Lee, and T. Zhang. How to characterize the landscape of overparameterized convolutional neural networks. *Advances in Neural Information Processing Systems*, 33:3797–3807, 2020.

[31] S. Gui, H. N. Wang, H. Yang, C. Yu, Z. Wang, and J. Liu. Model compression with adversarial robustness: A unified optimization framework. In *Advances in Neural Information Processing Systems*, pages 1285–1296, 2019.

[32] M. Guo, Y. Yang, R. Xu, Z. Liu, and D. Lin. When nas meets robustness: In search of robust architectures against adversarial attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 631–640, 2020.

[33] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016.

[34] Y. Guo, C. Zhang, C. Zhang, and Y. Chen. Sparse dnns with improved adversarial robustness. In *Advances in neural information processing systems*, pages 242–251, 2018.

[35] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[38] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[39] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[40] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

[41] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[42] M. E. E. Khan, A. Immer, E. Abedi, and M. Korzepa. Approximate inference turns deep

networks into gaussian processes. *Advances in neural information processing systems*, 32, 2019.

[43] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[44] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[45] A. Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009.

[46] J. Lamperti. *Stochastic processes : a survey of the mathematical theory / J. Lamperti.* Applied mathematical sciences (Springer-Verlag New York Inc.); v. 23. Springer-Verlag, New York, 1977. ISBN 0387902759.

[47] G. R. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine learning research*, 5(Jan): 27–72, 2004.

[48] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[50] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, volume 32, pages 8572–8583. Curran

Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/0d1a9651497a38d8b1c3871c84528bd4-Paper.pdf.

[51] N. Lee, T. Ajanthan, and P. Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2018.

[52] B. Li, S. Wang, Y. Jia, Y. Lu, Z. Zhong, L. Carin, and S. Jana. Towards practical lottery ticket hypothesis for adversarial training. *arXiv preprint arXiv:2003.05733*, 2020.

[53] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[54] T. Liu and F. Zenke. Finding trainable sparse networks through neural tangent transfer. In *International Conference on Machine Learning*, pages 6336–6347. PMLR, 2020.

[55] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJlnB3C5Ym.

[56] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[57] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[58] T. Maho, T. Furon, and E. Le Merrer. Surfree: A fast surrogate-free black-box attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10430–10439, June 2021.

[59] A. G. d. G. Matthews, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations*, 2018.

[60] C. A. Micchelli, M. Pontil, and P. Bartlett. Learning the kernel function via regularization. *Journal of machine learning research*, 6(7), 2005.

[61] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X.

[62] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[63] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018.

[64] R. Novak, L. Xiao, Y. Bahri, J. Lee, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2018.

[65] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020. URL `https://github.com/google/neural-tangents`.

[66] C. S. Ong, A. J. Smola, and R. C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6(36):1043–1071, 2005.

[67] P. Pan, S. Swaroop, A. Immer, R. Eschenhagen, R. Turner, and M. E. E. Khan. Continual deep learning by functional regularisation of memorable past. *Advances in Neural Information Processing Systems*, 33:4453–4464, 2020.

[68] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[69] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Advances in neural information processing systems*, 29, 2016.

[70] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[71] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Jan. 2006.

[72] M. Reed and B. Simon. *Methods of modern mathematical physics*, volume 1. Elsevier, 1972.

[73] A. Renda, J. Frankle, and M. Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2019.

[74] T. G. J. Rudner, Z. Chen, Y. W. Teh, and Y. Gal. Rethinking Function-Space Variational Inference in Bayesian Neural Networks. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2021.

[75] P. Samangouei, M. Kabkab, and R. Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BkJ3ibb0-`.

[76] S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2016.

[77] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[78] B. Schölkopf, A. J. Smola, F. Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[79] L. Schott, J. Rauber, M. Bethge, and W. Brendel. Towards the first adversarially robust neural network model on mnist. In *Seventh International Conference on Learning Representations (ICLR 2019)*, pages 1–16, 2019.

[80] V. Sehwag, S. Wang, P. Mittal, and S. Jana. Hydra: Pruning adversarially robust neural networks. *Advances in Neural Information Processing Systems*, 33:19655–19666, 2020.

[81] A. Shafahi, M. Najibi, M. A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/7503cfacd12053d309b6bed5c89de212-Paper.pdf.

[82] X. Shi and A. A. Ding. Understanding and quantifying adversarial examples existence in linear classification. *arXiv preprint arXiv:1910.12163*, 2019.

[83] X. Shi, P. Zheng, A. Ding, Y. Gao, and W. Zhang. Finding dynamics preserving adversarial winning tickets. *arXiv preprint arXiv:2202.06488*, 2022.

[84] A. Sinha, H. Namkoong, and J. Duchi. Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hk6kPgZA-.

[85] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[86] S. Sun, G. Zhang, J. Shi, and R. Grosse. Functional variational bayesian neural networks. In *International Conference on Learning Representations*, 2018.

[87] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2014.

[88] M. K. Titsias, J. Schwarz, A. G. d. G. Matthews, R. Pascanu, and Y. W. Teh. Functional regularisation for continual learning with gaussian processes. In *International Conference on Learning Representations*, 2019.

[89] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34, 2021.

[90] A. R. Triki, M. Berman, and M. B. Blaschko. Function norms and regularization in deep networks. *arXiv preprint arXiv:1710.06703*, 2017.

[91] Z. Tu, J. Zhang, and D. Tao. Theoretical analysis of adversarial learning: A minimax approach. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/16bda725ae44af3bb9316f416bd13b1b-Paper.pdf`.

[92] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[93] C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019.

[94] S. Wang, N. Liao, L. Xiang, N. Ye, and Q. Zhang. Achieving adversarial robustness via sparsity. *arXiv preprint arXiv:2009.05423*, 2020.

[95] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020.

[96] C. Williams and M. Seeger. Using the nystroem method to speed up kernel machines. *Advances in Neural Information Processing Systems 13*, 2001.

[97] E. Wong, L. Rice, and J. Z. Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJx040EFvH`.

[98] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018.

[99] G. Yang and S. Schoenholz. Mean field residual networks: On the edge of chaos. *Advances in neural information processing systems*, 30, 2017.

[100] S. Ye, K. Xu, S. Liu, H. Cheng, J.-H. Lambrechts, H. Zhang, A. Zhou, K. Ma, Y. Wang, and X. Lin. Adversarial robustness vs. model compression, or both? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 111–120, 2019.

[101] W. Zeng and R. Urtasun. Mlprune: Multi-layer pruning for automated neural network compression.(2019). In *URL https://openreview. net/forum*, 2019.

[102] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[103] Y. Zhang, O. Plevrakis, S. S. Du, X. Li, Z. Song, and S. Arora. Over-parameterized adversarial training: An analysis overcoming the curse of dimensionality. *Advances in Neural Information Processing Systems*, 33:679–688, 2020.

[104] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, 33:21285–21296, 2020.

[105] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.