University of California
Santa Barbara

Overparameterization in neural networks: from application to theory

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Kaiqi Zhang

Committee in charge:

Professor Yu-Xiang Wang, Chair
Professor Shiyu Chang
Professor Yufei Ding
Professor Ramtin Pedarsani

June 2023

The Dissertation of Kaiqi Zhang is approved.

Professor Shiyu Chang

Professor Yufei Ding

Professor Ramtin Pedarsani

Professor Yu-Xiang Wang, Committee Chair

May 2023

Overparameterization in neural networks: from application to theory

Kaiqi Zhang

Dedication here

Acknowledgements

First of all, I would like to express my heartfelt gratitude to my PhD advisor, Professor Yu-Xiang Wang, for his continuous guidance and unwavering support throughout my doctoral journey. His patience, kindness, rich knowledge and profound insights into research directions have been invaluable in shaping me into a researcher. I am deeply indebted to him for helping me overcome numerous challenges and obstacles along the winding path of my PhD.

I would also like to extend my sincere appreciation to my former PhD advisor, Professor Zheng Zhang, for introducing me to the captivating field of machine learning. He played a pivotal role in shaping my thinking, work ethic, and academic development during the early stages of my career.

In addition to my advisors, I am immensely grateful for the collaboration and contributions of Xiyuan Zhang, Cole Hawkins, Ming Yin, Zixuan Zhang, Minshuo Chen, Professor Cong Hao, Professor Mengdi Liu, and Professor Tuo Zhao. Their invaluable support and joint efforts were instrumental in completing the research projects presented in this thesis. I would also like to acknowledge the support of my office mates, including Professor Chunfeng Cui, Zichang He, Zhuotong Chen, Jianyu Xu, Chong Liu, Yuqing Zhu, Dheeraj Baby, and many others, whose discussions and interactions have greatly enriched my understanding and exploration of new areas within our field.

Lastly, I would like to express my deepest gratitude to my parents for their unwavering love and support throughout my academic journey. Their unconditional belief in me has been a constant source of motivation. I would also like to thank my girlfriend, Yimeng Liu, for her support in both my career and personal life. And last but not least, a special mention goes to my cat, Princess, who has been by my side throughout these years.

Curriculum Vitæ

Kaiqi Zhang

Education

2018-2023	Ph.D. in Electrical and Computer Engineering, University of California, Santa Barbara.
2016-2018	M.S. in Electrical and Computer Engineering, University of California, Davis.
2012-2016	B.S. in Electronic Engineering. Tsinghua University.

Publications

K. Zhang, X. Zhang, and Z. Zhang, *Tucker tensor decomposition on fpga*, in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2019

C. Cui, K. Zhang, T. Daulbaev, J. Gusak, I. Oseledets, and Z. Zhang, *Active subspace of neural networks: Structural analysis and universal attacks*, *SIAM Journal on Mathematics of Data Science* **2** (2020), no. 4 1096–1122

K. Zhang, C. Hawkins, X. Zhang, C. Hao, and Z. Zhang, *On-fpga training with ultra memory reduction: A low-precision tensor method*, in *ICLR Workshop on Hardware Aware Efficient Training*, 2021

K. Zhang, C. Hawkins, and Z. Zhang, *General-purpose bayesian tensor learning with automatic rank determination and uncertainty quantification*, *Frontiers in Artificial Intelligence* **4** (2022) 668353

K. Zhang, M. Yin, and Y.-X. Wang, *Why quantization improves generalization: Ntk of binary weight neural networks*, *arXiv preprint arXiv:2206.05916* (2022)

K. Zhang and Y.-X. Wang, *Deep learning meets nonparametric regression: Are weight-decayed dnns locally adaptive?*, *arXiv preprint arXiv:2204.09664* (2022)

Abstract

Overparameterization in neural networks: from application to theory

by

Kaiqi Zhang

Neural networks are rapidly increasing in size, leading to a common occurrence of overparameterization in deep learning. This presents challenges in both the theory and application of deep learning. From a theoretical standpoint, it remains an open question as to why neural networks generalize well despite overparameterization. From an application perspective, overparameterization leads to significant computation and storage costs, which limits the practical application of deep neural networks.

This thesis presents our attempt to address both issues. In terms of application, we propose training a low-rank tensorized neural network to compress the model and reduce the computation cost during both training and inference. We also apply Bayesian methods to evaluate the uncertainty of this model. In terms of theory, we apply a new method — neural tangent kernel (NTK) — to study the training dynamics of an infinitely wide neural network. We compare the eigenvalues of the NTK of a vanilla neural network with that of a binary weight neural network, and show that the latter decays faster. This explains why binary weight neural networks have lower generalization gap empirically. We also examine the effect of weight decay on a neural network, and demonstrate that it induces sparsity in both a parallel neural network and a ResNet, thus prove that neural networks are locally adaptive, which is not present in any linear method, including kernels.

For the problems discussed above, we present both theoretical analyses of our method or statement, and numerical experiments to validate our conclusions.

Contents

Chapter 1

Introduction

1.1 Background

Over the past few decades, neural networks have been rapidly increasing in its width and depth. With this increase, the number of parameters within a neural network has also grown rapidly, resulting in a phenomenon called overparameterization. In an overparameterized model, the number of parameters is not only larger than necessary but also exceeds the size of the training set.

Overparameterization has become a significant problem in both empirical studies and theoretical analysis. Empirically, overparameterization greatly increases the time and cost to train, store, and deploy a model. This is because the large number of parameters requires more computational resources to train the model, and the trained model also occupies more memory space for storage and deployment.

On the theoretical front, it can be found that while a slightly overparameterized neural network often overfit, a strongly overparameterized neural network can generalize well. This phenomenon is called "double descent" and it remains largely unexplained.

1.2 Contributions

The contributions of this article is two-folded. From the empirical prospective, we propose several methods to reduce the number of parameters in a neural network with only a small loss in accuracy. We design an algorithm to train a neural network with reduced number of parameters directly, and this greatly reduces the computation and storage cost to train a neural network. We demonstrate that this training algorithm can be implemented on an embedded device.

From the theoretical prospective, we aim to provide an explanation why neural networks generalize well despite overparameterization, and why overparameterized neural networks outperform other methods. On the limit that the width of a neural network is infinity, we show that training a binary weight neural network is similar to kernel learning with Gaussian kernel. As for a neural network with finite width, we show that training a parallel neural network or a ResNeXt induces sparsity, which reduces the number of parameters after training and avoids overfitting. We show that these neural networks enjoy strong adaptivity, which does not exist in traditional models, and this explains why neural networks outperform traditional machine learning methods including kernel methods.

Chapter 2

Reducing Overparameterization: Bayesian Tensorized Nerual Networks

2.1 Introduction

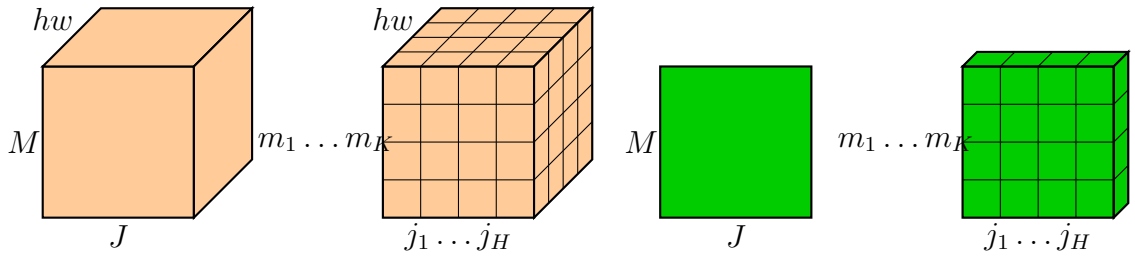
Overparameterization increases the computation and storage cost of neural networks and limits its application. In order to reduce the number of parameters in a neural network, many methods have been proposed, including pruning [7, 8], quantization [9, 10], knowledge distillation [11], and low-rank approximation [12, 13, 14, 15]. In this section, we focus on another method to reduce the number of parameters in a neural network called tensorized neural network [16, 17, 15, 18]. The key idea is to tensorize its convolution kernels and fully connected weights into higher order tensors, as shown in Fig. 2.1. Consequently, different tensor decomposition method such as Tucker decomposition and

This work has been published as K. Zhang, C. Hawkins, and Z. Zhang, *General-purpose bayesian tensor learning with automatic rank determination and uncertainty quantification*, *Frontiers in Artificial Intelligence* **4** (2022) 668353.

tensor-train (TT) decomposition can be employed to compress the weights. We design a low-rank-inducing prior for the tensorized neural networks such that the models can be further compressed during training.

On the other hand, a well known problem of deep neural networks is that deep neural networks are not uncertainty-aware. As a result, deep neural networks are often over-confident about its prediction, even when the prediction is incorrect. A solution to this problem is Bayesian neural networks, which can provide uncertainty estimations for both the model parameters and the predictive results. Instead of using a single maximum of posterior (MAP) model for prediction, Bayesian neural networks can estimate the posterior distribution of the prediction to a data point conditioned on the training samples.

Following Neal [19], we propose training a Bayesian tensorized neural networks using using Hamiltonian Monte Carlo (HMC) [20]. HMC approach is an attractive method to solve tensor learning problems because it avoids the random walks of a standard Markov-Chain Monte Carlo (MCMC) method and leads to significantly lower computational cost. Due to the huge amount of training data in many tensor learning problems, estimating the full gradient can be computationally expensive. Therefore, we replace the full gradient in a tensor learning problem with the stochastic gradient [21] while achieving a similar level of accuracy. In summary, we design an efficient and scalable method to train a Bayesian tensorized neural network.



(a) The convolutional kernel (b) The fully connected weights

Figure 2.1: Unfolding of the convolutional kernel and the fully connected weights

2.2 Problem formulation

2.2.1 Tensorized neural networks

The majority of the parameters in a typical neural network lay in the weight of the fully-connected layers and convolution layers. Motivated by this, tensorized neural networks was proposed to compress a neural network. In a tensorized neural network, the weights in the fully-connected layers and/or convolution layers are reshaped into tensors, which is defined below.

Definition 2.1 *A tensor* $\mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$ *is a high-dimensional array of order* n*. Here the order* n *(also known as "way") is the total number of dimensions. For a general* n*-th order tensor* $\mathcal{W}$*, its element indexed by* $(i_1, i_2 \cdots, i_n)$ *is denoted as* $\mathcal{W}[i_1 i_2 \cdots i_n]$*.*

Then a tensor decomposition method is applied to decompose the weight tensor into the product of a series of smaller tensors or matrices. There are three commonly used tensor decomposition methods: Canonical polyadic (CP)decomposition [22], tensor Tucker decomposition [23], and tensor train decomposition [24]. We focus on tensor train decomposition, because this often yield the best cost-accuracy trade-off, while we note that this method can be applied to all the decomposition methods.

For a weight matrix $\mathbf{W}$ of size $M \times J$, one can decompose $M = \prod_{k=1}^{K} m_k$ and $J = \prod_{k=1}^{K} j_k$, then reformulate $\mathbf{W}$ as a $2K$-dimension tensor $\mathcal{W}$ with size $m_1 \times j_1 \times$

$\cdots \times m_K \times j_K$. Afterwards, $\mathcal{W}$ is approximated by a low-rank *tensor-train decomposition*

$$\mathcal{W} = [\![\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(K)}]\!]_{TT}, \tag{2.1}$$

where $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1} \times m_k \times j_k \times R_k}$ is called the *TT core*, R_k is the *TT rank*, $R_0 = R_K = 1$, and $[\![\cdot]\!]_{TT}$ denotes the tensor-train product [24]. The convolutional layers can be decomposed in a similar way. The convolution kernel $\mathcal{C}$ is a 4-th dimension tensor in $M \times J \times H \times W$, where H and W denote the height and width of the convolution window. This tensor can be further viewed as a $(2K+2)$-dimensional tensor with size $m_1 \times j_1 \times \cdots \times m_K \times j_K \times H \times W$. In our experiments $H = W = 3$ remain unchanged, and we only compress along the remaining dimensions, i.e.,

$$\mathcal{C} = [\![\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(2K)}]\!]_{TT}. \tag{2.2}$$

The shape of each factors $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots \mathcal{G}^{(2K-1)}, \mathcal{G}^{(2K)}$ are $m_1 \times R_1, R_1 \times j_1 \times R_2, \ldots, R_{2K-2} \times m_K \times R_{2K-1}, R_{2K-1} \times j_K \times H \times W$, respectively.

2.2.2 Bayesian model

Given the training data $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, we want to find a low-rank tensor $\mathcal{W}$ in the TT format to describe the weight matrices or convolution filters such that $\mathbf{y} = g(\mathbf{x}, \mathcal{W})$. Our goal is to estimate the posterior density

$$P(\Theta|\mathcal{D}) \propto \prod_{n=1}^{N} P(D_n|\Theta)P(\Theta). \tag{2.3}$$

Here $P(\mathcal{D}|\Theta) = \prod_{n=1}^{N} P(D_n|\Theta)$ is a likelihood function, $P(\Theta)$ is a prior probability density. A key advantage of this Bayesian parameterized description is as follows: by properly

choosing a prior density $P(\Theta)$, one can control the structure of Θ and thus automatically enforce a low-rank representation for $\mathcal{X}(\Theta)$ based on the observed data $\mathcal{D}$. Doing so overcome the difficulty of rank determination in an optimization-based tensor learning. Our choice of the likelihood function will be discussed in Section 2.2.3 and the prior distribution $P(\Theta)$ will be discussed in Section 2.2.4.

2.2.3 Likelihood function

The choice of the likelihood function is determined by the training task. We provide two examples below, corresponding to the classification problems and the regression problems respectively.

Classification problems

Suppose the training dataset is $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}$. In most classification problems, the neural network can give a likelihood $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \Theta)$ directly, where $f(\boldsymbol{x}_i; \Theta)$ is the propagation function of the network, $\hat{\boldsymbol{y}}_i$ is a vector and each element denotes the probability that x_i belongs to one class. It is usually the softmax of output of the last linear layer.

Suppose $\boldsymbol{y}_i$ is a vector with size C, C is the total number of classes,

$$\boldsymbol{y}_{ic} = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ in class } c \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

The negative log likelihood is

$$-\log P(\boldsymbol{y}_i|\Theta) = \langle \boldsymbol{y}_i, -\log f(x_i; \Theta) \rangle . \tag{2.5}$$

Regression Problem

In regression problems, it is usually assumed that

$$P(\boldsymbol{y}_i|\Theta) = \mathcal{N}(\boldsymbol{y}_i|f(\boldsymbol{x}_i;\Theta)), \tag{2.6}$$

$$-\log P(\boldsymbol{y}_i|\Theta) = \frac{1}{2}(\boldsymbol{y}_i - f(\boldsymbol{x}_i;\Theta))^2/\sigma^2, \tag{2.7}$$

where σ is a hyperparameter denoting the variance.

2.2.4 Rank Determination

Here a Gaussian prior is placed over each tensor factor and a Gamma prior is placed over $\Lambda^{(k)}$,

$$\begin{aligned} P(\mathcal{G}^{(k)}|\Lambda^{(k-1)},\Lambda^{(k)}) &= \prod_{i,j}\mathcal{N}\big(\mathcal{G}^{(k)}(i,:,j)\big|0,(c_k\lambda_i^{(k-1)}\lambda_j^{(k)})^{-1}\big), \\ P(\Lambda^{(k)}) &= \prod_{r=1}^{R_k}\text{Gamma}(\lambda_r^{(k)}|\alpha,\beta), \\ P(\Theta) &= \prod_{k=1}^{d}P(\mathcal{G}^{(k)}|\Lambda^{(k-1)},\Lambda^{(k)})\prod_{k=1}^{d-1}P(\Lambda^{(k)}). \end{aligned} \tag{2.8}$$

where α and β are constants. Once the estimated parameter $\lambda_r^{(k)}$ is larger than a threshold, we delete one horizontal slice of $\mathcal{G}^{(k)}$ and one frontal slice of $\mathcal{G}^{(k+1)}$.

A threshold ϵ can be set to determine the rank. A tensor slice can be eliminated, and correspondingly the rank can be reduced when

$$\hat{\lambda}_r^{(k)} \geq \log(\frac{1}{2}S_kR_{k-1} + \frac{1}{2}S_{k+1}R_{k+1} + \alpha) + \log\beta - \epsilon,$$

where $S_k = M_kJ_k$ for the fully connected layers and $S_{2k-1} = M_k, S_{2k} = J_k$ for the

convolutional layers.

2.3 Bayesian training

2.3.1 Stochastic Gradient HMC (SGHMC) Solver

Now we need to estimate the hidden tensor factors and hyper-parameters by computing the posterior density in (2.3). Existing methods [25, 26, 25, 26] does not apply to generalized tensor learning problems where resulting Bayesian models violate the required strong assumptions. Therefore, we we employ Hamiltonian Monte Carlo (HMC) [20] to make our framework applicable to a broad class of tensor learning problems.

The HMC method avoids the random walks in a standard MCMC framework by simulating the following dynamic system:

$$\frac{d\Theta}{dt} = \mathbf{M}^{-1}\mathbf{p}, \qquad \frac{d\mathbf{p}}{dt} = -\nabla U(\Theta). \tag{2.9}$$

Here $\mathbf{p}$ is the auxiliary momentum variable with the same dimension as Θ, $\mathbf{M}$ is a mass matrix. Here $U(\Theta)$ is the potential energy which is equal to the negative log posterior:

$$U(\Theta) = -\log P(\Theta|\mathcal{D}) = -\sum_{n=1}^{N} \log P(D_n|\Theta) - \log P(\Theta). \tag{2.10}$$

The HMC method starts from an initial guess of Θ, and its steady-state distribution converges to our desired posterior density $P(\Theta|\mathcal{D})$.

A standard HMC becomes inefficient when we solve a tensor learning problem with massive training samples, because computing the gradient requires estimating $\nabla \log P(D_n|\Theta)$ for every index n over the whole data set. This often happens in completing a huge-size tensor data set or training a tensorized neural network. To reduce the cost, we use the

stochastic unbiased estimator of $U(\Theta)$:

$$\tilde{U}(\Theta) = -\frac{N}{|\mathcal{B}|}\sum_{D_i\in\mathcal{B}} \log P(D_i|\Theta) - \log P(\Theta) + \text{const.} \tag{2.11}$$

Here $\mathcal{B} \subset \mathcal{D}$ denotes a mini-batch with $|\mathcal{B}| \ll N$. Then one can update the parameters via $\frac{d\Theta}{dt} = \mathbf{M}^{-1}\mathbf{p}$ and $\frac{d\mathbf{p}}{dt} = -\nabla\tilde{U}(\Theta)$. To compensate the noise introduced by the stochastic gradient, we adopt the thermostats method [27] for our tensor learning framework. Specifically, a friction term c is introduced, i.e.,

$$\begin{aligned} &\frac{d\Theta}{dt} = \mathbf{M}^{-1}\boldsymbol{p}, \qquad \frac{d\mathbf{p}}{dt} = -\nabla\tilde{U}(\Theta) - c\boldsymbol{p}, \\ &\frac{dc}{dt} = \frac{1}{|\Theta|}\text{tr}(\boldsymbol{p}^T\mathbf{M}^{-1}\boldsymbol{p}) - 1. \end{aligned} \tag{2.12}$$

The friction term changes accordingly to keep the average kinetic energy $\frac{1}{2}\boldsymbol{p}^T\mathbf{M}^{-1}\boldsymbol{p}$ constant, thus keeping the distribution of samples invariant.

Our method employs a slightly modified leapfrog approach to solve the Hamiltonian system because it has a smaller integration error compared with other methods [20]:

$$\begin{aligned} \mathbf{p}_{t+\epsilon/2} &\leftarrow \mathbf{p}_t - \tfrac{1}{2}\epsilon(\nabla\tilde{U}(\Theta_t) + c_t\boldsymbol{p}_t), \\ \Theta_{t+\epsilon} &\leftarrow \Theta_t + \epsilon\mathbf{p}_{t+\epsilon/2}, \\ \mathbf{p}_{t+\epsilon} &\leftarrow \mathbf{p}_{t+\epsilon/2} - \tfrac{1}{2}\epsilon(\nabla\tilde{U}(\Theta_{t+\epsilon}) + c_t\boldsymbol{p}_{t+\epsilon/2}), \\ c_{t+\epsilon} &\leftarrow c_t + \epsilon(\tfrac{1}{|\Theta|}\text{tr}(\boldsymbol{p}^T\mathbf{M}^{-1}\boldsymbol{p}) - 1), \end{aligned} \tag{2.13}$$

where ϵ is the stepsize, t is the iteration index.

2.3.2 The Potential Function

For all hyperparameters $\lambda_r^{(k)}$, we sample $\hat{\lambda}_r^{(k)} = \log\lambda_r^{(k)}$ and use the log Gamma distribution as a prior. Denote the training dataset as $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$. **The potential**

Algorithm 1: SGHMC with thermostats

Input : The dataset $\mathcal{D}$, the potential $U(\Theta)$, the mass $\mathbf{M}$, the maximal number of samples T

Initialize Θ by minimizing $U(\Theta)$ using SGD, Adam, etc.

for $t = 1, 2, \ldots T$ **do**

  Sample the momentum $\mathbf{p} \sim \mathcal{N}(0, \mathbf{M})$.

  Draw a mini-batch $\mathcal{B} \subset \mathcal{D}$ to formulate the unbiased potential function $\tilde{U}(\Theta)$ by equation (2.11).

  for $i = 1$ *to* m **do**

    Update $\Theta, \boldsymbol{p}, c$ using (2.13)

  end

  $\Theta^{(t)} \leftarrow \Theta$

end

Output: The sample set of $\{\Theta^{(t)}\}_{t=1}^{T}$.

function can be computed as

$$U(\Theta) = -\log P(\Theta|\mathcal{D}) = \sum_{n=1}^{N} \text{loss}(\mathbf{y}_n, g(\mathbf{x}_n, \Theta)) - \log P(\Theta), \tag{2.14}$$

where loss$(\cdot)$ is the negative log likelihood and $g(\cdot)$ denotes the neural network. The loss function can be the cross entropy loss for classification problems and the mean square error loss for regression problems. After getting the potential function, we can apply the SGHMC framework to draw samples for the parameters Θ.

2.3.3 More General Models

The above descriptions of the prior, the likelihood, and the potential function are all based on a low-rank tensor-train representation. Our Bayesian tensor learning framework can also be applied to other decomposition of network parameters such as the CP decomposition and Tucker decomposition.

Bayesian Tucker Factorization

The Tucker decomposition projects the original tensor $\mathcal{X}$ into a smaller kernel tensor $\mathcal{G}$,

$$\mathcal{X} = \mathcal{G} \bigotimes_{k=1}^{d} \mathbf{U}^{(k)}. \tag{2.15}$$

Similar to Zhao et al. [28], the priors of $\mathbf{U}^{(k)}$ and $\mathcal{G}$ are set as

$$P(\mathbf{U}^{(k)}|\Lambda^{(k)}) = \prod_{i_k=1}^{I_k} \mathcal{N}(\mathbf{U}^{(k)}(i_k,:)|0,(\Lambda^{(k)})^{-1}) \tag{2.16}$$

and

$$\begin{aligned} &P(\mathcal{G}|\Lambda^{(1)},\ldots,\Lambda^{(d)}) \\ &= \prod_{r_1,\ldots,r_d} \mathcal{N}\left(\mathcal{G}(r_1,\ldots,r_d) \,\middle|\, 0, \beta \prod_{k=1}^{d} \left(\lambda_{r_k}^{(k)}\right)^{-1}\right) \end{aligned} \tag{2.17}$$

respectively. Here β is a constant scaling factor. For simplicity, we assume β is a constant instead of a random variable, which is different from Zhao et al. [28]. $\Lambda^{(k)}$ follows from the Gamma distribution

$$P(\Lambda^{(k)}) = \prod_{r=1}^{R_k} \text{Gamma}(\lambda_r^{(k)}|a,b).$$

Here, $\Lambda^{(k)}$ is shared between $\mathbf{U}^{(k)}$ and $\mathcal{G}$. In summary, the prior of the unknown parameters $\Theta = \{\mathcal{G}, \mathbf{U}^{(k)}, \Lambda^{(k)}\}$ is

$$P(\Theta) = P(\mathcal{G}|\Lambda^{(1)}\ldots\Lambda^{(1)}) \prod_{k=1}^{d} P(U^{(k)}|\Lambda^{(k)}) P(\Lambda^{(k)})$$

.

CP Decomposition

Tensor CP decomposition represents a tensor with the sum of a few rank-1 tensors, namely,

$$\mathcal{X} = \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(d)} = [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!].$$

With the parameters $\Theta = \{\mathbf{A}^{(k)}, \Lambda, \tau\}$, the prior distribution satisfies $P(\Theta) = \prod_{k=1}^{d} P(\mathbf{A}^{(k)}|\Lambda)P(\Lambda)P(\tau)$ where

$$\begin{aligned} -\log P(\mathbf{A}^{(k)}|\Lambda) &= \frac{1}{2}\sum_{r=1}^{R}\left(|\mathbf{A}^{(k)}(:,r)|^2\lambda_r - \sum_{k=1}^{d} I_k \log \lambda_r\right), \\ -\log P(\Lambda) &= \sum_{r=1}^{R} -(a-1)\log\lambda_r + b\lambda_r. \end{aligned} \tag{2.18}$$

The negative log prior of τ is

$$-\log P(\tau) = -(c-1)\log\tau + d\tau. \tag{2.19}$$

In this work, instead of using λ_r and τ directly, we use the log Gamma distribution $\hat{\tau} = \log\tau$ and the inverse Gamma distribution $\hat{\lambda} = \lambda^{-1}$. Their prior distributions are

$$P(\hat{\tau}) = \frac{\exp(c\hat{\tau})\exp(-e^{\hat{\tau}})/d)}{d^c\mathcal{T}(c)} \tag{2.20}$$

and

$$P(\hat{\lambda}) = \frac{b^a}{\mathcal{T}(a)}(1/\hat{\lambda})^{a+1}\exp(-b/\hat{\lambda}) \tag{2.21}$$

respectively.

Combing equations (2.18), (2.20) and (2.21), we have the negative log prior

$$\begin{aligned} -\log P(\Theta) = \sum_{r=1}^{R} \Bigg(& \frac{1}{2} \sum_{k=1}^{d} \left(|\mathbf{A}^{(k)}(:,r)|^2 / \hat{\lambda}_r + I_k \log \hat{\lambda}_r \right) \\ & + (\alpha+1) \log \hat{\lambda}_r + \frac{\beta}{\hat{\lambda}_r} \Bigg) \\ & - c\hat{\tau} + \exp \hat{\tau}/d. \end{aligned} \tag{2.22}$$

2.4 Numerical Experiments

In this section, we present numerical experiments of our Bayesian tensor learning framework on both tensor completion and tensorized neural network tasks. We omit the numerical results on tensor regression which is easier than tensorized neural networks.

2.4.1 2-layer NN for Fashion-MNIST

We first consider the Fashion-MNIST dataset [29] by a two layer neural network. The first layer (FC1) is a 784×500 fully connected layer with a ReLU activation and the second layer (FC2) is a 500×10 fully connected layer with the softmax activation. We convert FC1 as a 8-th order tensor and FC2 as a 4-th order tensor for the tensor-train decomposition. For the Tucker decomposition, we convert FC1 as a 4-th order tensor and FC2 into a 3-th order tensor.

2.4.2 6-layer CNN for CIFAR-10

We build a 6-layer convolutional neural network (CNN) containing 4 convolution layers and 2 fully connected layers. Each convolution layer has a kernel size of 3×3 and padding of 1. The number of channels in each convolution layer is 128, 256, 256,

Table 2.1: Results of different networks on two datasets. LL: predictive log likelihood (the larger the better). TT: tensor train decomposition. Tucker: Tucker decomposition. BF: Bayesian low rank prior.

Dataset	Network	#Parameters (compression ratio)	MAP		Bayesian	
			LL	Accuracy	LL	Accuracy
Fashion-MNIST	NN	3.97×10^5 (1×)	-0.7118	88.91%	-0.6730	89.41%
	TT-NN	2.63×10^4 (15.1×)	-0.6687	87.07%	-0.6337	87.78%
	BF-TT-NN	4.02×10^3 (98.8×)	-0.3317	88.24%	-0.3254	88.64%
	Tucker-NN	2.57×10^5 (1.54×)	-1.1673	87.20%	-1.0984	87.53%
	BF-Tucker-NN	3.10×10^4 (12.8×)	-1.2948	87.18%	-0.4405	88.18%
CIFAR-10	CNN	9.91×10^6 (1×)	-0.5337	91.54%	-0.5370	91.53%
	TT-CNN	6.93×10^5 (14.3×)	-0.6077	89.00%	-0.5329	90.13%
	BF-TT-CNN	7.83×10^4 (127×)	-0.3936	86.68%	-0.3623	88.01%

256, respectively. The size of the first fully connected layer (FC1) is 512. A batch normalization layer and a ReLU activation layer is placed after each convolution and fully-connected layer. A maxpooling layer with kernel size of 2×2 is placed after the second and the fourth convolution layer.

2.4.3 Results

We use the ADAM method to minimize the negative posterior to get an initial point, then shrink the rank according to Section 2.2.4. Afterwards, we generate $T = 450$ samples.The accuracy of this model is evaluated using two criterions: the predictive log likelihood (LL) and the prediction accuracy. The results for different benchmarks using different tensor formats are shown in Table 2.1. We compare the proposed Bayesian

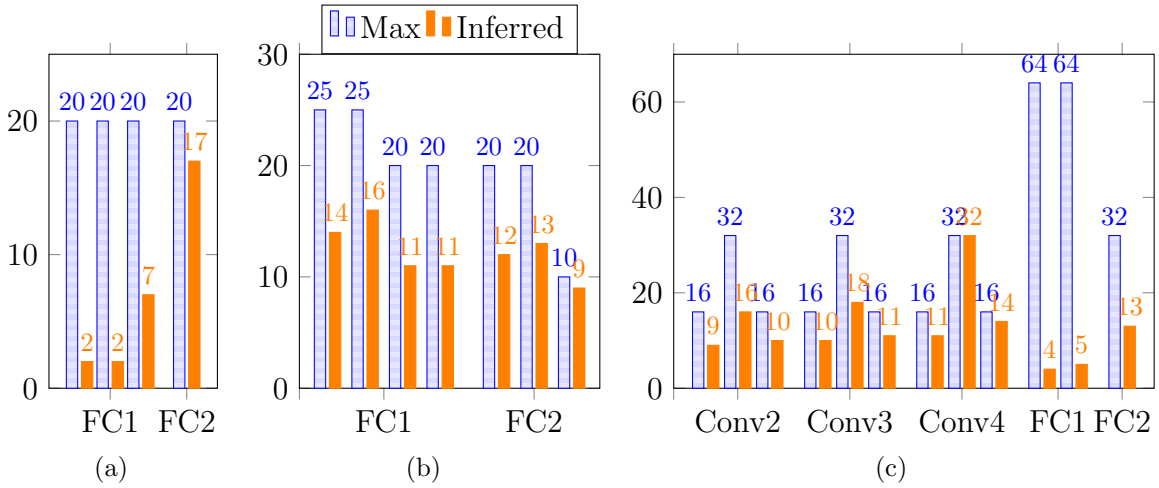


Figure 2.2: The inferred TT rank at different layers. (a): 2 TT-FC layers for Fashion-MNIST. (b): 2 Tucker-FC layers for Fashion-MNIST. (c): 4 TT-Conv and 2 TT-FC layers for CIFAR-10

learning with the optimization method that maximize a posterior (MAP) directly. It is shown that our tensor learning framework outperforms MAP in almost every case in terms of both the accuracy and the log likelihood (LL). The improvement in log likelihood indicates that our model can predict the uncertainty better than the MAP method. Besides, our method achieves a compression ratio of up to 98.8× in Fashion-MNIST and 127× in Cifar-10 in terms of the number of model parameters compared with the baseline network.

We also show the estimated tensor-train ranks of the estimated weight matrices and convolution filters in Figure 2.2. Clearly, our Bayesian tensor learning framework can perform model compression in the training process with automatic rank determination.

2.5 Conclusion

We presented applying Hamiltonian Monte Carlo (HMC) to train a Bayesian tensorized neural network representations. A low-rank inducing prior is proposed to reduce

the rank during training stage, which enables rank determination during training. Our method has demonstrated a significant compression ratio in the end-to-end training of tensorized neural networks, as well as better accuracy than the maximum-a-posterior training.

Chapter 3

Reducing Overparameterization: Quantized Tensorized Neural Network Training on FPGA

3.1 Introduction

Despite great success in vast applications, modernized neural networks are often over-parameterized, leading to prohibitive memory and computing costs in both training and inference. To overcome this issue, many neural network accelerators targeting efficient inference on FPGA and ASIC [30, 31, 32, 33, 34] have been proposed to improve the inference throughput and energy efficiency.

On the other hand, training is much more challenging, as it involves not only forward propagation but also backward propagation. As a result, most of the training tasks are still done on high-performance computing platforms such as clusters and cloud servers.

This work has been released as K. Zhang, C. Hawkins, X. Zhang, C. Hao, and Z. Zhang, *On-fpga training with ultra memory reduction: A low-precision tensor method*, in *ICLR Workshop on Hardware Aware Efficient Training*, 2021.

Besides the expensive hardware and economic cost, these training methods can cause a huge environmental impact as well. The study [35] shows that training some common natural language processing models on the cloud could emit 5× as much carbon dioxide compared with the lifetime emissions of an average American car. This has motivated us to train neural networks on resource-constrained platforms with much lower energy cost. Meanwhile, the increasing concerns about data privacy have become another driving force for training on edge devices [36]. Most of the above-mentioned post-training approaches [7, 8, 9, 10, 11, 12, 13, 14, 15] do not help reducing the training cost except quantization. By utilizing low-precision quntized arithmetic in optimization solvers [37, 38, 39, 40, 41, 42], one can reduce the cost per parameter during training, but the memory cost reduction is limited to *a single order* of magnitude even if the most recent ultra low-precision 4-bit training [40] can be employed. As a result, training neural networks on FPGA still remains an extremely challenging task.

Can we achieve orders-of-magnitude memory and variable reduction in training? If we can achieve this ambitious goal, then it becomes possible to train many large neural networks on FPGA and on other resource-constraint platforms. In this paper, we will show that it is possible to achieve this goal by exploiting tensor computation [43] and low-precision arithmetic together. Tensors are a high-dimensional extension of matrices, and tensor decomposition methods have outperformed many existing matrix compression algorithms by exploiting hidden low-rank structures in high dimensions. Recently, tensor decomposition has achieved orders-of-magnitude parameter reduction in post-training compression of deep neural networks [14, 15, 17, 18]. The methods have boosted the inference performance on various platforms [14, 44, 45, 46]. However, training a tensorized neural network from scratch is challenging. The training cost and model performance are controlled by tensor ranks, which are unknown *a priori*. Recent works [16, 47, 48] train a tensorized neural network with a fixed rank parameter, which often requires an

Table 3.1: Memory footprint of a d-dimension tensor of size $N \times N \ldots N$, its tensor train (TT) decomposition with rank R, and their quantized representation (LP). F is the word width of floating point (usually 32 or 64), and D is the word width of quantized representation (usually ≤ 8).

	Parameters	Bits
Tensor	N^d	FN^d
TT-Tensor	dNR^2	$FdNR^2$
LP-Tensor	N^d	DN^d
LP-TT-Tensor	dNR^2	$DdNR^2$

expensive manual search and massive training runs.

Paper Contributions. This paper presents, for the first time, an *end-to-end* neural network training framework on FPGA with *orders-of-magnitude memory reduction*. This work is based on two ideas: (1) a rank-adaptive tensorized model that automatically reduces training variables and model complexity in training; and (2) a low-precision tensor optimization solver that further reduces the hardware cost of each training variable. As shown Table 3.1, by combining these two methods, we can achieve higher memory reduction ratios than using any single method alone. With a largely reduced memory footprint in training, our method can be implemented on various edge devices with very limited on-chip memory and computation capacity, which is beyond the capability of existing full-size low-precision training. The specific contributions are summarized below:

- We propose a rank-adaptive tensorized model for end-to-end training. This model employs a Bayesian method for automatic tensor rank determination and acheives orders-of-magnitude model compression in the training process.

- We propose a low-precision framework to train the proposed tensorized neural network. This can further reduce the memory footprint. Together with the above rank-adaptive tensor compression, this method makes it possible to store all model parameters with limited on-chip memory in a training process.

- We design an embedded FPGA accelerator for the proposed low-precision tensorized end-to-end training framework. Our FPGA design achieves up to 128 FLOPs per clock cycle, and achieves 50× speedup compared with embedded CPU.
- We implement our algorithm with a two-layer neural network on a Xilinx MPSoC, which stores all model parameters on chip and achieves 82.08% testing accuracy on the Fashion MNIST dataset.

3.2 Tensor Train Neural Network

Training neural networks on edge devices is largely constrained by model size and computational cost. The FLOPS required is often so high that only expensive GPUs can finish training runs in reasonable time. Low-rank tensor compression is a promising solution to reduce both computation and memory cost [15, 16, 17].

To compress the layers of pre-trained models, different decompositions have been studied [49, 50, 51, 52]. Among these methods, tensor train decompostion often yields the highest compression ratio with little accuracy loss. Therefore, in this work we focus on the tensor-train decomposition [24].

A fully-connected layer takes the form of $\mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ where $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias, and $\mathbf{x}$ is the input vector. The majority of the parameters in a layer are in the weight matrix $\mathbf{W}$. To compress the weights of a fully-connected layer, we apply the Tensor-Train Matrix format as shown in Definition 3.1.

In the fully-connected layer of a neural network, the weight matrix $\mathbf{W}$ contains many parameters. To achieve high compression ratios we reshape it into a high dimensional tensor $\mathcal{W}$ with the same elements as $\mathbf{W}$, and use the tensor-train decomposition to it.

Definition 3.1 *Let* $\mathbf{W} \in \mathbb{R}^{I \times J}$ *be a matrix and let* $i = \prod_{n=1}^{d} i_n, j = \prod_{n=1}^{d} j_n$ *be a*

factorization of its dimensions. To apply the tensor-train matrix format we reshape $\mathbf{W}$ *into a tensor* $\mathcal{W}$ *with dimensions* $i_1 \times \cdots \times i_d \times j_1 \times \cdots \times j_d$*. The explicit reshape scheme is given in [16]. The* **tensor-train matrix** *(TTM) factorization applies tensor train decomposition to* $\mathcal{W}$ *and expresses it as a series of matrix products.*

$$\mathcal{W}_{i_1,\ldots,i_d,j_1,\ldots,j_d} = \mathcal{G}^{(1)}_{:,i_1,j_1,:}\mathcal{G}^{(2)}_{:,i_2,j_2,:}\ldots\mathcal{G}^{(d)}_{:,i_d,j_d,:}$$

Each **TT-core** $\mathcal{G}^{(d)} \in \mathbb{R}^{R_{n-1}\times I_n \times J_n \times R_n}$, or **tensor factor**, is an order 4 tensor. The tuple $(R_0, R_1, R_2, \ldots, R_d)$ is the **TT-rank** and as before with $R_0 = R_d = 1$. The Tensor-Train Matrix factorization requires $\sum_n R_{n-1} I_n J_n R_n$ parameters, which is usually much smaller than the original matrix with $\prod_n I_n J_n$ number of parameters.

Low-rank optimization and Bayesian inference are the two main approaches used for rank determination in tensor completion. The first approach relies on the generalization of the matrix nuclear norm [53] to tensors. Popular approaches achieve rank reduction by relying on tensor unfolding operators at the cost of high computational expense for high-order tensors. The second approach, Bayesian inference, utilizes low-rank priors to deduce the tensor rank in CP or Tucker tensor completion [25]. The tensor-train decomposition differs from CP or Tucker in that the the ranks of different tensor factors may couple with each other. The observed data is a linear mapping of a tensor in tensor completion and it is a nonlinear mapping in neural networks. This nonlinearity prevents us from directly applying previous work on tensor rank determination to tensorized neural network.

3.3 Rank-Adaptive Tensorized Neural Network

Most existing methods to produce tensorized neural networks require training an uncompressed model first, which is computationally expensive. In this section, we propose to train a low-rank tensorized neural network from scratch by directly updating the unknown tensor cores via stochastic gradient-descent optimization. This approach does not form the large uncompressed weight matrices, therefore it can greatly reduce the memory footprint and training cost. To reduce tensor ranks, we start with a model with higher tensor ranks, and apply a low-rank regularizer to shrink the tensor rank during the training process. This approach leads to a better trade-off between accuracy and cost (in terms of memory and computing) than existing fixed-rank training [16, 47, 48] that require combinatoral rank search and multiple training runs.

3.3.1 Hierarchical Bayes Model

Determining a proper tensor rank is NP hard. Failure to get an appropriate rank estimation may cause high training cost or low accuracy. In this work, instead of setting the rank as a prior, we use a Hierarchical Bayes model to infer the optimal rank in the training process. The hierarchical structure is shown in Figure 3.1.

We introduce a set of additional parameters $\{\boldsymbol{\lambda}^{(n)}\}$ to determine the actual Tensor-Train ranks. Our goal is to determine tensor factors with a low TT-rank so we select a prior density that specifies a prior belief that the TT-rank is low. Specifically, $\lambda_i^{(k)}$ will be larger if the values in the tensor slice associated with it is larger. It will in turn influence the regularizer, and penalties more if $\lambda_i^{(k)}$ is smaller. At the end of training process, some of $\lambda_k^{(n)}$ will be small, and the whole slice of $\mathcal{G}^{(n)}$ will be close to zero, leading to a rank reduction in the n-th mode.

Additionally, we place a Log-Uniform prior on the hyperparameters $\boldsymbol{\lambda^{(n)}}, 1 \leq n \leq$

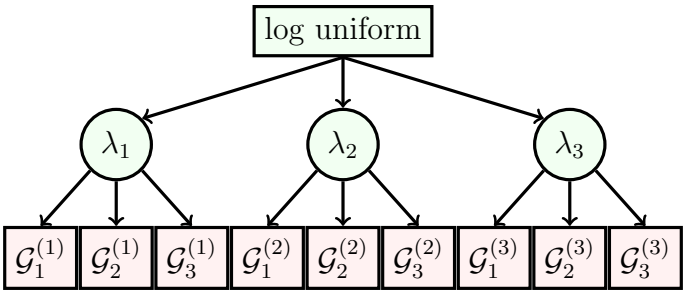


Figure 3.1: Hierarchical Bayes model for low rank TT neural network.

$d-1$:

$$p(\boldsymbol{\lambda}^{(\boldsymbol{n})}) = \prod_{i=1}^{R} p(\lambda_i^{(n)}), \quad p(\lambda_i^{(n)}) \sim {\lambda^{(n)}}^{-1/2}. \tag{3.1}$$

We make two observations about this choice of hyper-prior. Firstly, the prior density, and therefore the entire Bayesian model, does not contain any manually tuned hyper-parameters. This enables us to perform one-shot tensorized training on edge devices without multiple hyperparamter tuning runs. Secondly, the prior density enforces sparsity in the vector $\boldsymbol{\lambda}$ and therefore induces structural rank-sparsity on the low-rank tensor factors.

Fully Bayesian training to fit the model parameters is prohibitively expensive. Instead we convert the low-rank prior into the form of a regularizer by taking the negative log, which leads to (3.3).

3.3.2 Objective Function

To train a neural network for classification, the typical objective to minimize is the cross entropy loss between the predicted label $f(x)$ and the ground-truth label y:

$$\frac{1}{m}\sum_{i=1}^{m} \mathrm{CE}(f(x_i), y_i) \tag{3.2}$$

Algorithm 2: Low-rank tensorized neural network training algorithm.

Input : dataset $(x_i, y_i) \in \mathcal{D}$
ndomly uniformly initialize weights. **while** *Not converge* **do**
  Randomly draw a minibatch $\mathcal{B} \subset \mathcal{D}$, calculate the target function (3.2) + (3.3) and its gradient with respect to $\mathcal{G}$. Update $\mathcal{G}_i^{(l)}$ along (stochastic) gradient direction using ADAM. Update $\boldsymbol{\lambda}$ with (3.4)
end
Output: $\mathcal{G}_i^{(l)}$

where m is the size of training set and CE stands for cross entropy loss. In some cases an additional nonnegative, convex function, eg. L-2 norm function, is added to the objective function as a *regularizer* to avoid overfitting or to make the optimization landscape smoother. In our work, we add a regularizer to shrink the tensor rank:

$$\sum_{1\le n\le d-1}\sum_{k}\frac{\|g^{(n)}_{:,:,:,k}\|_F^2}{\lambda_k^{(n)}}+\sum_{k}\frac{1+r_{n-1}i_nj_n}{2}\log(\lambda_k^{(n)}) \tag{3.3}$$

Note that the second term does not depend on $\mathcal{G}$, so it is not included when computing the gradient. The algorithm to minimize the target function is shown in Algorithm 2, and it is explained in the following sections.

3.3.3 Update rule

In optimization process, we updates the tensor factors using Adam algorithm [54]. On the other hand, we update the rank parameters $\boldsymbol{\lambda}$ by minimizing equation (3.3) analytically with respect to $\boldsymbol{\lambda}$:

$$\lambda_k^{(n)}=\frac{2}{1+r_{n-1}i_nj_n}\sum_{r,i,j}{g^{(n)}_{r,i,j,k}}^2 \tag{3.4}$$

We alternate between updates of tensor factors $\mathcal{G}$ and updates of the rank parameters $\boldsymbol{\lambda}$ to minimize the objective function.

3.4 Low-Precision Tensorized Training

3.4.1 Training Quantized Weights

The update rule of Stochastic gradient descent (SGD) is

$$\mathcal{G}^{(t+1)} = \mathcal{G}^{(t)} - \eta_t \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\mathcal{G}} \ell(f(\boldsymbol{x}_i; \mathcal{G}), y_i)$$

where $\mathcal{B} \subset \mathcal{D}$ is the minibatch used in this step, $\eta > 0$ is the stepsize, ℓ is the loss function, eg. mean square error (MSE) or cross entropy loss, and $f(\cdot; \mathcal{G})$ represents the neural network given the trainable tensor factors $\{\mathcal{G}\}$. In a quantized neural network each value in $\mathcal{G}$ is chosen from a discrete set $\{k\Delta | -B \leq k\Delta < B, k \in \mathbb{N}\}$, where $B = 2^{b-1}\delta$ is the bound on the quantized values, b is the number of bits to represent a value, and Δ is the quantization precision.

When the stepsize η is small, which is often the case when getting close to the local minima, the update in a single step can be smaller than the quantization precision Δ, which prevents the empirical loss from further decreasing. In order to avoid this problem, two methods have been proposed: stochastic rounding [39] and Binary connect (BC) [41]. Their convergence has been analysed in [55] which demonstrates that BC has faster convergence speed and better stability than stochastic rounding.

In our work, we use the BinaryConnect algorithm to train a tensorized neural network. The BinaryConnect algorithm keeps a high precision copy of all the low precision parameters in a buffer. In each iteration, the gradients are accumulated in the buffer and the low precision parameters are updated by quantizing the buffer. The update process is

$$\hat{\mathcal{G}}^{(t+1)} = \tilde{\mathcal{G}}^{(t)} - \eta_t \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\mathcal{G}} \ell(f(\boldsymbol{x}_i; \mathcal{G}), y_i) \tag{3.5}$$

$$\mathcal{G}^{(t+1)} = Q(\tilde{\mathcal{G}}^{(t+1)}) \tag{3.6}$$

Where Q is the quantization function.

$$Q(x) = \arg\min_y |y - x|, y \in \{k\Delta | -B \leq k\Delta \leq B, k \in \mathbb{N}\}$$

The quantization precision is chosen to avoid clipping:

$$B = \max |\mathcal{G}_i^{(l)}|, \qquad \Delta_i^{(l)} = \frac{\max \mathcal{G}_i^{(l)}}{2^b}$$

Notice that in such neural networks, the biases are usually not quantized, as this often greatly hurt the accuracy, while saving only a small amount of computing and storage resources.

3.4.2 Straight Through Estimator

Quantization functions are not differentiable and backpropagation can't be used directly to train a neural network with quantized activations. In order to solve this problem, the straight through estimator (STE) [56] has been proposed as an approximation to the gradient of the quantization function. The idea of STE is to use the gradient of a smooth function is the backpropagation. In [56], the gradient of a 0-1 activation function is approximated with the gradient of sigmoid function. In [42], it is suggested to use $1_{|r|<1}$ as the approximated gradient of quantization function. This method is often called saturated STE. A detailed analysis of different types of STE is presented in [57]. It is proven

that both ReLU and clipped ReLU as STE leads to guaranteed convergence, while simple passthrough can result in divergence in some cases. In this paper, we use clipped ReLU and leaky clipped ReLU function as STE. Considering a ReLU-like quantized activation function

$$\sigma(x) = \arg\min_{y \in \mathcal{Y}} |y - x|, \mathcal{Y} = \{0 \leq k\Delta \leq B\} \tag{3.7}$$

the backpropagation rule can be written as

$$\frac{\partial}{\partial y} = \mathbf{1}(0 \leq x \leq B)\frac{\partial}{\partial y} \tag{3.8}$$

where $\mathbf{1}(\cdot)$ is the indicator function.

Table 3.2: Fixed point expression used.

Values	bits used
Weights	4
Activation	8
Gradients	16

We summatize our fixed point expression in Table 3.2. We use 4 bits to represent the weights, 8 bits for activation, and 16 bits for gradients of both activation and model parameters (tensor factors). The bias has the same representation as activation (8 bits). In computing the gradients of factors, gradients in a single minibatch are shifted before accumulated to 16 bits to avoid overflow. Since PE are shared between forward and backward propagation, they are designed to handle 16 bits data and 4 bits weights. In forward propagation, only the 8 LSB in data are used, and in backward propagation, all 16 bits are used. This allows the model trained with our method portable to devices with only 8 bits by 4 bits for inference.

3.4.3 Automatic scale selection

When using fixed point representation, the scaling factor of each value needs to be carefully designed to avoid overflow or large quantization error. Besides, if the scaling factors differ by a factor of a power of 2, then a simple bit shift is needed, which takes almost no hardware resource or time.

In the training process of neural network, the scale of activation an gradients can vary by several orders of magnitude during the training process, which makes it impossible to use a fixed scaling factor throughout the training process. In order to deal with this problem, we used a variable scaling factor, and introduced a mechanism to determine the scaling factor on the fly. The scaling factors of all values are enforced to be a power of 2 so that data conversion requires a simple bit shift. We allow a different scaling factor for each activation, gradient or intermediate result, while it is shared between different samples or channels. The scaling factors of the weights are fixed, and with 4 bits the available range is $[-1, 0.875]$. To avoid overflow and make sure the values approximately zero mean, weights are clipped to the range $[-0.91, 0.91]$ after each iteration. To determine the scaling factor of the activation and gradients, we keep track of the mean of absolute value during training process, and enforce it to lay in the range between 0.1 and 0.3 by dynamically adjusting the scaling factor. This allows a small margin to avoid overflow, while making the most use of the bits to reduce quantization error.

3.5 FPGA implementation

3.5.1 Overall design

In this section we introduce the FPGA implementation for our proposed on-device rank-adaptive tensorized training for neural networks. The overall FPGA design is shown

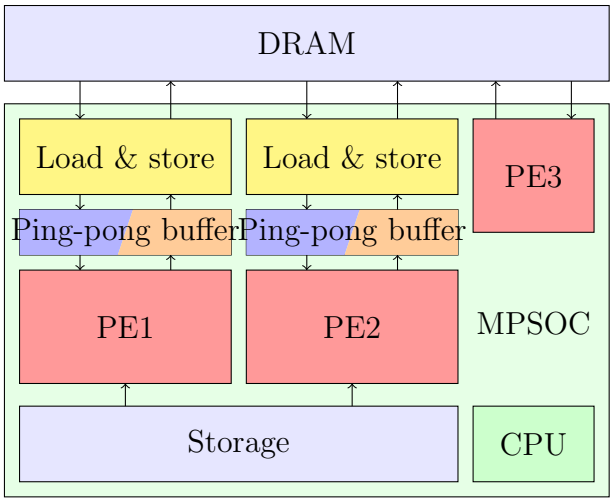
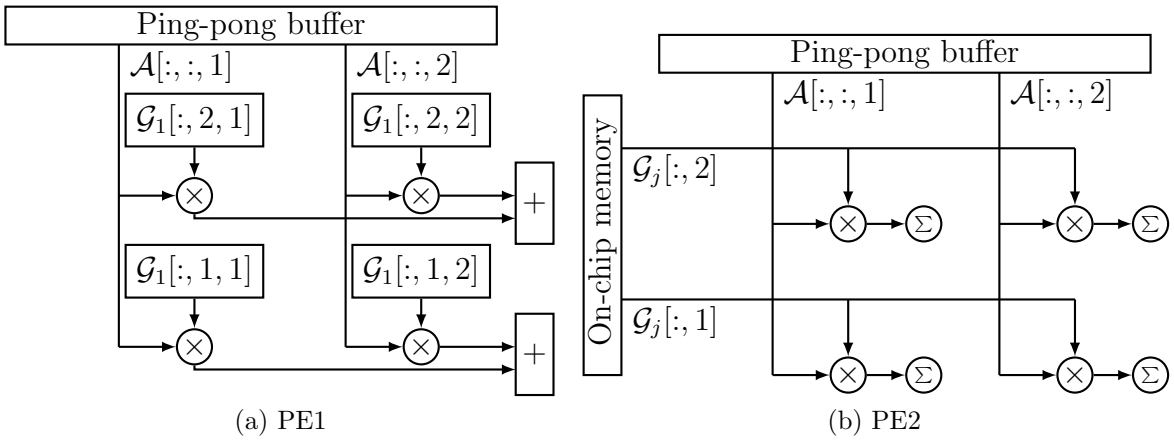


Figure 3.2: Overall view of hardward design

in Figure 3.2. During training, the training samples, activations, and gradients are stored in the off-chip DRAM. Thanks to our low-rank tensorization, all the model parameters can be stored in the on-chip BRAM. The overall training involves three steps: forward propagation, backward propagation, and model parameter update. The forward and backward propagation are executed on FPGA programmable logic; updates to the tensor factors $\mathcal{G}$ and rank parameters $\boldsymbol{\lambda}$ are executed on the embedded ARM core, which usually take less than 1% of total computing cost. We design three processing elements (PEs) to compute the forward and backward propagation: PE1 and PE2 are used in forward propagation, while PE1, PE2, and PE3 are used in backward propagation. Thus, PE1 and PE2 are shared by forward and backward propagation to reduce resource usage. PE2 will execute tensor contraction along the last dimension, while PE1 will handle contraction along other dimensions. The data, activations and gradients involved in the computation of PE1 and PE2 are cached by ping-pong buffers. Because PE3 only performs outer product operation, which is memory bounded and cannot benefic from a buffer, it reads and writes the activations and gradients from the DRAM directly.



(a) PE1 (b) PE2

Figure 3.3: Hardware architecture of PE1 and PE2

3.5.2 PE design

We designed two PE to handle tensor contraction. The reason for that is there are two kinds of tensor contraction operations, depending on whether the dimension to contract is the last dimension in a tensor or not. Data locality in these two cases are significantly different. In order to deal with that, we designed two kinds of PE to handle these two different kinds of processes.

The first PE (PE1) does tensor contraction along the last dimension. The design of PE1 is shown in Figure 3.3a. It consists a two-dimension multiplier array. Each column of the multipliers share the same data/activation/gradient from the Ping-pong buffer, and an accumulator is placed at each row of the multiplier array. Each multiplier is equipped with a small block memory and the weights are loaded the block memory before performing a tensor operation.

The computation can be written as

$$abc \times bdc \to ad$$

The dataflow of this PE is shown in algorithm 3. In the partial tensor contraction (line

Algorithm 3: Process of PE1.

Input : operand 1 $\mathcal{A}$ in off-chip memory, operand 2 $\mathcal{G}$ in on-chip memory.
foreach *i from 1 to a step 8* **do**
  Pipeline:
  Load $\mathcal{A}[i:i+7,:,:]$ to ping-pong buffer
  Partial tensor contraction to $\mathcal{A}[i:i+7,:,:]$ and $\mathcal{G}$ to get $\mathcal{B}[i:i+7,:]$, store to another ping-pong buffer.
  Store $\mathcal{B}[i:i+7,:]$ to off-chip memory.
end
Output: Tensor $\mathcal{B}$ in off-chip memory.

5), we parallelize the computing along the last dimension (c) by a factor of 16, and parallelize along the first dimension (a) by a factor of 8. This requires 128 scalars from the first operand (data) and 16 continuous (and aligned) data from the second operand (weight) per clock cycle during computing. To simplify the design, we enforce C to be a multiplier of 16, which is equivalent to enforcing the last dimension of both input and output tensor, m_3 and n_3, to be a multiplier of 16, which brings an additional benefit that d is always a multiplier of 16.

The second PE (PE2) performs tensor contraction along dimensions other than the last dimension. The design of PE2 is shown in Figure 3.3b. It consists a two-dimension multiplier-and-accumulator array. Each column of the multipliers share the same data/activation/gradie from the Ping-pong buffer, and each row of the multipliers share the same tensor weight stored in the on-chip memory. Each multiplier is equipped with a small block memory to store the accumulated result.

The computation can be written as

$$abc \times bd \rightarrow adc$$

and shape of operands in each step are shown in Table 3.6.

The dataflow of this PE is shown in algorithm 4. In the partial tensor contraction

Algorithm 4: Process of PE2.

Input : operand 1 $\mathcal{A}$ in off-chip memory, operand 2 $\mathcal{G}$ in on-chip memory.
foreach *j from 1 to c step 16* **do**
  foreach *i from 1 to a step 1* **do**
    Pipeline:
    Load $\mathcal{A}[i, :, j : j + 15]$ to ping-pong buffer
    Partial tensor contraction to $\mathcal{A}[i, :, j : j + 15]$ and $\mathcal{G}$ to get $\mathcal{B}[i, :, j : j + 15]$
    store to another ping-pong buffer.
    Store $\mathcal{B}[i, :, j : j + 15]$ to off-chip memory.
  end
end
Output: Tensor $\mathcal{B}$ in off-chip memory.

(line 6), we parallelize the computing along the last dimension (c) of the first operand by a factor of 16, and parallelize along the last dimension of the second operand (d) by a factor of 8. Again, we enforce c to be a multiple of 16, which has been covered in the condition above. This requires 128 scalars from the first operand (data) and 8 continuous (and aligned) data from the second operand (weight) per clock cycle during computing.

The interface of both data and weight has a width of 16. This is enough for weight, but not enough for data, which is used more heavily during computation. To deal with this limitation we split the data into parts. In the first PE, data is split by dimension a. In the second PE, data is split by dimensions a and c. We split the computation into three steps: loading data from DRAM to the on-chip buffer, performing the multiply-and-accumulate, and storing the result back to DRAM. Making use of the ping-pong buffer, the second step, multiply-and-accumulate, can be executed in parallel with other steps.

We introduce the third PE to perform outer product. The throughput of this step is bounded by memory bandwidth of storing. Due to this limitation, this PE consists only a one-dimensional multiplier array so that the throughput of computation matches the throughput of memory. The computing is parallarized along the last dimension m_3

only by a factor of 16. Elements of the second operand is cached, while the first operand is read from DRAM directly and write through. The detailed operation of each PE is deferred to Section 3.8.

3.5.3 Memory management

In many other neural network accelerators, both data and model parameters are placed on off-chip DRAM due to limited on chip memory, and are loaded to on-chip memory only when needed. This incurs overhead in latency and power. In our design, thanks to the reduction in the number of parameters due to tensorization and quantization, it is possible to store all the tensor factors (and bias) on-chip through the entire training process. This reduces the overhead of data movement between on-chip memory and off-chip DRAM memory. In summary, we should expect the throughput to be close to 128 Flops/cycle.

3.6 Experiments and results

3.6.1 MLP

To test the performance of our accelerator, we implemented it for a 2-layer tensorized neural network for MNIST-like dataset. We used C++ to implement fixed point tensor contraction and used Pytorch to implement high level methods (ADAM, rank parameters update). In order to fit the requirement on the shape of tensors, we zero pad the input to 28×32 and decompose it to $7 \times 4 \times 2 \times 16$. There are 512 neurons in the hidden layer decomposed into $4 \times 4 \times 2 \times 16$ for the first layer, and 32×16 for the second layer. The output is decomposed into 1×16. We trained this model for FashionMNIST dataset [58], which has the same shape and size as MNIST dataset but is more complicated

Table 3.3: Fashion MNIST training result

Method	Training accuracy	Testing accuracy	Model parameters	Memory in bits
Vanella	95.75%	89.27%	4.67×10^5	1.49×10^7
Floating, w/o prior	92.54%	88.03%	1.48×10^4	4.74×10^5
Fixed, w/o prior	88.31%	86.67%	1.48×10^4	6.13×10^4
Floating, w/ prior	90.17%	87.88%	1.08×10^4	3.46×10^5
Fixed, w/ prior	85.45%	84.86%	1.22×10^4	5.11×10^4

and can better We train the model for 30 epochs and compare both standard floating point computation in Pytorch (Floating) and our simulator (Fixed), and training with or without the low rank prior. We report the epoch with highest testing accuracy, and show our result in Table 3.3. We also listed the memory footprint of the model parameters (tensor factors) achieved after training.

3.6.2 Implementation on FPGA

We implemented the forward and backward propagation process on a Avnet Ultra96-V2 board. This board is equipped with a Xilinx Zynq UltraScale+ XCZU3EG MPSoC and 2GB off-chip memory. The resource utilization is listed in Table 3.5. For larger neural networks, utilization of LUTs and DSPs will not increase as PEs can be reused for computing across layers, while only the utilization of BRAM will increase, as weights are stored on chip during training. A maximum of 114MHz clock rate can be achieved.

We compared the time and memory usage to tensorized neural network training between our implementation on FPGA and that on a embedded computer. The embedded computer we use is an Raspberry Pi 3B with Quad Core 1.2GHz ARM processor. We used Pytorch and Tensorly module to implement training algorithm on it. For FPGA, we set the clock rate to 100MHz. Only the time on forward and backward propagation is included, as the rest part (optimizer) is the same on both devices. The memory usage

Table 3.4: Time and memory use comparison between floating point training on embedded computer (RPi) and fixed point training on FPGA (FPGA)

	Time (s/batch)	Memory (MB)
RPi	5.34	1.49
FPGA	0.09	20.06

Table 3.5: Resource utilization of 2-layer tensorized neural network

resource	used	available	utilization
LUT	56131	70560	79.55%
FF	30155	141120	21.37%
DSP	278	360	77.22%
BRAM	77	432	17.82%

shown in this table excludes the model parameters and training data. For embedded computer, we measured the memory usage by taking the maximum one, minus the memory usage after loading data and initializing the model but before training.

3.7 Conclusion

In this work, we proposed a new algorithm to train quantized tensorized neural networks. By training end-to-end compressed neural networks, our approach produce a compressed model from scratch while saving hardware resources during the training phase. Our algorithm uses Bayes rule to determine the rank from the training data, and achieved up to 335× reduction in memory cost compared to the base model with only a slight loss in accuracy.

3.8 Detailed operations

3.8.1 Forward propagation

The forward and backward propagation of neural network involves tensor contraction. Here we use a tensor train with three factors as an example, denote the input dimensions as m_1, m_2, m_3, output dimensions as n_1, n_2, n_3, and the rank as r_1, r_2. the forward propagation involves the following computation in Einstein summation convention:

$$m_1 m_2 m_3 \times m_3 r_2 n_3 \rightarrow m_1 m_2 r_2 n_3 \tag{3.9}$$

$$m_1 m_2 r_2 n_3 \times m_2 r_2 r_1 n_2 \rightarrow m_1 r_1 n_2 n_3 \tag{3.10}$$

$$m_1 r_1 n_2 n_3 \times m_1 r_1 n_1 \rightarrow n_1 n_2 n_3 \tag{3.11}$$

In these expressions, the first operand is the data (input or intermediate results), and the second operand is the tensor factors of the weights.

3.8.2 Backward propagation

In back propagation, there are two tasks:

- To compute the gradients with respect to the inputs of the layer.
- To compute the gradients with respect to the model parameters (tensor factors) of the layer.

To compute the gradients with respect to the inputs, the computation in Einstein

summation convention is shown below:

$$n_1 n_2 n_3 \times n_1 m_1 r_1 \rightarrow m_1 r_1 n_2 n_3 \tag{3.12}$$

$$m_1 r_1 n_2 n_3 \times r_1 n_2 m_2 r_2 \rightarrow m_1 m_2 r_2 n_3 \tag{3.13}$$

$$m_1 m_2 r_2 n_3 \times r_2 m_3 n_3 \rightarrow m_1 m_2 m_3 \tag{3.14}$$

The first equation is to compute the gradients directly. The second is to compute the gradients with respect to full weights and then accumulate them and compute the gradients with respect to the factors. The former method is more efficient if batch size is small and the compressed model is small, while the latter is more efficient if it is the opposite. In our work, we are starting with a model with large rank (larger model), the latter method is more efficient. In this work, we are implementing the second method. The first step, computing the gradient with respect to the full weight matrix, requires a simple outer product:

$$m_1 m_2 m_3 \times n_1 n_2 n_3 \rightarrow n_1 m_1 n_2 m_2 n_3 m_3$$

The first operand of this PE is the input to this layer during forward propagation, and the second operand is the gradient of the output. After the gradient has been accumulated in a batch, the gradient with respect to the factors can be computed by contracting the

Eq.	PE	a	b	c	d
(3.9)	PE1	m_1m_2	1	m_3	r_2n_3
(3.10)	PE2	m_1	m_2r_2	n_3	r_1n_2
(3.11)	PE2	1	m_1r_1	n_2n_3	n_1
(3.12)	PE2	1	n_1	n_2n_3	m_1r_1
(3.13)	PE2	m_1	r_1n_2	n_3	m_2r_2
(3.14)	PE1	m_1m_2	r_2	n_3	m_3
(3.15)	PE1	$n_1m_1n_2m_2$	1	n_3m_3	r_2
(3.16)	PE1	n_1m_1	1	$n_2m_2r_2$	r_1
(3.17)	PE2	1	n_1m_1	$n_2m_2r_2$	r_1
(3.18)	PE2	1	n_1m_1	$n_2m_2n_3m_3$	r_1
(3.19)	PE2	1	$r_1n_2m_2$	n_3m_3	r_2

Table 3.6: PE and operand of each expression.

gradient of full weight with the tensor factors:

$$n_1m_1n_2m_2n_3m_3 \times r_2n_3m_3 \to n_1m_1n_2m_2r_2 \tag{3.15}$$

$$n_1m_1n_2m_2r_2 \times r_1n_2m_2r_2 \to n_1m_1r_1 \tag{3.16}$$

$$n_1m_1n_2m_2r_2 \times n_1m_1r_1 \to r_1n_2m_2r_2 \tag{3.17}$$

$$n_1m_1n_2m_2n_3m_3 \times n_1m_1r_1 \to r_1n_2m_2n_3m_3 \tag{3.18}$$

$$r_1n_2m_2n_3m_3 \times r_1n_2m_2r_2 \to r_2n_2m_3 \tag{3.19}$$

Note that the result of the first expression is shared to get the gradient of the first and second tensor factor.

To compute the gradient with respect to the tensor factors, as in Equation (3.15)-(3.19), and the first and second PE can be reused here. This puts additional requirement on the shape and rank of the tensor factors. A sufficient condition is that all the ranks are a multiple of 16. Since we are using models with rank determination, the rank of the final model will a rank smaller than this pre-specified maximum rank. The final rank is not necessarily a multiplier of 16.

Chapter 4

Infinite Overparameterization: NTK of BinaryWeight Neural Networks

4.1 Introduction

Traditional statistical learning techniques (e.g., VC-dimension [59]) rely on the number of parameters to study the generalization ability of a machine learning method. Because of overparameterization, the traditional statistical learning techniques based on uniform convergence do not satisfactorily explain the generalization ability of neural networks. Furthermore, Zhang et al. [60] showed that neural networks can perfectly fit the training data even if the labels are random, yet it generalized well when the data are not random. This seems to suggest that the model capacity of a neural network depends on not only the model, but also the dataset. Recent studies [61] managed to understand the empirical performance in a number of different aspects, including modeling stochastic gradient (SGD) with stochastic differential equation (SDE) [62], studying the geometric

This work has been released as K. Zhang, M. Yin, and Y.-X. Wang, *Why quantization improves generalization: Ntk of binary weight neural networks*, *arXiv preprint arXiv:2206.05916* (2022).

structure of loss surface [63], and overparameterization – a particular asymptotic behavior when the number of parameters of the neural network tends to infinity [64, 65, 66, 67]. Recently, it was proven that the training process of neural network in the overparameterized regime corresponds to kernel regression with Neural Tangent Kernel (NTK) [68]. A line of work [69, 70, 71, 72] further studied Mercer's decomposition of NTK and proved that it is similar to a Laplacian kernel in terms of the eigenvalues.

It has been found that by quantizing the parameters in a neural network, the memory footprint and computing cost can be greatly decreased with little to no loss in accuracy [39]. Furthermore, Hubara et al. [73], Courbariaux et al. [41] argued that quantization serves as an implicit regularizer and thus should increase the generalizability of neural network comparing to its full precision version. However, there is no formal theoretical investigation of this statement to the best of our knowledge.

In this paper, we propose modeling a two-layer binary weight neural network using a model with continuous parameters. Specifically, we assume the binary weights are drawn from the Bernoulli distribution where the parameters of the distribution (or the mean of the weights) are trainable parameters. We propose a *quasi neural network*, which has the same structure as a vanilla neural network but has a different activation function, and prove one can analytically approximate the expectation of output of this binary weight neural network with this quasi neural network. Using this model, our main contributions are as follows:

- Under the overparameterized regime, we prove that the gradient computed from BinaryConnect algorithm is approximately an unbiased estimator of the gradient of the quasi neural network, hence such a quasi neural network can model the training dynamic of binary weight neural network.

- We study the NTK of two-layer binary weight neural networks by studying the

"quasi neural network", and show that the eigenvalue of this kernel decays at an exponential rate, in contrast with the polynomial rate in a ReLU neural network [72, 71]. We reveal the similarity between the Reproducing kernel Hilbert space (RKHS) of this kernel with Gaussian kernel, and it is a strict subset of function as the RKHS of NTK in a ReLU neural network. This indicates that the model capacity of binary weight neural network is smaller than that with real weights, and explains higher training error and lower generalization gap observed empirically.

4.2 Related work

Quantized neural networks. There is a large body of work that focuses on training neural networks with quantized weights [74, 37, 39, 75, 76], including considering radically quantizing the weights to binary [42, 77] or ternary [78] values, which often comes at a mild cost on the model's predictive accuracy. Despite all these empirical works, the theoretical analysis of quantized neural networks and their convergence is not well studied. Many researchers believed that quantization adds noise to the model, which serves as an implicit regularizer and makes neural networks generalize better [73, 41], but this statement is instinctive and has never been formally proved to the best of our knowledge. One may argue that binary weight neural networks have a smaller parameter space than its real weight counterpart, yet Ding et al. [79] showed that a quantized ReLU neural network with enough parameters can approximate any ReLU neural network with arbitrary precision. These seemingly controversy results motivate us to find another way to explain the stronger generalization ability that is observed empirically.

Theory of deep learning and NTK. A notable recent technique in developing the theory of neural networks is the neural tangent kernel (NTK) [68]. It draws the connection

between an over-parameterized neural network and the kernel learning. This makes it possible to study the generalization of overparameterized neural network using more mature theoretical tools from kernel learning [80, 81].

The expressive power of kernel learning is determined by the RKHS of the kernel. Many researches have been done to identify the RKHS. Bach [69], Bietti et al. [70] studied the spectral properties of NTK of a two-layer neural network without bias. Geifman et al. [71] further studied the NTK with bias and showed that the RKHS of two layer neural networks contains the same set of functions as RKHS of the Laplacian kernel. Chen et al. [72] expanded this result to arbitrary layer neural networks and showed that RKHS of arbitrary layer neural network is equivalent to Laplacian kernel. All these works are based on neural networks with real weights, and to the best of our knowledge, we are the first to study the NTK and generalization of binary weight neural networks.

4.3 Preliminary

4.3.1 Neural tangent kernel

It has been found that an overparameterized neural network has many local minima. Furthermore, most of the local minima are almost as good as the global minima [82]. As a result, in the training process, the model parameters often do not need to move far away from the initialization point before reaching a local minimum [83, 84, 85]. This phenomenon is also known as lazy training [86]. This allows one to approximate a neural network with a model that is nonlinear in its input and linear in its parameters. Using the connection between feature map and kernel learning, the optimization problem reduces to kernel optimization problem. More detailed explanation can be found below:

Denote Θ as the collection of all the parameters in a neural network f_Θ before an

iteration, and Θ^+ as the parameters after this iteration. Let in denote fixed distribution in the input space. In this paper, it is a discrete distribution induced by the training dataset. Using Taylor expansion, for any testing data $\tilde{x}$, let the stepsize be η, the first-order update rule of gradient descent can be written as ($l_{\text{loss}}(\cdot)$ be the differentiable loss function and the label is omitted)

$$\begin{aligned}
\Theta^+ - \Theta &= \eta \mathbb{E}_{x\sim in}\left[\nabla_\Theta \text{loss}(f_\Theta(x))\right] \\
&= \eta \mathbb{E}_{x\sim in}\left[\nabla_\Theta f_\Theta(x)\ \text{loss}'(f_\Theta(x))\right] \\
f_{\Theta^+}(x') - f_\Theta(x') &= \eta \nabla_\Theta f_\Theta(x') \cdot \mathbb{E}_{x\sim in}\left[\nabla_\Theta f_\Theta(x)\ \text{loss}'(f_\Theta(x))\right] \\
&= \eta \mathbb{E}_{x\sim in}\left[\nabla_\Theta f_\Theta(x') \cdot \nabla_\Theta f_\Theta(x)\ \text{loss}'(f_\Theta(x))\right] \\
&:= \eta \mathbb{E}_{x\sim in}\left[\mathcal{K}(x,x')\ \text{loss}'(f_\Theta(x))\right].
\end{aligned}$$

This indicates that the learning dynamics of overparameterized neural network is approximating the kernel learning with the limiting kernel (in the almost surely sense) to be defined as:

$$\mathcal{K}(x,x') := \lim_{\text{width}\to\infty} \nabla_\Theta^\top f_\Theta(x) \cdot \nabla_\Theta f_\Theta(x').$$

Here as the width of the neural network tends to infinity, the number of parameters will also go to infinity. The limiting kernel $\mathcal{K}$ is usually referred as the neural tangent kernel (NTK). As the width of the hidden layers in this neural network tends to infinity, this kernel convergences to its expectation over Θ [68].

4.3.2 Exponential kernel

A common class of kernel functions used in machine learning is the exponential kernel, which is a radial basis function kernel with the general form

$$\mathcal{K}(x, x') = \exp(-(c\|x - x'\|)^{\gamma}),$$

where $c > 0$ and $\gamma \geq 1$ are constants. When $\gamma = 1$, this kernel is known as the Laplacian kernel, and when $\gamma = 2$, it is known as the Gaussian kernel.

According to Moore-Aronszajn theorem, each symmetric positive definite kernel uniquely induces a Reproducing kernel Hilbert space (RKHS). RKHS determines the functions that can be learned using a kernel. It has been found that the RKHS of NTK in a ReLU neural network is the same as Laplacian kernel [71, 72], and the empirical performance of a neural network is close to that of kernelized linear classifiers with exponential kernels in many datasets [71].

4.3.3 Training neural networks with quantized weights

Among various methods to train a neural network, BinaryConnect (BC) [41] is often one of the most efficient and accurate method. The key idea is to introduce a real-valued buffer θ and use it to accumulate the gradients. The weights will be quantized just before forward and backward propagation, which can benefit from the reduced computing complexity. The update rule was shown in (3.5)-(3.6).

4.4 Approximation of binary weight neural network

4.4.1 Notations

In this paper, we use $w_{\ell,ij}$ to denote the binary weights in the ℓ-th layer, $\theta_{\ell,ij}$ to denote its real-valued counterpart, and $b_{\ell,i}$ to denote the (real valued) bias. Θ is the collection of all the real-valued model parameters which will be specified in Section 4.4.2. The number of neurons in the ℓ-th hidden layer is d_ℓ, the input to the ℓ-th linear layer is $\boldsymbol{x}_\ell$ and the output is $\boldsymbol{y}_\ell$. d denote the number of input features. Besides, we use $\boldsymbol{x}$ to denote the input to this neural network, y to denote the output and z to denote the label.

We focus on the mean and variance under the randomness of stochastic rounding. Denote

$$\mu_{\ell,i} := \mathbb{E}[x_{\ell,i}|\boldsymbol{x},\Theta], \quad \sigma^2_{\ell,i} := \mathrm{Var}[x_{\ell,i}|\boldsymbol{x},\Theta],$$

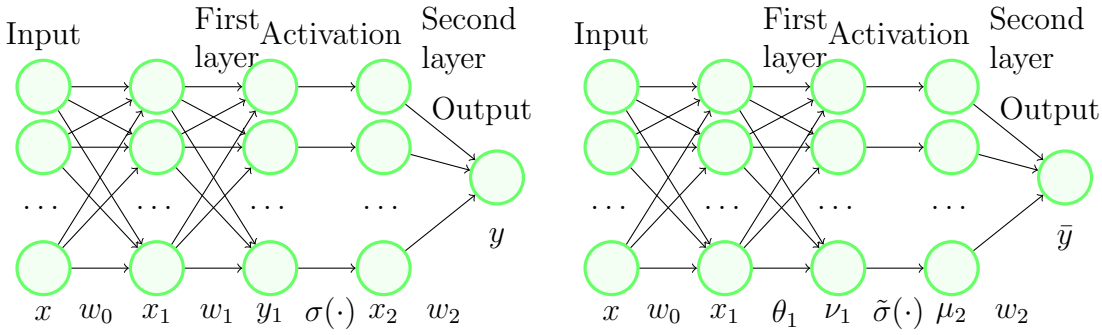
$$\nu_{\ell,i} := \mathbb{E}[y_{\ell,i}|\boldsymbol{x},\Theta], \quad \varsigma^2_{i,\ell} := \mathrm{Var}[y_{\ell,i}|\boldsymbol{x},\Theta], \quad \bar{y} := \mathbb{E}[y|\Theta].$$

We use $\sigma(x) = \max(x, 0)$ to denote ReLU activation function, and in to denote the (discrete) distribution of training dataset. $\mathbb{E}_{in}[\cdot] := \mathbb{E}_{(\boldsymbol{x},z)\sim in}[\cdot]$ denotes the expectation over training dataset, or "sample average". We use bold symbol to denote a collection of parameters or variables $\boldsymbol{w}_2 = [w_{2,j}], \boldsymbol{b}_2 = [b_{2,j}], \boldsymbol{\nu}_1 = [\nu_{1,j}], \boldsymbol{\theta}_1 = [\theta_{1,ij}], i \in [d_1], j \in [d_2]$.

4.4.2 Problem statement

In this work, we target on stochastic quantization [87], which often yields higher accuracy empirically compared with deterministic rounding [41]. This also creates a smooth connection between the binary weights in a neural network and its real-valued parameters.

Let $w_{\ell,ij} = Quantize(\theta_{\ell,ij}), \theta_{\ell,ij} \in [-1, 1]$ be the binary weights from stochastic quan-



(a) Binary weight neural network we focus on.

(b) Quasi neural network.

tization function, which satisfy Bernoulli distribution:

$$w_{\ell,ij} = \begin{cases} +1, & \text{with probability } p_{\ell,ij} = \frac{\theta_{\ell,ij}+1}{2}, \\ -1, & \text{with probability } 1 - p_{\ell,ij}. \end{cases} \tag{4.1}$$

This relationship leads to $\mathbb{E}[w_{\ell,ij}|\theta_{\ell,ij}] = \theta_{\ell,ij}$.

We focus on a ReLU neural network with one hidden layer and two fully connect layers, which was also studied in Bach [69], Bietti et al. [70] except quantization. Besides, we add a linear layer ("additional layer") in front of this neural network to project the input to an infinite dimension space. We randomly initialize the weights in this layer and leave it fixed (not trainable) throughout the training process. Furthermore, we quantize the weights in the first fully connect layer $w_{1,ij}$ and add a real-valued buffer $\theta_{1,ij}$ which determines the distribution of $w_{1,ij}$ as in (4.1), and leave the second layers not quantized. It is a common practice to leave the last layer not quantized, because this often leads to better empirical performance. If the second layer is quantized as well, the main result of this paper will not be changed. This can be easily checked by extending Lemma 4.2 into the second layer.

Remark 4.1 *In many real applications, e.g. computer vision, the dimension of data are often very large ($\approx 10^3$) while they are laying in the lower dimension linear subspace, so*

we can take the raw input in these applications as the output of the additional layer, and the NN in this case is a two-layer NN where the first layer is quantized.

The set of all the real-valued parameters is $\Theta = \{\theta_{\ell_1,ij}, w_{\ell_2,ij}, b_{\ell,i}\}$. The neural network can be expressed as

$$x_{1,i} = \frac{1}{\sqrt{d}} \sum_{k=1}^{d} w_{0,ki} x_k + b_{0,i}, \forall i \in [d_1]; \quad y_{1,j} = \sqrt{\frac{c}{d_1}} \sum_{i=1}^{d_1} w_{1,ij} x_{1,i} + b_{1,j}, \forall j \in [d_2];$$

$$x_{2,j} = \sigma(y_{1,j}), \forall j \in [d_2]; \qquad y = \frac{1}{\sqrt{d_2}} \sum_{j=1}^{d_2} w_{2,j} x_{2,j} + b_2.$$

We follow the typical setting of NTK papers [71] in initializing the parameters except the quantized parameters. As for the quantized parameters, we only need to specify the real-valued buffer of the weights in the first layer $\theta_{1,ij}$.

Assumption 4.1 *We randomly initialize the weights in the "additional layer" and second linear layer independently as* $w_{0,ki}, w_{2,j} \sim \mathcal{N}(0,1)$*, and initialize all the biases to 0. The real-valued buffer of the weights are initialized independently identical with zero mean, variance* $\mathrm{Var}[\theta]$ *and bounded in* $[-1,1]$*.*

Remark 4.2 *Our theory applies to any initial distribution of* $\theta_{1,ij}$ *as long as it satisfies the constraint above. One simple example is the uniform distribution in* $[-1,1]$*, which has variance* $\mathrm{Var}[\theta] = 1/3$*.*

4.4.3 Quasi neural network

Given a fixed input and real-value model parameters Θ, under the randomness of stochastic rounding, the output of this binary weight neural network is a random variable. Furthermore, as the width of the neural network d_1 tends to infinity, we define a parameter sequence $\{\Theta_{d_1}\}$ and prove that with parameters from this sequence, the output

of a linear layer tends to Gaussian distribution according to central limit theorem (CLT). We propose a method to determine the distribution of output and using the model parameters. Specifically, we give a closed form equation to compute the mean and variance of output of all the layers $\mu_\ell, \sigma_\ell, \nu_\ell, \varsigma_\ell$, and then marginalize over random initialization of Θ to further simplify this equation. We prove that ς_ℓ converges to a constant almost surely using the law of large number (LLN), and simplify the expression by replacing them with the constant. This allows us to compute μ_ℓ, ν_ℓ using a neural-network-style function for given Θ. We call this function *quasi neural network*, which is given below:

$$\begin{aligned} x_{1,i} &= \frac{1}{\sqrt{d}} \sum_{k=1}^{d} w_{0,ki} x_k + b_{0,i}, \forall i \in [d_1]; \quad \nu_{1,j} = \sqrt{\frac{c}{d_1}} \sum_{i=1}^{d_1} \theta_{1,ij} x_{1,j} + \beta b_{1,i}, \forall j \in [d_2]; \\ \mu_{2,j} &= \tilde{\sigma}(\nu_{1,j}), \forall j \in [d_2]; \qquad\qquad \bar{y} = \frac{1}{\sqrt{d_2}} \sum_{j=1}^{d_2} w_{2,j} \mu_{2,j} + \beta b_2. \end{aligned} \tag{4.2}$$

In Section 4.4.3, we study the distribution of the output of each layer in a binary weight neural network (BWNN) conditioned on the set of real-valued parameter Θ. In Section 4.4.3, we prove that the conditioned variance of the output of the first linear layer studied above converges almost surely to a constant which does not depend on the data (input). This simplifies the expression computed in Section 4.4.3 to the form of quasi neural network (4.2), and also give a closed-form expression to $\tilde{\sigma}(\cdot)$ in (4.2). In Section 4.4.3, we prove that conditioned on the set of real-valued parameter, the expectation of the gradients of BWNN equals the gradient of quasi neural network on the overparameterization limit. This indicates that the training dynamics of BWNN at initialization is the same as training the quasi neural network directly. The training dynamics beyond initialization are discussed in Section 4.4.3. Before jumping to the proof, we make the following assumptions:

Assumption 4.2 *After training the binary weight neural network as in (3.5)-(3.6), all*

the real-valued weights $\theta_{\ell,ij}$ stay in the range $[-1, 1]$.

Based on this assumption, we can ignore constraints that $\theta_{\ell,ij} \in [-1, 1]$ and the projected gradient descent reduces to gradient descent. Because of the lazy training property of the overparameterized neural network, the model parameters $\theta_{\ell,ij}$ stay close to the initialization point during the training process, so this assumption can be satisfied by initializing $\theta_{\ell,ij}$ with smaller absolute value and/or applying weight decay during training. On the other hand, a common trick in a quantized neural network is to change the quantization level gradually during the training process to avoid (or reduce) overflow. With this trick, Assumption 4.2 are often naturally satisfied, but it introduces the quantization level as a trainable parameter.

Assumption 4.3 *The Euclidean norm of the input is 1:*

$$\|\boldsymbol{x}\|_2 = 1, \forall \boldsymbol{x} \in \mathcal{D} \subseteq \mathbb{R}^d.$$

where $\mathcal{D}$ denotes the training dataset.

This is a common assumption in studying NTK [69, 70], and can be satisfied by normalizing the input data.

Conditioned distribution of the outputs of each layer

First we recognize that as the model parameters are initialized randomly, there are "bad" initialization that will mess up our analysis. For example, all of $\theta_{1,ij}$ are initialized to 1 (or -1) while they are drawn from a uniform distribution. Fortunately, as the width d_1, d_2 grows to infinite, the probability of getting into these "bad" initialization goes to 0. We make this statement formal in the following part.

Definition 4.1 *"Parameter sequence". Define the parameter sequence, indexed by d_1, d_2, as*

$$\Theta^{(d_1,d_2)} = \Big\{ \mathbf{W}_0^{(d_1,d_2)} \in \mathbb{R}^{d,d_1}, \boldsymbol{b}_0^{(d_1,d_2)} \in \mathbb{R}^{d_1}, \boldsymbol{\theta}_1^{(d_1,d_2)} \in \mathbb{R}^{d_1,d_2}, \boldsymbol{b}_1^{(d_1,d_2)} \in \mathbb{R}^{d_2}, \\ \mathbf{W}_2^{(d_1,d_2)} \in \mathbb{R}^{d_2}, b_2^{(d_1,d_2)} \in \mathbb{R} \Big\},$$

where $\mathbf{W}_0 = \{w_{0,ki}\}, \boldsymbol{b}_0 = \{b_{0,i}\}, \boldsymbol{\theta}_1 = \theta_{1,ij}, \boldsymbol{b}_1 = \{b_{1,j}\}, \mathbf{W}_2 = \{w_{1,j}\}$, the superscripts are omitted, such that for all $d_1 \le d_1', d_2 \le d_2'$, $\Theta^{(d_1,d_2)}, \Theta^{(d_1',d_2')}$ satisfy

$$\mathbf{W}_0^{(d_1,d_2)} = \mathbf{W}_0^{(d_1',d_2')}[:, 1:d_1], \quad \boldsymbol{b}_0^{(d_1,d_2)} = \boldsymbol{b}_0^{(d_1',d_2')}[1:d_1], \quad \boldsymbol{\theta}_1^{(d_1,d_2)} = \boldsymbol{\theta}_1^{(d_1',d_2')}[1:d_1, 1:d_2], \\ \boldsymbol{b}_1^{(d_1,d_2)} = \boldsymbol{b}_1^{(d_1',d_2')}[1:d_2], \quad \mathbf{W}_{2,j}^{(d_1,d_2)} = w_2^{(d_1',d_2')}[1:d_2], \quad b_2^{(d_1,d_2)} = b_2^{(d_1',d_2')},$$

$\forall k \in [d], i \le d_1, j \le d_2$.

Remark 4.3 *This definition states that for any two terms (sets of parameters) in the "parameter sequence", the overlapping parameters are always equal.*

Definition 4.2 *"Good Initialization sequence". For any finite d_2, we call the set of parameters sequence defined in Definition 4.1 as a "Good Initialization" $\{\Theta^{(d_1)}\} \in \mathcal{G}$ if it satisfies:*

- $\forall k, k' \in [d], \lim_{d_1 \to \infty} \frac{1}{d_1} \sum_{i=1}^{d_1} w_{0,ki} w_{0,k'i} = \delta_{k,k'}$,
- $\forall k, k', k'' \in [d], \lim_{d_1 \to \infty} \frac{1}{d_1} \sum_{i=1}^{d_1} |w_{0,ki} w_{0,k'i} w_{0,k''i}| \le \sqrt{\frac{8}{\pi}}$,
- $\forall k, k' \in [d], \forall j \in [d_2], \lim_{d_1 \to \infty} \frac{1}{d_1} \sum_{i=1}^{d_1} w_{0,ki} w_{0,k'i} \theta_{1,ij}^2 = \mathrm{Var}[\theta] \delta_{k,k'}$, *where*

$$\delta_{k,k'} = \begin{cases} 1 & k = k' \\ 0 & k \ne k'. \end{cases}$$

Here, we omit the superscript (d_1, d_2) again in the statement for the parameters w's. d is the input dimension. d_1, d_2 are defined in (4.2).

The following Proposition 4.1 guarantees the "Good initialization sequence" in Definition 4.2 holds true with probability 1. The proof can be found in Section 4.8.1.

Proposition 4.1 *Under the assumption that all the parameters are initialized as in Assumption 4.1, for any finite d_2, the probability that the sequence defined in Definition 4.1 is a "Good Initialization sequence" is 1:*

$$Pr(\{\Theta^{(d_1,d_2)}, d_1 = 1, 2, \ldots\} \in \mathcal{G}) = 1.$$

Lemma 4.2 *Given any fixed x, and any fixed "Good Initialization sequence" $\{\Theta_{d_1}\} \in \mathcal{G}$ denoted as Θ in short, for any fixed j, define the random sequence $y_{1,j}^{(d_1)} = f_{\Theta_{d_1}}(x)$. on the limit $d_1 \to \infty$, the distribution of $y_{1,j}^{(d_1)}$ converge to* Gaussian distribution *with mean $\nu_{1,j}$ and variance $\varsigma_{1,j}^2$ which can be computed by:*

$$y_{1,j}|\Theta \to \mathcal{N}(\nu_{1,j}, \varsigma_{1,j}^2), \quad \nu_{1,j} = \sqrt{\frac{c}{d_1}} \sum_{i=1}^{d_1} \theta_{1,ij} x_{1,i} + b_{1,j}, \quad \varsigma_{1,j}^2 = \frac{c}{d_1} \sum_{i=1}^{d_1} (1 - \theta_{1,ij}^2) x_{1,i}^2 \tag{4.3}$$

This lemma can be proved by Lyapunov central limit theorem and sum of expectation. See Section 4.8.1 for the details.

Lemma 4.3 *Assume that the input to a ReLU layer $y_{1,j}$ satisfy Gaussian distribution with mean $\nu_{1,j}$ and variance $\varsigma_{1,j}^2$*

$$y_{1,j} \sim \mathcal{N}(\nu_{1,j}, \varsigma_{1,j}^2).$$

Denote

$$g_j = \varphi\left(\frac{\nu_j}{\varsigma_j}\right), s_j = \Phi\left(\frac{\nu_j}{\varsigma_j}\right), \tag{4.4}$$

where $\varphi(x)$ denotes standard Gaussian function and $\Phi(x)$ denotes its integration:

$$\varphi(x) = \sqrt{\frac{1}{2\pi}} \exp\left(-\frac{1}{2}x^2\right), \quad \Phi(x) = \int_{-\infty}^{x} \varphi(y)dy.$$

Then the output $x_{2,j}$ has mean $\mu_{2,i}$ and variance $\sigma^2_{2,i}$, with

$$\begin{aligned} \mu_{2,j} &:= \mathbb{E}[x_{2,j}] = g_j \varsigma_{1,j} + s_j \nu_{1,j}, \\ \sigma^2_{2,j} &:= \mathrm{Var}[x_{2,j}] = (\varsigma^2_{1,j} + \nu^2_{1,j}) s_j + \nu_{1,j}\sigma_{1,j} g_{1,j} - \nu^2_{1,j}. \end{aligned} \tag{4.5}$$

The proof can be found in Section 4.8.1. From Lemma 4.2 we know that on the limit $d_1 \to \infty$, conditioned on Θ and $\boldsymbol{x}$, for any j, $y_{1,j}$ converge to Gaussian distribution. From continuous mapping theorem, the distribution of $x_{2,j}$ converge to that shown in Lemma 4.3 so its mean $\mu_{2,j}$ and variance $\sigma_{2,j}$ converge to that computed in Lemma 4.3.

Equations (4.3) and (4.5) provide a method to calculate the mean and variance of output conditioned on the input and real-valued model parameters and allow us to provide a closed-form equation of quasi neural network. We will simplify this equation in Section 4.4.3.

Convergence of conditioned variance

In this part, we assume that the model parameters are chosen from "Good Initialization sequence", which is almost surely on the limit $d_1 \to \infty$ as is proven in Proposition 4.1, and study the distribution of $\nu_{1,j}$ and $\varsigma_{1,j}$.

Theorem 4.4 *For any fixed "Good Initialization sequence" $\{\Theta_{d_1}\} \in \mathcal{G}$, on the limit $d_1 \to \infty$, for any finite d_2, $\nu_{1,j}$ converges to Gaussian distribution which are independent*

of each other, and $\varsigma_{1,j}^2$ *converges a.s. to*

$$\tilde{\varsigma}_1^2 = \frac{c}{d}(1 - \mathrm{Var}[\theta]).$$

With this approximation, we can replace the variance $\varsigma_{1,i}$ in Equation (4.5) with $\tilde{\varsigma}_1$ and leave the mean of output in the linear layer as the only variable in the quasi neural network. Formal proof can be found in Section 4.8.1. Note the propagation function in the linear layer (the first equation in (4.3)) is also a linear function in x and θ. This motivates us to compute $\bar{y}$ using a neural network-like function as is given in (4.2), where $\tilde{\sigma}(\cdot)$ is

$$\tilde{\sigma}(\nu_{1,j}) = \mathbb{E}[\sigma(y_{1,j})|\nu_{1,j}] = \tilde{\varsigma}_1 \phi\left(\frac{\nu_{1,i}}{\tilde{\varsigma}_1}\right) + \nu_{1,j}\Phi\left(\frac{\nu_{1,j}}{\tilde{\varsigma}_1}\right). \tag{4.6}$$

This equation gives a closed-form connection between the mean of output of neural network $\bar{y}$ and the real-valued model parameter Θ, and allows up to apply existing tools for analyzing neural networks with real-valued weight to analysis binary weight neural network. Its derivative in the sense of Calculus is:

$$\tilde{\sigma}'(\nu_{1,j}) = \Phi\left(\frac{\nu_{1,j}}{\tilde{\varsigma}_1}\right). \tag{4.7}$$

The proof of derivative can be found in Section 4.8.1.

Gradient of quasi neural network

In this part, we compute the gradients using binary weights as in BinaryConnect Algorithm, and make sense of the gradient in (4.7) by proving that it is the expectation of gradients under the randomness of stochastic rounding.

Theorem 4.5 *The expectation of gradients to output with respect to weights computed by sampling the quantized weights equals the gradients of "quasi neural network" defined*

above in (4.2) *satisfy*

$$\lim_{d_2\to\infty}\lim_{d_1\to\infty}\sqrt{d_2}\left(\frac{\partial\bar{y}}{\partial\theta_{1,ij}}-\mathbb{E}\left[\frac{\partial y}{\partial w_{1,ij}}\middle|\Theta^{(d_1,d_2)}\right]\right)=0,$$
$$\lim_{d_2\to\infty}\lim_{d_1\to\infty}\sqrt{d_2}\left(\frac{\partial\bar{y}}{\partial b_{1,j}}-\mathbb{E}\left[\frac{\partial y}{\partial b_{1,j}}\middle|\Theta^{(d_1,d_2)}\right]\right)=0,$$
$$\lim_{d_2\to\infty}\lim_{d_1\to\infty}\sqrt{d_1d_2}\left(\frac{\partial\bar{y}}{\partial w_{2,j}}-\mathbb{E}\left[\frac{\partial y}{\partial w_{2,j}}\middle|\Theta^{(d_1,d_2)}\right]\right)=0.$$

Theorem 4.6 *For MSE loss,* $\text{loss}(y)=\frac{1}{2}(y-z)^2$*, where* z *is the ground-truth label, the gradient of the loss converges to*

$$\lim_{d_2\to\infty}\lim_{d_1\to\infty}\sqrt{d_2}\left(\frac{\partial\text{loss}(\bar{y})}{\partial\theta_{1,ij}}-\mathbb{E}\left[\frac{\partial\text{loss}(y)}{\partial w_{1,ij}}\middle|\Theta^{(d_1,d_2)}\right]\right)=0,$$
$$\lim_{d_2\to\infty}\lim_{d_1\to\infty}\sqrt{d_2}\left(\frac{\partial\text{loss}(\bar{y})}{\partial b_{1,j}}-\mathbb{E}\left[\frac{\partial\text{loss}(y)}{\partial b_{1,j}}\middle|\Theta^{(d_1,d_2)}\right]\right)=0,$$
$$\lim_{d_2\to\infty}\lim_{d_1\to\infty}\sqrt{d_1d_2}\left(\frac{\partial\text{loss}(\bar{y})}{\partial w_{2,j}}-\mathbb{E}\left[\frac{\partial\text{loss}(y)}{\partial w_{2,j}}\middle|\Theta^{(d_1,d_2)}\right]\right)=0.$$

In other words, the BinaryConnect algorithm provides an unbiased estimator to the gradients for the quasi neural network on this limit of overparameterization. The proof can be found in Section 4.8.1 and Section 4.8.1 respectively.

Theorem 4.4 and Theorem 4.6 conclude that for an infinite wide neural network, the BinaryConnect algorithm is equivalent to training quasi neural network with stochastic gradient descent (SGD) directly. Furthermore, this points out the gradient flow of BinaryConnect algorithm and allows us to study this training process with neural tangent kernel (NTK).

Asymptotics during training

So far we have studied the distribution of output during initialization. To study the dynamic of binary weight neural network during training, one need to extend these

results to any parameter during training $\Theta(t), t \in [0, T]$. Fortunately, motivated by [68], we can prove that as $d_1, d_2 \to \infty$, the model parameters $\Theta(t)$ stays asymptotically close to initialization for any finite T, so-called "lazy training", so the above results apply to the entire training process.

Lemma 4.7 *For all T such that $\int_{t=0}^{T} \|\bar{y}(t) - z\|_{in} dt$ stays stochastically bounded, where $\|\cdot\|_{in}$ is defined in Section 4.4.1, as $d_2 \to \infty, d_1 \to \infty$, $\|\boldsymbol{w}_2(T) - \boldsymbol{w}_2(0)\|, \|\boldsymbol{b}_1(T) - \boldsymbol{b}_1(0)\|, \|\boldsymbol{\theta}_1(T) - \boldsymbol{\theta}_1(0)\|_F$ are all stochastically bounded, $\|\boldsymbol{\nu}_1(t) - \boldsymbol{\nu}_1(0)\|$ and $\int_{t=0}^{T} \left\|\frac{\partial \boldsymbol{\nu}_1(t)}{\partial t}\right\| dt$ is stochastically bounded for all x.*

The proof can be found in Section 4.8.1. Note that $\|\boldsymbol{w}_2\| = O(\sqrt{d_2}), \|\boldsymbol{\theta}_1\|_F = O(\sqrt{d_1 d_2})$, this results indicates that as $d_2 \to \infty$, the varying of the parameter is much smaller than the initialization, or so-called "lazy training". Making use of this result, we further get the follow result:

Lemma 4.8 *Under the condition of Lemma 4.7, Lyapunov's condition holds for all T so $y_{1,j}$ converges to Gaussian distribution conditioned on the model parameters $\Theta(T)$. Furthermore, $\varsigma_{1,j}(T) \to \varsigma_{1,t}(0)$, which equals $\tilde{\varsigma}_1$ almost surely.*

The proof can be found in Section 4.8.1. This result shows that the analysis in Section 4.4.3 applies to the entire training process, and allows us to study the dynamics of binary weight neural network using quasi neural network.

4.5 Capacity of Binary Weight Neural Network

As has been found in [68], the dynamics of an overparameterized neural network trained with SGD is equivalent to kernel gradient descent where the kernel is NTK. As a result, the effective capacity of a neural network is equivalent to the RKHS of its NTK.

In the following part, we will study the NTK of binary weight neural network using the approximation above, and compare it with Gaussian kernel.

4.5.1 NTK of three-layer binary weight neural networks

We consider the NTK binary weight neural network by studying this "quasi neural network" defined as the limiting kernel

$$\lim_{d_2\to\infty}\lim_{d_1\to\infty}\sum_{i=1,j=1}^{d_1,d_2}\frac{\partial\bar{y}}{\partial\theta_{1,ij}}\frac{\partial\bar{y}'}{\partial\theta_{1,ij}}+\sum_{j=1}^{d_2}\frac{\partial\bar{y}}{\partial b_{1,j}}\frac{\partial\bar{y}'}{\partial b_{1,j}}+\sum_{j=1}^{d_2}\frac{\partial\bar{y}}{\partial w_{2,j}}\frac{\partial\bar{y}'}{\partial w_{2,j}}\overset{a.s.}{=\!=}\mathcal{K}_{BWNN}(x,x') \tag{4.8}$$

where $\Theta := \{w_{1,ij}, b_{1,j}, b_{2,j}\}$ denotes all the trainable parameters. We omitted the terms related to b_2 (which is a constant) in this equation.

First prove that the change of kernel asymptotically converges to 0 during training process.

Theorem 4.9 *Under the condition of Lemma 4.7, $\mathcal{K}(x,x')(T) \to \mathcal{K}(x,x')(0)$ at rate $1/\sqrt{d_2}$ for any x, x'.*

The proof can be found in Section 4.8.2. Using Assumption 4.3, we confine the input on the hypersphere $\S^{d-1} = \{x \in \mathbb{R}^d : \|x\|_2 = 1\}$. One can easily tell that it is positive definite, so we can apply Mercer's decomposition [88] to it.

To find the basis and eigenvalues to this kernel, we apply spherical harmonics decomposition to this kernel, which is common among studying of NTK [69, 70]:

$$\mathcal{K}_{BWNN}(x,x') = \sum_{k=1}^{\infty} u_k \sum_{j=1}^{N(d,k)} Y_{k,j}(x)Y_{k,j}(x'), \tag{4.9}$$

where d denotes the dimension of x and x', $Y_{k,j}$ denotes the spherical harmonics of order k. This suggests that NTK of binary weight neural network and exponential kernel can

be spanned by the same set of basis function. The key question is the decay rate of u_k with k.

Theorem 4.10 *The limit of NTK of a binary weight neural network can be decomposed using* (4.9)*. If* $k \gg d$*, then*

$$\mathrm{Poly}_1(k)C^{-k} \leq u_k \leq \mathrm{Poly}_2(k)C^{-k}. \tag{4.10}$$

where $\mathrm{Poly}_1(k)$ *and* $\mathrm{Poly}_2(k)$ *denote polynomials of* k*, and* C *is a constant.*

In contrast, Geifman et al. [71] shows that for NTK in the continuous space, it holds that

$$C_1 k^{-d} \leq u_k \leq C_2 k^{-d},$$

with constants C_1 and C_2. Because its decay rate is slower than that of the binary weight neural network, its RKHS covers a strict superset of functions [71].

Proof Sketch: We first compute NTK of quasi neural network, which depends on the distribution of $\mu_{1,j}$. As is shown in Theorem 4.4, $\mu_{1,j}$ converge to Gaussian distribution on the limit of infinite wide neural network. To find the joint distribution of $\mu_{1,j}$ and $\mu'_{1,j}$ given arbitrary two inputs x, x', we combine the first linear layer in the quasi neural network with the "additional layer" in front of it (the first two equations in (4.2)). This allows up to reparameterize $\mu_{1,j}$ as

$$\mu_{1,j} = \langle w_j, x \rangle,$$

where $w_j \sim \mathcal{N}(0, \frac{\mathrm{cVar}[\theta]}{d} I)$ denotes the fused weight. A key component in computing the NTK has the form

$$\mathbb{E}[\sigma(\mu_1)\sigma(\mu'_1)] = \mathbb{E}[\sigma(\langle w, x \rangle)\sigma'(\langle w, x \rangle)] = \mathbb{E}_{\|w\|}\mathbb{E}[\sigma(\langle w, x \rangle)\sigma'(\langle w, x \rangle)|\|w\|].$$

The second equation comes from the law of total expectation. We use 2-norm in this expression. The inner expectation is equivalent to integration on a sphere, and can be computed by applying sphere harmonics decomposition to $\sigma(\cdot)$. The squared norm of the fused weight $\|w\|^2$ satisfy Chi-distribution, and we use momentum generating function to finish computing. ■

4.5.2 Comparison with Gaussian Kernel

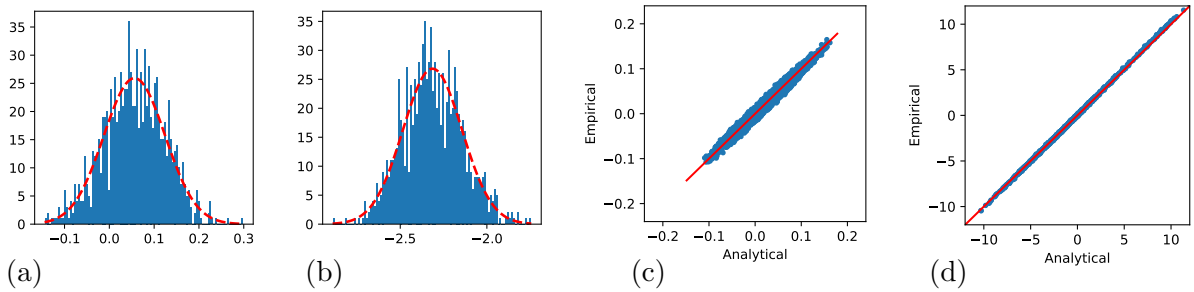
Even if the input to a neural network x is constrained on a unit sphere, the first linear layer (together with the additional linear layer in front of it) will project it to the entire $\mathbb{R}^d$ space with Gaussian distribution. In order to simulate that, we define a kernel by randoming the scale of x and x' beforing taking them into a Gaussian kernel.

$$\mathcal{K}_{RGauss}(x, x') = \mathbb{E}_{\kappa}[\mathcal{K}_{Gauss}(\kappa x, \kappa x')],$$

where $\mathcal{K}_{Gauss}(x, x') = \exp\left(-\frac{\|x-x'\|^2}{\xi^2}\right)$ is a Gaussian kernel, $\kappa \sim \chi_d$ satisfy Chi distribution with d degrees of freedom. This scaling factor projects a random vector uniformly distributed on a unit sphere to Gaussian distributed. The corresponding eigenvalue satisfy

$$A_1 C^{-k} \leq u_k \leq A_2 C^{-k}, \tag{4.11}$$

where A_1, A_2, C are constants that depend on ξ. The dominated term in both (4.10) and (4.11) have an exponential decay rate C^{-k}, which suggests the similarity between NTK of binary weight neural network and Gaussian kernel. In comparison, Bietti et al. [70], Geifman et al. [71] showed that the eigenvalue of NTK decay at rate k^{-d}, which is slower that binary weight neural network or Gaussian kernel. Furthermore, Aronszajn's



(a) (b) (c) (d)

Figure 4.1: Approximation of quasi neural network. (a)(b): before (a) and after (b) training, histogram of output under fixed model parameter (blue), and fitted with Gaussian distribution (red). (c)(d): $\mathbb{E}(y|\Theta)$ computed from quasi neural network (horizontal axis) and by Monte Carlo (Vertical axis). The red line shows $y = x$.

inclusion theorem suggests $\mathcal{H}_{\mathcal{K}_{BWNN}} \subset \mathcal{H}_{\mathcal{K}_{NN}}$, where $\mathcal{K}_{NN}$ denotes the NTK of real-valued weight neural network. In other words, the expressive power of binary weight neural network is weaker than its real valued counterpart on the limit that the width goes to infinity. Binary weight neural networks are less venerable to noise thanks to the smaller expressive power at the expense of failing to learn some "high frequency" components in the target function. This explains that binary weight neural network often achieve lower training accuracy and smaller generalization gap compared with real weight neural network.

4.6 Numerical result

4.6.1 Quasi neural network

In this part, we empirically verify the approximation of quasi neural network by comparing the inference result of quasi neural network with that achieved by Monte Carlo. The architecture is the same as that mentioned in Section 4.4.2, with 1600 hidden neurons. We train this neural network on MNIST dataset [89] by directly applyng gradient descent to the quasi neural network. To reduce overflow, we add weight decay of 0.001 during

training. Figure 4.1(a)(b) shows the histogram of output under stochastic rounding before and after training. We arbitrarily choose one input sample from the testing set and get 1000 samples under different stochastic rounding. This result supports our statement that the distribution of pre-activation (output of linear layer) conditioned on real-valued model parameters converge to Gaussian distribution. Figure 4.1(c)(d) compares the mean of output by quasi neural network approximation (horizontal axis) with that computed using Monte Carlo (vertical axis). These alignments further supports our method of approximating binary weight neural network with quasi neural network.

4.6.2 Generalization gap

Toy dataset

We compare the performance of the neural network with/without binary weight and kernel learning using the same set of 90 small scale UCI datasets with less than 5000 data points as in Geifman et al. [71], Arora et al. [90]. We report the training accuracy and testing accuracy of both vanilla neural network (NN) and binary weight neural network (BWNN) in Figure 4.2. To further illustrate the difference, we list the paired T-test result of neural network (NN) against binary weight neural network (BWNN), and Gaussian kernel (Gaussian) against Laplace kernel (Laplace) using in Table 4.1. In this table, t-stats and p-val denotes the t-statistic and two-sided p-value of the paired t-test between two classifiers, and $<$ and $>$ denotes the percentage of dataset that the first classifier gets lower or higher testing accuracy or generalization bound (training accuracy - testing accuracy), respectively.

As can be seen from the results, although the Laplacian kernel gets higher training accuracy than the Gaussian kernel, its testing accuracy is almost the same as the latter one. In other words, the former has smaller generalization gap than the latter which can

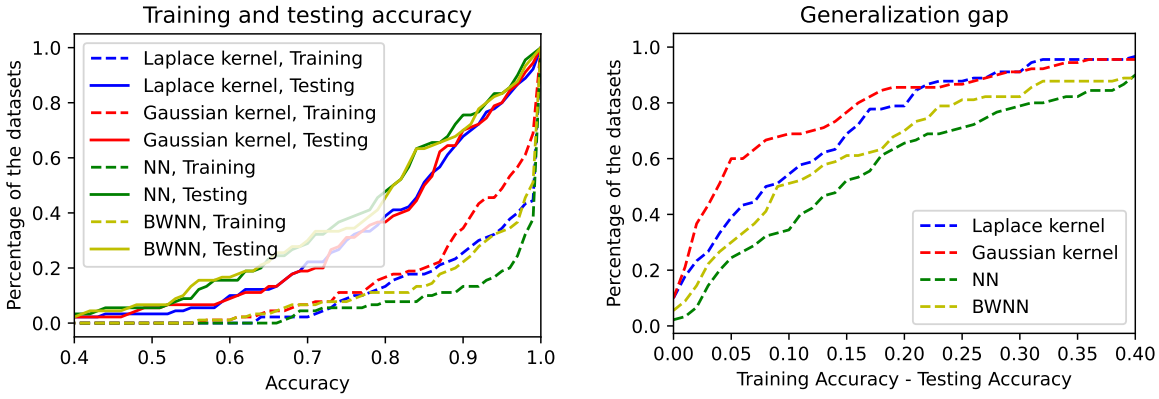


Figure 4.2: Accuracy and generalization gap on selected 90 UCI datasets. The lines show the accuracy metric of a classifier from the lowest to the highest against their percentiles of datasets.

Table 4.1: Pairwise performance comparison on selected 90 UCI datasets.

Classifier	Testing				Training-Testing			
	t-stats	p-val	<	>	t-stats	p-val	<	>
NN-BWNN	0.7471	0.4569	53.33%	41.11%	4.034	0.000	26.67%	67.77%
Laplace-Gaussian	0.4274	0.6701	51.11%	33.33%	3.280	0.001	37.78%	53.33%

also be observed in Table 4.1. Similarly, a neural network gets higher training accuracy than a binary weight neural network but gets similar testing accuracy.

MNIST-like dataset

We compare the performance of neural networks with binary weights (Binary) with its counterpart with real value weights (Real). We take the number of training samples as a parameter by random sampling the training set and use the original test set for testing. The experiments are repeated 10 times and the mean and standard derivation is shown in Figure 4.3. In the MNIST dataset, the performance of neural networks with or without quantization is similar. This is because MNIST [89] is simpler and less vulnerable to overfitting. On the other hand, the generalization gap with weight quantized is much

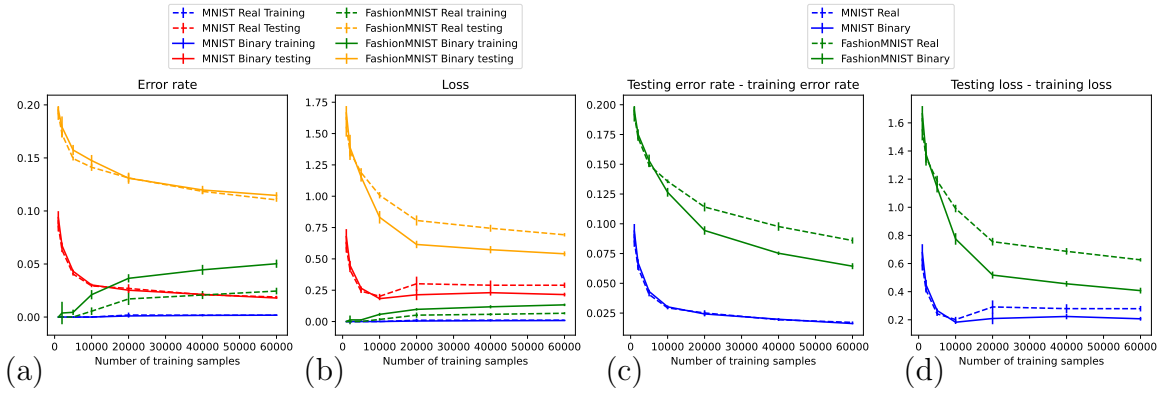


Figure 4.3: Training/testing error rate and loss of neural networks with/without binary weight. (a) Training and testing error rate. (b) Training and testing loss. (c) Testing error rate - Training error rate. (d) Testing loss - Training loss.

smaller than without it in FashionMNIST [29] dataset, which matches our prediction.

4.7 Discussion

In this paper, we propose a quasi neural network to approximate the binary weight neural network. The parameter space of quasi neural network is continuous, and its gradient can be approximated using the BinaryConnect algorithm. We study the expressive power of the binary weight neural network by studying the RKHS of its NTK and showed its similarity with the Gaussian kernel. We empirically verify that quantizing the weights can reduce the generalization gap, similar to Gaussian Kernel versus the Laplacian kernel. This result can be easily generalized to a neural network with other weight quantization methods, i.e. using more bits. Yet there are several questions to be answered by future work:

1. In this work, we only quantize the weights, while much empirical work to quantize both the weights and the activations has been done. Can we use a similar technique to study the expressive power of that neural network?

2. We study the NTK of a two-layer neural network with one additional linear liner in front of it, and only the weights in the first layer are quantized. It remains to be answered whether a multi-layer neural network allow similar approximation, and whether using more layers can increase its expressive power.

4.8 Proofs of technical results

4.8.1 Gaussian approximation in quantized neural network

Proof of Proposition 4.1

Proof: To prove the first statement in Definition 4.2 holds a.s., observe that fixing k, k' and taking i as the variable, $w_{0,ki}w_{0,k'i}$ are independent from each other. Furthermore, $\mathbb{E}[w_{0,ki}w_{0,k'i}] = \delta_{k,k'}$ has identical mean for different i. In addition, since w is bounded, $\sum_{i=1}^{\infty} \mathrm{Var}[w_{0,ki}w_{0,k'i}]/i^2 \leq \sum_{i=1}^{\infty} C/i^2 < \infty$. By the strong law of large number (SLLN) Lemma 4.13, the first statement is proved. The third statement can be proved similarly, observing that $\mathbb{E}[w_{0,ki}w_{0,k'i}\theta_{1,ij}^2] = \delta_{k,k'}\mathrm{Var}[\theta]$ and that both w and θ are bounded (which guarantees $\sum_{i=1}^{\infty} \mathrm{Var}[w_{0,ki}w_{0,k'i}\theta_{1,ij}^2]/i^2 < \infty$).

To prove the second statement in Definition 4.2 holds a.s., since geometric mean is no larger than cubic mean,

$$|w_{0,ki}||w_{0,k'i}||w_{0,k'i}| \leq \frac{1}{3}(|w_{0,ki}|^3 + |w_{0,k'i}|^3 + |w_{0,k'i}|^3),$$

Since $w_{0,ki} \sim \mathcal{N}(0,1)$, the expectation of the right hand side equals $\sqrt{\frac{8}{\pi}}$. We apply SLLN (Lemma 4.13) again to finish the proof. ■

Proof of Lemma 4.2

We first compute the conditioned mean and variance $\nu_{1,j}$ and $\varsigma_{1,j}$. Notice that for any d_1, conditioned on any Θ, x_1 is deterministic,

$$
\begin{aligned}
\nu_{1,j} &= \mathbb{E}_{w_1}\left[y_{1,j}|\Theta\right] \\
&= \sqrt{\frac{c}{d_1}}\sum_{i=1}^{d_1}\mathbb{E}_{w_1}[w_{1,ij}]x_{1,i} + \beta b_{1,j} \\
&= \sqrt{\frac{c}{d_1}}\sum_{i=1}^{d_1}\theta_{1,ij}x_{1,i} + \beta b_{1,j} \\
\varsigma_{1,j} &= \mathrm{Var}_{w_1}\left[y_{1,j}|\Theta\right] \\
&= \mathbb{E}_{w_1}\left[y_{1,j}^2\middle|\Theta\right] - \mathbb{E}_{w_1}\left[y_{1,j}|\Theta\right]^2 \\
&= \frac{c}{d_1}\sum_{i=1}^{d_1}\sum_{i'=1}^{d_1}\mathbb{E}_{w_1}\left[w_{1,ij}w_{1,i'j}|\Theta\right]x_{1,i}x_{1,i'} + 2\beta b_{1,j}\sqrt{\frac{c}{d_1}}\sum_{i=1}^{d_1}\mathbb{E}\left[w_{1,ij}|\Theta\right]x_{1,i} \\
&\quad - \frac{c}{d_1}\sum_{i=1}^{d_1}\sum_{i'=1}^{d_1}\theta_{1,ij}\theta_{1,i'j}x_{1,i}x_{1,i'} - 2\beta b_{1,j}\sqrt{\frac{c}{d_1}}\sum_{i=1}^{d_1}\theta_{1,ij}x_{1,i} \\
&= \frac{c}{d_1}\sum_{i=1}^{d_1}\sum_{i'=1}^{d_1}(\mathbb{E}[w_{1,ij}w_{1,i'j}|\Theta] - \theta_{1,ij}\theta_{1,i'j})x_{1,i}^2 \\
&= \frac{c}{d_1}\sum_{i=1}^{d_1}(\mathbb{E}[w_{1,ij}^2|\Theta] - \theta_{1,ij}^2)x_{1,i}^2 \\
&= \frac{c}{d_1}\sum_{i=1}^{d_1}(1-\theta_{1,ij}^2)x_{1,i}^2.
\end{aligned}
$$

The second line is because $\mathbb{E}_{w_1}[w_{1,ij}w_{1,i'j}|\Theta] = \mathbb{E}_{w_1}[w_{1,ij}|\Theta]\mathbb{E}[w_{1,i'j}|\Theta] = \theta_{1,ij}\theta_{1,i'j}$ when $i \neq i'$.

Next, we need to prove that for any "good initialization sequence" $\{\Theta_{d_1}\} \in \mathcal{G}$, $\{y_{1,i}^{(d_1)}\}$ converge to Gaussian distribution conditioned on $\Theta \in \mathcal{G}$ by verifying Lyapunov's condi-

tion. Note that for any $j \in [d_2]$,

$$y_{1,j} = \sqrt{\frac{c}{d_1}} \sum_{i=1}^{d_1} w_{1,ij} x_{1,i} + b_{1,j}$$

Define $X_i = w_{1,ij} x_{1,i}$. As mentioned above, its mean and variance (conditioned on Θ) is

$$\mathbb{E}_{w_1}[X_i|\Theta] = \theta_{1,ij} x_{1,i}, \quad \mathrm{Var}_{w_1}[X_i|\Theta] = \mathbb{E}_{w_1}[X_i^2|\Theta] - \mathbb{E}_{w_1}[X_i|\Theta]^2 = (1 - \theta_{1,ij}^2) x_{1,i}^2$$

Since $\Theta \in \mathcal{G}$, $\forall j \in [d_2]$ for some finite d_2,

$$\begin{aligned}
\lim_{d_1 \to \infty} \frac{1}{d_1} \sum_{i=1}^{d_1} \mathrm{Var}_{w_1}[X_i|\Theta] &= \lim_{d_1 \to \infty} \frac{1}{d_1} \sum_{i=1}^{d_1} (1 - \theta_{1,ij}^2) x_{1,i}^2 \\
&= \lim_{d_1 \to \infty} \frac{1}{d d_1} \sum_{i=1}^{d_1} (1 - \theta_{1,ij}^2) \Big(\sum_{k=1}^{d} w_{0,ki} x_k \Big)^2 \\
&= \lim_{d_1 \to \infty} \frac{1}{d d_1} \sum_{i=1}^{d_1} \sum_{k,k'=1}^{d} (1 - \theta_{1,ij}^2) w_{0,ki} w_{0,k'i} x_k x_{k'} \\
&= \lim_{d_1 \to \infty} \sum_{k,k'=1}^{d} (1 - \mathrm{Var}[\theta]) \delta_{k,k'} x_k x_{k'} = 1 - \mathrm{Var}[\theta]
\end{aligned} \tag{4.12}$$

The fourth equality comes from the definition of $\mathcal{G}$, and the fifth equality is because

$\|x\|_2 = 1$. The third order absolute momentum can be bounded by

$$\begin{aligned}
&\lim_{d_1\to\infty} \frac{1}{d_1}\sum_{i=1}^{d_1} \mathbb{E}_{w_1}\left[|X_i - \mathbb{E}_{w_1}[X_i|\Theta]|^3\middle|\Theta\right] \\
&= \lim_{d_1\to\infty} \frac{1}{d_1}\sum_{i=1}^{d_1} \mathbb{E}_{w_1}\left[|(w_{1,ij} - \theta_{1,ij})x_{1,i}|^3\middle|\Theta\right] \\
&\leq \lim_{d_1\to\infty} \frac{1}{d_1}\sum_{i=1}^{d_1} \mathbb{E}_{w_1}\left[8|x_{1,i}|^3\middle|\Theta\right] \\
&= \lim_{d_1\to\infty} \frac{8}{d_1}\sum_{i=1}^{d_1} \left|\sum_{k=1}^{d} w_{0,ki}x_k\right|^3 \\
&\leq \lim_{d_1\to\infty} \frac{8}{d_1}\sum_{i=1}^{d_1} \left(\sum_{k=1}^{d} |w_{0,ki}x_k|\right)^3 \\
&= \lim_{d_1\to\infty} \frac{8}{d_1}\sum_{i=1}^{d_1} \sum_{k,k',k''=1}^{d} |w_{0,ki}w_{0,k'i}w_{0,k''i}||x_k x_{k'} x_{k''}| \\
&\leq 8\sqrt{\frac{8}{\pi}}d^3
\end{aligned} \tag{4.13}$$

The last inequality comes from the definition of "Good Initialization": for all $\Theta \in \mathcal{G}$,

$$\lim_{d_1\to\infty} \frac{1}{d_1}\sum_{i=1}^{d_1} w_{0,ki}w_{0,k'i}w_{0,k'i} \leq \sqrt{\frac{8}{\pi}},$$

and because $\|x\|_2 = 1, |x_k| \leq 1$ for all $k \in [d]$. Note that using the strong law of large number, one can prove that the third order absolute momentum converges almost surely to a constant that doesn't depend on d. On the other hand, we are proving a upper bound for all $\Theta \in \mathcal{G}$ which is stronger than almost surely converge.

$$
\begin{aligned}
&\lim_{d_1\to\infty} \frac{\sum_{i=1}^{d_1} \mathbb{E}_{w_1}[|X_i - \mathbb{E}_{w_1}[X_i|\Theta]|^3|\Theta]}{(\sum_{i=1}^{d_1} \mathrm{Var}_{w_1}[X_i|\Theta])^{3/2}} \\
&= \lim_{d_1\to\infty} \frac{\frac{1}{d_1}\sum_{i=1}^{d_1} \mathbb{E}_{w_1}[|X_i - \mathbb{E}_{w_1}[X_i|\Theta]|^3|\Theta]}{d_1^{1/2}(\frac{1}{d_1}\sum_{i=1}^{d_1} \mathrm{Var}_{w_1}[X_i|\Theta])^{3/2}} \\
&= \frac{\lim_{d_1\to\infty} \frac{1}{d_1}\sum_{i=1}^{d_1} \mathbb{E}_{w_1}[|X_i - \mathbb{E}_{w_1}[X_i|\Theta]|^3|\Theta]}{\lim_{d_1\to\infty} d_1^{1/2}(\frac{1}{d_1}\sum_{i=1}^{d_1} \mathrm{Var}_{w_1}[X_i|\Theta])^{3/2}} \\
&\leq \frac{8\sqrt{\frac{8}{\pi}}d^3}{\lim_{d_1\to\infty} d_1^{1/2}(1 - \mathrm{Var}[\theta])} \\
&= 0
\end{aligned}
$$

This proves that Lyapunov's condition for all "Good Initialization", so conditioned on $\Theta \in \mathcal{G}$, $y_{1,j}$ converges to Gaussian distribution.

Proof of Lemma 4.3

To compute $\mathbb{E}_{w_1}[x_{2,j}|\Theta]$ and $\mathrm{Var}_{w_1}[x_{2,j}|\Theta]$, we first compute $\mathbb{E}_{w_1}[\sigma(y_{1,j})|\Theta]$ and $\mathbb{E}_{w_1}[\sigma(y_{1,j})^2|\Theta]$. Recall $\sigma(x) = x\mathbf{1}(x \geq 0)$,

$$
\begin{aligned}
\mathbb{E}_{w_1}[\sigma(y_{1,j})|\Theta] &= \int_0^\infty x \frac{1}{\sqrt{2\pi}\varsigma_{1,j}} \exp\left(-\frac{1}{2}\frac{(x-\nu_{1,j})^2}{\varsigma_{1,j}^2}\right) dx \\
&= \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty (\varsigma_{1,j}y + \nu_{1,j}) \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy \\
&= \varsigma_{1,j} \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty \frac{1}{\sqrt{2\pi}} y \exp\left(-\frac{1}{2}y^2\right) dy + \mu_{\ell,i} \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy,
\end{aligned}
$$

$$\begin{aligned}\mathbb{E}_{w_1}[\sigma(y_{1,j})^2|\Theta] &= \int_0^\infty x^2 \frac{1}{\sqrt{2\pi}\varsigma_{1,j}} \exp\left(-\frac{1}{2}\frac{(x-\nu_{1,j})^2}{\varsigma_{1,j}^2}\right) dx \\ &= \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty (\varsigma_{1,j}y + \nu_{1,j})^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy \\ &= \varsigma_{1,j}^2 \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty y^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\hat{y}^2\right) dy + 2\varsigma_{1,j}\nu_{1,j} \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty y \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy \\ &\quad + \nu_{1,j}^2 \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy.\end{aligned}$$

We only need to compute

$$\int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty \frac{1}{\sqrt{2\pi}} y^\alpha \exp\left(-\frac{1}{2}y^2\right) dy.$$

For $\alpha = 0, 1, 2$. When $\alpha = 0$, this is integration to Gaussian function, and it is known that there's no analytically function to express that. For sack of simplicity, define it as $s_{1,j}$

$$s_{1,j} = \int_{\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy := \Phi(\frac{\nu_{1,j}}{\varsigma_{1,j}}).$$

When $\alpha = 1$, this integration can be simply solved by change of the variable and we denote it as $g_{1,j}$:

$$g_{1,j} = \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^\infty y \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy = \sqrt{\frac{1}{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\nu_{1,j}}{\varsigma_{1,j}}\right)^2\right).$$

When $\alpha = 2$, we can do integration by parts and express it using $s_{1,j}$ and $g_{1,j}$:

$$\begin{aligned}
&\int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^{\infty} y^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy \\
&= -\int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^{\infty} y \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) d\frac{1}{2}y^2 \\
&= \int_{-\frac{\nu_{1,j}}{\varsigma_{1,j}}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) dy - \frac{\nu_{1,j}}{\varsigma_{1,j}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\nu_{1,j}}{\varsigma_{1,j}}\right)^2\right) \\
&= s_{1,j} - \frac{\nu_{1,j}}{\varsigma_{1,j}} g_{1,j}.
\end{aligned}$$

Using the definition of mean and variance,

$$\mu_{2,j} = \mathbb{E}_{w_1}[\sigma(y_{1,j})|\Theta], \sigma_{2,i}^2 = \mathbb{E}_{w_1}[\sigma(y_{1,j})^2|\Theta] - \mathbb{E}_{w_1}[\sigma(y_{1,j})|\Theta]^2,$$

we can come to the result.

Proof of Theorem 4.4

In this part, we take $\Theta = \{w_0, \theta_1, w_2, b_0, b_1, b_2\}$ as the random variables and conditioned mean and variance derived above $\mu_1, \sigma_1, \nu_1, \varsigma_1$ as functions to Θ. From Eq. (4.3), as $d_1 \to \infty$, v_1 tend to iid Gaussian processes, and there covariance converges almost surely to its expectation. We then focus on computing the expectation of covariance.

For any $j \neq j'$, we take the expectation over random initialization of Θ:

$$\begin{aligned}
&\mathbb{E}_\Theta[\nu_{1,j}\nu_{1,j'}] \\
&= \mathbb{E}_\Theta\left[\frac{c}{d_1}\sum_{i=1}^{d_1}\sum_{i'=1}^{d_1}\theta_{1,ij}\theta_{1,i'j'}x_{1,i}x_{1,i'} + \beta\sqrt{\frac{c}{d_1}}\sum_{i=1}^{d_1}(\theta_{1,ij}x_{1,i}b_j + \theta_{1,ij'}x_{1,i}b_{j'}) + \beta^2 b_j b_{j'}\right] \\
&= \frac{c}{d_1}\sum_{i=1}^{d_1}\sum_{i'=1}^{d_1}\mathbb{E}_\Theta[\theta_{1,ij}]\mathbb{E}_\Theta[\theta_{1,i'j'}]\mathbb{E}_\Theta[x_{1,i}x_{1,i'}] + \beta^2\mathbb{E}_\Theta[b_j b_{j'}] \\
&\quad + \sqrt{\frac{c}{d_1}}\beta\sum_{i=1}^{d_1}(\mathbb{E}_\Theta[\theta_{1,ij}]\mathbb{E}_\Theta[x_{1,i}]\mathbb{E}_\Theta[b_j] + \mathbb{E}_\Theta[\theta_{1,ij'}]\mathbb{E}_\Theta[x_{1,i}]\mathbb{E}_\Theta[b_{j'}]) \\
&= 0
\end{aligned} \tag{4.14}$$

which indicates that they are independent.

Computation of $\varsigma_{1,j}$ was already finished implicitly in Section 4.8.1. We write it explicitly here. From (4.3), on the limit $d_1 \to \infty$,

$$\begin{aligned}
\varsigma_{1,j}^2 &= \frac{c}{d_1}\sum_{i=1}^{d_1}(1-\theta_{1,ij}^2)x_{1,i}^2 \\
&= \frac{c}{dd_1}\sum_{i=1}^{d_1}(1-\theta_{1,ij}^2)\sum_{k,k'=1}^{d} w_{0,ki}w_{0,k'i}x_k x_{k'} \\
&= \frac{c}{d}\sum_{k,k'=1}^{d} x_k x_{k'}\frac{1}{d_1}\sum_{i=1}^{d_1} w_{0,ki}w_{0,k'i}(1-\theta_{1,ij}^2) \\
&= \frac{c}{d}\sum_{k,k'=1}^{d} x_k x_{k'}\delta_{k,k'}(1-\mathrm{Var}[\theta]) \\
&= \frac{c}{d}(1-\mathrm{Var}[\theta])\|x\|_2^2 \\
&= \frac{c}{d}(1-\mathrm{Var}[\theta])
\end{aligned}$$

The fourth line comes from the definition of $\mathcal{G}$.

Derivative of activation function in quasi neural network

Let

$$\tilde{\sigma}(x) = \tilde{\varsigma}_1 \phi\left(\frac{x}{\tilde{\varsigma}}\right) + x\Phi\left(\frac{x}{\tilde{\varsigma}}\right),$$

Its derivative is

$$\begin{aligned}\tilde{\sigma}'(x) &= \varphi'\left(\frac{x}{\tilde{\varsigma}}\right) + \Phi\left(\frac{x}{\tilde{\varsigma}}\right) + \frac{x}{\tilde{\varsigma}}\Phi'\left(\frac{x}{\tilde{\varsigma}}\right) \\ &= -\frac{x}{\tilde{\varsigma}}\varphi\left(\frac{x}{\tilde{\varsigma}}\right) + \Phi\left(\frac{x}{\tilde{\varsigma}}\right) + \frac{x}{\tilde{\varsigma}}\varphi\left(\frac{x}{\tilde{\varsigma}}\right) \\ &= \Phi\left(\frac{x}{\tilde{\varsigma}}\right)\end{aligned}$$

The second line is because

$$\begin{aligned}\Phi'(x) &= \varphi(x), \\ \varphi'(x) &= \frac{d}{dx}\sqrt{\frac{1}{2\pi}}\exp\left(-\frac{1}{2}x^2\right) \\ &= -x\sqrt{\frac{1}{2\pi}}\exp\left(-\frac{1}{2}x^2\right) \\ &= -x\varphi(x).\end{aligned}$$

Proof of Theorem 4.5

To make the proof more general, we make $\varsigma_{1,j}$ a parameter of the activation function in quasi neural network as $\tilde{\sigma}(\cdot;\varsigma_{1,j})$. To get the derivative with respect to $\theta_{1,ij}$, we first get the derivative with respect to $\nu_{1,j}$.

$$\frac{\partial \bar{y}}{\partial \nu_{1,j}} = \frac{\partial \bar{y}}{\partial \mu_{2,j}}\frac{\partial \mu_{2,j}}{\partial \nu_{1,j}} = \sqrt{\frac{1}{d_2}} w_{2,j}\tilde{\sigma}'(\nu_{1,j};\varsigma_{1,j})$$

then apply chain rule:

$$\frac{\partial \bar{y}}{\partial w_{2,j}} = \sqrt{\frac{c}{d_2}}\mu_{2,j}, \tag{4.15}$$

$$\frac{\partial \bar{y}}{\partial b_{1,j}} = \frac{\partial \bar{y}}{\partial \nu_{1,j}}\frac{\partial \nu_{1,j}}{\partial b_{1,j}} = \sqrt{\frac{c}{d_2}}\beta w_{2,j}\tilde{\sigma}'(\nu_{1,j};\varsigma_{1,j}), \tag{4.16}$$

$$\frac{\partial \bar{y}}{\partial \theta_{1,ij}} = \frac{\partial \bar{y}}{\partial \nu_{1,j}}\frac{\partial \nu_{1,j}}{\partial \theta_{1,ij}} = \sqrt{\frac{c}{d_1 d_2}}w_{2,j}x_{1,i}\tilde{\sigma}'(\nu_{1,j};\varsigma_{1,j}). \tag{4.17}$$

On the other hand, let's first write down the gradient with respect to weights w_{ij} in quantized neural network and take their expectation conditioned on Θ:

$$\mathbb{E}_{w_1}\left[\frac{\partial y}{\partial w_{2,j}}\middle|\Theta^{(d_1,d_2)}\right] = \sqrt{\frac{c}{d_2}}\mathbb{E}_{w_1}\left[x_{2,j}\middle|\Theta^{(d_1,d_2)}\right], \tag{4.18}$$

$$\mathbb{E}_{w_1}\left[\frac{\partial y}{\partial b_{2,j}}\middle|\Theta^{(d_1,d_2)}\right] = \mathbb{E}_{w_1}\left[\frac{\partial \bar{y}}{\partial y_{1,j}}\frac{\partial y_{1,j}}{\partial b_{2,j}}\middle|\Theta\right] = \sqrt{\frac{c}{d_2}}\beta w_{2,j}\mathbb{E}_{w_1}\left[\sigma'(y_{1,j})\middle|\Theta^{(d_1,d_2)}\right], \tag{4.19}$$

$$\mathbb{E}_{w_1}\left[\frac{\partial y}{\partial w_{1,ij}}\middle|\Theta^{(d_1,d_2)}\right] = \mathbb{E}_{w_1}\left[\frac{\partial y}{\partial y_{1,j}}\frac{\partial y_{1,j}}{\partial w_{1,ij}}\middle|\Theta^{(d_1,d_2)}\right] = \sqrt{\frac{c}{d_1 d_2}}w_{2,j}x_{1,i}\mathbb{E}_{w_1}\left[\sigma'(y_{1,j})\middle|\Theta^{(d_1,d_2)}\right], \tag{4.20}$$

By definition, $\mu_{2,j} = \lim_{d_1\to\infty}\mathbb{E}_{w_1}[x_{2,j}|\Theta^{(d_1,d_2)}]$. On the other hand, from (4.6), one can tell using continuous mapping theorem that

$$\tilde{\sigma}'(\nu_{1,j};\varsigma_{1,j}) = \Phi\left(\frac{\nu_{1,j}}{\varsigma_{1,j}}\right) = \lim_{d_1\to\infty} P[y_{1,j}\geq 0] = \lim_{d_1\to\infty}\mathbb{E}_{w_1}\left[\sigma'(y_{1,j})\middle|\Theta^{(d_1,d_2)}\right],$$

Taking them into (4.15)-(4.20) finishes the proof.

Proof of Theorem 4.6

Observe that conditioned on Θ, $y_{1,j}$ depends only on $\{w_{1,ij}, i\in[d_1]\}$, and that $\{w_{1,ij}, i\in[d_1]\}\cap\{w_{1,ij'}, i\in[d_1]\} = \emptyset$ for $j\neq j'$. Because of that, $y_{1,j}$ are independent of each other. Similarly, $x_{2,j}$ are independent of each other conditioned on Θ. For

MSE loss,

$$\text{loss}(y) = \frac{1}{2}(y-z)^2, \frac{dl(y)}{dy} = y - z,$$

According to the chain rule

$$\frac{\partial \text{loss}(\bar{y})}{\partial \theta} = \frac{\partial \text{loss}(\bar{y})}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial \theta} = (\bar{y} - z) \frac{\partial \bar{y}}{\partial \theta}, \tag{4.21}$$

for any $\theta \in \{\theta_{1,ij}, b_{1,j}, w_{2,j}\}$, which leads to

$$\frac{\partial \text{loss}(\bar{y})}{\partial w_{2,j}} = \sqrt{\frac{c}{d_2}} (\bar{y} - z) \mu_{2,j}, \tag{4.22}$$

$$\frac{\partial \text{loss}(\bar{y})}{\partial b_{1,j}} = \frac{\partial \bar{y}}{\partial \nu_{1,j}} \frac{\partial \nu_{1,j}}{\partial b_{1,j}} = \sqrt{\frac{c}{d_2}} \beta w_{2,j} (\bar{y} - z) \tilde{\sigma}'(\nu_{1,j}; \varsigma_{1,j}), \tag{4.23}$$

$$\frac{\partial \text{loss}(\bar{y})}{\partial \theta_{1,ij}} = \frac{\partial \bar{y}}{\partial \nu_{1,j}} \frac{\partial \nu_{1,j}}{\partial \theta_{1,ij}} = \sqrt{\frac{c}{d_1 d_2}} w_{2,j} x_{1,i} (\bar{y} - z) \tilde{\sigma}'(\nu_{1,j}; \varsigma_{1,j}). \tag{4.24}$$

On the other hand, in the original binary weight neural network, according to the chain rule,

$$\mathbb{E}_{w_1} \left[\frac{\partial \text{loss}(y)}{\partial w_{2,j}} \middle| \Theta^{(d_1,d_2)} \right] = \sqrt{\frac{c}{d_2}} \mathbb{E}_{w_1} \left[(y-z) x_{2,j} \middle| \Theta^{(d_1,d_2)} \right], \tag{4.25}$$

$$\mathbb{E}_{w_1} \left[\frac{\partial \text{loss}(y)}{\partial b_{1,j}} \middle| \Theta^{(d_1,d_2)} \right] = \sqrt{\frac{c}{d_2}} \beta w_{2,j} \mathbb{E}_{w_1} \left[(y-z) \sigma'(y_{1,j}) \middle| \Theta^{(d_1,d_2)} \right], \tag{4.26}$$

$$\mathbb{E}_{w_1} \left[\frac{\partial \text{loss}(y)}{\partial w_{1,ij}} \middle| \Theta^{(d_1,d_2)} \right] = \sqrt{\frac{c}{d_1 d_2}} w_{2,j} x_{1,i} \mathbb{E}_{w_1} \left[(y-z) \sigma'(y_{1,j}) \middle| \Theta^{(d_1,d_2)} \right], \tag{4.27}$$

Note that y is not independent form $x_{2,j}$ or $\sigma'(y_{1,j})$, which is the main challenge of the proof. To deal with this problem, we bound the difference between (4.22)-(4.24) and

(4.25)-(4.27), which requires bounding their covariance.

$$\begin{aligned}
\mathbb{E}_{w_1}\left[x_{2,j}y|\Theta^{(d_1,d_2)}\right] &= \sqrt{\frac{c}{d_2}}\mathbb{E}_{w_1}\left[x_{2,j}\sum_{j=1}^{d_2} w_{2,j}x_{2,j}\middle|\Theta^{(d_1,d_2)}\right] \\
&= \sqrt{\frac{c}{d_2}}\left(\mathbb{E}_{w_1}\left[x_{2,j}^2 w_{2,j}\middle|\Theta^{(d_1,d_2)}\right] + \sum_{j'\neq j}\mathbb{E}_{w_1}\left[x_{2,j}x_{2,j'}w_{2,j'}\middle|\Theta^{(d_1,d_2)}\right]\right) \\
\lim_{d_1\to\infty}\mathbb{E}_{w_1}\left[x_{2,j}y|\Theta^{(d_1,d_2)}\right] &= \sqrt{\frac{c}{d_2}}\left((\mu_{2,j}^2+\sigma_{2,j}^2)w_{2,j} + \sum_{j'\neq j}\mu_{2,j}\mu_{2,j'}w_{2,j'}\right) \\
&= \sqrt{\frac{c}{d_2}}\left(\sigma_{2,j}^2 w_{2,j} + \sum_{j'=1}^{d_2}\mu_{2,j}\mu_{2,j'}w_{2,j'}\right)
\end{aligned} \tag{4.28}$$

Notice that by definition

$$\sqrt{\frac{c}{d_2}}\sum_{j'=1}^{d_2}\mu_{2,j}\mu_{2,j'}w_{2,j'} = \mathbb{E}_{w_1}\left[x_{2,j}\middle|\Theta^{(d_1,d_2)}\right]\mathbb{E}_{w_1}\left[y\middle|\Theta^{(d_1,d_2)}\right]$$

The second term equals $\mathbb{E}_{w_1}[x_{2,j}|\Theta]\mathbb{E}_{w_1}[y|\Theta]$ and the first term converges to 0 when $d_2\to\infty$. Taking it into (4.22) and (4.25) finishes the proof of the first equation.

Similarly,

$$
\begin{aligned}
&\mathbb{E}_{w_1}\left[\sigma'(y_{1,j})y\middle|\Theta^{(d_1,d_2)}\right] \\
&= \mathbb{E}_{w_1}\left[\sqrt{\frac{c}{d_2}}\sigma'(y_{1,j})\sum_{j=1}^{d_2} w_{2,j}\sigma(y_{1,j})\middle|\Theta^{(d_1,d_2)}\right] \\
&= \sqrt{\frac{c}{d_2}}\Bigg(\mathbb{E}_{w_1}\left[\sigma(y_{1,j})\sigma'(y_{1,j})w_{2,j}\middle|\Theta^{(d_1,d_2)}\right] \\
&\qquad + \sum_{j'\neq j}\mathbb{E}_{w_1}\left[\sigma(y_{1,j'})\sigma'(y_{1,j})w_{2,j'}\middle|\Theta^{(d_1,d_2)}\right]\Bigg) \qquad (4.29) \\
&\lim_{d_1\to\infty} E_{w_1}\left[\sigma'(y_{1,j})y\middle|\Theta^{(d_1,d_2)}\right] \\
&= \sqrt{\frac{c}{d_2}}\left(\mathbb{E}_{w_1}\left[\sigma(y_{1,j})w_{2,j}\middle|\Theta^{(d_1,d_2)}\right]w_{2,j} + \sum_{j'\neq j}\mathbb{E}_{w_1}[\sigma'(y_{1,j})|\Theta^{(d_1,d_2)}]\mu_{2,j'}w_{2,j'}\right), \\
&= \sqrt{\frac{c}{d_2}}\left((1-\mathbb{E}_{w_1}[\sigma'(y_{1,j})|\Theta])\mu_{2,j}w_{2,j} + \sum_{j'=1}^{d_2}\mathbb{E}_{w_1}[\sigma'(y_{1,j})|\Theta]\mu_{2,j'}w_{2,j'}\right)
\end{aligned}
$$

Notice that by definition

$$
\sqrt{\frac{c}{d_2}}\sum_{j'=1}^{d_2}\mu_{2,j'}w_{2,j'} = \lim_{d_1\to\infty}\mathbb{E}\left[y\middle|\Theta^{(d_1,d_2)}\right]
$$

and the first term converges to 0 when $d_2\to\infty$. Taking it into (4.23)(4.24)(4.26)(4.27) finishes the proof.

Proof of Lemma 4.7

In this part, we denote $\dot{a} := \frac{\partial a}{\partial t}$ for $a \in \{w_\ell, \theta_\ell, b_\ell\}$, and express each time-depent variable as a function of time t. We define an inner product under the distribution of training dataset

$$
\langle \boldsymbol{a}, \boldsymbol{b}\rangle_{in} = \mathbb{E}_{in}[a(x)b(x)],
$$

and the corresponding norm

$$\|\boldsymbol{a}\|_{in} = \sqrt{\langle \boldsymbol{a}, \boldsymbol{a} \rangle_{in}} = \sqrt{\mathbb{E}_{in}[a(x)^2]}.$$

If $\boldsymbol{a}(x)$ is a vector, $\|\boldsymbol{a}\|_{in} := \sqrt{\mathbb{E}_{in}[\|\boldsymbol{a}(x)\|^2]}$. Note this inner product and norm define a Hilbert space (not to be confused with the RKHS induced by a kernel), so by Cauchy-Schwarz inequality,

$$|\langle \boldsymbol{a}, \boldsymbol{b} \rangle_{in}| \leq \|\boldsymbol{a}\|_{in} \|\boldsymbol{b}\|_{in}, \forall \boldsymbol{a}, \boldsymbol{b}.$$

As is shown in 4.4.3, on the limit $d_1, \to \infty$, the dynamics of training this neural network using gradient descent can be written as:

$$\begin{aligned}
\dot{w}_{2,j}(t) &= \sqrt{\frac{c}{d_2}} \mathbb{E}_{in}[(\bar{y}(t) - z)\mu_{2,j}(t)] \\
\dot{b}_{1,j}(t) &= \sqrt{\frac{c}{d_2}} \mathbb{E}_{in}[\beta w_{2,j}(t)(\bar{y}(t) - z)\tilde{\sigma}'(\nu_{1,j}(t), \varsigma_{1,j}(t))], \\
\dot{\theta}_{1,ij}(t) &= \sqrt{\frac{c}{d_1 d_2}} \mathbb{E}_{in}[w_{2,j}(t) x_{1,i}(\bar{y}(t) - z)\tilde{\sigma}'(\nu_{1,j}(t); \varsigma_{1,j}(t))]
\end{aligned}$$

where dot denotes the derivative with respect to t. Note the activation function $\tilde{\sigma}(\cdot; \varsigma_{1,j}(t))$ depends on $\varsigma_{1,j}$, which makes it time dependent. One can further write down the dynamics of $\nu_{1,j}(t)$ as

$$\dot{\nu}_{1,j}(t) = \sqrt{\frac{1}{d_1}} \sum_{i=1}^{d_1} \dot{\theta}_{1,ij}(t) x_{1,i}(t) + \dot{b}_{1,j}(t)$$

Rewrite these two differential equations in matrix form:

$$\begin{aligned}
\dot{\boldsymbol{w}}_2(t) &= \sqrt{\frac{c}{d_2}}\mathbb{E}_{in}[(\bar{y}(t)-z)\boldsymbol{\mu}_2(t)] \\
\dot{\boldsymbol{b}}_1(t) &= \beta\sqrt{\frac{c}{d_2}}\mathbb{E}_{in}[(\bar{y}(t)-z)(\tilde{\sigma}'(\boldsymbol{\nu}_1(t))\circ\boldsymbol{w}_2(t))], \\
\dot{\boldsymbol{\theta}}_1(t) &= \sqrt{\frac{c}{d_1d_2}}\mathbb{E}_{in}\big[(\bar{y}(t)-z)\boldsymbol{x}_1\otimes\big(\tilde{\sigma}'(\boldsymbol{\nu}_1(t))\circ\boldsymbol{w}_2(t)\big)\big], \\
\dot{\boldsymbol{\nu}}_1(t) &= \sqrt{\frac{1}{d_1}}\dot{\boldsymbol{\theta}}_1\boldsymbol{x}_1+\dot{\boldsymbol{b}}_1
\end{aligned}$$

where $\circ$ denotes elementwise product and $\otimes$ denotes outer product. Here we slightly abuse the notation $\tilde{\sigma}(\cdot)$, which represents elementwise operation when applied to a vector. Their norm are bounded by

$$\begin{aligned}
\frac{\partial}{\partial t}\|\boldsymbol{w}_2(t)-\boldsymbol{w}_2(0)\| &\le \sqrt{\frac{c}{d_2}}\mathbb{E}_{in}[(\bar{y}(t)-z)\|\boldsymbol{\mu}_2(t)\|] = \sqrt{\frac{c}{d_2}}\langle\bar{y}(t)-z,\boldsymbol{\mu}_2(t)\rangle_{in} \\
&\le \sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{\mu}_2(t)\|_{in} \le \sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{\nu}_1(t)\|_{in}
\end{aligned} \tag{4.30}$$

$$\begin{aligned}
\frac{\partial}{\partial t}\|\boldsymbol{b}_1(t)-\boldsymbol{b}_1(0)\| &\le \beta\sqrt{\frac{c}{d_2}}\mathbb{E}_{in}[(\bar{y}(t)-z)\|\tilde{\sigma}'(\boldsymbol{\nu}_1(t))\circ\boldsymbol{w}_2(t)\|] \\
&\le \beta\sqrt{\frac{c}{d_2}}\mathbb{E}_{in}[(\bar{y}(t)-z)\|\boldsymbol{w}_2(t)\|] = \beta\sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{w}_2(t)\|,
\end{aligned} \tag{4.31}$$

$$\begin{aligned}
\frac{\partial}{\partial t}\|\boldsymbol{\theta}_1(t)-\boldsymbol{\theta}_1(0)\|_F &\le \sqrt{\frac{c}{d_1d_2}}\mathbb{E}_{in}\big[(\bar{y}(t)-z)\|\boldsymbol{x}_1\otimes\big(\tilde{\sigma}'(\boldsymbol{\nu}_1(t))\circ\boldsymbol{w}_2(t)\big)\|_F\big] \\
&\le \sqrt{\frac{c}{d_1d_2}}\mathbb{E}_{in}\big[(\bar{y}(t)-z)\|\boldsymbol{x}_1\|\|\boldsymbol{w}_2(t)\|\big] \\
&\le \sqrt{\frac{c}{d_1d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{x}_1\|_{in}\|\boldsymbol{w}_2(t)\| \\
&= \sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{w}_2(t)\|,
\end{aligned} \tag{4.32}$$

$$
\begin{aligned}
\forall \boldsymbol{x}_1, \quad \frac{\partial}{\partial t}\|\boldsymbol{\nu}_1(t)-\boldsymbol{\nu}_1(0)\| &\le \int_{t=0}^{T}\Big\|\frac{\partial \boldsymbol{\nu}_1(t)}{\partial t}\Big\|dt \\
&\le \sqrt{\frac{1}{d_1}}\frac{\partial}{\partial t}\|\boldsymbol{\theta}_1(t)-\boldsymbol{\theta}_1(0)\|_{op}\|\boldsymbol{x}_1\| + \frac{\partial}{\partial t}\|\boldsymbol{b}_1(t)-\boldsymbol{b}_1(0)\| \\
&\le \sqrt{\frac{c}{d_1^2 d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{w}_2(t)\|\|\boldsymbol{x}_1\|_{in}\|\boldsymbol{x}_1\| \\
&\quad + \beta\sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{w}_2(t)\| \\
&= (1+\beta)\sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{w}_2(t)\|, \\
\frac{\partial}{\partial t}\|\boldsymbol{\nu}_1(t)-\boldsymbol{\nu}_1(0)\|_{in} &\le (1+\beta)\sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{w}_2(t)\|.
\end{aligned}
\tag{4.33}
$$

Here we make use of the fact that $\tilde{\phi}'(x) \le 1$, $\tilde{\phi}(x) \le x$ regardless of the value of $\varsigma_{1,j}(t)$, that $\lim_{d_1\to\infty}\|\boldsymbol{x}_1\|_{in}/\sqrt{d_1} = 1$ as long as $\Theta \in \mathcal{G}$, and that w_0 is not updated during training. In the last equation, we make use of $\dot{\boldsymbol{\theta}}_1 = \frac{\partial}{\partial t}(\boldsymbol{\theta}_1(t)-\boldsymbol{\theta}_1(0)), \dot{\boldsymbol{b}}_1 = \frac{\partial}{\partial t}(\boldsymbol{b}_1(t)-\boldsymbol{b}_1(0))$.

Define $A(t) = \sqrt{\frac{c}{d_2}}\sqrt{1+\beta}(\|\boldsymbol{w}_2(t)-\boldsymbol{w}_2(0)\| + \|\boldsymbol{w}_2(0)\|) + \sqrt{\frac{c}{d_2}}(\|\boldsymbol{\nu}_1(t)-\boldsymbol{\nu}_1(0)\|_{in} + \|\boldsymbol{\nu}_1(0)\|_{in})$, then

$$
\begin{aligned}
\dot{A}(t) &\le \sqrt{1+\beta}\sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{\nu}_1(t)\|_{in} + (1+\beta)\sqrt{\frac{c}{d_2}}\|\bar{y}(t)-z\|_{in}\|\boldsymbol{w}_2(t)\| \\
&\le \sqrt{1+\beta}A(t)
\end{aligned}
$$

Observe that $A(0)$ is stochastically bounded. Using Grönwall's Lemma, for any finite T:

$$
A(T) \le A(0)\exp\Big(\int_{t=0}^{T}\sqrt{1+\beta}dt\Big) = A(0)\exp(\sqrt{1+\beta}T)
$$

so $A(T)$ is stochastically bounded for all finite T as $d_2 \to \infty$. Furthermore,

$$
\sqrt{\frac{c}{d_2}}\|\boldsymbol{w}_2(T)\| \le \sqrt{\frac{c}{d_2}}(\|\boldsymbol{w}_2(T)-\boldsymbol{w}_2(0)\| + \|\boldsymbol{w}_2(0)\|)
$$

which is also stochastically bounded. Integrating (4.30)-(4.33) from 0 to T finishes the proof.

Proof of Lemma 4.8

From (4.3), it's easy to get the dynamics of ς_1:

$$\begin{aligned}
\frac{\partial \varsigma_{1,j}^2(t)}{\partial t} &= -\frac{2c}{d_1}\sum_{i=1}^{d_1}\theta_{1,ij}(t)\dot{\theta}_{1,ij}(t)x_{1,i}^2 \\
|\varsigma_{1,j}^2(T)-\varsigma_{1,j}^2(0)| &\le \frac{2c}{d_1}\sum_{i=1}^{d_1}x_{1,i}^2\int_{t=0}^{T}|\theta_{1,ij}(t)||\dot{\theta}_{1,ij}(t)|dt \\
&\le \frac{2c}{d_1}\sum_{i=1}^{d_1}x_{1,i}^2\int_{t=0}^{T}|\dot{\theta}_{1,ij}(t)|dt \\
&\le \frac{2c}{d_1}\sqrt{\frac{c}{d_1d_2}}\sum_{i=1}^{d_1}x_{1,i}^2\int_{t=0}^{T}\mathbb{E}_{in}\big|w_{2,j}(t)x_{1,i}(\bar{y}(t)-z)\tilde{\sigma}'(\nu_{1,j}(t);\varsigma_{1,j}(t))\big|dt \\
&\le \frac{2c}{d_1}\sqrt{\frac{c}{d_1d_2}}\sum_{i=1}^{d_1}x_{1,i}^2\int_{t=0}^{T}|w_{2,j}(t)|\mathbb{E}_{in}\big|x_{1,i}(\bar{y}(t)-z)\big|dt \\
&\le \frac{2c}{d_1}\sqrt{\frac{c}{d_1d_2}}\sum_{i=1}^{d_1}x_{1,i}^2\int_{t=0}^{T}|w_{2,j}(t)|\|x_{1,i}\|_{in}\|\bar{y}(t)-z\|_{in}dt \\
&\le \frac{2c}{d_1^{3/2}}\sum_{i=1}^{d_1}x_{1,i}^2\|x_{1,i}\|_{in}\int_{t=0}^{T}C(t)\|\bar{y}(t)-z\|_{in}dt \\
&\le \frac{2c}{d_1^{3/2}}\sum_{i=1}^{d_1}x_{1,i}^2\|x_{1,i}\|_{in}\max_{t\in[0,T]}C(t)\int_{t=0}^{T}\|\bar{y}(t)-z\|_{in}dt \quad a.s.
\end{aligned}$$

Here we assume that $\sqrt{\frac{c}{d_2}}\|\boldsymbol{w}_2(t)\|$ is stochastically bounded by $C(t)$. Since $C(t)$ is finite for all $t\in[0,T]$, it's easy to check the term after max operator is stochastically bounded. The remaining task is to bound term before max operator. From standard Gaussian process analysis, $x_{1,i}$ satisfy Gaussian distribution. From the law of large number (LLN),

as $d_1 \to \infty$,

$$\frac{1}{d_1}\sum_{i=1}^{d_1} x_{1,i}^2 \|x_{1,i}\|_{in} = \mathbb{E}[x_{1,i}^2 \|x_{1,i}\|_{in}]$$

almost surely, where the expectation is taken over w_1, and this limit is also bounded. Because of that, as $d_1, d_2 \to \infty$, the difference $|\varsigma_{1,j}^2(T) - \varsigma_{1,j}^2(0)|$ converges to 0 at rate $\frac{1}{\sqrt{d_2}}$.

Notice that the proof of Lyapunov's condition (4.13) doesn't depend on time T from the third line. Since $\varsigma_{1,j}(T)$ stochastically converges to $\varsigma_{1,j}(0)$ for all finite T, Lyapunov's condition holds for all T thus $x_{2,j}$ always converges to Gaussian distribution conditioned on model parameter.

4.8.2 NTK of neural networks with quantized weights

Spherical harmonics

This subsection briefly reviews the relevant concepts and properties of spherical harmonics. Most part of this subsection comes from Bach [69] Section D.1. and Bietti et al. [70] Section C.1.

According to Mercer's theorem, any positive definite kernel can be decomposed as

$$\mathcal{K}(x, x') = \sum_i \lambda_i \Phi(x)\Phi(x'),$$

where $\Phi(\cdot)$ is called the feature map. Furthermore, any zonal kernel on the unit sphere, i.e., $\mathcal{K}(x, x') = \mathcal{K}(x^T x')$ for any $x, x' \in \mathbb{R}^d, \|x\|_2 = \|x'\|_2 = 1$, including exponential kernels and NTK, can be decomposed using spherical harmonics (4.9):

$$\mathcal{K}(x, x') = \sum_{k=1}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} Y_{k,j}(x) Y_{k,j}(x').$$

Legendre polynomial. We have the additional formula

$$\sum_{j=1}^{N(d,k)} Y_{k,j}(x)Y_{k,j}(x') = N(d,k)P_k(x^T x'),$$

where

$$N(d,k) = \frac{(2k+d-2)(k+d-3)!}{k!(d-2)!}.$$

The polynomial P_k is the k-th Legendre polynomial in d dimension, also known as Gegenbauer polynomials:

$$P_k(t) = \left(-\frac{1}{2}\right)^k \frac{\Gamma\left(\frac{d-1}{2}\right)}{\Gamma\left(k+\frac{d-1}{2}\right)}(1-t^2)^{(3-d)/2}\left(\frac{d}{dt}\right)^k (1-t^2)^{k+(d-3)/2}.$$

It is even (resp. odd) when k is odd (reps. even). Furthermore, they have the orthogonal property

$$\int_{-1}^{1} P_k(t)P_j(t)(1-t^2)^{(d-3)/2}dt = \delta_{ij}\frac{w_{d-1}}{w_{d-2}}\frac{1}{N(d,k)},$$

where

$$w_{d-1} = \frac{2\pi^{d-2}}{\Gamma(d/2)}$$

denotes the surface of sphere $\S^{d-1}$ in d dimension, and this leads to the integration property

$$\int P_j(\langle w,x\rangle)P_k(\langle w,x\rangle)d\tau(w) = \frac{\delta_{jk}}{N(p,k)}P_k(\langle x,y\rangle)$$

for any $x, y \in \S^{d-1}$. $\tau(w)$ is the uniform measure on the sphere.

NTK of quasi neural network

We start the proof of the Theorem 4.10 by the following lemmas:

Lemma 4.11 *The NTK of a binary weight neural network can be simplified as*

$$\begin{aligned}\mathcal{K}(x,x') &= \left(\frac{c}{d}\langle x,x'\rangle + \beta^2\right)\Sigma^{(0)} + \Sigma^{(1)},\\ \Sigma^{(0)} &= \mathbb{E}\left[\tilde{\sigma}'(\mu)\,\tilde{\sigma}'(\mu')\right], \quad \Sigma^{(1)} = \mathbb{E}\left[\tilde{\sigma}(\mu)\,\tilde{\sigma}(\mu')\right],\end{aligned} \tag{4.34}$$

where $[\mu,\mu'] \sim \mathcal{N}(0,\Sigma)$,

$$\Sigma = \mathbb{E}[x_{1,i}x'_{1,i}] = \frac{c}{d}\mathrm{Var}[\theta]\begin{bmatrix}1 & x^Tx'\\ x^Tx' & 1\end{bmatrix}$$

are the pre-activation of the second layer.

Proof:

$$\begin{aligned}\mathcal{K}(x,x') &= \sum_{i=1,j=1}^{d_1,d_2}\frac{\partial\bar{y}}{\partial\theta_{1,ij}}\frac{\partial\bar{y}'}{\partial\theta_{1,ij}} + \sum_{j=1}^{d_2}\frac{\partial\bar{y}}{\partial b_{1,j}}\frac{\partial\bar{y}'}{\partial b_{1,j}} + \sum_{j=1}^{d_2}\frac{\partial\bar{y}}{\partial w_{2,j}}\frac{\partial\bar{y}'}{\partial w_{2,j}}\\ &= \frac{c}{d_1d_2}\sum_{i=1,j=1}^{d_1,d_2}x_{1,i}x'_{1,i}w_{2,j}^2\tilde{\sigma}'(\nu_{1,j})\tilde{\sigma}'_2(\nu'_{1,j})\\ &\quad + \frac{\beta^2}{d_2}\sum_{j=1}^{d_2}\tilde{\sigma}'(\nu_{1,j})\tilde{\sigma}'(\nu'_{1,j}) + \frac{1}{d_2}\sum_{j=1}^{d_2}\tilde{\sigma}(\nu_{1,j})\tilde{\sigma}(\nu'_{1,j})\\ &= \frac{c}{d_1d_2}\sum_{i=1}^{d_1}x_{1,i}x'_{1,i}\sum_{j=1}^{d_2}w_{2,j}^2\tilde{\sigma}'(\nu_{1,j})\tilde{\sigma}'(\nu'_{1,j})\\ &\quad + \frac{\beta^2}{d_2}\sum_{j=1}^{d_2}w_{2,j}^2\tilde{\sigma}'(\nu_{1,j})\tilde{\sigma}'(\nu'_{1,j}) + \frac{1}{d_2}\sum_{j=1}^{d_2}\tilde{\sigma}(\nu_{1,j})\tilde{\sigma}(\nu'_{1,j})\\ &= (\frac{c}{d}\langle x,x'\rangle + \beta^2)\mathbb{E}[\tilde{\sigma}'(\nu)\tilde{\sigma}'(\nu')] + \mathbb{E}[\tilde{\sigma}(\nu)\tilde{\sigma}(\nu')] \quad a.s.\end{aligned}$$

where (ν,ν') has the same distribution as $(\nu_{2,j},\nu'_{2,j})$ for any j. We make use of the fact $\mathbb{E}[w_{2,j}^2]=1$, and from central limit theorem, $x_{1,i},x'_{1,i}$ and $\mu_{1,i},\mu'_{1,i}$ converge to joint

Gaussian distribution for any fixed x, x' as $d_1 \to \infty$

$$\begin{aligned}\mathbb{E}[x_{1,i}x'_{1,i}] &= \frac{1}{d}\mathbb{E}[\sum_{k=1}^{d} w_{ki}x_k \sum_{k'=1}^{d} w_{k'i}x'_{k'}] \\ &= \frac{1}{d}\mathbb{E}[\sum_{k=1}^{d} w_{ki}^2 x_k x'_k] \\ &= \frac{1}{d}\langle x, x'\rangle\end{aligned}$$

Similarly,

$$\mathbb{E}[\mu_{1,i}^2] = \frac{c}{d_1}\sum_{i=1}^{d_1}\mathbb{E}[\theta_{1,ij}^2]\mathbb{E}[x_{1,j}^2] = \frac{c}{d}\mathrm{Var}[\theta]$$

$$\mathbb{E}[\mu_{1,i}\mu'_{1,i}] = \frac{c}{d_1}\sum_{i=1}^{d_1}\mathbb{E}[\theta_{1,ij}^2]\mathbb{E}[x_{1,j}x'_{1,j}] = \frac{c}{d}\mathrm{Var}[\theta]\langle x, x'\rangle$$

■

Proof of Theorem 4.9

Remind that as is proved in Theorem 4.8, $\varsigma_{1,j}(T) \to \varsigma_{1,t}(0)$ for any T satisfying a mild condition, and $\varsigma_{1,t}(0)$ is nonzero almost surely. Making use the fact that $\tilde{\sigma}(\cdot;\varsigma)$ is continuous with respect to ς, and its first and second order derivative is stochastically bounded, the change of kernel $\mathcal{K}$ induced by $\varsigma_{1,j}$ converges to 0 as $d_1, d_2 \to \infty$. This reduces to this quasi neural network to a standard neural network with activation function $\tilde{\sigma}(\cdot)$, which is twice differentiable and has bounded second order derivative. From Theorem 2 in [68], the kernel during training converges to the one during initialization. For the ease of the

readers, we restate the proof below. On the limit $d_2 \to \infty, d_1 \to \infty$,

$$
\begin{aligned}
&\mathcal{K}(x,x')(t) - \mathcal{K}(x,x')(0) \\
&= \frac{c\langle x,x'\rangle + \beta^2}{d_2} \sum_{j=1}^{d_2} \big(w_{2,j}^2(t)\tilde{\sigma}'(\nu_{1,j}(t))\tilde{\sigma}'(\nu'_{1,j}(t)) - w_{2,j}^2(0)\tilde{\sigma}'(\nu_{1,j}(0))\tilde{\sigma}'(\nu'_{1,j}(0)) \big) \\
&\quad + \frac{1}{d_2} \sum_{j=1}^{d_2} \big(\tilde{\sigma}(\nu_{1,j}(t))\tilde{\sigma}(\nu'_{1,j}(t)) - \tilde{\sigma}(\nu_{1,j}(0))\tilde{\sigma}(\nu'_{1,j}(0)) \big) \\
&= \frac{c\langle x,x'\rangle + \beta^2}{d_2} \Bigg(\sum_{j=1}^{d_2} \big(w_{2,j}^2(t) - w_{2,j}^2(0) \big) \tilde{\sigma}'(\nu_{1,j}(t))\tilde{\sigma}'(\nu'_{1,j}(t)) \\
&\quad + \sum_{j=1}^{d_2} w_{2,j}^2(0) \big(\tilde{\sigma}'(\nu_{1,j}(t)) - \tilde{\sigma}'(\nu_{1,j}(0)) \big) \tilde{\sigma}'(\nu'_{1,j}(t)) \\
&\quad + \sum_{j=1}^{d_2} w_{2,j}^2(0) \tilde{\sigma}'(\nu_{1,j}(0)) \big(\tilde{\sigma}'(\nu'_{1,j}(t)) - \tilde{\sigma}'(\nu'_{1,j}(0)) \big) \Bigg) \\
&\quad + \frac{1}{d_2} \sum_{j=1}^{d_2} \tilde{\sigma}(\nu_{1,j}(t)) \big(\tilde{\sigma}(\nu'_{1,j}(t))\tilde{\sigma}(\nu'_{1,j}(0)) \big) \\
&\quad + \frac{1}{d_2} \sum_{j=1}^{d_2} \tilde{\sigma}(\nu'_{1,j}(0)) \big(\tilde{\sigma}(\nu_{1,j}(t)) - \tilde{\sigma}(\nu_{1,j}(0)) \big)
\end{aligned}
$$

$$
\begin{aligned}
&|\mathcal{K}(x,x')(t) - \mathcal{K}(x,x')(0)| \\
&\leq \left| \frac{c\langle x,x'\rangle + \beta^2}{d_2} \right| \Bigg(\sum_{j=1}^{d_2} w_{2,j}^2(0)\tilde{\sigma}'(\nu'_{1,j}(t)) \big| \tilde{\sigma}'(\nu_{1,j}(t)) - \tilde{\sigma}'(\nu_{1,j}(0)) \big| \\
&\quad + \sum_{j=1}^{d_2} w_{2,j}^2(0)\tilde{\sigma}'(\nu_{1,j}(0)) \big| \tilde{\sigma}'(\nu'_{1,j}(t)) - \tilde{\sigma}'(\nu'_{1,j}(0)) \big| \\
&\quad + \sum_{j=1}^{d_2} |w_{2,j}(t) - w_{2,j}(0)||w_{2,j}(t) + w_{2,j}(0)||\tilde{\sigma}'(\nu_{1,j}(t))\tilde{\sigma}'(\nu'_{1,j}(t))| \Bigg) \\
&\quad + \frac{1}{d_2} \sum_{j=1}^{d_2} \tilde{\sigma}(\nu_{1,j}(t)) \big| \tilde{\sigma}(\nu'_{1,j}(t))\tilde{\sigma}(\nu'_{1,j}(0)) \big| \\
&\quad + \frac{1}{d_2} \sum_{j=1}^{d_2} \tilde{\sigma}(\nu'_{1,j}(0)) \big| \tilde{\sigma}(\nu_{1,j}(t)) - \tilde{\sigma}(\nu_{1,j}(0)) \big|
\end{aligned}
$$

From Theorem 4.8, and observing that $\tilde{\sigma}'(x), \tilde{\sigma}''(x)$ are bounded by constants, one can verify that each summation term is stochastically bounded by $\sqrt{d_2}$, so as $d_2 \to \infty$, $\mathcal{K}(t) - \mathcal{K}(0)$ converges to 0 at rate $\sqrt{d_2}$.

Spherical harmonics decomposition to activation function

Following Bach [69], we start by studying the decomposition of action in quasi neural network (4.6) and its gradients (4.7): for arbitrary fixed $c > 0$, $-1 \leq t \leq 1$, we can decompose equation (4.6) and (4.7) as

$$\tilde{\sigma}(ct) = \sum_{k=0}^{\infty} \lambda_k N(d,k) P_k(t), \tag{4.35}$$

$$\tilde{\sigma}'(ct) = \sum_{k=0}^{\infty} \lambda'_k N(d,k) P_k(t), \tag{4.36}$$

where P_k is the k-th Legendre polynomial in dimension d.

Lemma 4.12 *The decomposition of activation function in the quasi neural network (4.35) satisfies*

1. $\lambda_k = 0$ *if* k *is odd,*
2. $\lambda_k > 0$ *if* k *is even,*
3. $\lambda_k \asymp \mathrm{Poly}(k)(C/\sqrt{k})^{-k}$ *as* $k \to \infty$ *when* k *is even, where* $\mathrm{Poly}(k)$ *denotes a polynomial of* k*, and* C *is a constant.*

Its gradient (4.36) satisfies

1. $\lambda'_k = 0$ *if* k *is even,*
2. $\lambda'_k > 0$ *if* k *is odd,*

3. $\lambda'_k \asymp \text{Poly}(k)(C/\sqrt{k})^{-k}$ *as* $k \to \infty$ *when* k *is odd, where* $\text{Poly}(k)$ *denotes a polynomial of* k*, and* C *is a constant.*

Proof:

Let's start with the derivative of activation function in quasi neural network:

$$\tilde{\sigma}'(t) = \Phi(\hat{c}t), -1 \leq t \leq 1,$$

where $\hat{c}$ is a constant. We introduce the auxiliary parameters $x, w \in \mathbb{R}^d$ s.t. $\|x\|_2 = \|w\|_2 = 1$ and let $t = w^T x$ By Cauchy–Schwarz inequality, $-1 \leq w^T x \leq 1$. Following [69], we have the following decomposition to $\tilde{\sigma}'(w^T x)$:

$$\tilde{\sigma}'(w^T x) = \sum_{k=1}^{\infty} \lambda'_k N(d,k) P_k(w^T x),$$

where $N(d,k)$ and $P_k(\cdot)$ are defined in section 4.8.2, λ'_k can be computed by

$$\begin{aligned}\lambda'_k &= \frac{w_{d-1}}{w_d} \int_{-1}^{1} \tilde{\sigma}'(t) P_k(t) (1-t^2)^{(d-2)/2} dt \\ &= \left(-\frac{1}{2}\right)^k \frac{\Gamma((d-1)/2)}{\Gamma(k+(d-1)/2)} \frac{w_{d-1}}{w_d} \int_{-1}^{1} \tilde{\sigma}'(t) \left(\frac{d}{dt}\right)^k (1-t^2)^{k+(d-3)/2} dt.\end{aligned}$$

To solve this itegration, we can apply Taylor decomposition to $\tilde{\sigma}'(\cdot)$:

$$\tilde{\sigma}'(ct) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n \hat{c}^{2n+1}}{2^n n!(2n+1)} t^{2n+1}. \tag{4.37}$$

We will study the following polynomial integration first

$$\int_{-1}^{1} t^{\alpha} \left(\frac{d}{dt}\right)^k (1-t^2)^{k+(d-3)/2} dt.$$

When $\alpha < k$, this integration equals 0 as P_k is orthogonal to all polynomials of degree less

than k. If $(\alpha - k) \mod 2 \neq 0$, this integration is 0 because the function to be integrated is an odd function. For $\alpha \geq k$ and $k \equiv \alpha \mod 2$ (k is odd), using successive integration by parts,

$$\begin{aligned}\int_{-1}^{1} t^{\alpha}\left(\frac{d}{dt}\right)^{k}(1-t^2)^{k+(d-3)/2}dt &= (-1)^k\frac{\alpha!}{(\alpha-k)!}\int_{-1}^{1} t^{\alpha-k}(1-t^2)^{k+(d-3)/2}dt \\ &= (-1)^k\frac{\alpha!}{(\alpha-k)!}\int_{-\pi/2}^{\pi/2}\sin^{\alpha-k}(x)\cos^{2k+(d-2)}(x)dx \\ &= (-1)^k C_d\frac{\alpha!(2k+d-3)!!}{(\alpha-k)!!(\alpha+k+d-2)!!},\end{aligned} \tag{4.38}$$

where C_d is a constant that depends only on d mod 2.

Combining (4.37) and (4.38), we have $\lambda_k = 0$ when k is even and $k \neq 0$. When k is odd,

$$\lambda'_k = \left(-\frac{1}{2}\right)^k \frac{\Gamma((d-1)/2)}{\Gamma(k+(d-1)/2)}\frac{w_{d-1}}{w_d}\frac{C_d}{\sqrt{2\pi}}\sum_{\alpha=k:2}^{\infty}\hat{c}^{\alpha}(-1)^{(\alpha-1)/2}\frac{(\alpha-2)!!(2k+d-3)!!}{(\alpha-k)!!(\alpha+k+d-2)!!}.$$

Following [69, 71] we take d as a constant and take k to infinity. Let $\beta = (\alpha-k)/2 \geq 0$ we have

$$\begin{aligned}\lambda'_k &= (-1)^{(k+1)/2}\left(\frac{1}{2}\right)^k \frac{\Gamma((d-1)/2)}{\Gamma(k+(d-1)/2)}\frac{w_{d-1}}{w_d}\frac{C_d}{\sqrt{2\pi}}\sum_{\beta=0}^{\infty}\frac{(-1)^{\beta}\hat{c}^{2\beta+k}(2\beta+k-2)!!(2k+d-3)!!}{(2\beta)!!(2\beta+2k+d-2)!!} \\ &\asymp (-1)^{(k+1)/2}\left(\frac{1}{2}\right)^k \frac{\Gamma((d-1)/2)}{\Gamma(k+(d-1)/2)}\frac{w_{d-1}}{w_d}\frac{C_d}{\sqrt{2\pi}}\sum_{\beta=0}^{\infty}\frac{(-1)^{\beta}\hat{c}^{2\beta+k}\Gamma(\beta+k/2)\Gamma(k+(d-1)/2)}{\beta!\Gamma(\beta+k+d/2)2^{\beta-k/2}} \\ &:= (-1)^{(k+1)/2}\left(\frac{1}{2}\right)^k \frac{\Gamma((d-1)/2)}{\Gamma(k+(d-1)/2)}\frac{w_{d-1}}{w_d}\frac{C_d}{\sqrt{2\pi}}\sum_{\beta=0}^{\infty} g(\beta,k).\end{aligned}$$

where $\asymp$ means the radio converge to a constant which doesn't depend on k or β as

$k \to \infty$. Here we introduced the function $g(\beta, k)$ for simplification, and it satisfies

$$\frac{g(\beta,k)}{g(\beta-1,k)} = -\frac{\hat{c}^2(2\beta+k-3)}{2\beta(2\beta+2k+d-2)},$$

which indicates that $g(\beta, k)$ decays at factorial rate when $\beta > \hat{c}^2/2$. If $k \gg \hat{c}^2/2$, $\beta \ll k$ regime dominates the summation.

Using Stirling's approximation, one can easily prove

$$\Gamma(k+x) \asymp \Gamma(k)k^x$$

When $k \gg d$,

$$\begin{aligned} g(\beta,k) &= \frac{(-1)^\beta \hat{c}^{2\beta+k}\Gamma(\beta+k/2)\Gamma(k+(d-1)/2)}{\beta!\Gamma(\beta+k+d/2)2^{\beta-k/2}} \\ &\asymp \left(-\frac{1}{4}\right)^\beta \hat{c}^{2\beta+k}\Gamma(k+(d-1)/2)\frac{2^{k/2}\Gamma(k/2)}{\Gamma(k)k^{d/2}\beta!} \\ &= \hat{c}^k\Gamma(k+(d-1)/2)\frac{2^{k/2}\Gamma(k/2)}{\Gamma(k)k^{d/2}}\left(-\frac{\hat{c}^2}{4}\right)^\beta\frac{1}{\beta!} \end{aligned}$$

This splits $g(\beta, k)$ into two parts: the first part depends only on k and the rest part only depends on β. The summation of the second part over β yields

$$\sum_{\beta=0}^{\infty}\left(-\frac{\hat{c}^2}{4}\right)^\beta\frac{1}{\beta!} = \exp(-\frac{\hat{c}^2}{4}),$$

Using Stirling's approximation

$$\gamma(x+1) \asymp \sqrt{2\pi x}(x/e)^x,$$

this leads to the expression for λ_k:

$$\begin{aligned}\lambda_k' &\asymp (-1)^{(k+1)/2}\left(\frac{1}{2}\right)^k \frac{\Gamma((d-1)/2)}{\Gamma(k+(d-1)/2)}\hat{c}^k\Gamma(k+(d-1)/2)\frac{2^{k/2}\Gamma(k/2)}{\Gamma(k)k^{d/2}}\exp(-\frac{\hat{c}^2}{4}) \\ &\asymp (-1)^{(k+1)/2}\left(\frac{\hat{c}}{2}\right)^k \frac{2^{k/2}\Gamma(k/2)}{\Gamma(k)k^{d/2}}\exp(-\frac{\hat{c}^2}{4}) \\ &\asymp (-1)^{(k+1)/2}\left(\frac{\hat{c}}{2}\sqrt{\frac{e}{k}}\right)^k k^{-d/2}\exp(-\frac{\hat{c}^2}{4})\end{aligned}$$

Similarly, the activation function of quasi neural network has the Tayler expansion

$$\begin{aligned}\tilde{\sigma}(x) &= \varsigma_\ell\varphi\left(\hat{c}t\right) + x\Phi\left(\hat{c}t\right) \\ &= \frac{t}{2} + \sum_{n=0}^{\infty}\frac{(-1)^n\hat{c}^{2n+1}}{2^{n+1}(n+1)!(2n+1)}t^{2n+2}.\end{aligned}$$

So $\lambda_k = 0$ when k is odd, and when k is even:

$$\lambda_k = (-1)^{(k+1)/2}\left(\frac{1}{2}\right)^k \frac{\Gamma((d-1)/2)}{\Gamma(k+(d-1)/2)}\frac{w_{d-1}}{w_d}\frac{C_d}{\sqrt{2\pi}}\sum_{\beta=0}^{\infty}\frac{(-1)^\beta\hat{c}^{2\beta+k}(2\beta+k-2)!!(2k+d-3)!!}{(2\beta)!!(2\beta+2k+d-2)!!}$$

Furthermore, when $k \gg d$,

$$\lambda_k \asymp (-1)^{k/2}k^{-\frac{d}{2}}\left(\frac{\hat{c}}{2}\sqrt{\frac{e}{k}}\right)^k \exp\left(-\frac{\hat{c}^2}{4}\right),$$

■

Computing covariance matrix

In this part, we prove Theorem 4.10 by computing $\Sigma^{(0)}$ and $\Sigma^{(1)}$.

Theorem 4.10 *NTK of a binary weight neural network can be decomposed using*

equation (4.9). If $k \gg d$, then

$$\mathrm{Poly}_1(k)(C)^{-k} \leq u_k \leq \mathrm{Poly}_2(k)(C)^{-k}$$

where $\mathrm{Poly}_1(k)$ *and* $\mathrm{Poly}_2(k)$ *denote polynomials of* k*, and* C *is a constant.*

We make use of the results in Section 4.8.2, and remind that λ_k, λ_k' depends on $\hat{c}$, we make this explicit as $\lambda_k(\hat{c}), \lambda_k'(\hat{c})$. We introduce an auxiliary parameter $w \sim \mathcal{N}(0, I)$, and denote $\tilde{c} = \sqrt{\frac{c\mathrm{Var}[\Theta]}{d\ \tilde{\varsigma}^2}} = \sqrt{\frac{\mathrm{Var}[\theta]}{1-\mathrm{Var}[\theta]}}, \tilde{w} = w/\|w\|_2$, then the decomposition of kernel (4.9) can be computed by

$$\begin{aligned}
\Sigma^{(1)} &= \mathbb{E}_\theta\left[\tilde{\sigma}\left(\mu\right)\tilde{\sigma}\left(\mu\right)\right] \\
&= \mathbb{E}_{w\sim\mathcal{N}(0,I)}\left[\tilde{\sigma}(\tilde{c}\langle w, x\rangle)\tilde{\sigma}(\tilde{c}\langle w, x'\rangle)\right] \\
&= \mathbb{E}_{\|w\|}\int \tilde{\sigma}(\tilde{c}\langle \tilde{w}, x\rangle)\tilde{\sigma}(\tilde{c}\langle \tilde{w}, x'\rangle)d\tau(\tilde{w}) \\
&= \mathbb{E}_{\|w\|}\sum_{k=0}^{\infty}(\lambda_k(\tilde{c}\|w\|))^2 N(p,k)P_k(\langle x, x'\rangle), \\
\Sigma^{(0)} &= \mathbb{E}_\theta\left[\tilde{\sigma}'\left(\mu\right)\tilde{\sigma}'\left(\mu\right)\right] \\
&= \mathbb{E}_{\|w\|}\sum_{k=0}^{\infty}(\lambda_k'(\tilde{c}\|w\|))^2 N(p,k)P_k(\langle x, x'\rangle).
\end{aligned}$$

First compute $\Sigma^{(0)}$. According to Lemma 16 in Bietti et al. [70],

$$u_{0,k} = \mathbb{E}_{w\sim\mathcal{N}(0,I)}[{\lambda_k'}^2] = \mathbb{E}_{\|w\|}[{\lambda_k'}^2].$$

Remind that

$$\lambda_k'(\tilde{c}\|w\|) \asymp (-1)^{k/2}k^{-d/2}\left(\frac{\tilde{c}\|w\|}{2}\sqrt{\frac{e}{k}}\right)^k \exp\left(-\frac{\tilde{c}^2\|w\|^2}{4}\right).$$

$$
\begin{aligned}
u_{0,k} &= \mathbb{E}_{\|w\|}(\lambda_k'(\tilde{c}\|w\|))^2 \\
&\asymp \mathbb{E}_{\|w\|} k^{-d} \left(\frac{\tilde{c}^2\|w\|^2 e}{4k}\right)^k \exp\left(-\frac{\tilde{c}^2\|w\|^2}{2}\right) \\
&= k^{-d}\,(k/e)^{-k}\,\mathbb{E}_{\tilde{c}\|w\|}\left(\frac{\tilde{c}^2\|w\|^2}{4}\right)^k \exp\left(-\frac{\tilde{c}^2\|w\|^2}{2}\right)
\end{aligned}
$$

Because $w \sim \mathcal{N}(0,1)$, $\|w\|_2^2$ satisfy Chi-square distribution, and its momentum generating function is

$$M_X(t) = \mathbb{E}[\exp(t\|w\|^2)] = (1-2t)^{-d/2}$$

It's k-th order derivative is

$$M_X^{(k)} = \mathbb{E}[\|w\|^{2k}\exp(t\|w\|^2)] = \frac{(d+2k-2)!!}{(d-2)!!}(1-2t)^{-\frac{d}{2}-k}$$

Let $t = -\tilde{c}^2/2$, we get

$$
\begin{aligned}
\mathbb{E}\left[\|w\|^{2k}\exp\left(-\frac{\tilde{c}^2\|w\|^2}{2}\right)\right] &= \frac{(d+2k-2)!!}{(1+\tilde{c}^2)^{d/2+k}(d-2)!!} \\
&\asymp 2^k\frac{\Gamma(k+d/2)}{\Gamma(d/2)}(1+\tilde{c}^2)^{-k-d/2} \\
&\asymp \left(\frac{2k}{(1+\tilde{c}^2)e}\right)^{d/2+k}\sqrt{\frac{1}{k}}
\end{aligned}
$$

so

$$
\begin{aligned}
u_{0,k} &\asymp \left(\frac{\tilde{c}}{2}\right)^{2k} k^{-d}\,(k/e)^{-k}\left(\frac{2k}{(1+\tilde{c}^2)e}\right)^{d/2+k} \\
&\asymp k^{-(d-1)/2}\left(\frac{\tilde{c}^2}{2(1+\tilde{c}^2)}\right)^k
\end{aligned}
$$

when k is odd, and 0 when k is even.

Similarly,

$$u_{1,k} \asymp \left(\frac{\tilde{c}}{2}\right)^{2k} k^{-d} (k/e)^{-k} \left(\frac{2k}{(1+\tilde{c}^2)e}\right)^{d/2+k}$$
$$\asymp k^{-(d-1)/2} \left(\frac{\tilde{c}^2}{2(1+\tilde{c}^2)}\right)^{k}$$

when k is even, and 0 when k is odd.

Finally, using the recurrence relation

$$tP_k(t) = \frac{k}{2k+d-3} P_{k-1}(t) + \frac{k+d-3}{2k+d-3} P_{k+1}(t)$$

taking them into (4.34) finishes the proof.

Gaussian kernel

$$\begin{aligned}\mathcal{K}_{RGauss}(x,x') &= \mathbb{E}[\mathcal{K}_{Gauss}(\kappa x, \kappa x')] \\ &= \mathbb{E}\left[\exp\left(-\frac{\kappa^2\|x-x'\|}{\xi^2}\right)\right] \\ &= \mathbb{E}\left[\exp\left(-\frac{\|x-x'\|}{(\xi/\kappa)^2}\right)\right]\end{aligned}$$

This indicates that this kernel can be decomposed using spherical harmonics (4.9), and

when $k \gg d$, the coefficient

$$
\begin{aligned}
u_k &= \mathbb{E}\left[\exp\left(-\frac{2\kappa^2}{\xi^2}\right)\left(\frac{\xi}{\kappa}\right)^{d-2} I_{k+d/2-1}\left(\frac{2\kappa^2}{\xi^2}\right)\Gamma\left(\frac{d}{2}\right)\right] \\
&\asymp \mathbb{E}\left[\exp\left(-\frac{2\kappa^2}{\xi^2}\right)\Gamma\left(\frac{d}{2}\right)\sum_{j=0}^{\infty}\frac{1}{j!\Gamma(k+d/2+j)}\left(\frac{\kappa^2}{\xi^2}\right)^{k+2j}\right] \\
&= \sum_{j=0}^{\infty}\frac{\Gamma(d/2)}{j!\Gamma(k+d/2+j)}\mathbb{E}\left[\left(\frac{\kappa^2}{\xi^2}\right)^{k+2j}\exp\left(-\frac{2\kappa^2}{\xi^2}\right)\right] \\
&= \sum_{j=0}^{\infty}\frac{\Gamma(d/2)}{j!\Gamma(k+d/2+j)}\frac{\Gamma(k+2j+d/2)}{\Gamma(d/2)}\left(\frac{2}{\xi^2}\right)^{k+2j}\left(\frac{1}{1+4/\xi^2}\right)^{(k+2j+d/2)} \\
&\asymp \left(\frac{2}{\xi^2}\right)^{k}\left(1+\frac{4}{\xi^2}\right)^{(-k-d/2)}\sum_{j=0}^{\infty}\frac{1}{j!}\left(\frac{k(2/\xi^2)^2}{(1+4/\xi^2)^2}\right)^{j} \\
&\asymp \left(\frac{2}{4+\xi^2}\right)^{k}\exp\left(\left(\frac{2}{4+\xi^2}\right)^{2}k\right).
\end{aligned}
$$

Note that $\frac{2}{4+\xi^2}\exp\left(\left(\frac{2}{4+\xi^2}\right)^2\right)$ is always smaller than 1 so u_k is always decreasing with k.

4.8.3 Additional Lemmas

Lemma 4.13 (Kolmogorov's Strong Law of Large Number (SLLN)) *Suppose $X_1, X_2, \ldots$ are independent variables such that $E[X_n] = \mu$ and $\sum_n \mathrm{Var}[X_n]/n^2 < \infty$. Then, $\frac{\sum_{i=1}^n X_i}{n} \to \mu$ a.e..*

Lemma 4.14 (Continuous mapping theorem) *Let $\{X_n\}$, X be random elements defined on a metric space S. Suppose a function $g : S \to S'$ (where S' is another metric*

Table 4.2: More results in UCI dataset experiment.

Classifier	Training			Testing		
	Accuracy	P90	P95	Accuracy	P90	P95
NN	96.19±8.03%	96.67%	91.11%	77.62±16.10%	73.33%	56.67%
BWNN	93.55±10.39%	84.44%	76.67%	77.83±16.57%	77.78%	54.44%
Laplacian	93.52±9.65%	85.56%	76.67%	81.62±14.72%	97.78%	91.11%
Gaussian	91.08±10.63%	76.67%	58.89%	81.40±14.85%	95.56%	87.78%

space) has the set of discontinuity points D_g such that $\Pr[X \in D_g] = 0$. *Then*

$$\begin{aligned} X_n \xrightarrow{\mathrm{d}} X \quad &\Rightarrow \quad g(X_n) \xrightarrow{\mathrm{d}} g(X) \\ X_n \xrightarrow{\mathrm{p}} X \quad &\Rightarrow \quad g(X_n) \xrightarrow{\mathrm{p}} g(X) \\ X_n \xrightarrow{a.s.} X \quad &\Rightarrow \quad g(X_n) \xrightarrow{a.s.} g(X) \end{aligned} \tag{4.39}$$

4.9 Additional information about numerical result

4.9.1 Toy dataset

In neural networks (NN) experiment, we used three layers with the first layer fixed. The number of hidden neural is 512. In neural network with binary weights (BWNN) experiment, the setup is the same as NN except the second layer is Binary. We used BinaryConnect method with stochastic rounding. We used gradient descent with learning rate searched from $10^{-3}, 10^{-2}, 10^{-1}$. For Laplacian kernel and Gaussian kernel, we searched kernel bandwidth from $2^{-2}\mu$ to $2^{2}\mu$ by power of 2, and μ is the medium of pairwise distance. The SVM cost value parameter is from 10^{-2} to 10^{4} by power of 2.

More results are listed in Table 4.2. Accuracy are shown in the format of mean ± std. P90 and P95 denotes the percentage of dataset that a model achieves at least 90% and 95% of the highest accuracy, respectively.

4.9.2 MNIST-like dataset

Similar to the toy dataset experiment, we used three layer neural networks with the first layer fixed, and only quantize the second layer. The number of neurons in the hidden layer is 2048. The batchsize if 100 and ADAM optimizer with learning rate 10^{-3} is used.

Chapter 5

Finite Overparameterization: Local adaptivity of Weight Decayed DNNs

5.1 Introduction

The theory based on NTK discussed in the last section can explain why neural networks generalize despite overparameterization, yet it fails to explain why neural networks outperform traditional machine learning methods including kernel methods. To solve this problem, in this paper, we study DNNs in nonparametric regression problems, aiming to separate it from other methods from the prospective of adaptivity.

Nonparametric regression is a classical branch of statistical theory and methods with more than half a century of associated literatures [91, 92, 93, 94, 95, 96, 97]. Nonparametric regression addresses the fundamental problem:

- Let $y_i = f(x_i) + \text{Noise}$ for $i = 1, ..., n$. How can we estimate a function f using data points $(x_1, y_1), ..., (x_n, y_n)$ in conjunction with the knowledge that f belongs to a

This work has been published as K. Zhang and Y.-X. Wang, *Deep learning meets nonparametric regression: Are weight-decayed dnns locally adaptive?*, *arXiv preprint arXiv:2204.09664* (2022).

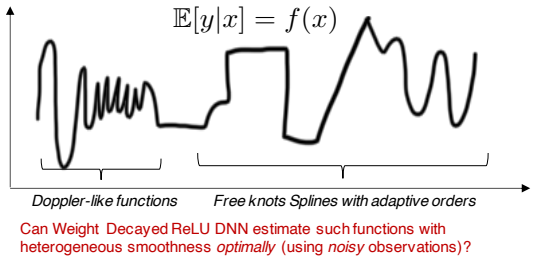


Figure 5.1: Illustration of a function with heterogeneous smoothness and the problem of locally adaptive nonparametric regression.

function class $\mathcal{F}$?

Function class $\mathcal{F}$ typically imposes only weak regularity assumptions such as smoothness, which makes nonparametric regression widely applicable to real-life applications under weak assumptions.

Local adaptivity. We say a nonparametric regression technique is *locally adaptive* if it can cater to local differences in smoothness, hence allowing more accurate estimation of functions with varying smoothness and abrupt changes. A subset of nonparametric regression techniques were shown to have the property of *local adaptivity* [98] in both theory and practice. These include wavelet smoothing [94], locally adaptive regression splines [LARS, 98], trend filtering [99, 100] and adaptive local polynomials [101, 102]. In light of such a distinction, it is natural to consider the following question: *Are NNs* locally adaptive*, i.e., optimal in learning functions with heterogeneous smoothness?*

This is a timely question to ask, partly because the bulk of recent theory of NN leverages its asymptotic Reproducing Kernel Hilbert Space (RKHS) in the overparameterized regime [68, 103, 104]. RKHS-based approaches, e.g., kernel ridge regression with any fixed kernels are *suboptimal* in estimating functions with heterogeneous smoothness [105]. Therefore, existing deep learning theory based on RKHS does not satisfactorily explain the advantages of neural networks over kernel methods.

We build upon the recent work of Suzuki [106] and Parhi et al. [107] who provided encouraging first answers to the question above. Specifically, Parhi et al. [107, Theorem

8] showed that a two-layer *truncated power function* activated neural network with a non-standard regularization is equivalent to the LARS. This connection implies that such NNs achieve the minimax rate for the (high order) bounded variation (BV) classes. A detailed discussion is provided in Section 5.8.1. Suzuki [106] showed that multilayer ReLU DNNs can achieve minimax rate for the Besov class, but requires the artificially imposed sparsity-level of the DNN weights to be calibrated according to parameters of the Besov class, thus is quite difficult to implement in practice.

Oono et al. [108], Liu et al. [109] replaced the sparse neural network with Resnet-style CNN and achieved the same rate, but they similarly require carefully choosing the number of parameters for *each* nonparametric class. We show that ℓ_2 regularization suffices for *mildly overparameterized* DNNs to achieve the optimal "local adaptive" rates for *many* nonparametric classes at the same time.

Parallel neural networks. We restrict our attention on a special network architecture called *parallel neural network* [110, 111] which learns an ensemble of subnetworks — each being a multilayer ReLU DNNs. Parallel NNs have been shown to be more well-behaved both theoretically [110, 112, 113, 111, 114] and empirically [115, 116]. On the other hand, many successful NN architectures such as SqueezeNet, ResNext and Inception (see [111] and the references therein) use the idea similar to a parallel NN.

Weight decay, also known as square ℓ_2 **regularization**, is one of the most popular regularization techniques for preventing overfitting in DNNs. It is called "weight decay" because each iteration of the gradient descent (or SGD) shrinks the parameter towards 0 multiplicatively. Many tricks in deep learning, including early stopping [117], quantization [73], and dropout [118] behaves like ℓ_2 regularization. Thus even though we focus on the exact minimizer of the regularized objective, it may explain the behavior of SGD in practice.

Summary of results. Our main contributions are:

Table 5.1: Comparison with the results in the literature

	# layers	Activation	Function space	Minimax rate	Remark
Parhi et al. [107, 119]	2	truncated power	BV^m	Yes	Non-standard activation and regularization (when $m > 1$).
Schmidt-Hieber [120]	≥ 3	ReLU	Hölder	Up to a log factor	With sparsity constraint.
Suzuki [106]	≥ 3	ReLU	Besov & m-Besov	Up to a log factor	With sparsity constraint.
Ours	≥ 3	ReLU	Besov & BV	Up to $n^{o(1)}$ factor	Requires only ℓ_2 regularization.

1. We prove that the (standard) ℓ_2 regularization in training an L-layer *parallel* ReLU-activated neural network is equivalent to a sparse ℓ_p penalty term (where $p = 2/L$) on the linear coefficients of a learned representation (Proposition 5.5).

2. We show that the estimation error of ℓ_2 regularized parallel NN can be close to the minimax rate for estimating functions in Besov space. Notably, the method can adapt to different smoothness parameter, which is not the case for many other methods.

3. We find that deeper models achieve closer to the optimal error rate. This result helps explain why deep neural networks can achieve better performance than shallow ones empirically.

Besides, we have the following technical contributions which could be of separate interest:

- We provide a way to bound the complexity of an overparameterized neural network. Specifically, we bound the metric entropy of a parallel neural network in Theorem 5.6, and the bound does not depend on the number of subnetworks.

- We propose a method to handle unconstrained function subspace when bounding the estimation error as in Equation (5.6).

The above results separate parallel NNs with any linear methods such as kernel ridge regression. To the best of our knowledge, we are the first to demonstrate that standard techniques (ℓ_2 regularization and ReLU activation) suffice for DNNs in achieving the optimal rates for estimating BV and Besov functions. The comparison with previous works is shown in Table 5.1. More discussion about related works are shown in Section 5.2.

5.2 Related works

NN and kernel methods. Jacot et al. [68] draws the connection between neural networks and kernel methods. However, it has been found that neural networks often outperform any kernel method, especially when the learning rate is relatively large [121]. A series of work tried to distinguish NN from kernel methods by providing examples of function spaces that NN provably outperform kernel methods [122, 123]. However, these papers did not consider the local adaptivity of nerual networks, which provides a more systematic explanation.

NN and splines. Besides Parhi et al. [107] which we discussed earlier, Parhi et al. [124, 119] also leveraged the connections between NNs and splines. Parhi et al. [124] focused on characterizing the variational form of multi-layer NN. Parhi et al. [119] showed that two-layer ReLU activated NN achieves minimax rate for a BV class of order 1 but did not cover multilayer NNs nor BV class with order > 1, which is our focus.

Weight-decay regularization with sparsity-inducing penalties. The connection between weight-decay regularization with sparsity-inducing penalties in two-layer NNs is folklore and used by Neyshabur et al. [125], Savarese et al. [126], Ongie et al. [127], Ergen et al. [128, 114], Parhi et al. [107, 119], Pilanci et al. [129]. The key underlying technique — an application of the AM-GM inequality (which we used in this paper as well) — can be traced back to Srebro et al. [130] (see a recent exposition by

Tibshirani [131]). Tibshirani [131] also generalized the result to multi-layered NNs, but with a simple (element-wise) connections. Besides, Ergen et al. [132] proved that training a two-layer convolution neural network (CNN) with weight decay induces sparsity, and points to a potential extension to these works including our work.

Finally, it was brought to our attention that while Savarese et al. [126] mainly consider two-layer NNs, a set of results about L-layer parallel NNs was presented in Appendix C of their paper, which essentially contains same arguments we used for proving the equivalence to an $\ell_{2/L}$ regularized optimization problem in Proposition 5.5. The difference is they applied the insight to understand the interpolation regime while we focused on analyzing MSE in the noisy case.

Proposition 5.5, is the Savarese et al. [126] showed that a parallel networks of depth L have an inductive bias for the $L_{2/L}$ sparse model, and explicit weight decay causes the solutions of these networks to have a sparse last layer with at most n nonzero weights.

Resnet-type convolution neural networks. A recent series of work [108, 109] proves that an arbitrary parallel neural network can be approximated by a resnet-type convolution neural networks. These works do not require the model to be sparse, thus are easier to train, yet they still require the architecture (the width and depth of each residual block, the number of residual blocks) to be tuned based on the dataset, and the estimation error analysis is based on the number of parameters. Besides, the number of residual block need to increase with n, making the entire too deep to train in practice.

Approximation and estimation. The approximation-theoretic and estimation-theoretic research for neural network has a long history too [133, 134, 135, 120, 106]. Most existing work considered the Holder, Sobolev spaces and their extensions, which contain only homogeneously smooth functions and cannot demonstrate the advantage of NNs over kernels. The exceptions including Suzuki [106], Oono et al. [108], Liu et al. [109] which, as we discussed earlier, requires modifications to NN architecture for each

Table 5.2: Symbols used in this paper

symbol	Meaning		
$a/\boldsymbol{a}/\mathbf{A}$	scalars / vectors / matrices.	$[a,b]$	$\{x \in \mathbb{R} : a \le x \le b\}$
$B^{\alpha}_{p,q}$	Besov space.	$[n]$	$\{x \in \mathbb{N} : 1 \le x \le n\}$.
$\lvert\cdot\rvert_{B^{\alpha}_{p,q}}$	Besov quasi-norm .	$\lVert\cdot\rVert_F$	Frobenius norm.
$\lVert\cdot\rVert_{B^{\alpha}_{p,q}}$	Besov norm.	$\lVert\cdot\rVert_p$	ℓ_p-norm.
$M_m(\cdot)$	m^{th} order Cardinal B-spline bases.	d	Dimension of input.
$M_{m,k,\boldsymbol{s}}(\cdot)$	m^{th} order Cardinal B-spline basis function of resolution k at position $\boldsymbol{s}$.	M	# subnetworks in a parallel NN.
		L	# layers in a (parallel) NN.
		w	Width of a subnetwork.
$\sigma(\cdot)$	ReLU activation function.	n	# samples.
$\mathbf{W}_j^{(\ell)}, \boldsymbol{b}_j^{(\ell)}$	Weight and bias in the ℓ-th layer in the j-th subnetwork.	$\mathbb{R}, \mathbb{Z}, \mathbb{N}$	Set of real numbers, integers, and nonnegative integers.

class. In contrast, we require tuning only the standard weight decay parameter. Most importantly, in all previously works, the estimation error of the model (eg. the covering number) depends on the number of nonzero parameters in the model, while our work provides a bound that depends on the norm of the weights instead of the number of subnetworks.

5.3 Preliminary

5.3.1 Notation and Problem Setup.

We denote regular font letters as scalars, bold lower case letters as vectors and bold upper case letters as matrices. $a \lesssim b$ means $a \le Cb$ for some constant C that does not depend on a or b, and $a \eqsim b$ denotes $a \lesssim b$ and $b \lesssim a$. See Table 5.2 for the full list of symbols used.

Let f_0 be the target function to be estimated. The training dataset is $\mathcal{D}_n := \{(\boldsymbol{x}_i, y_i), y_i = f_0(\boldsymbol{x}_i) + \epsilon_i, i \in [n]\}$, where x_i are fixed and ϵ_i are zero-mean, inde-

pendent Gaussian noises with variance σ^2. In the following discussion, we assume $\boldsymbol{x}_i \in [0,1]^d, f_0(x_i) \in [-1,1], \forall i$.

We will be comparing estimators under the mean square error (MSE), defined as $\mathrm{MSE}(\hat{f}) := \mathbb{E}_{\mathcal{D}_n} \frac{1}{n} \sum_{i=1}^n (\hat{f}(\boldsymbol{x}_i) - f_0(\boldsymbol{x}_i))^2$. The optimal worst-case MSE is described by $R(\mathcal{F}) := \min_{\hat{f}} \max_{f_0 \in \mathcal{F}} \mathrm{MSE}(\hat{f})$. We say that $\hat{f}$ is optimal if $\mathrm{MSE}(\hat{f}) \lesssim R(\mathcal{F})$. The empirical (square error) loss is defined as $\hat{L}(\hat{f}) := \frac{1}{n} \sum_{i=1}^n (\hat{f}(\boldsymbol{x}_i) - y_i)^2$. The corresponding population loss is $L(\hat{f}) := \mathbb{E}[\frac{1}{n} \sum_{i=1}^n (\hat{f}(\boldsymbol{x}_i) - y_i')^2 | \hat{f}]$ where y_i' are new data points. It is clear that $\mathbb{E}[L(\hat{f})] = \mathrm{MSE}[\hat{f}] + \sigma^2$.

5.3.2 Besov Spaces and Bound Variation Space

Besov space

Definition 5.1 *Modulus of smoothness: For a function $f \in L^p(\Omega)$ for some $1 \le p \le \infty$, the r-th modulus of smoothness is defined by*

$$w_{r,p}(f,t) = \sup_{h \in \mathbb{R}^d : \|h\|_2 \le t} \|\Delta_h^r(f)\|_p,$$

$$\Delta_h^r(f) := \begin{cases} \sum_{j=0}^r \binom{r}{j} (-1)^{r-j} f(x+jh), & \text{if } x \in \Omega, x + rh \in \Omega, \\ 0, & \text{otherwise.} \end{cases}$$

Definition 5.2 *Besov space: For $1 \le p, q \le \infty, \alpha > 0, r := \lceil \alpha \rceil + 1$, define*

$$|f|_{B_{p,q}^\alpha} = \begin{cases} \Big(\int_{t=0}^\infty (t^{-\alpha} w_{r,p}(f,t))^q \frac{dt}{t} \Big)^{\frac{1}{q}}, & q < \infty \\ \sup_{t>0} t^{-\alpha} w_{r,p}(f,t), & q = \infty, \end{cases}$$

and define the norm of Besov space as:

$$\|f\|_{B^{\alpha}_{p,q}} = \|f\|_p + |f|_{B^{\alpha}_{p,q}}.$$

A function f is in the Besov space $B^{\alpha}_{p,q}$ if $\|f\|_{B^{\alpha}_{p,q}}$ is finite.

Here $\alpha \geq 0$ determines the smoothness of functions, $1 \leq p \leq \infty$ determines the averaging (quasi-)norm over locations, $1 \leq q \leq \infty$ determines the averaging (quasi-)norm over scale which plays a relatively minor role. Smaller p is more forgiving to inhomogeneity and loosely speaking, when the function domain is bounded, smaller p induces a larger function space. On the other hand, it is easy to see from definition that $B^{\alpha}_{p,q} \subset B^{\alpha}_{p,q'}$, if $q < q'$. Without loss of generalizability, in the following discussion we will only focus on $B^{\alpha}_{p,\infty}$. When $p = 1$, the Besov space allows higher inhomogeneity, and it is more general than the Sobolev or Hölder space. Note that the Besov space for $0 < p, q < 1$ is also defined, but in this case it is a quasi-Banach space instead of a Banach space and will not be covered in this paper.

Functions in Besov space can be decomposed using B-spline basis functions. Any function f in Besov space $B^{\alpha}_{p,q}, \alpha > d/p$ can be decomposed using B-spline of order $m, m > \alpha$: let $\boldsymbol{x} \in \mathbb{R}^d$,

$$f(\boldsymbol{x}) = \sum_{k=0}^{\infty} \sum_{\boldsymbol{s} \in J(k)} c_{k,\boldsymbol{s}}(f) M_{m,k,\boldsymbol{s}}(\boldsymbol{x}) \tag{5.1}$$

where $J(k) := \{2^{-k}\boldsymbol{s} : \boldsymbol{s} \in [-m, 2^k + m]^d \subset \mathbb{Z}^d\}$, $M_{m,k,\boldsymbol{s}}(\boldsymbol{x}) := M_m(2^k(\boldsymbol{x} - \boldsymbol{s}))$, and $M_k(\boldsymbol{x}) = \prod_{i=1}^{d} M_k(x_i)$ is the cardinal B-spline basis function which can be expressed as

a polynomial:

$$\begin{aligned} M_m(x) &= \frac{1}{m!}\sum_{j=1}^{m+1}(-1)^j\binom{m+1}{j}(x-j)_+^m \\ &= ((m+1)/2)^m\frac{1}{m!}\sum_{j=1}^{m+1}(-1)^j\binom{m+1}{j}\Big(\frac{x-j}{(m+1)/2}\Big)_+^m, \end{aligned} \tag{5.2}$$

Furthermore, the norm of Besov space is equivalent to the sequence norm:

$$\|\{c_{k,\boldsymbol{s}}\}\|_{b^\alpha_{p,q}} := \Big(\sum_{k=0}^{\infty}(2^{(\alpha-d/p)k}\|\{c_{k,\boldsymbol{s}}(f)\}_{\boldsymbol{s}}\|_p)^q\Big)^{1/q} \eqsim \|f\|_{B^\alpha_{p,q}}.$$

See e.g. Dũng [136, Theorem 2.2] for the proof.

The Besov space is closely connected to other function spaces including the Hölder space ($\mathcal{C}^\alpha$) and the Sobolev space (W_p^α). Specifically, if the domain of the functions is d-dimensional [106, 137],

- $\forall\alpha\in\mathbb{N}$, $B^\alpha_{p,1}\subset W_p^\alpha\subset B^\alpha_{p,\infty}$, and $B^\alpha_{2,2}=W_2^\alpha$.
- For $0<\alpha<\infty$ and $\alpha\in\mathcal{N}, \mathcal{C}^\alpha = B^\alpha_{\infty,\infty}$.
- If $\alpha > d/p$, $B^\alpha_{p,q}\subset\mathcal{C}^0$.

Bounded variation (BV) space

is a more interpretable class of functions with spatially heterogeneous smoothness [94]. It is defined through the total variation (TV) of a function.

Definition 5.3 *Total Variation (TV): The total variation (TV) of a function f on an interval $[a,b]$ is defined as*

$$TV(f) = \sup_{\mathcal{P}}\sum_{i=1}^{n_{\mathcal{P}}-1}|f(x_{i+1})-f(x_i)|$$

where the $\mathcal{P}$ is taken among all the partitions of the interval $[a, b]$.

In many applications, functions with stronger smoothness conditions are needed, which can be measured by high order total variation.

Definition 5.4 *High order total variation: the m-th order total variation is the total variation of the $(m-1)$-th order derivative*

$$TV^{(m)}(f) = TV(f^{(m-1)})$$

Definition 5.5 *Bounded variation (BV): The m-th order bounded variation class is the set of functions whose total variation (TV) is bounded.*

$$BV(m) := \{f : TV(f^{(m)}) < \infty\}.$$

Bounded variation class is tightly connected to Besov classes. Specifically [138]:

$$B_{1,1}^{m+1} \subset BV(m) \subset B_{1,\infty}^{m+1} \tag{5.3}$$

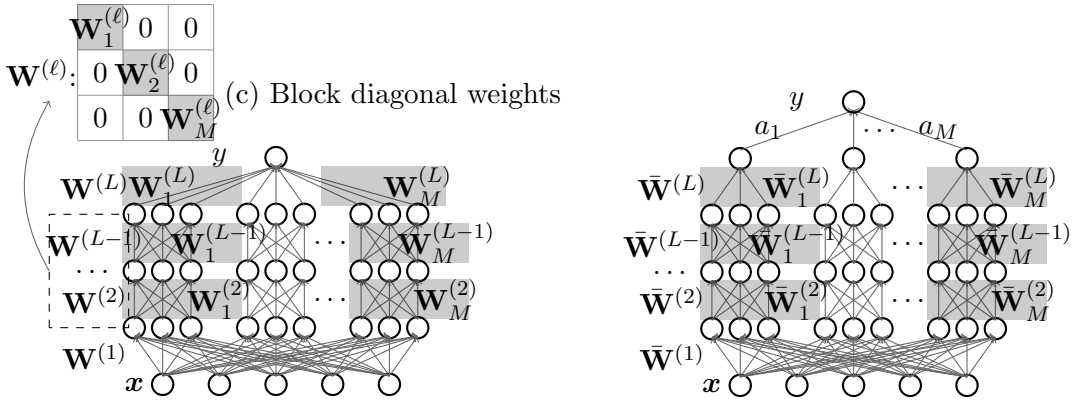
This allows the results derived for the Besov space to be easily applied to BV space.

Other Function Spaces

Definition 5.6 *Hölder space: let $m \in \mathbb{N}$, the m-th order Holder class is defined as*

$$\mathcal{C}^m = \left\{ f : \max_{|a|=k} \frac{|D^a f(x) - D^a f(z)|}{\|x - z\|_2} < \infty, \forall x, z \in \Omega \right\}$$

where D^a denotes the weak derivative.



(a) Parallel NN with ℓ_2 regularization

(b) Sparse Regression with Learned Representation

Figure 5.2: Parallel neural network and the equivalent sparse regression model we discovered.

Note that fraction order of Hölder space can also be defined. For simplicity, we will not cover that case in this paper.

Definition 5.7 *Sobolev space: let $m \in \mathcal{N}, 1 \leq p \leq \infty$, the Sobolev norm is defined as*

$$\|f\|_{W_p^m} := \left(\sum_{|a| \leq m} \|D^a f\|_p^p \right)^{1/p},$$

the Sobolev space is the set of functions with finite Sobolev norm:

$$W_p^m := \{f : \|f\|_{W_p^m} < \infty\}.$$

Minimax MSE

It is well known that minimax rate for Besov and 1D BV classes are $O(n^{-\frac{2\alpha}{2\alpha+d}})$ and $O(n^{-(2m+2)/(2m+3)})$ respectively . The minimax rate for *linear estimators* in 1D BV classes is known to be $O(n^{-(2m+1)/(2m+2)})$ [98, 94].

5.4 Main Results: Parallel ReLU DNNs

Consider a parallel neural network containing M multi layer perceptrons (MLP) with ReLU activation functions called *subnetworks*. Each subnetwork has width w and depth L. The input is fed to all the subnetworks, and the output of the parallel NN is the summation of the output of each subnetwork. The architecture of a parallel neural network is shown in Figure 5.2a. This parallel neural network is equivalent to a vanilla neural network with block diagonal weights in all but the first and the last layers (Figure 5.2(c)). Let $\mathbf{W}_j^{(\ell)}$ and $\boldsymbol{b}_j^{(\ell)}$ denote the weight and bias in the ℓ-th layer in the j-th subnetwork respectively. Training this model with ℓ_2 regularization returns:

$$\underset{\{\mathbf{W}_j^{(\ell)},\boldsymbol{b}_j^{(\ell)}\}}{\arg\min}\ \hat{L}(f)+\lambda\sum_{j=1}^{M}\sum_{\ell=1}^{L}\left\|\mathbf{W}_j^{(\ell)}\right\|_F^2, \tag{5.4}$$

where $f(x)=\sum_{j=1}^{M} f_k(x)$ denotes the parallel neural network, $f_j(\cdot)$ denotes the j-th subnetwork, and $\lambda>0$ is a fixed scaling factor. We choose not to regularize the bias terms $\boldsymbol{b}_j^{(\ell)}$ to provide a cleaner equivalent model (Proposition 5.5). If the bias terms are regularized, the result will be similar. Besides, we ignore the computation issue and focus on the global optimal solution to this problem. In practice, in deep neural network, the solution obtained using gradient descent-style methods are often close to the global optimal solution [65].

Theorem 5.1 *For any fixed $\alpha-d/p>1, q\geq 1, L\geq 3$, define $m=\lceil\alpha-1\rceil$. For any $f_0\in B_{p,q}^{\alpha}$, given an L-layer parallel neural network satisfying*

- *The width of each subnetwork is* ***fixed*** *satisfying $w\geq O(md)$. See Theorem 5.9 for the detail.*
- *The number of subnetworks is* ***large enough****: $M\gtrsim n^{\frac{1-2/L}{2\alpha/d+1-2/(pL)}}$.*

Under the assumption as in Lemma 5.18, with proper choice of the parameter of regularizaton λ that depends on $\mathcal{D}, \alpha, d, L$, the solution $\hat{f}$ parameterized by (5.4) satisfies

$$\mathrm{MSE}(\hat{f}) = C(w, L)\tilde{O}\big(n^{-\frac{2\alpha/d(1-2/L)}{2\alpha/d+1-2/(pL)}}\big) + e^{-c_6 L}. \tag{5.5}$$

where $\tilde{O}$ shows the scale up to a logarithmic factor, $c_6 > 0$ is a numerical constant from Theorem 5.9, $C(w, L) \eqsim (w^{4-4/L}L^{2-4/L})^{\frac{2\alpha/d}{2\alpha/d+1-2/(pL)}}$ depends polynomially on L.

We explain the proof idea in the next section,but defer the extended form of the theorem and the full proof to Section 5.8.5. Before that, we comment on a few interesting aspects of the result.

Near optimal rates and the effect of depth. The first term in the MSE bound is the estimation error and the second term is (part of) the approximation error of this NN. Recall that the minimax rate of a Besov class is $O(n^{-\frac{2\alpha}{2\alpha+d}})$. The gap between the estimation error and the minimax rate is because the minimax rate can be achieved by an ℓ_0 sparse model, while the parallel NN is equivalent to an ℓ_p sparse model (will be shown in Proposition 5.5), which is an approximation to ℓ_0. As the depth parameter L increases, $p = 2/L$ gets closer to 0, the MSE can get arbitrarily close to the minimax rate and the trailing constant term in (5.5) can be arbitrarily small. Close to the optimal rate can be achieved if we choose $L \gtrsim \log n$:

Corollary 5.2 *Under the conditions of Theorem 5.1, for any $f_0 \in B^{\alpha}_{p,q}$, there is a numerical constant C such that when we choose $C \log n \le L \le 100C \log n$,*

$$\mathrm{MSE}(\hat{f}) = \tilde{O}(n^{-\frac{2\alpha}{2\alpha+d}(1-o(1))}),$$

where $\tilde{O}$ hides only logarithmic factors and the $o(1)$ factor in the exponent is $O(1/\log(n))$.

Sparsity and comparison with standard NN. We also note that the result does not

depend on M as long as M is large enough. This means that the neural network can be arbitrarily overparameterized while not overfitting. The underlying reason is *sparsity*. As it will become clearer in Section 5.5.1, ℓ_2 regularized training of a parallel L-layer ReLU NNs is equivalent to a sparse regression problem with an ℓ_p penalty assigned to the coefficient vector of a learned dictionary. Here $p = 2/L$ which promotes even sparser solutions than an ℓ_1 penalty. Such ℓ_p sparsity does not exist in standard deep neural networks to the best of our knowledge, which indicates that parallel neural networks may be superior over standard neural networks in local adaptivity.

Adaptivity to function spaces. For any fixed $L, \tilde{m}$, our result shows the parallel neural network with width $w = O(\tilde{m}d)$ can achieve close to the minimax rate for any Besov class as long as $\alpha \leq \tilde{m}$. In other words, neural networks can adapt to smoothness parameter by tuning only the regularizaton parameter. As will be shown in Theorem 5.6, overestimating α with $\tilde{m}$ only changes the logarithmic terms in the MSE bound — a mild price to pay for a more adaptive method.

Hyperparameter tuning. We provide an explicit choice of λ in Lemma 5.18 underlying our theoretical result. Empirically, it can be determined empirically, e.g. using cross validation.

Fixed design v.s. random design. We mainly focus on bounding the error at sample covariates (the *fixed design* problem) to be comparable to classical nonparameteric regression results. One can easily apply the technique in this paper to achieve the estimation error bound on the random design problem:

Theorem 5.3 *Under the same condition as Theorem 5.1, the solution $\hat{f}$ parameterized by (5.4) satisfies*

$$\mathbb{E}_{\mathcal{D}}\mathbb{E}_f \mathrm{MSE}(\hat{f}) \leq \tilde{O}\left(\left(\frac{w^{4-4/L}L^{2-4/L}}{n^{1-2/L}}\right)^{\frac{2\alpha/d}{2\alpha/d+1-2/(pL)}} + e^{-c_6 L}\right)$$

where $\tilde{O}$ shows the scale up to a logarithmic factor, and c_6 is the constant defined in Theorem 5.9, $\mathbb{E}_{\mathcal{D}}$ indicates that the expectation is taken with respect to the training set $\mathcal{D}$, $\mathbb{E}_f$ indicates that the expectation is taken with respect to the domain of f.

The proof is similar to that of Theorem 5.1. The main difference lays in the proof of the estimation error. For $f_\perp$ part, the estimation error can be bounded using VC-dimension, which is 1. For $f_\parallel$ part, the estimation error can be bounded using its covering number, e.g. Lemma 8 in Schmidt-Hieber [120].

Representation learning and adaptivity. The results also shed a light on the role of representation learning in DNN's ability to adapt. Specifically, different from the two-layer NN in [107], which achieves the minimax rate of $BV(m)$ by choosing appropriate activation functions using each m, each subnetwork of a parallel NN can learn to approximate the spline basis of an arbitrary order, which means that if we choose L to be sufficiently large, such Parallel NN with optimally tuned λ is simultaneously near optimal for $m = 1, 2, 3, \ldots$. In fact, even if different regions of the space has different *orders* of smoothness, the paralle NN will still be able to learn appropriate basis functions in each local region. To the best of our knowledge, this is a property that none of the classical nonparametric regression methods possess.

Synthesis v.s. analysis methods. Our result could also inspire new ideas in estimator design. There are two families of methods in non-parametric estimation. One called *synthesis* framework which focuses on constructing appropriate basis functions to encode the contemplated structures and regress the data to such basis, e.g., wavelets [94]. The other is called *analysis* framework which uses analysis regularization on the data directly (see, e.g., RKHS methods [96] or trend filtering [99]). It appears to us that parallel NN is doing both simultaneously. It has a parametric family capable to synthesizing an $O(n)$ subset of an exponentially large family of basis, then *implicitly* use sparsity-

inducing analysis regularization to select the relevant basis functions. In this way the estimator does not actually have to explicitly represent that exponentially large set of basis functions, thus computationally more efficient.

Bounded variation classes. Thanks to the Besov space embedding of the BV class (5.3), our theorem also implies the result for the BV class in $1D$.

Corollary 5.4 *If the target function is in bounded variation class $f_0 \in BV(m)$, For any fixed $L \geq 3$, for a neural network satisfying the requirements in Theorem 5.1 with $d = 1$ and with proper choice of the regularization factor λ, the NN $\hat{f}$ parameterized by (5.8) satisfies*

$$\text{MSE}(\hat{f}) = C(w, L)\tilde{O}(n^{-\frac{(2m+2)(1-2/L)}{2m+3-2/L}}) + O(e^{-c_6 L}),$$

where $C(w, L)$ is the same as in (5.5) except replacing α with m.

It is known that any linear estimators such as kernel smoothing and smoothing splines cannot have an error lower than $O(n^{-(2m+1)/(2m+2)})$ for $BV(m)$ [94]. When $L > O(m^2)$, the first term in the MSE of NN decreases with n faster than that of the linear methods. When n is large enough, there exists L such that the MSE of NN is strictly smaller than that of any linear method. This partly explains the advantage of DNNs over kernels.

5.5 Proof Overview

We start by first proving that a parallel neural network trained with ℓ_2 regularization is equivalent to an ℓ_p-sparse regression problem with representation learning (Section 5.5.1); which helps decompose its MSE into an estimation error and approxmation error. Then we bound the two terms under an ℓ_p-sparse constrained problem setting in Section 5.5.2 and Section 5.5.3 respectively.

Notably, we adapted the generic statistical learning machinery (a self-bounding argument) for studying this constrained ERM problem [106, Proposition 4] to bound the estimation error. This adaption is non-trival because there is an *unconstrained* subspace with no bounded metric entropy. Specifically, Proposition 5.16 shows that the MSE of the regression problem can be bounded by

$$\mathrm{MSE}(\hat{f}) = O\Bigg(\underbrace{\inf_{f\in\mathcal{F}} \mathrm{MSE}(f)}_{\text{approximation error}} + \underbrace{\frac{\log \mathcal{N}(\mathcal{F}_{\|}, \delta, \|\cdot\|_\infty) + d(\mathcal{F}_\perp)}{n} + \delta}_{\text{estimation error}} \Bigg) \tag{5.6}$$

in which $\mathcal{F}$ decomposes into $\mathcal{F}_{\|} \times \mathcal{F}_\perp$, where $\mathcal{F}_\perp$ is an unconstrained subspace with finite dimension, and $\mathcal{F}_{\|}$ is a compact set in the orthogonal complement with a δ-covering number of $\mathcal{N}(\mathcal{F}_{\|}, \delta, \|\cdot\|_\infty)$ in $\|\cdot\|_\infty$-norm. This decomposes MSE into an approximation error and an estimation error. The novel analysis of these two represents the major technical contribution of this paper.

5.5.1 Equivalence to ℓ_p Sparse Regression

It is widely known that ReLU function is 1-homogeneous: $\sigma(ax) = a\sigma(x), \forall a \geq 0, x \in \mathbb{R}$. In any consecutive two layers in a neural network (or a subnetwork), one can multiply the weight and bias in one layer with a positive constant, and divide the weight in another layer with the same constant. The neural network after such transformation is equivalent to the original one:

$$\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} = \frac{1}{c}\mathbf{W}^{(2)}\sigma(c\mathbf{W}^{(1)}\boldsymbol{x} + c\boldsymbol{b}^{(1)}), \quad \forall c > 0, \boldsymbol{x}. \tag{5.7}$$

This property can be applied to *each subnetwork* (instead of the entire model in a standard NN), and we can reformulate (5.4) to an ℓ_p sparsity-regularized problem:

Proposition 5.5 *There exists an one-to-one mapping between $\lambda > 0$ and $\lambda' > 0$ such that (5.4) is equivalent to the following problem:*

$$\begin{aligned} &\underset{\{\bar{\mathbf{W}}_j^{(\ell)}, \bar{\boldsymbol{b}}_j^{(\ell)}, a_j\}}{\arg\min} \hat{L}\Big(\sum_{j=1}^{M} a_j \bar{f}_j\Big) + \lambda' \|\{a_j\}\|_{2/L}^{2/L} \\ &s.t.\ \|\bar{\mathbf{W}}_j^{(1)}\|_F \le c_1\sqrt{d}, \forall j \in [M];\ \|\bar{\mathbf{W}}_j^{(\ell)}\|_F \le c_1\sqrt{w}, \forall j \in [M], 2 \le \ell \le L, \end{aligned} \tag{5.8}$$

where $\bar{f}_j(\cdot)$ is a subnetwork with parameters $\bar{\mathbf{W}}_j^{(\ell)}, \bar{\boldsymbol{b}}_j^{(\ell)}$.

This equivalent model is demonstrated in Figure 5.2b. The proof, which we defer to Section 5.8.2, uses AM-GM inequality and the observation that the optimal solution will have norm-equalized weights per layer. The constraint $\|\bar{\mathbf{W}}_j^{(1)}\|_F \lesssim \sqrt{d}, \|\bar{\mathbf{W}}_j^{(\ell)}\|_F \lesssim \sqrt{w}, \forall \ell > 1$ is typical in deep learning for better numerical stability. The equivalent model in Proposition 5.5 is also a parallel neural network, but it appends one layer with parameters $\{a_k\}$ at the end of the neural network, and the constraint on the Frobenius norm is converted to the $2/L$ norm on the factors $\{a_k\}$. Since $L \gg 2$ in a typical application, $2/L \ll 1$ and this regularizer can enforce a sparser model than that in Section 5.8.1. The same technique can also be used to prove that an ℓ_2 constrained neural network is equivalent to the $\ell_{2/L}$ constrained model as in (5.9).

There are two useful implications of Proposition 5.5. First, it gives an intuitive explanation on how a regularized Parallel NN works. Specifically, it can be viewed as a sparse linear regression with representation learning. Secondly, the conversion into the constrained form allows us to decompose the MSE into two terms as in (5.6) and bound them separately.

We emphasize that Proposition 5.5 by itself is not new. The same result was previously obtained by Savarese et al. [126] Appendix C (see Section 5.2 for more details) and the key proof techniques date back to at least Burer et al. [139]. Our novel contribution

is to leverage this folklore equivalence for proving new learning bounds.

5.5.2 Estimation Error Analysis

Previous results that bound the covering number of neural networks [135, 106] depends on the width of the neural networks explicitly, which cannot be applied when analysing a potentially infinitely wide neural network. In this section, we leverage the ℓ_p-norm bounded coefficients to avoid the dependence in M in the covering number bound, and focus on a constrained optimization problem:

$$\underset{\{\bar{\mathbf{W}}_j^{(\ell)}, \bar{\boldsymbol{b}}_j^{(\ell)}, a_j\}}{\arg\min} \hat{L}\Big(\sum_{j=1}^{M} a_j \bar{f}_j\Big), \quad s.t. \|\{a_j\}\|_{2/L}^{2/L} \le P', \tag{5.9}$$

and $\{\bar{\mathbf{W}}_j^{(\ell)}, \bar{\boldsymbol{b}}_j^{(\ell)}\}$ satisfy the same constraint as in (5.8). The connection between the regularized problem and the constrained problem is defered to Lemma 5.18.

Theorem 5.6 *The covering number of the model defined in (5.9) apart from the bias in the last layer satisfies*

$$\log \mathcal{N}(\mathcal{F}, \delta) \lesssim w^{2+2/(1-2/L)} L^2 \sqrt{d} P'^{\frac{1}{1-2/L}} \delta^{-\frac{2/L}{1-2/L}} \log(wP'/\delta). \tag{5.10}$$

This theorem provides a bound of estimation error for an arbitrarily wide parallel neural network as long as the total Frobenius norm is bounded. The proof can be found in Section 5.8.3. It requires the following lemma, whose proof is deferred to Section 5.8.3:

Lemma 5.7 *Let $\mathcal{G} \subseteq \{\mathbb{R}^d \to [-c_3, c_3]\}$ be a set with covering number satisfying $\log \mathcal{N}(\mathcal{G}, \delta) \lesssim k \log(1/\delta)$ for some finite c_3, and for any $g \in \mathcal{G}, |a| \le 1$, we have $ag \in \mathcal{G}$. The covering*

number of $\mathcal{F} = \left\{ \sum_{i=1}^{M} a_i g_i \middle| g_i \in \mathcal{G}, \|a\|_p^p \leq P, 0 < p < 1 \right\}$ *for any* $P > 0$ *satisfies*

$$\log \mathcal{N}(\mathcal{F}, \epsilon) \lesssim k P^{\frac{1}{1-p}} (\delta / c_3)^{-\frac{p}{1-p}} \log(c_3 P / \delta)$$

up to a double logarithmic factor.

5.5.3 Approximation Error Analysis

The approximation error analysis involves two steps. We first analyse how a subnetwork can approximate a B-spline basis, which is defered to Section 5.8.4. Then we show that a sparse linear combination of B-spline bases approximates Besov functions. Both add up to the total error in approximating Besov functions with a parallel neural network (Theorem 5.9).

Proposition 5.8 *Let* $\alpha - d/p > 1, r > 0$. *For any function in Besov space* $f_0 \in B_{p,q}^{\alpha}$ *and any positive integer* $\bar{M}$, *there is an* $\bar{M}$*-sparse approximation using B-spline basis of order* m *satisfying* $0 < \alpha < \min(m, m - 1 + 1/p)$*:* $\check{f}_{\bar{M}} = \sum_{i=1}^{\bar{M}} a_{k_i, \boldsymbol{s}_i} M_{m, k_i, \boldsymbol{s}_i}$ *for any positive integer* $\bar{M}$ *such that the approximation error is bounded as* $\|\check{f}_{\bar{M}} - f_0\|_r \lesssim \bar{M}^{-\alpha/d} \|f_0\|_{B_{p,q}^{\alpha}}$, *and the coefficients satisfy*

$$\|\{2^{k_i} a_{k_i, \boldsymbol{s}_i}\}_{k_i, \boldsymbol{s}_i}\|_p \lesssim \|f_0\|_{B_{p,q}^{\alpha}}.$$

The proof as well as the remark can be found in Section 5.8.4.

Theorem 5.9 *Under the same condition as Proposition 5.8, for any positive integer* $\bar{M}$, *any function in Besov space* $f_0 \in B_{p,q}^{\alpha}$ *can be approximated by a parallel neural network with no more than* $O(\bar{M})$ *number of subnetworks satisfying:*

1. *Each subnetwork has width* $w = O(md)$ *and depth* L.

2. *The weights in each layer satisfy* $\|\bar{\mathbf{W}}_k^{(\ell)}\|_F \leq O(\sqrt{w})$ *except the first layer* $\|\bar{\mathbf{W}}_k^{(1)}\|_F \leq O(\sqrt{d})$,

3. *The scaling factors have bounded* $2/L$*-norm:* $P' := \|\{a_j\}\|_{2/L}^{2/L} \lesssim \bar{M}^{1-2/(pL)}$.

4. *The approximation error is bounded by*

$$\|\tilde{f} - f_0\|_r \leq (c_4 \bar{M}^{-\alpha/d} + c_5 e^{-c_6 L}) \|f\|_{B_{p,q}^{\alpha}}$$

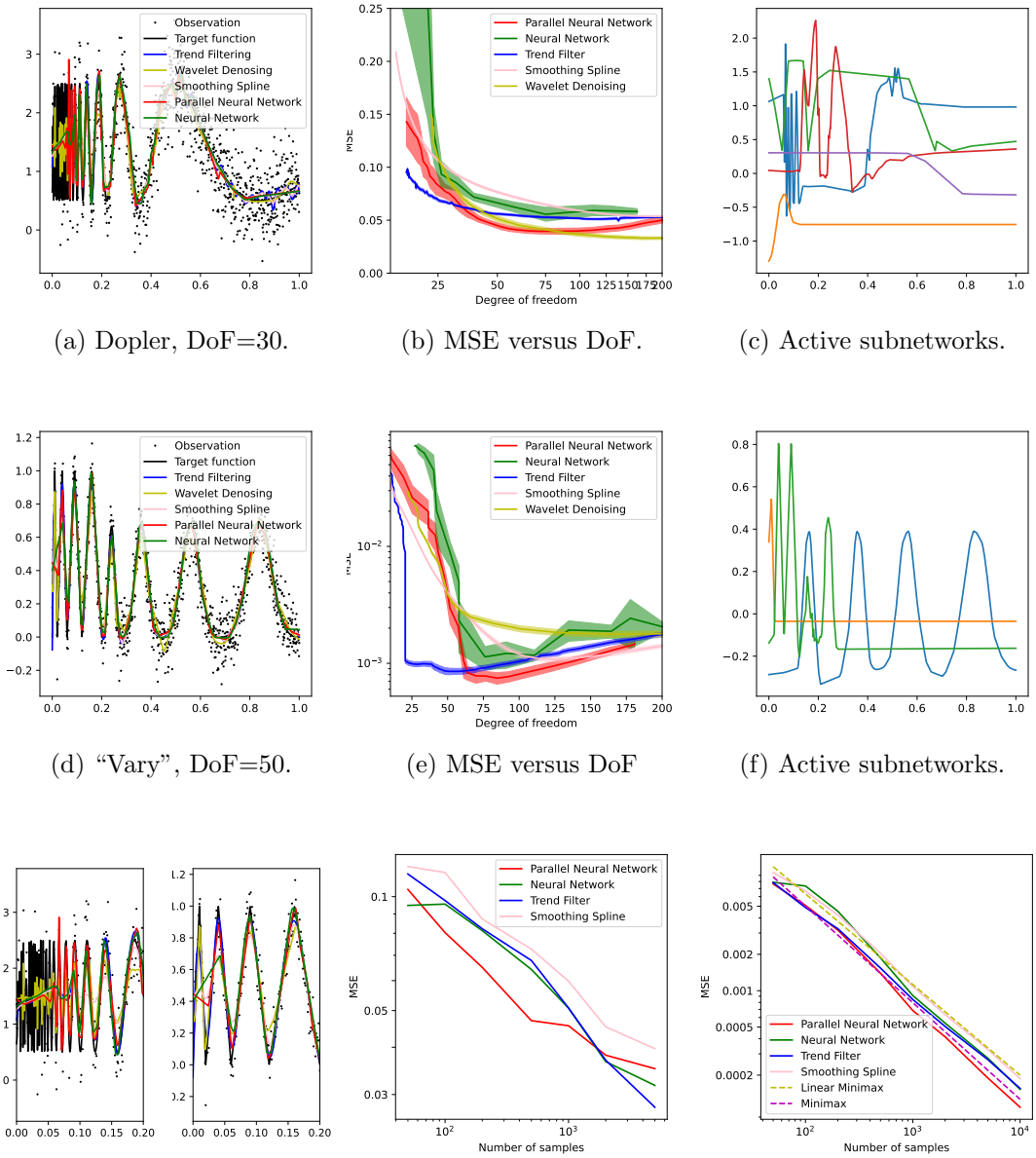
where c_4, c_5, c_6 *are constants that depend only on* m, d *and* p.

Here $\bar{M}$ is the number of "active" subnetworks, which is not to be confused with the number of subnetworks at initialization. The proof can be found in Section 5.8.4.

Using the estimation error in Theorem 5.6 and approximation error in Theorem 5.9, by choosing $\bar{M}$ to minimax the total error, we can conclude the sample complexity of parallel neural networks using ℓ_2 regularization, which is the main result (Theorem 5.1) of this paper. See Section 5.8.5 for the detail.

5.6 Experiment

We empirically compare a parallel neural network (PNN) and a vanilla ReLU neural network (NN) with smoothing spline, trend filtering (TF) [99], and wavelet denoising. Trend filtering can be viewed as a more efficient discrete spline version of locally adaptive regression spline and enjoys the same optimal rates for the BV classes. Wavelet denoising is also known to be minimax-optimal for the BV classes. The results are shown in Figure 5.3. We use two target functions: a Doppler function whose frequency is decreasing(Figure 5.3(a)-(c)(h)), and a combination of piecewise linear function and piecewise



(a) Dopler, DoF=30. (b) MSE versus DoF. (c) Active subnetworks.

(d) "Vary", DoF=50. (e) MSE versus DoF (f) Active subnetworks.

(g) Zoom in to (a)(d) (h) MSE versis n, Dopler (i) MSE versis n, "Vary"

Figure 5.3: Numerical experiment results of the Doppler function (a-c,h), and "vary" function (d-f,g). All the "active" subnetworks are plotted in (c)(f). The horizontal axis in (b) is not linear.

cubic function, or "vary" function (Figure 5.3(d)-(f)(i)). We repeat each experiment 10 times and take the average. The shallow area in Figure 5.3(b)(e) shows 95% confidence interval by inverting the Wald's test. The degree of freedom (DoF) is computed based

on Tibshirani [140].

As can be shown in the figure, both TF and wavelet denoising can adapt to the different levels of smoothness in the target function, while smoothing splines tend to be oversmoothed where the target function is less smooth (the left side in (a)(d), enlarged in (g)). The prediction of PNN is similar to TF and wavelet denoising and shows local adaptivity. Besides, the MSE of PNN almost follows the same trend as TF and wavelet denoising which is consistent with our theoretical understanding that the error rate of neural network is closer to locally adaptive methods. Notably PNN, TF and wavelet denoising achieve lower error at a much smaller degree-of-freedom than smoothing splines.

There are some mild drops in the best MSE one can achieve with Parallel NN vs TF in both examples. We are surprised that the drop is small because Parallel NN needs to learn the basis functions that TF essentially hard-coded. The additional price to pay for using a more adaptive and more flexible representation learning method seems not high at all.

In Figure 5.3(c)(f), we give the output *all* the "active" subnetwork, i.e. the subnetworks whose output is not a constant. Notice that the number of active subnetworks is much smaller than the initialization. This is because ℓ_2 regularization in weights induces ℓ_p sparsity and the weight in most of the subnetworks reduces towards 0 after training. More details are shown in Section 5.9.

In Figure 5.3(h)(i), we plot the MSE versus the number of training samples for "Doppler" and "Vary" respectively. It is clear that parallel NN works the best overall. In (i), we further compare the scaling of the MSE against the minimax rate ($n^{-4/5}$) and the minimax linear rate ($n^{-3/4}$), i.e., the best rate kernel methods could achieve. As is predicted by our theory, when n is large, the MSE of parallel neural networks and trend filtering decreases at almost the same rate as the minimax rate, while smoothing splines, as expected, is converging at the (suboptimal) minimax linear rate. Interestingly,

vanilla NN seems to converge at the optimal rate too on this example. It remains an open question whether vanila NN is merely "lucky" on this example, or it also achieves the minimax rate for all functions in BV(m).

5.7 Conclusion and Discussion

In this paper, we show that a deep parallel neural network can be locally adaptive with standard ℓ_2 regularization. This confirms that neural networks can be nearly optimal in learning functions with heterogeneous smoothness which separates them from kernel methods.

Specifically, we prove that training an L layer parallel neural network with standard ℓ_2 regularization is equivalent to an $\ell_{2/L}$-penalized regression model with representation learning. Since in typical application $L \gg 2$, standard regularization promotes a sparse linear combination of the learned bases. Using this method, we proved that a parallel neural network can achieve close to the minimax rate in the Besov space and bounded variation (BV) space by tuning the regularization factor.

Our result reveals that one do not need to specify the smoothness parameter α (or m) when training a parallel neural network. With only an estimation of the upper bound of α (or m), parallel neural networks can adapt to different degree of smoothness, or choose different parameters for different regions of the domain of the target function. This property shows the strong adaptivity of deep neural networks.

On the other hand, as the depth of neural network L increases, $2/L$ tends to 0 and the error rate moves closer to the minimax rate of Besov and BV space. This indicates that when the sample size is large enough, deeper models have smaller error than shallower models, and helps explain why empirically deep neural networks has better performance than shallow neural networks.

5.8 Proofs of technical results

5.8.1 Two-layer Neural Network with Truncated Power Activation Functions

We start by recapping the result of Parhi et al. [107] and formalizing its implication in estimating BV functions. Parhi et al. [107] considered a two layer neural network with truncated power activation function. Let the neural network be

$$f(x) = \sum_{j=1}^{M} v_j \sigma^m(w_j x + b_j) + c(x), \tag{5.11}$$

where w_j, v_j denote the weight in the first and second layer respectively, b_j denote the bias in the first layer, $c(x)$ is a polynomial of order up to m, $\sigma^m(x) := \max(x, 0)^m$. Parhi et al. [107, Theorem 8] showed that when M is large enough, The optimization problem

$$\min_{\boldsymbol{w},\boldsymbol{v}} \hat{L}(f) + \frac{\lambda}{2} \sum_{j=1}^{M} (|v_j|^2 + |w_j|^{2m}) \tag{5.12}$$

is equivalent to the locally adaptive regression spline:

$$\min_{f} \hat{L}(f) + \lambda TV(f^{(m)}(x)), \tag{5.13}$$

which optimizes over arbitrary functions that is m-times weakly differentiable. The latter was studied in Mammen et al. [98], which leads to the following MSE:

Theorem 5.10 *Let $M \geq n - m$, and $\hat{f}$ be the function (5.11) parameterized by the minimizer of* (5.12)*, then*

$$\mathrm{MSE}(\hat{f}) = O(n^{-(2m+2)(2m+3)}).$$

We show a simpler proof in the univariate case due to Tibshirani [141]:

Proof: As is shown in Parhi et al. [107, Theorem 8], the minimizer of (5.12) satisfy

$$|v_j| = |w_j|^m, \forall k$$

so the TV of the neural network f_{NN} is

$$\begin{aligned} TV^{(m)}(f_{NN}) &= TV^{(m)}c(x) + \sum_{j=1}^{M} |v_j||w_j|^m TV^{(m)}(\sigma^{(m)}(x)) \\ &= \sum_{j=1}^{M} |v_j||w_j|^m \\ &= \frac{1}{2}\sum_{j=1}^{M}(|v_j|^2 + |w_j|^{2m}) \end{aligned}$$

which shown that (5.12) is equivalent to the locally adaptive regression spline (5.13) as long as the number of knots in (5.13) is no more than M. Furthermore, it is easy to check that any spline with knots no more than M can be expressed as a two layer neural network (5.12). It suffices to prove that the solution in (5.13) has no more than $n - m$ number of knots.

Proposition 1 in Mammen et al. [98] showed that there is a solution to (5.13) $\hat{f}(x)$ such that $\hat{f}(x)$ is a mth order spline with a finite number of knots but did not give a bound. Let the number of knots be M, we can represent $\hat{f}$ using the truncated power basis

$$\hat{f}(x) = \sum_{j=1}^{M} a_j(x - t_j)_+^m + c(x) := \sum_{j=1}^{M} a_j \sigma_j^{(m)}(x) + c(x)$$

where t_j are the knots, $c(x)$ is a polynomial of order up to m, and define $\sigma_j^{(m)}(x) = (x - t_j)_+^m$.

Mammen et al. [98] however did not give a bound on M. Parhi et al. [107]'s Theorem 1

implies that $M \leq n - m$. Its proof is quite technical and applies more generally to a higher dimensional generalization of the BV class.

Tibshirani [141] communicated to us the following elegant argument to prove the same using elementary convex analysis and linear algebra, which we present below.

Define $\Pi_m(f)$ as the $L^2(P_n)$ projection of f onto polynomials of degree up to m, $\Pi_m^{\perp}(f) := f - \Pi_m(f)$. It is easy to see that

$$\Pi_m^{\perp} f(x) = \sum_{j=1}^{M} a_j \Pi_m^{\perp} \sigma_j^{(m)}(x)$$

Denote $f(x_{1:n}) := \{f(x_1), \ldots, f(x_n)\} \in \mathbb{R}^n$ as a vector of all the predictions at the sample points.

$$\Pi_m^{\perp} \hat{f}(x_{1:n}) = \sum_{j=1}^{M} a_j \Pi_m^{\perp} \sigma_j^{(m)}(x_{1:n}) \in \Pi_m^{\perp} \mathrm{Conv}\{\pm \sigma_j^{(m)}(x_{1:n})\} \cdot \sum_{j=1}^{M} |a_j|$$
$$\in \mathrm{Conv}\{\pm \Pi_m^{\perp} \sigma_j^{(m)}(x_{1:n})\} \cdot \sum_{j=1}^{M} |a_j|$$

where Conv denotes the convex hull of a set. The convex hull $\mathrm{Conv}\{\pm \sigma_j^{(m)}(x_{1:n})\} \cdot \sum_{j=1}^{M} |a_j|$ is an n-dimensional space, and polynomials of order up to m is an $m+1$ dimensional space, so the set defined above has dimension $n-m-1$. By Carathéodory's theorem, there is a subset of points in this space

$$\{\Pi_m^{\perp} \sigma_{j_k}^{(m)}(x_{1:n})\} \subseteq \{\Pi_m^{\perp} \sigma_j^{(m)}(x_{1:n})\}, 1 \leq k \leq n - m$$

such that

$$\Pi_m^{\perp} f(x) = \sum_{k=1}^{n-m} \tilde{a}_k \Pi_m^{\perp} \sigma_{j_k}^{(m)}(x), \sum_{k=1}^{n-m} |a_k| \leq 1$$

In other word, there exist a subset of knots $\{\tilde{t}_j, j \in [n-m]\}$ that perfectly recovers

$\Pi_m^{\perp}\hat{f}(x)$ at all the sample points, and the TV of this function is no larger than $\hat{f}$.

This shows that

$$\tilde{f}(x) = \sum_{j=1}^{n-m} \tilde{a}_j (x - t_j)_+^m, s.t. \tilde{f}(x_i) = f(x_i)$$

for all x_i in n onbservation points.

The MSE of locally adaptivity regressive spline (5.13) was studied in Mammen et al. [98, Section 3], which equals the error rate given in Theorem 5.10. ■

This indicates that the neural network (5.11) is minimax optimal for $BV(m)$.

Let us explain a few the key observations behind this equivalence. (a) The truncated power functions (together with an mth order polynomial) spans the space of an mth order spline. (b) The neural network in (5.11) is equivalent to a free-knot spline with M knots (up to reparameterization). (c) A solution to (5.13) is a spline with at most $n-m$ knots [107, Theorem 8]. (d) Finally, by the AM-GM inequality

$$|v_j|^2 + |w_j|^{2m} \geq 2|v_j||w_j|^m = 2|c_j|$$

where $c_j = v_j|w_j|^m$ is the coefficient of the corresponding jth truncated power basis. The mth order total variation of a spline is equal to $\sum_j |c_j|$. It is not hard to check that the loss function depends only on c_j, thus the optimal solution will always take "=" in the AM-GM inequality.

5.8.2 Equivalence Between Parallel Neural Networks and p-norm Penalized Problems

Proof of Proposition 5.5

We make use of the property from (5.7) to minimize the constraint term in (5.9) while keeping this neural network equivalent to the original one. Specifically, let $\mathbf{W}^{(1)}, \boldsymbol{b}^{(1)}, \dots \mathbf{W}^{(L)}, \boldsymbol{b}^{(L)}$ be the parameters of an L-layer neural network.

$$f(x) = \mathbf{W}^{(L)}\sigma(\mathbf{W}^{(L-1)}\sigma(\dots\sigma(\mathbf{W}^{(1)}x + \boldsymbol{b}^{(1)})\dots) + \boldsymbol{b}^{(L-1)}) + \boldsymbol{b}^{(L)},$$

which is equivalent to

$$f(x) = \alpha_L\tilde{\mathbf{W}}^{(L)}\sigma(\alpha_{L-1}\tilde{\mathbf{W}}^{(L-1)}\sigma(\dots\sigma(\alpha_1\tilde{\mathbf{W}}^{(1)}x + \tilde{\boldsymbol{b}}^{(1)})\dots) + \tilde{\boldsymbol{b}}^{(L-1)}) + \tilde{\boldsymbol{b}}^{(L)},$$

as long as $\alpha_\ell > 0, \prod_{\ell=1}^{L}\alpha^L = \prod_{\ell=1}^{L}\|\mathbf{W}^{(\ell)}\|_F$, where $\tilde{\mathbf{W}}^{(\ell)} := \frac{\mathbf{W}^{(\ell)}}{\|\mathbf{W}^{(\ell)}\|_F}$. By the AM-GM inequality, the ℓ_2 regularizer of the latter neural network is

$$\sum_{\ell=1}^{L}\|\alpha_\ell\tilde{\mathbf{W}}^{(\ell)}\|_F^2 = \sum_{\ell=1}^{L}\alpha_\ell^2 \geq L\left(\prod_{\ell=1}^{L}a_\ell\right)^{2/L} = L\left(\prod_{\ell=1}^{L}\|\mathbf{W}^{(\ell)}\|_F\right)^{2/L}$$

and equality is reached when $\alpha_1 = \alpha_2 = \cdots = \alpha_L$. In other word, in the problem (5.4), it suffices to consider the network that satisfies

$$\|\mathbf{W}_j^{(1)}\|_F = \|\mathbf{W}_j^{(2)}\|_F = \cdots = \|\mathbf{W}_j^{(L)}\|_F, \forall j \in [M], \ell \in [L]. \tag{5.14}$$

Using (5.7) again, one can find that the neural network is also equivalent to

$$f(x) = \sum_{j=1}^{M} a_j \bar{\mathbf{W}}^{(L)} \sigma(\bar{\mathbf{W}}_j^{(L-1)} \sigma(\dots \sigma(\bar{\mathbf{W}}_j^{(1)} x + \bar{\boldsymbol{b}}_j^{(1)}) \dots) + \bar{\boldsymbol{b}}_j^{(L-1)}) + \bar{\boldsymbol{b}}_j^{(L)},$$

where

$$\|\bar{\mathbf{W}}_j^{(\ell)}\|_F \leq \beta^{(\ell)}, a_j = \frac{\prod_{\ell=1}^{L} \|\mathbf{W}_j^{(\ell)}\|_F}{\prod_{\ell=1}^{L} \beta^{(\ell)}} = \frac{\|\mathbf{W}_j^{(1)}\|_F^L}{\prod_{\ell=1}^{L} \beta^{(\ell)}} = \frac{(\sum_{\ell=1}^{L} \|\mathbf{W}_j^{(\ell)}\|_F^2 / L)^{L/2}}{\prod_{\ell=1}^{L} \beta^{(\ell)}}, \tag{5.15}$$

where the last two equality comes from the assumption (5.14). Choosing $\beta^{(\ell)} = c_1\sqrt{w}$ expect $\ell = 1$ where $\beta^{(1)} = c_1\sqrt{d}$, and scaling $\bar{\boldsymbol{b}}^{(\ell)}$ accordingly and taking the regularizer in (5.4) into (5.15) finishes the proof.

5.8.3 Covering Number of Parallel Neural Networks

Proof of Theorem 5.6

The proof relies on the covering number of each subnetwork in a parallel neural network (Lemma 5.11), observing that $|f(x)| \leq 2^{L-1} w^{L-1} \sqrt{d}$ under the condition in Lemma 5.11, and then apply Lemma 5.7. We argue that our choice of condition on $\|\boldsymbol{b}^{(\ell)}\|_2$ in Lemma 5.11 is sufficient to analyzing the model apart from the bias in the last layer, because it guarantees that $\sqrt{w}\|\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(x)\|_2 \leq \|\boldsymbol{b}^{(\ell)}\|_2$. This leads to

$$\|\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(\boldsymbol{x})\|_\infty \leq \|\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(\boldsymbol{x})\|_2 \leq \sqrt{w}\|\boldsymbol{b}^{(\ell)}\|_2 \leq \|\boldsymbol{b}^{(\ell)}\|_\infty$$

If this condition is not met, $\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(\boldsymbol{x}) + b^{(\ell)}$ is either always positive or always negative for all feasible $\boldsymbol{x}$ along at least one dimension. If $(\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(\boldsymbol{x}) + b^{(\ell)})_i$ is always negative, one can replace $b^{(\ell)})_i$ with $-\max_{\boldsymbol{x}} \|\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(\boldsymbol{x})\|_\infty$ without changing the output of this model for any feasible $\boldsymbol{x}$. If $(\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(\boldsymbol{x}) + b^{(\ell)})_i$ is always positive, one can replace $b^{(\ell)})_i$

with $\max_{\boldsymbol{x}} \|\mathbf{W}^{(\ell)}\mathcal{A}_{\ell-1}(\boldsymbol{x})\|_\infty$, and adjust the bias in the next layer such that the output of this model is not changed for any feasible $\boldsymbol{x}$. In either cases, one can replace the bias $\boldsymbol{b}^{(\ell)}$ with another one with smaller norm while keeping the model equivalent except the bias in the last layer.

Lemma 5.11 *Let $\mathcal{F} \subseteq \{f : R^d \to \mathbb{R}\}$ denote the set of L-layer neural network (or a subnetwork in a parallel neural network) with width w in each hidden layer. It has the form*

$$\begin{aligned}
&f(x) = \mathbf{W}^{(L)}\sigma(\mathbf{W}^{(L-1)}\sigma(\dots\sigma(\mathbf{W}^{(1)}x + \boldsymbol{b}^{(1)})\dots) + \boldsymbol{b}^{(L-1)}) + \boldsymbol{b}^{(L)},\\
&\mathbf{W}^{(1)} \in \mathbb{R}^{w\times d}, \|\mathbf{W}^{(1)}\|_F \le \sqrt{d}, \boldsymbol{b}^{(1)} \in \mathbb{R}^w, \|\boldsymbol{b}^{(1)}\|_2 \le \sqrt{dw},\\
&\mathbf{W}^{(\ell)} \in \mathbb{R}^{w\times w}\|\mathbf{W}^{(\ell)}\|_F \le \sqrt{w}, \boldsymbol{b}^{(\ell)} \in \mathbb{R}^w, \|\boldsymbol{b}^{(\ell)}\|_2 \le 2^{\ell-1}w^{\ell-1}\sqrt{dw}, \quad \forall \ell = 2,\dots L-1,\\
&\mathbf{W}^{(L)} \in \mathbb{R}^{1\times w}, \|\mathbf{W}^{(L)}\|_F \le \sqrt{w}, b^{(L)} = 0
\end{aligned} \tag{5.16}$$

and $\sigma(\cdot)$ is the ReLU activation function, the input satisfy $\|x\|_2 \le 1$, then the supremum norm δ-covering number of $\mathcal{F}$ obeys

$$\log \mathcal{N}(\mathcal{F}, \delta) \le c_7 L w^2 \log(1/\delta) + c_8$$

where c_7 is a constant depending only on d, and c_8 is a constant that depend on d, w and L.

Proof of Lemma 5.11

First study two neural networks which differ by only one layer. Let g_ℓ, g'_ℓ be two neural networks satisfying (5.16) with parameters $\mathbf{W}_1, \boldsymbol{b}_1, \dots, \mathbf{W}_L, \boldsymbol{b}_L$ and $\mathbf{W}'_1, \boldsymbol{b}'_1, \dots, \mathbf{W}'_L, \boldsymbol{b}'_L$ respectively. Furthermore, the parameters in these two models are the same except the

ℓ-th layer, which satisfy

$$\|\mathbf{W}_\ell - \mathbf{W}'_\ell\|_F \leq \epsilon, \|\boldsymbol{b}_\ell - \boldsymbol{b}'_\ell\|_2 \leq \tilde{\epsilon}.$$

Denote the model as

$$g_\ell(x) = \mathcal{B}_\ell(\mathbf{W}_\ell \mathcal{A}_\ell(\boldsymbol{x}) + \boldsymbol{b}_\ell), g'_\ell(x) = \mathcal{B}_\ell(\mathbf{W}'_\ell \mathcal{A}_\ell(\boldsymbol{x}) + \boldsymbol{b}'_\ell)$$

where $\mathcal{A}_\ell(\boldsymbol{x}) = \sigma(\mathbf{W}_{\ell-1}\sigma(\dots\sigma(\mathbf{W}_1 x + \boldsymbol{b}_1)\dots) + \boldsymbol{b}_{\ell-1})$ denotes the first $\ell - 1$ layers in the neural network, and $\mathcal{A}_\ell(x) = \mathbf{W}_L\sigma(\dots\sigma(\mathbf{W}_{\ell+1}\sigma(x) + \boldsymbol{b}_{\ell+1})\dots) + \boldsymbol{b}_L)$ denotes the last $L - \ell - 1$ layers, with definition $\mathcal{A}_1(\boldsymbol{x}) = \boldsymbol{x}, \mathcal{B}_L(\boldsymbol{x}) = \boldsymbol{x}$.

Now focus on bounding $\|\mathcal{A}(\boldsymbol{x})\|$. Let $\mathbf{W} \in \mathbb{R}^{m\times m'}, \|\mathbf{W}\|_F \leq \sqrt{m'}, \boldsymbol{x} \in \mathbb{R}^{m'}, \boldsymbol{b} \in \mathbb{R}^m, \|\boldsymbol{b}\|_2 \leq \sqrt{m}$

$$\begin{aligned}\|\sigma(\mathbf{W}x + \boldsymbol{b})\|_2 &\leq \|\mathbf{W}\boldsymbol{x} + \boldsymbol{b}\|_2 \\ &\leq \|\mathbf{W}\|_2\|\boldsymbol{x}\|_2 + \|\boldsymbol{b}\|_2 \\ &\leq \|\mathbf{W}\|_F\|\boldsymbol{x}\|_2 + \|\boldsymbol{b}\|_2 \\ &\leq \sqrt{m'}\|\boldsymbol{x}\|_2 + \sqrt{m}\end{aligned}$$

where we make use of $\|\cdot\|_2 \leq \|\cdot\|_F$. Because of that,

$$\begin{aligned}&\|\mathcal{A}_2(\boldsymbol{x})\|_2 \leq \sqrt{d} + \sqrt{dw} \leq 2\sqrt{dw}, \\ &\|\mathcal{A}_3(\boldsymbol{x})\|_2 \leq \sqrt{w}\|\mathcal{A}_2(\boldsymbol{x})\|_2 + 2w\sqrt{dw} \leq 4w\sqrt{dw}, \\ &\qquad\dots \\ &\|\mathcal{A}_\ell(\boldsymbol{x})\|_2 \leq \sqrt{w}\|\mathcal{A}_{\ell-1}(\boldsymbol{x})\|_2 \leq 2\sqrt{dw}(2w)^{\ell-2}.\end{aligned} \tag{5.17}$$

Then focus on $\mathcal{B}(\boldsymbol{x})$. Let $\mathbf{W} \in \mathbb{R}^{m\times m'}, \|\mathbf{W}\|_F \leq \sqrt{m'}, \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^{m'}, \boldsymbol{b} \in \mathbb{R}^m, \|\boldsymbol{b}\|_2 \leq$

$\sqrt{m}$. Furthermore, $\|\boldsymbol{x}-\boldsymbol{x}'\|_2 \le \epsilon$, then

$$\|\sigma(\mathbf{W}\boldsymbol{x}+\boldsymbol{b})-\sigma(\mathbf{W}\boldsymbol{x}'+\boldsymbol{b})\|_2 \le \|\mathbf{W}(\boldsymbol{x}-\boldsymbol{x}')\|_2 \le \|\mathbf{W}\|_F\|\boldsymbol{x}-\boldsymbol{x}'\|_2$$

which indicates that $\|\mathcal{B}(\boldsymbol{x})-\mathcal{B}(\boldsymbol{x})'\|_2 \le (\sqrt{w})^{L-\ell}\|\boldsymbol{x}-\boldsymbol{x}'\|_2$

Finally, for any $\mathbf{W},\mathbf{W}' \in \mathbb{R}^{m\times m'}, \boldsymbol{x}\in\mathbb{R}^{m'}, \boldsymbol{b},\boldsymbol{b}'\in\mathbb{R}^m$, one have

$$\begin{aligned}\|(\mathbf{W}\boldsymbol{x}+\boldsymbol{b})-(\mathbf{W}'\boldsymbol{x}+\boldsymbol{b}')\|_2 &= \|(\mathbf{W}-\mathbf{W}')\boldsymbol{x}+(\boldsymbol{b}-\boldsymbol{b}')\|_2\\ &\le \|\mathbf{W}-\mathbf{W}'\|_2\|\boldsymbol{x}\|_2+\|\boldsymbol{b}-\boldsymbol{b}'\|_2.\\ &\le \|\mathbf{W}-\mathbf{W}'\|_F\|\boldsymbol{x}\|_2+\sqrt{m}\|\boldsymbol{b}-\boldsymbol{b}'\|_\infty.\end{aligned}$$

In summary,

$$\begin{aligned}|g_\ell(\boldsymbol{x})-g'_\ell(\boldsymbol{x})| &= |\mathcal{B}_\ell(\mathbf{W}_\ell\mathcal{A}_\ell(\boldsymbol{x})+\boldsymbol{b}_\ell)-\mathcal{B}_\ell(\mathbf{W}'_\ell\mathcal{A}_\ell(\boldsymbol{x})+\boldsymbol{b}'_\ell)|\\ &\le (\sqrt{w})^{L-\ell}\|(\mathbf{W}_\ell\mathcal{A}_\ell(\boldsymbol{x})+\boldsymbol{b}_\ell)-(\mathbf{W}'_\ell\mathcal{A}_\ell(\boldsymbol{x})+\boldsymbol{b}'_\ell)\|_2\\ &\le (\sqrt{w})^{L-\ell}(\|\mathbf{W}_\ell-\mathbf{W}'_\ell\|_F\|\mathcal{A}_\ell(\boldsymbol{x})\|_2+\|\boldsymbol{b}_\ell-\boldsymbol{b}'_\ell\|_2)\\ &\le 2^{(\ell-1)}w^{(L+\ell-3)/2}d^{1/2}\epsilon+w^{(L-\ell)/2}\bar{\epsilon}\end{aligned}$$

Let $f(x), f'(x)$ be two neural networks satisfying (5.16) with parameters $W_1,b_1,\ldots,W_L,b_L$ and $W'_1,b'_1,\ldots,W'_L,b'_L$ respectively, and $\|W_\ell-W'_\ell\|_F\le\epsilon_\ell, \|b_\ell-b'_\ell\|_F\le\tilde{\epsilon}_\ell$. Further define f_ℓ be the neural network with parameters $W_1,b_1,\ldots,W_\ell,b_\ell,W'_{\ell+1},b'_{\ell+1},\ldots,W'_L,b'_L$, then

$$\begin{aligned}|f(x)-f'(x)| &\le |f(x)-f_1(x)|+|f_1(x)-f_2(x)|+\cdots+|f_{L-1}(x)-f'(x)|\\ &\le \sum_{\ell=1}^{L}2^{(\ell-2)}d^{1/2}w^{(L+\ell-3)/2}\epsilon+w^{(L-\ell)/2}\bar{\epsilon}\end{aligned}$$

For any $\delta > 0$, one can choose

$$\epsilon_\ell = \frac{\delta}{2^\ell w^{(L+\ell-3)/2} d^{1/2}}, \tilde{\epsilon}_\ell = \frac{\delta}{2w^{(L-\ell)/2}}$$

such that $|f(x) - f'(x)| \leq \delta$.

On the other hand, the ϵ-covering number of $\{\mathbf{W} \in \mathbb{R}^{m\times m'} : \|\mathbf{W}\|_F \leq \sqrt{m'}\}$ on Frobenius norm is no larger than $(2\sqrt{m'}/\epsilon + 1)^{m\times m'}$, and the $\bar{\epsilon}$-covering number of $\{\boldsymbol{b} \in \mathbb{R}^m : \|\boldsymbol{b}\|_2 \leq 1\}$ on infinity norm is no larger than $(2/\bar{\epsilon} + 1)^m$. The entropy of this neural network can be bounded by

$$\log \mathcal{N}(f; \delta) \leq w^2 L \log(2^{L+1} w^{L-1}/\delta + 1) + wL \log(2^{L-1} w^{(L-1)/2} d^{1/2}/\delta + 1)$$

Proof of Lemma 5.7

Let ϵ be a positive constant. Without the loss of generality, we can sort the coefficients in descending order in terms of their absolute values. There exists a positive integer $\mathcal{M}$ (as a function of ϵ), such that $|a_i| \geq \epsilon$ for $i \leq \mathcal{M}$, and $|a_i| < \epsilon$ for $i > \mathcal{M}$.

By definition, $\mathcal{M}\epsilon^p \leq \sum_{i=1}^{\mathcal{M}} |a_i|^p \leq P$ so $\mathcal{M} \leq P/\epsilon^p$, and $|a_i|^p \leq P, |a_i| \leq P^{1/p}$ for all i. Furthermore,

$$\sum_{i>m} |a_i| = \sum_{i>\mathcal{M}} |a_i|^p |a_i|^{1-p} < \sum_{i>\mathcal{M}} |a_i|^p \epsilon^{1-p} \leq P\epsilon^{1-p}$$

Let $\tilde{g}_i = \arg\min_{g\in\tilde{\mathcal{G}}} \|g - \frac{a_i}{P^{1/p}} g_i\|_\infty$ where $\tilde{\mathcal{G}}$ is the δ'-convering set of $\mathcal{G}$. By definition of

the covering set,

$$\left\| \sum_{i=1}^{M} a_i g_i(x) - \sum_{i=1}^{\mathcal{M}} P^{1/p} \tilde{g}_i(x) \right\|_\infty \leq \left\| \sum_{i=1}^{\mathcal{M}} (a_i g_i(x) - P^{1/p} \tilde{g}_i(x)) \right\|_\infty + \left\| \sum_{i=\mathcal{M}+1}^{M} a_i g_i(x) \right\|_\infty \leq \mathcal{M} P^{1/p} \delta' + c_3 P \epsilon^{1-p}. \tag{5.18}$$

Choosing

$$\epsilon = (\delta/2c_3P)^{\frac{1}{1-p}}, \delta' \asymp P^{-\frac{1}{p(1-p)}} (\delta/2c_3)^{\frac{1}{1-p}}/2, \tag{5.19}$$

we have $\mathcal{M} \leq P^{\frac{1}{1-p}} (\delta/2c_3)^{-\frac{p}{1-p}}, \mathcal{M} P^{1/p} \delta' \leq \delta/2, c_3 P \epsilon^{1-p} \leq \delta/2$, so (5.18) $\leq \delta$. One can compute the covering number of $\mathcal{F}$ by

$$\log \mathcal{N}(\mathcal{F}, \delta) \leq \mathcal{M} \log \mathcal{N}(\mathcal{G}, \delta') \lesssim k\mathcal{M} \log(1/\delta') \tag{5.20}$$

Taking (5.19) into (5.20) finishes the proof.

5.8.4 Proof of Approximation Error

Approximation of Neural Networks to B-spline Basis Functions

Lemma 5.12 *Let $M_{m,k,\boldsymbol{s}}$ be the B-spline of order m with scale 2^{-k} in each dimension and position $\boldsymbol{s} \in \mathbb{R}^d$: $M_{m,k,\boldsymbol{s}}(\boldsymbol{x}) := M_m(2^k(\boldsymbol{x} - \boldsymbol{s}))$, M_m is defined in (5.2). There exists a neural network with d-dimensional input and one output, with width $w_{d,m} = O(dm)$ and depth $L \lesssim \log(c_{d,m}/\epsilon)$ for some constant $c_{d,m}$ that depends only on m and d, approximates the B spline basis function $M_{m,k,\boldsymbol{s}}(\boldsymbol{x}) := M_m(2^k(\boldsymbol{x} - \boldsymbol{s}))$ as defined in Section 5.3.2. This neural network, denoted as $\tilde{M}_{m,k,\boldsymbol{s}}(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^d$, satisfy*

- $|\tilde{M}_{m,k,\boldsymbol{s}}(\boldsymbol{x}) - M_{m,k,\boldsymbol{s}}(\boldsymbol{x})| \leq \epsilon$, *if* $0 \leq 2^k(x_i - s_i) \leq m + 1, \forall i \in [d]$,
- $\tilde{M}_{m,k,\boldsymbol{s}}(\boldsymbol{x}) = 0$, *otherwise.*

- *The weight in each layer has bounded norm* $\|\mathbf{W}^{(\ell)}\|_F \lesssim 2^{k/L}\sqrt{w}$*, except the first layer where* $\|\mathbf{W}^{(1)}\|_F \leq 2^{k/L}\sqrt{d}$*.*

Note that the product of the coefficients among all the layers are proportional to 2^k, instead of 2^{km} when approximating truncated power basis functions. This is because the transformation from M_m to $M_{m,k,\boldsymbol{s}}$ only scales the domain of the function by 2^k, while the codomain of the function is not changed. To apply the transformation to the neural network, one only need to scale weights in the first layer by 2^k, which is equivalent to scaling the weights in each layer bt $2^{k/L}$ and adjusting the bias according.

As for the proof, we follow the method developed in Yarotsky [135], Suzuki [106], while putting our attention on bounding the Frobenius norm of the weights.

Lemma 5.13 (Yarotsky [135, Proposition 3]) *: There exists a neural network with two-dimensional input and one output* $f_\times(x, y)$*, with constant width and depth* $O(\log(1/\delta))$*, and the weight in each layer is bounded by a global constant* c_1*, such that*

- $|f_\times(x, y) - xy| \leq \delta, \forall\ 0 \leq x, y \leq 1$*,*
- $f_\times(x, y) = 0, \forall\ x = 0$ *or* $y = 0$*.*

We first prove a special case of Lemma 5.12 on the unscaled, unshifted B-spline basis function by fixing $k = 0, \boldsymbol{s} = 0$:

Proposition 5.14 *There exists a neural network with* d*-dimensional input and one output, with width* $w = w(d, m) \eqsim dm$ *and depth* $L \lesssim \log(c(m, d)/\epsilon)$ *for some constant* w, c *that depends only on* m *and* d*, denoted as* $\tilde{M}_m(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^d$*, such that*

- $|\tilde{M}_m(\boldsymbol{x}) - M_m(\boldsymbol{x})| \leq \epsilon$*, if* $0 \leq x_i \leq m + 1, \forall i \in [d]$*, while* $M_m(\cdot)$ *denote* m*-th order B-spline basis function,*
- $\tilde{M}_m(\boldsymbol{x}) = 0$*, if* $x_i \leq 0$ *or* $x_i \geq m + 1$ *for any* $i \in [d]$*.*

- *The weight in each layer has bounded norm* $\|\mathbf{W}^{(\ell)}\|_F \lesssim \sqrt{w}$.

Proof: We first show that one can use a neural network with constant width w_0, depth $L \eqsim \log(m/\epsilon_1)$ and bounded norm $\|W^{(1)}\|_F \leq O(\sqrt{d}), \|W^{(\ell)}\|_F \leq O(\sqrt{w}), \forall \ell = 2, \ldots, L$ to approximate truncated power basis function up to accuracy ϵ_1 in the range $[0, 1]$. Let $m = \sum_{i=0}^{\lceil \log_2 m \rceil} m_i 2^i, m_i \in \{0, 1\}$ be the binary digits of m, and define $\bar{m}_j = \sum_{j=0}^{i} m_i, \gamma = \lceil \log_2 m \rceil$, then for any x

$$\begin{aligned} x_+^m &= x_+^{\bar{m}_\gamma} \times \left(x_+^{2^\gamma}\right)^{m_\gamma} \\ [x_+^{\bar{m}_\gamma}, x_+^{2^\gamma}] &= [x_+^{\bar{m}_{\gamma-1}} \times \left(x_+^{2^{\gamma-1}}\right)^{m_{\gamma-1}}, x_+^{2^{\gamma-1}} \times x_+^{2^{\gamma-1}}] \\ &\ldots \\ [x_+^{\bar{m}_2}, x_+^4] &= [x_+^{\bar{m}_1} \times \left(x_+^2\right)^{m_1}, x_+^2 \times x_+^2] \\ [x_+^{\bar{m}_1}, x_+^2] &= [x_+^{\bar{m}_0} \times x_+^{m_0}, x_+ \times x_+] \end{aligned} \tag{5.21}$$

Notice that each line of equation only depends on the line immediately below. Replacing the multiply operator $\times$ with the neural network approximation shown in Lemma 5.13 demonstrates the architecture of such neural network approximation. For any $x, y \in [0, 1]$, let $|f_\times(x, y) - xy| \leq \delta, |x - \tilde{x}| \leq \delta_1, |y - \delta y| \leq \delta_2$, then $|f_\times(\tilde{x}, \tilde{y}) - xy| \leq \delta_1 + \delta_2 + \delta$. Taking this into (5.21) shows that $\epsilon_1 \eqsim 2^\gamma \delta \eqsim m\delta$, where ϵ_1 is the upper bound on the approximate error to truncated power basis of order m and δ is the approximation error to a single multiply operator as in Lemma 5.13.

A univariate B-spline basis can be expressed using truncated power basis, and ob-

serving that it is symmetric around $(m+1)/2$:

$$\begin{aligned} M_m(x) &= \frac{1}{m!}\sum_{j=1}^{m+1}(-1)^j\binom{m+1}{j}(x-j)_+^m \\ &= \frac{1}{m!}\sum_{j=1}^{\lceil (m+1)/2\rceil}(-1)^j\binom{m+1}{j}(\min(x, m+1-x)-j)_+^m \\ &= \frac{((m+1)/2)^m}{m!}\sum_{j=1}^{\lceil (m+1)/2\rceil}(-1)^j\binom{m+1}{j}\Big(\frac{\min(x, m+1-x)-j}{(m+1)/2}\Big)_+^m, \end{aligned}$$

A multivariate (d-dimensional) B-spline basis function can be expressed as the product of truncated power basis functions and thus can be decomposed as

$$\begin{aligned} M_m(\boldsymbol{x}) &= \prod_{i=1}^d M_m(x_i) \\ &= \frac{((m+1)/2)^{md}}{(m!)^d}\prod_{i=1}^d\left(\sum_{j=1}^{\lceil (m+1)/2\rceil}(-1)^j\binom{m+1}{j}\Big(\frac{\min(x_i, m+1-x)-j}{(m+1)/2}\Big)_+^m\right) \end{aligned} \tag{5.22}$$

Using Lemma 5.13, one can construct $m+1$ number of neural networks, and each of them has width w_0 and depth $L = O(\log(m/\epsilon_1)$, such that the $(j+1)$-th neural network approximates $(\frac{x-j}{(m+1)/2})_+^m$ with error no more than ϵ_1 for any $0 \le x \le (m+1)/2$. The weighted summation of these subnetworks can approximate the univariate B-spline basis function with error no more than

$$d((m+1)/2)^m\frac{1}{m!}\sum_{i=1}^{m+1}\binom{m+1}{j}\epsilon_1 \eqsim \frac{de^{2m}}{\sqrt{m}}\epsilon_1$$

where we applied Stirling's approximation.

A multivariate B-spline basis is the product of univariate B-spline basis along each

dimension

$$M_m(\boldsymbol{x}) = \prod_{i=1}^{d} M_m(x_i).$$

We can construct a neural network to approximate this function by parallizing d number of neural networks to approximate each B-spline basis function along each dimension, and use the last $L_1 \eqsim \log(d/\delta)$ layers to approximate their product. The totol approximation error of this function is bounded by

$$d\frac{((m+1)/2)^m}{m!}\sum_{j=1}^{m+1}\binom{m+1}{j}\epsilon_1 + (d-1)\delta \eqsim \frac{e^{2m}}{\sqrt{m}}d\epsilon_1 + d\delta$$

where δ and ϵ_1 has the same definition as above. Choosing $\delta = \frac{\epsilon}{d(e^{2m}\sqrt{m}+1)}$, and recall $\epsilon_1 \eqsim m\delta$ proves the approximation error.

■

The proof of the Lemma 5.12 for general $k, \boldsymbol{s}$ follows by appending one more layer in the front, as we show below. *Proof:* [Proof of Lemma 5.12] Using the neural network proposed in Proposition 5.14, one can construct a neural network for appropximating $M_{m,k,\boldsymbol{s}}$ by adding one layer before the first layer:

$$\sigma(2^k\mathbf{I}_d\boldsymbol{x} - 2^k\boldsymbol{s})$$

The unused neurons in the first hidden layer is zero padded. The Frobenius norm of the weight is $2^k\|\mathbf{I}_d\|_F = 2^k\sqrt{d}$. Following the proof of Proposition 5.5, rescaling the weight in this layer by 2^{-k}, and the weight matrix in the last layer by 2^k, and scaling the bias properly, one can verify that this neural network satisfy the statement. ■

Sparse approximation of Besov functions using B-spline wavelets

Proof of Proposition 5.8

Dũng [136, Theorem 3.1] Suzuki [106, Lemma 2] proposed an adaptive sampling recovery method that approximates a function in Besov space. The method is divided into two cases: when $p \geq r$, and when $p < r$.

When $p \geq r$, there exists a sequence of scalars $\lambda_{\boldsymbol{j}}, \boldsymbol{j} \in P^d(\mu), P_d(\mu) := \{\boldsymbol{j} \in \mathbb{Z}^d : |j_i| \leq \mu, \forall i \in d\}$ for some positive μ, for arbitrary positive integer $\bar{k}$, the linear operator

$$Q_{\bar{k}}(f, \boldsymbol{x}) = \sum_{\boldsymbol{s} \in J(\bar{k}, m, d)} a_{\bar{k}, \boldsymbol{s}}(f) M_{\bar{k}, \boldsymbol{s}}(\boldsymbol{x}), \quad a_{\bar{k}, \boldsymbol{s}}(f) = \sum_{\boldsymbol{j} \in \mathbb{Z}^d, P^d(\mu)} \lambda_{\boldsymbol{j}} \bar{f}(\boldsymbol{s} + 2^{-\bar{k}} \boldsymbol{j})$$

has bounded approximation error

$$\|f - Q_{\bar{k}}(f, x)\|_r \leq C 2^{-\alpha \bar{k}} \|f\|_{B^\alpha_{p,q}},$$

where $\bar{f}$ is the extrapolation of f, $J(\bar{k}, m, d) := \{\boldsymbol{s} : 2^{\bar{k}} \boldsymbol{s} \in \mathbb{Z}^d, -m/2 \leq 2^{\bar{k}} s_i \leq 2^{\bar{k}} + m/2, \forall i \in [d]\}$. See Dũng [136, 2.6-2.7] for the detail of the extrapolation as well as references for options of sequence $\lambda_{\boldsymbol{j}}$.

Furthermore, $Q_{\bar{k}}(f) \in B^\alpha_{p,q}$ so it can be decomposed in the form (5.1) with $M = \sum_{k=0}^{\bar{k}} (2^k + m - 1)^d \lesssim 2^{\bar{k} d}$ components and $\|\{\tilde{c}_{k, \boldsymbol{s}}\}_{k, \boldsymbol{s}}\| \lesssim \|Q_{\bar{k}}(f)\|_{B^\alpha_{p,q}} \lesssim \|f\|_{B^\alpha_{p,q}}$ where $\tilde{c}_{k, \boldsymbol{s}}$ is the coefficients of the decomposition of $Q_{\bar{k}}(f)$. Choosing $\bar{k} \approx \log_2 M/d$ leads to the desired approximation error.

On the other hand, when $p < r$, there exists a greedy algorithm that constructs

$$G(f) = Q_{\bar{k}}(f) + \sum_{k=\bar{k}+1}^{k^*} \sum_{j=1}^{n_k} c_{k, \boldsymbol{s}_j}(f) M_{k, \boldsymbol{s}_j}$$

where $\bar{k} \approx \log_2(M), k^* = [\epsilon^{-1} \log(\lambda M)] + \bar{k} + 1, n_k = [\lambda M 2^{-\epsilon(k - \bar{k})}]$ for some $0 < \epsilon <$

$\alpha/\delta - 1, \delta = d(1/p - 1/r), \lambda > 0$, such that

$$\|f - G(f)\|_r \leq \bar{M}^{-\alpha/d} \|f\|_{B^\alpha_{p,q}}$$

and

$$\sum_{k=0}^{\bar{k}} (2^k + m - 1)^d + \sum_{k=\bar{k}+1}^{k^*} n_k \leq \bar{M}.$$

See Dũng [136, Theorem 3.1] for the detail.

Finally, since $\alpha - d/p > 1$,

$$\begin{aligned} \|\{2^{k_i} c_{k_i,\boldsymbol{s}_i}\}_{k_i,\boldsymbol{s}_i}\|_p &\leq \sum_{k=0}^{\bar{k}} 2^k \|\{c_{k_i,\boldsymbol{s}_i}\}_{\boldsymbol{s}_i}\|_p \\ &= \sum_{k=0}^{\bar{k}} 2^{(1-(\alpha-d/p))k} (2^{(\alpha-d/p)k} \|\{c_{k_i,\boldsymbol{s}_i}\}_{\boldsymbol{s}_i}\|_p) \\ &\lesssim \sum_{k=0}^{\bar{k}} 2^{(1-(\alpha-d/p))k} \|f\|_{B^\alpha_{p,q}} \\ &\eqsim \|f\|_{B^\alpha_{p,q}} \end{aligned} \tag{5.23}$$

where the first line is because for arbitrary vectors $\boldsymbol{a}_i, i \in [n], \|\sum_{i=1}^n \boldsymbol{a}_i\|_p \leq \sum_{i=1}^n \|\boldsymbol{a}_i\|_p$, the third line is because the sequence norm of B-spline decomposition is equivalent to the norm in Besov space (see Section 5.3.2).

Remark 5.1 *The requirement in Proposition 5.8: $\alpha - d/p > 1$ is stronger than the condition typically found in approximation theorem $\alpha - d/p \geq 0$ [136], so-called "Boundary of continuity", or the condition in Suzuki [106] $\alpha > d(1/p - 1/r)_+$. This is because although the functions in $B^\alpha_{p,q}$ when $0 \leq \alpha - d/p < 1$ can be approximated by B-spline basis, the sum of weighted coefficients may not converge. One simple example is the step function $f_{step}(x) = \mathbf{1}(x \geq 0.5), f_{step} \in B^1_{1,\infty}$. Although it can be decomposed using first order B-spline basis as in (5.1), the summation of the coefficients is infinite. Actually one*

only needs a ReLU neural network with one hidden layer and two neurons to approximate this function to arbitrary precision, but the weight need to go to infinity.

Remark 5.2 *Note that when $\alpha - d/p = 1$, the sequence norm (5.23) is bounded (up to a factor of constant) by $k^*\|f\|_{B^\alpha_{p,q}}$, which can be proven by following (5.23) except the last line. This adds a logarithmic term with respect to $\bar{M}$ compared with the result in Proposition 5.8. This will add a logarithmic factor to the MSE. We will not focus on this case in this paper of simplicity.*

Sparse approximation of Besov functions using Parallel Neural Networks

Proof of Theorem 5.9

The proof is divided into three steps:

1. Bound the 0-norm and the p-norm of the coefficients of B-spline basis in order to approximate an arbitrary function in Besov space up to any $\epsilon > 0$.

2. Bound p'-norm of the coefficients of B-spline basis functions where $p' = 2/L, 0 < p' < 1$ using the results above .

3. Add the approximation of neural network to B-spline basis computed in Lemma 5.12 into Step 2.

Proof: Using Proposition 5.8, one can construct $\bar{M}$ number of NN according to Lemma 5.12, such that each NN represents one B-spline basis function. The weights in the last layer of each NN is scaled to match the coefficients in Proposition 5.8. Taking p' in Lemma 5.15 as $2/L$ and combining with Lemma 5.12 finishes the proof. ■

Lemma 5.15 *For any $a \in \mathbb{R}^{\bar{M}}$, $0 < p' < p$, it holds that:*

$$\|a\|_{p'}^{p'} \leq \bar{M}^{1-p'/p}\|a\|_p^{p'}.$$

Proof:

$$\sum_i |a_i|^{p'} = \langle \mathbf{1}, |\boldsymbol{a}|^{p'} \rangle \leq \left(\sum_i 1 \right)^{1-\frac{p'}{p}} \left(\sum_i (|a_i|^{p'})^{\frac{p}{p'}} \right)^{\frac{p'}{p}} = \bar{M}^{1-\frac{p'}{p}} \|a\|_p^{p'}$$

The first inequality uses a Holder's inequality with conjugate pair $\frac{p}{p'}$ and $1/(1-\frac{p'}{p})$. ■

5.8.5 Proof of the Main Theorem

Proof of Theorem 5.1

Proof: First recall the relationship between covering number (entropy) and estimation error:

Proposition 5.16 *Let $\mathcal{F} \subseteq \{\mathbb{R}^d \to [-F, F]\}$ be a set of functions. Assume that $\mathcal{F}$ can be decomposed into two orthogonal spaces $\mathcal{F} = \mathcal{F}_\| \times \mathcal{F}_\perp$ where $\mathcal{F}_\perp$ is an affine space with dimension of N. Let $f_0 \in \{\mathbb{R}^d \to [-F, F]\}$ be the target function and $\hat{f}$ be the least squares estimator in $\mathcal{F}$:*

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^n (y_i - f(x_i))^2, y_i = f_0(x_i) + \epsilon_i, \epsilon_i \sim \mathcal{N}(0, \sigma^2) i.i.d.,$$

then it holds that

$$\text{MSE}(\hat{f}) \leq \tilde{O}\Big(\arg\min_{f \in \mathcal{F}} \text{MSE}(f) + \frac{N + \log \mathcal{N}(\mathcal{F}_\|, \delta) + 2}{n} + (F + \sigma)\delta \Big).$$

The proof of Proposition 5.16 is defered to the section below. We choose $\mathcal{F}$ as the set of functions that can be represented by a parallel neural network as stated, the (null) space $\mathcal{F}_\perp = \{f : f(\boldsymbol{x}) = constant\}$ be the set of functions with constant output, which has dimension 1. This space captures the bias in the last layer, while the other parameters contributes to the projection in $\mathcal{F}_\|$. See Section 5.8.3 for how we handle the bias in the

other layers. One can find that $\mathcal{F}_{\|}$ is the set of functions that can be represented by a parallel neural network as stated, and further satisfy $\sum_{i=1}^{n} f(\boldsymbol{x}_i) = 0$. Because $\mathcal{F}_{\|} \subseteq \mathcal{F}$, $\mathcal{N}(\mathcal{F}_{\|}, \delta) \leq \mathcal{N}(\mathcal{F}, \delta)$ for all $\delta > 0$, and the latter is studied in Theorem 5.6.

In Theorem 5.1, the width of each subnetwork is no less than what is required in Theorem 5.9, while the depth and norm constraint are the same, so the approximation error is no more that that in Theorem 5.9. Choosing $r = 2, p = 2/L$, and taking Theorem 5.6 and Theorem 5.9 into this Proposition 5.16, one gets

$$\begin{aligned} \mathrm{MSE}(\hat{f}) &\lesssim \min_{f \in \mathcal{F}} \mathrm{MSE}(f) + \frac{w^{2+2/(1-2/L)} L^2 \sqrt{d} P'^{\frac{1}{1-2/L}} \delta^{-\frac{2/L}{1-2/L}} \log(wP'/\delta)}{n} + \delta \\ &\lesssim \bar{M}^{-2\alpha/d} + \frac{w^{2+2/(1-2/L)} L^2}{n} \bar{M}^{\frac{1-2/(pL)}{1-2/L}} \delta^{-\frac{2/L}{1-2/L}} (\log(\bar{M}/\delta) + 3) + \delta, \end{aligned} \tag{5.24}$$

where $\|f\|_{B^{\alpha}_{p,q}}, m$ and d taken as constants. By choosing

$$\delta \asymp \frac{w^{4-4/L} L^{2-4/L} \bar{M}^{1-2/(pL)}}{n^{1-2/L}}, \bar{M} \asymp \left(\frac{n^{1-2/L}}{w^{4-4/L} L^{2-4/L}} \right)^{\frac{1}{2\alpha/d+1-2/(pL)}},$$

we get

$$\mathrm{MSE}(\hat{f}) \leq \tilde{O}\left(\left(\frac{w^{4-4/L} L^{2-4/L}}{n^{1-2/L}} \right)^{\frac{2\alpha/d}{2\alpha/d+1-2/(pL)}} + e^{-c_6 L} \right) \tag{5.25}$$

where $\mathrm{MSE}(\hat{f})$ shows the MSE of the solution to constrained optimization problem (5.9) by optimally choosing $\bar{M}$ (or P').

Finally, under the assumption in Lemma 5.18, for any constrained optimization problem, there exists a regularized optimzation problem, whose MSE is not larger than the MSE of the constrained optimization problem up to a factor of a constant. This closes the connection between (5.8) and (5.9) and finishes the proof.

Note that the empirical risk minimizer (ERM) of the parallel nerual network satisfy that the $(2/L)$-norm of the coefficients of the parallel neural network satisfy that

$\|\{a_j\}\|_{2/L}^{2/L} = \|\{\tilde{a}_{j,\bar{M}}\}\|_{2/L}^{2/L}$ where $\{\tilde{a}_{j,\bar{M}}\}$ is the coefficient of the particular $\bar{M}$-sparse approximation, although $\{a_j\}$ is not necessarily $\bar{M}$ sparse. Empirically, one only need to guarantee that during initialization, the number of subnetworks $M \geq \bar{M}$ such that the $\bar{M}$-sparse approximation is feasible, thus the approximation error bound from Theorem 5.9 can be applied. Theorem 5.9 also says that $\|\{a_j\}\|_{2/L}^{2/L} = \|\{\tilde{a}_{j,\bar{M}}\}\|_{2/L}^{2/L} \lesssim \bar{M}^{1-2/pL}$, thus we can apply the covering number bound from Theorem 5.6 with $P' = \bar{M}^{1-2/pL}$. Finally, if λ is optimally chosen, then it achieves a smaller MSE than this particular λ', which has been proven to be no more than $O(\bar{M}^{-\alpha/d})$ and completes the proof.

■

Proof of Proposition 5.16

For any function $f \in \mathcal{F}$, define $f_\perp = \arg\min_{h\in\mathcal{F}_\perp} \sum_{i=1}^n (f(\boldsymbol{x}_i) - h(\boldsymbol{x}_i))^2$ be the projection of f to $\mathcal{F}_\perp$, and define $f_\parallel = f - f_\perp$ be the projection to the orthogonal complement. Note that $f_\parallel$ is not necessarily in $\mathcal{F}_\parallel$. However, if $f \in \mathcal{F}$, then $f_\parallel \in \mathcal{F}_\parallel$. $y_{i\perp}$ and $y_{i\parallel}$ are defined by creating a function f_y such that $f_y(\boldsymbol{x}_i) = y_i, \forall i$, e.g. via interpolation. Because $\mathcal{F}_\parallel$ and $\mathcal{F}_\perp$ are orthogonal, the empirical loss and population loss can be decomposed in the same way:

$$L_\parallel(f) = \frac{1}{n}\sum_{i=1}^n (f_\parallel(\boldsymbol{x}) - f_{0\parallel}(\boldsymbol{x}))^2 + \frac{n-N}{n}\sigma^2, \qquad L_\perp(f) = \frac{1}{n}\sum_{i=1}^n (f_\perp(\boldsymbol{x}) - f_{0\perp}(\boldsymbol{x}))^2 + \frac{N}{n}\sigma^2,$$

$$\hat{L}_\parallel(f) = \frac{1}{n}\sum_{i=1}^n (f_\parallel(\boldsymbol{x}) - y_{i\parallel})^2, \qquad \hat{L}_\perp(f) = \frac{1}{n}\sum_{i=1}^n (f_\perp(\boldsymbol{x}) - y_{i\perp}(\boldsymbol{x}))^2,$$

$$MSE_\parallel(f) = \mathbb{E}_{\mathcal{D}}\Big[\frac{1}{n}\sum_{i=1}^n (f_\parallel(\boldsymbol{x}) - f_{0\parallel}(\boldsymbol{x}))^2\Big], \qquad MSE_\perp(f) = \mathbb{E}_{\mathcal{D}}\Big[\frac{1}{n}\sum_{i=1}^n (f_\perp(\boldsymbol{x}) - f_{0\perp}(\boldsymbol{x}))^2\Big],$$

such that $L(f) = L_\parallel(f) + L_\perp(f), \hat{L}(f) = \hat{L}_\parallel(f) + \hat{L}_\perp(f)$. This can be verified by decomposing $\hat{f}, f_0$ and y into two orthogonal components as shown above, and observing that $\sum_{i=1}^n f_{1\perp}(\boldsymbol{x}_i) f_{2\parallel}(\boldsymbol{x}_i) = 0, \forall f_1, f_2$.

First prove the following claim

Claim 5.17 *Assume that* $\hat{f} = \arg\min_{f\in\mathcal{F}} \hat{L}(f)$ *is the empirical risk minimizer. Then* $\hat{f}_\perp = \arg\min_{f\in\mathcal{F}_\perp} \hat{L}_\perp(f), \hat{f}_\parallel = \arg\min_{f\in\mathcal{F}_\parallel} \hat{L}_\parallel(f)$*, where* $\hat{f}_\perp$ *is the projections of* $\hat{f}$ *in* $\mathcal{F}_\perp$*, and* $\hat{f}_\parallel = \hat{f} - \hat{f}_\perp$ *respectively.*

Proof: Since $\hat{f} \in \mathcal{F}$, by definition $\hat{f}_\parallel \in \mathcal{F}_\parallel$. Assume that there exist $\hat{f}'_\perp, \hat{f}'_\parallel$, and either $\hat{L}_\perp(\hat{f}'_\perp) < \hat{L}_\perp(\hat{f}_\perp)$, or $\hat{L}_\parallel(\hat{f}'_\parallel) < \hat{L}_\parallel(\hat{f}_\parallel)$. Then

$$\begin{aligned}\hat{L}(\hat{f}') &= \hat{L}(\hat{f}'_\perp + \hat{f}'_\parallel) = \hat{L}_\parallel(\hat{f}'_\perp + \hat{f}'_\parallel) + \hat{L}_\perp(\hat{f}'_\perp + \hat{f}'_\parallel) = \hat{L}_\parallel(\hat{f}'_\parallel) + \hat{L}_\perp(\hat{f}'_\perp) \\ &< \hat{L}_\parallel(\hat{f}_\parallel) + \hat{L}_\perp(\hat{f}_\perp) = \hat{L}_\parallel(\hat{f}_\perp + \hat{f}_\parallel) + \hat{L}_\perp(\hat{f}_\perp + \hat{f}_\parallel) = \hat{L}(\hat{f})\end{aligned}$$

which shows that $\hat{f}$ is not the minimizer of $\hat{L}(f)$ and violates the assumption.

■

Then we bound $\mathrm{MSE}_\perp(f)$**.** We convert this part into a finite dimension least square problem:

$$\begin{aligned}\hat{f}_\perp &= \arg\min_{f\in\mathcal{F}_\perp} \hat{L}_\perp(f) \\ &= \arg\min_{f\in\mathcal{F}_\perp} \frac{1}{n}\sum_{i=1}^n (f(\boldsymbol{x}_i) - f_{0\perp}(\boldsymbol{x}_i) - \epsilon_{i\perp})^2 \\ &= \arg\min_{f\in\mathcal{F}_\perp} \frac{1}{n}\sum_{i=1}^n (f(\boldsymbol{x}_i) - f_{0\perp}(\boldsymbol{x}_i) - \epsilon_{i\perp})^2 + \epsilon_{i\parallel}^2 \\ &= \arg\min_{f\in\mathcal{F}_\perp} \frac{1}{n}\sum_{i=1}^n (f(\boldsymbol{x}_i) - f_{0\perp}(\boldsymbol{x}_i) - \epsilon_{i\perp} - \epsilon_{i\parallel})^2 \\ &= \arg\min_{f\in\mathcal{F}_\perp} \frac{1}{n}\sum_{i=1}^n (f(\boldsymbol{x}_i) - f_{0\perp}(\boldsymbol{x}_i) - \epsilon_i)^2\end{aligned}$$

The forth line comes from our assumption that $\mathcal{F}_\perp$ is orthogonal to $\mathcal{F}_\parallel$, so $\forall f \in \mathcal{F}_\perp, f + f_{0\perp} + \epsilon_\perp$ is orthogonal to $\epsilon_\parallel$.

Let the basis function of $\mathcal{F}_\perp$ be $h_1, h_2, \ldots, h_N$, the above problem can be reparame-

terized as

$$\arg\min_{\boldsymbol{\theta}\in\mathbb{R}^N} \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta}-\boldsymbol{y}\|^2$$

where $\mathbf{X} \in \mathbb{R}^{n\times N}$: $X_i = h_j(\boldsymbol{x}_i), \boldsymbol{y} = \boldsymbol{y}_{0\perp} + \boldsymbol{\epsilon}, \boldsymbol{y}_{0\perp} = [f_{0\perp}(x_1),\ldots,f_{0\perp}(x_n)], \boldsymbol{\epsilon} = [\epsilon_1,\ldots,\epsilon_n]$. This problem has a closed-form solution

$$\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y}$$

Observe that $f_{0\perp} \in \mathcal{F}_\perp$, let $\boldsymbol{y}_{0\perp} = \mathbf{X}\boldsymbol{\theta}^*$,The MSE of this problem can be computed by

$$\begin{aligned} L(\hat{f}_\perp) &= \frac{1}{n}\|\mathbf{X}\boldsymbol{\theta}-\boldsymbol{y}_{0\perp}\|^2 = \frac{1}{n}\|\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta}^*+\boldsymbol{\epsilon})-\mathbf{X}\boldsymbol{\theta}^*\|^2 \\ &= \frac{1}{n}\|\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{\epsilon}\|^2 \end{aligned}$$

Observing that $\Pi := \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is an idempotent and independent projection whose rank is N, and that $\mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T] = \sigma^2\mathbf{I}$, we get

$$\mathrm{MSE}_\perp(\hat{f}_\perp) = \mathbb{E}[L(\hat{f}_\perp)] = \frac{1}{n}\|\Pi\boldsymbol{\epsilon}\|^2 = \frac{1}{n}\mathrm{tr}(\Pi\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T) = \frac{\sigma^2}{n}\mathrm{tr}(\Pi)$$

which concludes that

$$\mathrm{MSE}_\perp(\hat{f}) = O\Big(\frac{N}{n}\sigma^2\Big). \tag{5.26}$$

See also [142, Proposition 1].

Next we study $\mathrm{MSE}_\parallel(\hat{f})$**.** Denote $\tilde{\sigma}_\parallel^2 = \frac{1}{n}\sum_{i=1}^n \epsilon_{i\parallel}^2, E = \max_i|\epsilon_i|$. Using Jensen's inequality and union bound, we have

$$\exp(t\mathbb{E}[E]) \le \mathbb{E}[\exp(tE)] = \mathbb{E}[\max\exp(t|\epsilon_i|)] \le \sum_{i=1}^n \mathbb{E}[\exp(t|\epsilon_i|)] \le 2n\exp(t^2\sigma^2/2)$$

Taking expectation over both sides, we get

$$\mathbb{E}[E] \leq \frac{\log 2n}{t} + \frac{t\sigma^2}{2}$$

maximizing the right hand side over t yields

$$\mathbb{E}[E] \leq \sigma\sqrt{2\log 2n}.$$

Let $\tilde{\mathcal{F}}_{\|}$ be the covering set of $\mathcal{F}_{\|} = \{f_{\|} : f \in \mathcal{F}\}$. For any $\tilde{f}_{\|} \in \tilde{\mathcal{F}}_{\|}$,

$$\begin{aligned}
L_{\|}(f_j) - \hat{L}_{\|}(f_j) &= \frac{1}{n}\sum_{i=1}^{n}(f_{j\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))^2 - \frac{1}{n}\sum_{i=1}^{n}(\tilde{f}_{\|}(\boldsymbol{x}_i) - y_{i\|})^2 + \frac{n-N}{n}\sigma^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\epsilon_{i\|}(2\tilde{f}_{\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i) - y_{i\|}) + \frac{n-N}{n}\sigma^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\epsilon_{i}(2\tilde{f}_{\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i) - y_{i\|}) + \frac{n-N}{n}\sigma^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\epsilon_{i}(2\tilde{f}_{\|}(\boldsymbol{x}_i) - 2f_{0\|}(\boldsymbol{x}_i)) + \frac{n-N}{n}\sigma^2 - \tilde{\sigma}_{\|}^2
\end{aligned}$$

The first term can be bounded using Bernstein's inequality: let $h_i = \epsilon_i(f_{j\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))$, by definition $|h_i| \leq 2EF$,

$$\begin{aligned}
\mathrm{Var}[h_i] &= \mathbb{E}[\epsilon_i^2(\tilde{f}_{\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))^2] \\
&= (\tilde{f}_{\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))^2\mathbb{E}[\epsilon_i^2] \\
&= (\tilde{f}_{\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))^2\sigma^2
\end{aligned}$$

using Bernstein's inequality, for any $\tilde{f}_{\|} \in \tilde{\mathcal{F}}_{\|}$, with probably at least $1-\delta_p$,

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\epsilon_i(2\tilde{f}_{\|}(\boldsymbol{x}_i)-2f_{0\|}(\boldsymbol{x}_i)) &= \frac{2}{n}\sum_{i=1}^{n}h_i \\ &\le \frac{2}{n}\sqrt{2\sum_{i=1}^{n}\big(\tilde{f}_{\|}(\boldsymbol{x}_i)-f_{0\|}(\boldsymbol{x}_i)\big)^2\sigma^2\log(1/\delta_p)}+\frac{8EF\log(1/\delta_p)}{3n} \\ &= 2\sqrt{\Big(L_{\|}(\tilde{f}_{\|})-\frac{n-N}{n}\sigma^2\Big)\frac{2\sigma^2\log(1/\delta_p)}{n}}+\frac{8EF\log(1/\delta_p)}{3n} \\ &\le \epsilon\Big(L_{\|}(\tilde{f}_{\|})-\frac{n-N}{n}\sigma^2\Big)+\frac{8\sigma^2\log(1/\delta_p)}{n\epsilon}+\frac{8EF\log(1/\delta_p)}{3n}\end{aligned}$$

the last inequality holds true for all $\epsilon > 0$. The union bound shows that with probably at least $1-\delta$, for all $\tilde{f}_{\|} \in \tilde{\mathcal{F}}_{\|}$,

$$\begin{aligned}L_{\|}(\tilde{f}_{\|})-\hat{L}_{\|}(\tilde{f}_{\|}) \le \epsilon\Big(L_{\|}(\tilde{f}_{\|})-\frac{n-N}{n}\sigma^2\Big)+\frac{8\sigma^2\log(\mathcal{N}(\mathcal{F}_{\|},\delta)/\delta_p)}{n\epsilon}+\frac{8EF\log(\mathcal{N}(\mathcal{F}_{\|},\delta)/\delta_p)}{3n} \\ +\frac{n-N}{n}\sigma^2-\tilde{\sigma}_{\|}^2.\end{aligned}$$

By rearanging the terms and using the definition of $L(\tilde{f}_{\|})$, we get

$$(1-\epsilon)\Big(L_{\|}(\tilde{f}_{\|})-\frac{n-N}{n}\sigma^2\Big) \le \hat{L}_{\|}(\tilde{f}_{\|})+\frac{8\sigma^2\log(\mathcal{N}(\mathcal{F}_{\|},\delta)/\delta_p)}{n\epsilon}+\frac{8EF\log(\mathcal{N}(\mathcal{F}_{\|},\delta)/\delta_p)}{3n}-\tilde{\sigma}_{\|}^2.$$

Taking the expectation (over $\mathcal{D}$) on both sides, and notice that $\mathbb{E}[\tilde{\sigma}_{\|}^2]=\frac{n-N}{n}\sigma^2$. Furthermore, for any random variable $X, \mathbb{E}[X]=\int_{-\infty}^{\infty}xdP(X\le x)$, we get

$$\begin{aligned}&\max_{\tilde{f}_{\|}\in\tilde{\mathcal{F}}_{\|}}\Big((1-\epsilon)\mathrm{MSE}_{\|}(\tilde{f}_{\|})-\mathbb{E}[\hat{L}_{\|}(\tilde{f}_{\|})]\Big) \\ &\le \Big(\frac{8\sigma^2}{n\epsilon}+\frac{8F\sigma\sqrt{2\log 2n}}{3n}\Big)\Big(\log\mathcal{N}(\mathcal{F}_{\|},\delta)-\int_{\delta=0}^{1}\log(\delta_p)d\delta_p\Big)-\frac{n-N}{n}\sigma^2 \qquad (5.27)\\ &= \Big(\frac{8\sigma^2}{n\epsilon}+\frac{8F\sigma\sqrt{2\log 2n}}{3n}\Big)(\log\mathcal{N}(\mathcal{F}_{\|},\delta)+1)-\frac{n-N}{n}\sigma^2.\end{aligned}$$

where the integration can be computed by replacing δ with e^x. Though it is not integrable

under Riemann integral, it is integrable under Lebesgue integration.

Similarly, let $\check{f}_{\|} = \arg\min_{f\in\mathcal{F}_{\|}} L_{\|}(f)$,

$$L_{\|}(\check{f}_{\|}) - \hat{L}_{\|}(\check{f}_{\|}) = \frac{1}{n}\sum_{i=1}^{n}\epsilon_i(2\check{f}_{\|}(\boldsymbol{x}_i) - 2f_{0\|}(\boldsymbol{x}_i)) + \frac{n-N}{n}\sigma^2 - \tilde{\sigma}_{\|}^2$$

with probably at least $1-\delta_q$, for any $\epsilon > 0$,

$$-\frac{1}{n}\sum_{i=1}^{n}\epsilon_i(2\check{f}_{\|}(\boldsymbol{x}_i) - 2f_{0\|}(\boldsymbol{x}_i)) \le \epsilon\Big(L_{\|}(\check{f}_{\|}) - \frac{n-N}{n}\sigma^2\Big) + \frac{8\sigma^2\log(1/\delta_p)}{n\epsilon} + \frac{8EF\log(1/\delta_p)}{3n},$$
$$\hat{L}_{\|}(\check{f}_{\|}) \le (1+\epsilon)\Big(L_{\|}(\check{f}_{\|}) - \frac{n-N}{n}\sigma^2\Big) + \frac{8\sigma^2\log(1/\delta_p)}{n\epsilon} + \frac{8EF\log(1/\delta_q)}{3n} + \tilde{\sigma}_{\|}^2.$$

Taking the expectation on both sides,

$$\mathbb{E}[\hat{L}_{\|}(\check{f}_{\|})] \le (1+\epsilon)\mathrm{MSE}_{\|}(\check{f}_{\|}) + \frac{8\sigma^2}{n\epsilon} + \frac{8F\sigma\sqrt{2\log 2n}}{3n} + \frac{n-N}{n}\sigma^2. \tag{5.28}$$

Finally, let $\hat{f}_* := \arg\min_{f\in\tilde{\mathcal{F}}_{\|}}\sum_{i=1}^{n}(\hat{f}_{\|}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i))^2$ be the projection of $\hat{f}_{\|}$ in its δ-covering space,

$$\begin{aligned}
\mathrm{MSE}_{\|}(\hat{f}_{\|}) &= \mathbb{E}\Big[\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_{\|}(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))^2\Big] \\
&= \mathbb{E}\Big[\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_*(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_{\|}(\boldsymbol{x}_i) - \hat{f}_*(\boldsymbol{x}_i))(\hat{f}_{\|}(\boldsymbol{x}_i) + \hat{f}_*(\boldsymbol{x}_i) - 2f_{0\|}(\boldsymbol{x}_i))\Big] \\
&\le \mathbb{E}\Big[\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_*(\boldsymbol{x}_i) - f_{0\|}(\boldsymbol{x}_i))^2\Big] + 4F\delta \\
&= \mathrm{MSE}_{\|}(\hat{f}_*(\boldsymbol{x}_i)) + 4F\delta,
\end{aligned}$$

and similarly

$$\hat{L}_{\|}(\hat{f}_*) \le \hat{L}_{\|}(\hat{f}_{\|}) + (4F+2E)\delta. \tag{5.29}$$

We can conclude that

$$\begin{aligned}
\mathrm{MSE}_{\|}(\hat{f}_{\|}) &\leq \frac{1}{1-\epsilon}\Big(\mathbb{E}[\hat{L}_{\|}(\hat{f}_*)] + \Big(\frac{8\sigma^2}{n\epsilon} + \frac{8F\sigma\sqrt{2\log 2n}}{3n}\Big)(\log \mathcal{N}(\mathcal{F}_{\|}, \delta) + 1) - \frac{n-N}{n}\sigma^2\Big) \\
&\quad + 4F\delta \\
&\leq \frac{1}{1-\epsilon}\Big(\mathbb{E}[\hat{L}_{\|}(\hat{f}_{\|})] + (4F + \sigma\sqrt{8\log 2n})\delta \\
&\quad + \Big(\frac{8\sigma^2}{n\epsilon} + \frac{8F\sigma\sqrt{2\log 2n}}{3n}\Big)(\log \mathcal{N}(\mathcal{F}_{\|}, \delta) + 1) - \frac{n-N}{n}\sigma^2\Big) + 4F\delta \\
&\leq \frac{1}{1-\epsilon}\Big(\mathbb{E}[\hat{L}_{\|}(\check{f}_{\|})] + (4F + \sigma\sqrt{8\log 2n})\delta \\
&\quad + \Big(\frac{8\sigma^2}{n\epsilon} + \frac{8F\sigma\sqrt{2\log 2n}}{3n}\Big)(\log \mathcal{N}(\mathcal{F}_{\|}, \delta) + 1) - \frac{n-N}{n}\sigma^2\Big) + 4F\delta \\
&\leq \frac{1+\epsilon}{1-\epsilon}\mathrm{MSE}_{\|}(\check{f}_{\|}) + \frac{1}{n}\Big(\frac{8\sigma^2}{\epsilon} + \frac{8F\sigma\sqrt{2\log 2n}}{3}\Big)\Big(\frac{\log \mathcal{N}(\mathcal{F}_{\|}, \delta) + 2}{1-\epsilon}\Big) \\
&\quad + \Big(4F + \frac{4F + \sigma\sqrt{8\log 2n}}{1-\epsilon}\Big)\delta,
\end{aligned}$$

where the first line comes from (5.27), and second comes from (5.29), the thid line is because $\hat{f}_{\|} = \arg\min_{f\in\mathcal{F}_{\|}} \hat{L}_{\|}(f)$, and the last line comes from (5.28). We also use that fact that $\hat{L}_{\|}(\hat{f}) \leq \hat{L}_{\|}(f), \forall f$. Noticing that $\mathrm{MSE}(\hat{f}) = \mathrm{MSE}_{\|}(\hat{f}) + \mathrm{MSE}_{\perp}(\hat{f})$, combining this with (5.26) finishes the proof.

Lemma 5.18 *Assume that these exists $C_1, C_2 > 1$ (which may depend on the target function), for all $P' > 0$, there exists $\lambda > 0$, such that the soltion to the regularized optimization problem (5.8), denoted as $\tilde{f}$, satisfy*

$$C_1 P' \leq \|\{\tilde{a}_j\}\|_{2/L}^{2/L} \leq C_2 P',$$

then the MSE of the regularized optimization problem satisfy

$$\mathrm{MSE}(\tilde{f}) \leq C\mathrm{MSE}(\hat{f})$$

where C is a constant that depends on C_1, C_2, $\hat{f}$ is the solution to the constrained optimzation problem (5.9), and

$$\lambda \lesssim \frac{\mathrm{MSE}(\hat{f})}{P'} \lesssim n^{-(1-2/L)}$$

Proof: The MSE of the regularized problem can be achieved by taking our assumtion into (5.6). We only need to prove the selection of λ. We apply the decomposition as in Proposition 5.16, and only need to consider $\mathcal{F}_{\|}$, as $\mathcal{F}_{\perp}$ is not imfluenced by regularization or constrained. From the definition of $\tilde{f}$ and λ, we have

$$\begin{aligned}
\hat{L}(\tilde{f}) + \lambda \|\{\tilde{a}_j\}\|_{2/L}^{2/L} &\leq \hat{L}(\hat{f}) + \lambda \|\{\hat{a}_j\}\|_{2/L}^{2/L}, \\
\hat{L}_{\|}(\tilde{f}) + \lambda \|\{\tilde{a}_j\}\|_{2/L}^{2/L} &\leq L_{\|}(\hat{f}) + \lambda \|\{\hat{a}_j\}\|_{2/L}^{2/L}
\end{aligned}$$

From Proposition 5.16, we get

$$\begin{aligned}
&(1-\epsilon)\mathrm{MSE}(\tilde{f}) - O\left(\frac{\log \mathcal{N}(\|\{\tilde{a}_j\}\|_{2/L}^{2/L}, \delta)}{n}\right) + \lambda \|\{\tilde{a}_j\}\|_{2/L}^{2/L} \\
&\leq (1+\epsilon)\mathrm{MSE}(\hat{f}) + O\left(\frac{\log \mathcal{N}(\|\{\hat{a}_j\}\|_{2/L}^{2/L}, \delta)}{n}\right) + \lambda \|\{\hat{a}_j\}\|_{2/L}^{2/L}
\end{aligned} \tag{5.30}$$

Observing that $\mathrm{MSE}(\tilde{f}) \geq 0$, and $\frac{\log \mathcal{N}(\|\{\hat{a}_j\}\|_{2/L}^{2/L}, \delta)}{n} \eqsim MSE(\hat{f})$ for the optimally chosen P', taking the assumtion into the inequality proves the choice of λ. ■

Remark 5.3 *Define $R(\lambda) := R(\arg\min \hat{L}(f) + \lambda R(f))$, where $R(f) = \|\{a_j\}\|_{2/L}^{2/L}$ is the regularizer term of a parallel NN (f). Notice that $R(\lambda)$ is a non-increasing function of lambda (as proved below), the assumption in Lemma 5.18 is equivalent to that if $R(\lambda)$ contains any uncontinuous points, then the uncontinuous points should not be larger than $\frac{C_2}{C_1}$ in ratio. On the other hand, if λ is chosen as $\lambda = O(\frac{\mathrm{MSE}(\hat{f})}{P'})$, then from (5.30), we*

get

$$\lambda\|\{\tilde{a}_j\}\|_{2/L}^{2/L} \leq O(\mathrm{MSE}(\hat{f})) + O\left(\frac{\log \mathcal{N}(\|\{\hat{a}_j\}\|_{2/L}^{2/L}, \delta)}{n}\right)$$
$$\leq O(\mathrm{MSE}(\hat{f})) + \frac{1}{n}\tilde{O}((\|\{\hat{a}_j\}\|_{2/L}^{2/L})^{\frac{1}{1-2/L}})$$

If the constant term in λ is large enough, the above inequality yields two sets of solutions:

$$\|\{\tilde{a}_j\}\|_{2/L}^{2/L} \leq O\left(\|\{\hat{a}_j\}\|_{2/L}^{2/L} + \frac{1}{\lambda}\mathrm{MSE}(\hat{f})\right) = O(\|\{\hat{a}_j\}\|_{2/L}^{2/L}),$$

and

$$\|\{\tilde{a}_j\}\|_{2/L}^{2/L} \geq \tilde{O}\big((n\lambda)^{\frac{1-2/L}{2/L}}\big).$$

In the first case, one can easily see from (5.30) that $\mathrm{MSE}(\tilde{f}) \leq O(\mathrm{MSE}(\hat{f}))$, which says that the MSE of the regularized problem is close to the minimax rate; in the later case, the generalization gap of the regularized problem is bounded by $O(n^{\frac{1-2/L}{2/L}}\lambda^{L/2})$, which is much larger than the former case. So a sufficient condition of the above assumption is that the model does not overfit significantly (by orders of magnitude) more than the constrained version. In our experiment, we find that the latter case is very difficult to happen, possibly because of the implicit regularization during training, and the connection between λ and effective degree of freedom is actually smooth. Notably, as L gets larger, in the second case $\|\{\tilde{a}_j\}\|_{2/L}^{2/L}$ increases exponentially with L (the constant terms depends at most polynomially on L), which suggests that the latter case is less likely to happen for deep neural networks.

Claim 5.19 *For fixed $\mathcal{D}$, the regularized problem satisfy that $R(\lambda)$ as defined above is strictly non-increasing with λ.*

Proof: We provide a short proof by contradiction: suppose that there exists $lambda_1 < \lambda_2$, and the solution satisfy $(f_1) < R(f_2)$ where $R(f) = \|\{a_j\}\|_{2/L}^{2/L}$ is the regularizer term

of a parallel NN, f_1, f_2 are the solution to the regularized problem with $\lambda = \lambda_1, \lambda_2$ respectively. Then by definition of f_1, f_2, we have $\hat{L}(f_1) + \lambda_1 R(f_1) \leq \hat{L}(f_2) + \lambda_1 R(f_2)$, so $\lambda_1 \geq \frac{\hat{L}(f_1)-\hat{L}(f_2)}{R(f_2)-R(f_1)}$; $\hat{L}(f_2) + \lambda_2 R(f_2) \leq \hat{L}(f_1) + \lambda_2 R(f_1)$, so $\lambda_2 \leq \frac{\hat{L}(f_1)-\hat{L}(f_2)}{R(f_2)-R(f_1)}$ which is controversal to our assumption that $\lambda_1 < \lambda_2$. ■

5.9 Additional information about numerical result

5.9.1 Target Functions

The doppler function used in Figure 5.3(d)-(f) is

$$f(x) = \sin(4/(x + 0.01)) + 1.5.$$

The "vary" function used in Figure 5.3(g)-(i) is

$$\begin{aligned} f(x) = {} & M_1(x/0.01) + M_1((x - 0.02)/0.02) + M_1((x - 0.06)/0.03) \\ & + M_1((x - 0.12)/0.04) + M_3((x - 0.2)/0.02) + M_3((x - 0.28)/0.04) \\ & + M_3((x - 0.44)/0.06) + M_3((x - 0.68)/0.08), \end{aligned}$$

where M_1, M_3 are first and third order Cardinal B-spline bases functions respectively. We uniformly take 256 samples from 0 to 1 in the piecewise cubic function experiment, and uniformly 1000 samples from 0 to 1 in the doppler function and "vary" function experiment. We add zero mean independent (white) Gaussian noise to the observations. The standard derivation of noise is 0.4 in the doppler function experiment and 0.1 in the "vary" function experiment.

5.9.2 Training/Fitting Method

In the piecewise polynomial function ("vary") experiment, the depth of the PNN $L = 10$, the width of each subnetwork $w = 10$, and the model contains $M = 500$ subnetworks. The depth of NN is also 10, and the width is 240 such that the NN and PNN have almost the same number of parameters. In the doppler function experiment, the depth of the PNN $L = 12$, the width of each subnetwork $w = 10$, and the model contains $M = 2000$ subnetworks, because this problem requires a more complex model to fit. The depth of NN is 12, and the width is 470. We used Adam optimizer with learning rate of 10^{-3}. We first train the neural network layer by layer without weight decay. Specifically, we start with a two-layer neural network with the same number of subnetworks and the same width in each subnetwork, then train a three layer neural network by initializing the first layer using the trained two layer one, until the desired depth is reached. After that, we turn the weight decay parameter and train it until convergence. In both trend filtering and smoothing spline experiment, the order is 3, and in wavelet denoising experiment, we use sym4 wavelet with soft thresholding. We implement the trend filtering problem according to Tibshirani [99] using CVXPY, and use MOSEK to solve the convex optimization problem. We directly call R function *smooth.spline* to solve smoothing spline.

5.9.3 Post Processing

The degree of freedom of smoothing spline is returned by the solver in R, which is rounded to the nearest integer when plotting. To estimate the degree of freedom of trend filtering, for each choice of λ, we repeated the experiment for 10 times and compute the average number of nonzero knots as estimated degree of freedom. For neural networks,

we use the definition [140]:

$$2\sigma^2 \mathrm{df} = \mathbb{E}\|\boldsymbol{y}' - \hat{\boldsymbol{y}}\|_2^2 - \mathbb{E}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 \tag{5.31}$$

where df denotes the degree of freedom, σ^2 is the variance of the noise, $\boldsymbol{y}$ are the labels, $\hat{\boldsymbol{y}}$ are the predictions and $\boldsymbol{y}'$ are independent copy of y. We find that estimating (5.31) directly by sampling leads to large error when the degree of freedom is small. Instead, we compute

$$2\sigma^2 \hat{\mathrm{df}} = \hat{\mathbb{E}}\|\boldsymbol{y}_0 - \hat{\boldsymbol{y}}\|_2^2 - \hat{\mathbb{E}}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 + \hat{\mathbb{E}}\|\boldsymbol{y} - \bar{y}_0\|_2^2 - \|\boldsymbol{y}_0 - \bar{y}_0\|_2^2 \tag{5.32}$$

where $\hat{\mathrm{df}}$ is the estimated degree of freedom, $\mathbb{E}$ denotes the empirical average (sample mean), $\boldsymbol{y}_0$ is the target function and $\bar{y}_0$ is the mean of the target function in its domain.

Proposition 5.20 *The expectation of (5.32) over the dataset $\mathcal{D}$ equals (5.31).*

Proof:

$$\begin{aligned}
2\sigma^2 \hat{\mathrm{df}} &= \mathbb{E}_{\mathcal{D}}[\hat{\mathbb{E}}\|\boldsymbol{y}_0 - \hat{\boldsymbol{y}}\|_2^2 - \hat{\mathbb{E}}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 + \hat{\mathbb{E}}\|\boldsymbol{y} - \bar{y}_0\|_2^2 - \|\boldsymbol{y}_0 - \bar{y}_0\|_2^2] \\
&= \mathbb{E}\|\boldsymbol{y}_0 - \hat{\boldsymbol{y}}\|_2^2 - \mathbb{E}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 + \mathbb{E}_{\mathcal{D}}[\hat{\mathbb{E}}[(\boldsymbol{y} - \boldsymbol{y}_0)(\boldsymbol{y} + \boldsymbol{y}_0 - 2\bar{y}_0)]] \\
&= \mathbb{E}\|\boldsymbol{y}_0 - \hat{\boldsymbol{y}}\|_2^2 - \mathbb{E}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 + \mathbb{E}\Big[\sum_{i=1}^{n} \epsilon_i(2y_i + \epsilon_i - 2\bar{y}_0)\Big] \\
&= \mathbb{E}\|\boldsymbol{y}_0 - \hat{\boldsymbol{y}}\|_2^2 - \mathbb{E}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 + n\sigma^2 \\
&= \mathbb{E}\|\boldsymbol{y}' - \hat{\boldsymbol{y}}\|_2^2 - \mathbb{E}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2
\end{aligned}$$

where $\mathcal{D}$ denotes the dataset. In the third line, we make use of the fact that $\mathbb{E}[\epsilon_i] = 0, \mathbb{E}[\epsilon_i^2] = \sigma^2$, and in the last line, we make use of $\mathbb{E}[\epsilon_i'] = 0, \mathbb{E}[\epsilon_i'^2] = \sigma^2$, and ϵ_i' are independent of y_i and $y_{0,i}$ ■

One can easily check that a "zero predictor" (a predictor that always predict $\bar{y}_0$, and it

always predicts 0 if the target function has zero mean) always has an estimated degree of freedom of 0.

In Figure 5.3(h)(i), we take the minimum MSE over different choices of λ, and plot the average over 10 runs. Due to optimization issue, sometimes the neural networks are stuck at bad local minima and the empirical loss is larger than the global minimum by orders of magnitude. To deal with this problem, in Figure 5.3(h)(i), we manually detect these results by removing the experiments where the MSE is larger than 1.5 times the average MSE under the same setting, and remove them before computing the average.

5.9.4 More experimental results

Regularization weight vs degree-of-freedom

As we explained in the previous section, the degree of freedom is the exact information-theoretic measure of the generalization gap. A Larger degree-of-freedom implies more overfitting.

In figure Figure 5.4, we show the relationship between the estimated degree of freedom and the scaling factor of the regularizer λ in a parallel neural network and in trend filtering. As is shown in the figure, generally speaking as λ decreases towards 0, the degree of freedom should increase too. However, for parallel neural networks, if λ is very close to 0, the estimated degree of freedom will not increase although the degree of freedom is much smaller than the number of parameters — actually even smaller than the number of subnetworks. Instead, it actually decreases a little. This effect has not been observed in other nonparametic regression methods, e.g. trend filtering, which overfits every noisy datapoint perfectly when $\lambda \to 0$. But for the neural networks, even if we do not regularize at all, the among of overfitting is still relatively mild 30/256 vs 80/1000. In our experiments using neural networks, when λ is small, we denoise the estimated

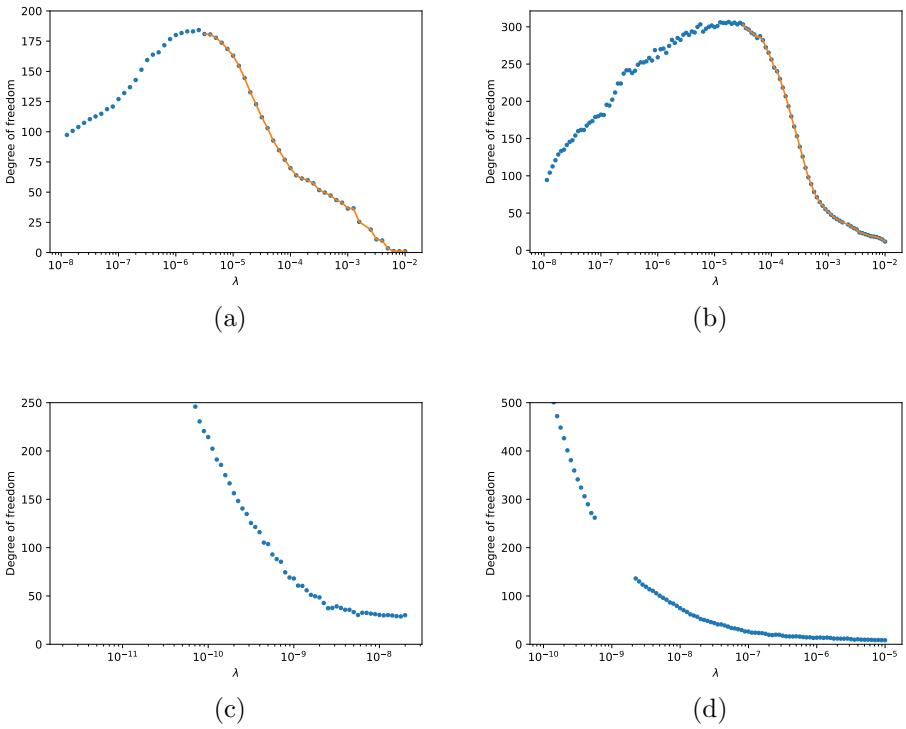


Figure 5.4: The relationship between degree of freedom and the scaling factor of the regularizer λ. The solid line shows the result after denoising. (a)(b)in a parallel NN. (c)(d) In trend filtering. (a)(c): the "vary" function. (b)(d) the doppler function.

degree of freedom using isotonic regression.

We do not know the exact reason of this curious observation. Our hypothesis is that it might be related to issues with optimization, i.e., the optimizer ends up at a local minimum that generalizes better than a global minimum; or it could be connected to the "double descent" behavior of DNN [143] under over-parameterization.

Chapter 6

Finite Overparameterization: Overparameterized ResNets for functions on manifolds

6.1 Introduction

In the previous chapter, we delved into the local adaptivity of deep neural networks in the context of nonparametric regression. Unfortunately, the practical applications of this theory are limited by the curse of dimensionality, a common issue in nonparametric regression. This phenomenon refers to the exponential increase in the number of samples required as the dimension of the data increases. While nonparametric regression techniques are typically restricted to low-dimensional data due to the curse of dimensionality, deep learning is frequently utilized for high-dimensional data and is observed to perform well in this case.

One possible explanation to this effect is that in real world applications, the high dimension data often follows a lower dimension latent space. In other words, the high di-

mension data lay on a low dimension manifold. An estimator only needs to Approximate the target function on this manifold, thus the sample complexity depends exponentially only on the latent dimension, thus overcome the problem of curse of dimensionality.

A series of work shows that neural networks can approximate functions on a low dimension manifold and overcome the curse of dimensionality [109]. In this work, we demonstrate that a weight decayed ResNet and ResNeXt can adapt to the smooth functions on a low dimension manifold. Notably, the adaptivity does not require tuning the architecture of the neural network, but only the weight decay parameter. Our investigation encompasses both vanilla neural networks and convolutional neural networks (CNNs), utilizing both ResNet and ResNeXt architectures.

6.2 Preliminary and related work

6.2.1 Besov function and smooth manifold

Besov spaces, denoted as $B^{\alpha}_{p,q}$, are a family of function spaces that are widely used in the field of functional analysis and harmonic analysis. The detained definition of Besov space is deferred to Section 5.3.2 and Section 6.5.2. It is parameterized by three parameters α, p and q, where α defines the smoothness of functions in the function space, p and q defines the norm to measure the smoothness. When $p = q = 2$, it reduces to Sobolev space. It is known that linear estimators, including kernel methods, can achieve the minimax rate on Sobolev space, but not on more generalized Besov space; the latter requires the estimator to be locally adaptivity. Estimators with this property includes wavelet smoothing [94], locally adaptive regression splines [LARS, 98], trend filtering [99, 100] and adaptive local polynomials [101, 102].

In real application, the pre-image of the target function may not be the Euclidean

space, but rather a manifold with lower *intrinsic dimension* denoted as d. A manifold is a topological space and there exists a continuous bijection mapping between this space and a d-dimensional Euclidean space. In typical applications, $d \ll D$, where D is the dimension of the Euclidean space called the *ambient dimension*. The detailed definition is deferred to Section 6.5.1.

It has been found that if the data are distributed near a low dimension smooth manifold, neural networks can explore this low dimension data structure and the performance of the neural networks can be improved [144, 145, 146, 147]. Our work aims to find if weight-decayed overparameterized neural networks can benefit from data structure.

6.2.2 ResNet and ResNeXt

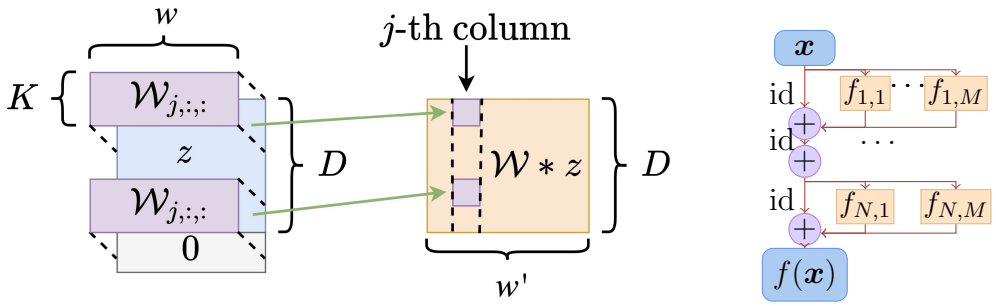


Figure 6.1: (a) Demonstration of the convolution operation $\mathcal{W} * z$, where the input is $z \in \mathbb{R}^{D \times w}$, and the output is $\mathcal{W} * z \in \mathbb{R}^{D \times w'}$. Here $\mathcal{W}_{j,:,:}$ is a $D \times w$ matrix for the j-th output channel. (b) Demonstration of the ConvResNeXt. $f_{1,1} \ldots f_{N,M}$ are the building blocks, each building block is a convolution neural network.

ResNets was proposed to mitigate the vanishing/exploding gradients in deep neural networks [148]. The key technique is to add identity mappings, also known as residual connections, that skip the connections of every L layers. This divides the neural networks into multiple building blocks. Each block can be represented as

$$y = f(\boldsymbol{x}; \mathbf{w}) + \boldsymbol{x}$$

where $f(\boldsymbol{x}; \mathbf{w})$ denotes a feedforward neural network.

ResNeXt [149] is an extension to the ResNet A parallel architecture is introduced to each building block, which enables multiple "paths" in each block. This approach allows for improving the performance without increasing the number of parameters. The detailed definition can be found in Section 6.3.

Initially, ResNet and ResNeXt were proposed for convolutional neural networks (CNNs), but the concept can be applied to vanilla neural networks as well. This paper investigates both vanilla neural networks and CNNs, applying the ResNet and ResNeXt architectures.

In cases where the number of blocks in a ResNeXt architecture is reduced to 1, it essentially becomes a parallel neural network, observing the impact of the residual connection is negligible in this case. Parallel neural networks will be discussed in detail in the next section.

It has been found that both parallel neural networks and ResNets achieve near-minimax rates on a wide range of function classes up to a logarithmic factor [106, 108, 109], which separates them from other neural network architectures.

6.2.3 Parallel neural network and ℓ_p sparse model

A series of work shows that regularization in neural networks induces sparsity. Parhi et al. [107, Theorem 8] showed that a specific regularization in a two-layer neural network is equivalent to enforcing ℓ_1 sparsity. The key technique, the AM-GM inequality, was applied in a variety of works including Srebro et al. [130] Tibshirani [131], etc. Zhang et al. [6] showed that training a parallel neural network with weight decay is equivalent to training an ℓ_p sparse model where $p = 2/L$, L is the number of layers. Making use of this result, they prove that weight decayed parallel neural networks are locally adaptive.

6.3 Main theorem

In this paper, we investigate ResNeXt, a neural network that utilizes residual connections and parallel architecture. Both feedforward neural networks and convolution neural networks can be used as the building blocks.

Definition 6.1 *Let the hidden dimension (or the number of channels) of the ResNeXt be h. The neural network comprises N residual blocks, each building block has a parallel architecture with M building blocks, and each building block contains L layers. If the feedforward neural network is used in the building blocks, denote the width of each block as w. If the convolution neural network is used in the building blocks, define the number of channels as w and the size of the kernel be K. A ResNeXt can be represented as*

$$f = \mathbf{W}_{out} \cdot \left(1 + \sum_{m=1}^{M} f_{N,m}\right) \circ \cdots \circ \left(1 + \sum_{m=1}^{M} f_{1,m}\right)$$
$$f_{n,m} = \mathbf{W}_L^{(n,m)} \star \mathrm{ReLU}(W_{L-1}^{(n,m)} \star \ldots \mathrm{ReLU}(\mathbf{W}_1^{(n,m)} \star \boldsymbol{x})),$$

where $\star$ denotes the matrix-vector product when the feedforward neural network is used in the building blocks, and convolution operation when convolution neural network is used in the building blocks.

With a weight decay regularization applied on the residual blocks and the last fully connected layer separately, let the sum norm of all the residual blocks be bounded by B_{res} and the norm of the last linear layer be bounded by B_{out}:

$$\sum_{n=1}^{N}\sum_{m=1}^{M}\sum_{\ell=1}^{L} \|\mathbf{W}_\ell^{(n,m)}\|_F^2 \leq B_{res}, \quad \|\mathbf{W}_{out}\|_F^2 \leq B_{out}.$$

All the blocks above contains no bias. The missing representation power from the bias can be compensated by appending a padding layer in front of the model. Specifically,

the input is padded with a scalar '1'. If the building blocks are feedforward neural networks, an additional scalar "0" is padded to the input, such that $h = D + 2$, where D is the dimension of the input; If the building blocks are convolution neural networks, an additional channel containing '0's is padded such that $h = 2$ and the dimension of the input to the ResNeXt is $w_{in} = D + 1$. The channel with "0" is used to accumulate the output. Besides, another linear layer is included after all the building blocks to project the hidden space to a 1-dimensional output space.

Without the loss of generalization, assume that target function is bounded $f_0(\boldsymbol{x}) \in [0, 1], \forall \boldsymbol{x} \in \mathcal{M}$. Correspondingly, assume that the output of the neural network is clipped to $[0, 1]$.

6.3.1 Approximation theory

In this section, we provide a universal approximation error of ResNeXts for Besov functions on a smooth manifold:

Theorem 6.1 *For any Besov function f_0 on a smooth manifold satisfying $p, q \geq 1, \alpha - d/p > 1$,*

$$\|f_0\|_{B^{\alpha}_{p,q}(\mathcal{M})} \leq C_f,$$

for any $P > 0$, for any ResNeXt architecture defined in Section 6.3 with the feedforward neural network as the building blocks and with parameters $M, N, L, B_{res}, B_{out}$ satisfying $L \geq 3$,

$$\begin{aligned} MN \geq C_{\mathcal{M}} P, \quad w \geq C_1(dm + D), \quad B_{res} \leq C_2 L, \\ B_{out} \leq C_3 C_f^2 ((dm + D)L)^L (C_{\mathcal{M}} P)^{L-2/p}, \end{aligned} \tag{6.1}$$

there exists an instance f of this ResNeXt class, such that

$$\|f - f_0\|_{\infty} \leq C_f C_{\mathcal{M}} \left(C_4 P^{-\alpha/d} + C_5 \exp(-C_6 L \log P) \right), \tag{6.2}$$

where C_1, C_2, C_3 are universal constants and C_4, C_5, C_6 are constants that only depends on d and m, d is the intrinsic dimension of the manifold and m is an integer satisfying $0 < \alpha < \min(m, m - 1 + 1/p)$,

The approximation error of the network is bounded by the sum of two terms. The first term is a polynomial decay term that decreases with the size of the neural network, and represents the trailing term of B-spline approximation. The second term reflects the approximation error of neural networks to piecewise polynomials, and it decreases exponentially with the number of layers. The proof is deferred to Section 6.4.1 and the appendix.

When the building blocks of the ResNeXt are the convolution neural networks, a similar result can be proven:

Theorem 6.2 *Under the same condition as Theorem 6.1, for any ResNeXt architecture defined in Section 6.3 with the convolution neural network as the building blocks and with parameters $M, N, B_{res}, B_{out}, K$ and depth L' satisfying $L' = \bar{L} + L_0 - 1, \bar{L} \geq 3$, where $L_0 = \lceil \frac{h-1}{K-1} \rceil$, and*

$$\begin{aligned} &MN \geq C_{\mathcal{M}} P, \quad w \geq C_1(dm + D), \quad B_{res} \leq C_2 L/K, \\ &B_{out} \leq C_3 C_f^2 ((dm + D)LK)^L (C_{\mathcal{M}} P)^{L-2/p}, \end{aligned} \tag{6.3}$$

there exists an instance of this ResNeXt class that has the approximation error

$$\|f - f_0\|_\infty \leq C_f C_{\mathcal{M}} \left(C_4 P^{-\alpha/d} + C_5 \exp(-C_6 \bar{L} \log P) \right), \tag{6.4}$$

where $C_1, C_2, C_3, C_4, C_5, C_6$ are the same constants as in Theorem 6.1.

This theorem is the direct result of Theorem 6.1 and Section 6.6.4, of which the latter shows that any L-layer feedforward neural network can be reformulated as an $L + L_0 - 1$-

layer convolution neural network.

6.3.2 Estimation theory

Theorem 6.3 *For any error function L that is 1-Lipschitz in its first argument, and the architecture of the neural network satisfy the condition in Theorem 6.1 and Theorem 6.2, for any function f_0 in the Besov space on a smooth manifold*

$$f_0 \in B^{\alpha}_{p,q}(\mathcal{M}), f_0(\boldsymbol{x}) \in [0,1] \ \forall \boldsymbol{x} \in \mathcal{M} \subseteq [-1,1]^D,$$

the empirical risk minimize $\hat{f} = \arg\min_f \mathbb{E}_n[\text{loss}(\hat{f}(x), y)]$ of a ResNeXt in (6.1) satisfies

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[\text{loss}(\hat{f}(x), y)] \leq \mathbb{E}_{\mathcal{D}}[L(f_0)] + C_7\Big(\frac{K^{-\frac{2}{L-2}} w^{\frac{3L-4}{L-2}} L^{\frac{3L-2}{L-2}}}{n}\Big)^{\frac{\alpha/d(1-2/L)}{2\alpha/d(1-1/L)+1-2/(pL)}} \\ + C_8 \exp(-C_6(L - L_0)),\end{aligned}$$

where the logarithmic terms are omitted. C_6 is the constant defined in Theorem 6.1 and Theorem 6.2, C_7, C_8 are constants that depend on $C_f, C_{\mathcal{M}}, d, m$. $K = 1, L_0 = 1$ when the building blocks are feedforward neural networks ; K is the size of the convolution kernel, $L_0 = \lceil \frac{h-1}{K-1} \rceil$ when the building blocks are convolution neural networks.

Furthermore, for the mean square error loss

$$\text{MSE}(f) = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - f_0(\boldsymbol{x}_i))^2$$

A tighter bound can be established:

$$\begin{aligned}\text{MSE}(\hat{f}) \leq C_9 \mathbb{E}_{\mathcal{D}}[\text{MSE}(f_0)] + C_{10}\Big(\frac{K^{-\frac{2}{L-2}} w^{\frac{3L-4}{L-2}} L^{\frac{3L-2}{L-2}}}{n}\Big)^{\frac{2\alpha/d(1-2/L)}{2\alpha/d(1-1/L)+1-2/(pL)}} \\ + C_{11} \exp(-C_6(L - L_0)).\end{aligned}$$

where C_9, C_{10} and C_{11} are some universal constants.

We would like to make the following remarks about the result:

• **Strong adaptivity:** by setting the width of the neural network to $w = 2C_1$, the model can adapt to any Besov functions on any smooth manifold satisfying $dm \leq D$ by tuning only the regularization parameter. This increases the estimation error by no more than a constant term, which is a mild price to pay for a more adaptive method.

• **No curse of dimensionality:** the above rate only depends polynomially on the ambient dimension D and exponentially on the hidden dimension d. Since in real data, the hidden dimension d can be much smaller than the ambient dimension D, this result shows that neural networks can overcome the curse of dimensionality when the data are on a low-dimension manifold.

• **Overparameterization is fine:** the number of building blocks in a ResNet and ResNeXt does not influence the estimation error as long as it is large enough. This matches the empirical observations that neural networks generalize well despite overparameterization.

• **Close to minimax rate:** with weight decay, overparameterized ResNet and ResNeXt can achieve close to the minimax rate in estimating functions in this class, and when the number of training samples n is large enough, deeper models and achiever lower estimation error. As a reference, the lower bound of the 1-Lipschitz error for any estimator θ is

$$\min_{\theta} \max_{f^* \in B^{\alpha}_{p,q}} L(\theta(\mathcal{D}), f^*) \gtrsim n^{-\frac{\alpha/d}{2\alpha/d+1}},$$

where $\gtrsim$ notation hides a factor of constant. The proof can be found in Section 6.6.8.

• **Deeper is better:** with larger L, the error rate decays faster and get closer to the minimax rate. This indicates that deeper model can achieve better performance than shallower models when the training set is large enough.

By choosing $L = O(\log(n))$, the second term in the error can be eliminated and close to the minimax rate can be achieved:

Corollary 6.4 *For any ResNet or ResNeXt satisfying the condition in Theorem 6.3, and the depth of each block is $L = O(\log(n))$, then the estimation error of the empirical risk minimizer under 1-lipschitz loss is*

$$\mathbb{E}_{\mathcal{D}}[\text{loss}(\hat{f}(x), y)] \leq \mathbb{E}_{\mathcal{D}}[\text{loss}(f_0)] + \tilde{O}(n^{-\frac{\alpha/d}{2\alpha/d+1}(1-o(1))}),$$

and that of the MSE loss is

$$\mathbb{E}_{\mathcal{D}}[\text{MSE}(\hat{f}(x), y)] \leq \tilde{O}(\mathbb{E}_{\mathcal{D}}[\text{MSE}(f_0)] + n^{-\frac{2\alpha/d}{2\alpha/d+1}(1-o(1))}),$$

where f_0 is any function satisfying the condition in Theorem 6.3, $\tilde{O}(\cdot)$ shows that the logarithmic term is omitted.

The proof of Theorem 6.3 is deferred to Section 6.4.2 and Section 6.6.7. The key technique is computing the critical radius of the local Gaussian complexity by bounding the covering number of weight-decayed ResNets and ResNeXts. This technique provides a tighter bound than choosing a single radius of the covering number as in Suzuki [106], Zhang et al. [6], for example. The covering number of an overparameterized ResNeXt with norm constraint is one of the key contribution of this paper.

6.4 Proof overview

6.4.1 Approximation error

We follow the method in Liu et al. [109] to construct a neural network that achieves the approximation error we claim. It is divided into the following steps:

1. Decompose the target function into the sum of locally supported functions.
2. Locally approximate the decomposed functions using cardinal B-spline basis functions.
3. Approximate the cardinal B-spline basis functions using neural networks.
4. Use a parallel neural network to Approximate the target function.

Decompose the target function into the sum of locally supported functions.

From the definition of the Besov function on a smooth manifold, we can decompose the target function as the sum of locally supported Besov functions on a low dimensional subspace. This decomposition exists for any construction of atlas of the manifold $\mathcal{M}$. See Section 6.5.1 for the detail.

In this work, we adopt a similar approach to [109] and partition $\mathcal{M}$ using a finite number of open balls on $\mathbb{R}^D$. Specifically, we define $B(\boldsymbol{c}_i, r)$ as the set of unit balls with center $\boldsymbol{c}_i$ and radius r such that their union covers the manifold of interest, i.e., $\mathcal{M} \subseteq \cup_{i=1}^{C_\mathcal{M}} B(\boldsymbol{c}_i, r)$. This allows us to partition the manifold into subregions $U_i = B(\boldsymbol{c}_i, r) \cup \mathcal{M}$, and further decompose a smooth function on the manifold into sum of locally supported smooth functions with linear projections.

Lemma 6.5 *Approximating Besov function on a smooth manifold using B-spline: Let $f \in B^\alpha_{p,q}(\mathcal{M})$. There exists a decomposition of f:*

$$f(\boldsymbol{x}) = \sum_{i=1}^{C_\mathcal{M}} \tilde{f}_i \circ \phi_i(\boldsymbol{x}) \times \mathbf{1}(\boldsymbol{x} \in B(\boldsymbol{c}_i, r))$$

and $\tilde{f}_i \in B^\alpha_{p,q}$, $\sum_{i=1}^{C_\mathcal{M}} \|\tilde{f}_i\|_{B^\alpha_{p,q}} \leq C\|f\|_{B^\alpha_{p,q}(\mathcal{M})}$, $\phi_i : \mathcal{M} \to \mathbb{R}^d$ are linear projections, $B(\boldsymbol{c}_i, r)$ denotes the unit ball with radius r and center $\boldsymbol{c}_i$.

The lemma is inferred by the existence of the partition of unity, which is shown below for reference:

Proposition 6.6 (Existence of a $\mathcal{C}^\infty$ partition of unity, Proposition 1 in [109]) *Let $\{U_\alpha\}_{\alpha\in\mathcal{A}}$ be a locally finite cover of a smooth manifold $\mathcal{M}$. There is a $\mathcal{C}^\infty$ partition of unity $\{\rho_\alpha\}_{\alpha\in\mathcal{A}}$ such that* $\text{supp}(\rho_\alpha) \in U_\alpha$.

Locally approximate the decomposed functions using cardinal B-spline basis functions.

In the second step, we decompose the locally supported Besov functions achieved in the first step using B-spline basis functions. The existence of the decomposition was proven in [136], and was applied in a series of words Zhang et al. [6], Suzuki [106], Liu et al. [109]. The difference between our result and previous work is that we define a norm on the coefficients and bound this norm, instead of bounding the maximum value.

Proposition 6.7 *For any function in the Besov space on a compact smooth manifold $f^* \in B^s_{p,q}(\mathcal{M})$, any $N \geq 0$, there exists an approximated to f^* using cardinal B-spline basis functions:*

$$\tilde{f} = \sum_{i=1}^{C_\mathcal{M}} \sum_{j=1}^{P} a_{i,k_j,\boldsymbol{s}_j} M_{m,k_j,\boldsymbol{s}_j} \circ \phi_i \times \mathbf{1}(x \in B(\boldsymbol{c}_i, r))$$

where m is a integer satisfying $0 < \alpha < min(m, m-1+1/p)$, $M_{m,k,\boldsymbol{s}} = M_m(2^k(\cdot - \boldsymbol{s}))$, M_m denotes the B-spline basis function defined in (5.2), the approximation error is bounded by

$$\|f - \tilde{f}\|_\infty \leq C_{12} C_\mathcal{M} P^{-\alpha/d}$$

and the coefficients satisfy

$$\|\{2^{k_j} a_{i,k_j,\boldsymbol{s}_j}\}_{i,j}\|_p \leq C_{13} \|f\|_{B^\alpha_{p,q}(\mathcal{M})}$$

The proof is deferred to Section 6.6.1. As will be shown below, the scaled coefficients $2^{k_j} a_{i,k_j,\boldsymbol{s}_j}$ corresponds to the total norm of the parameters in the neural network to approximate the B-spline basis function, so this lemma

Approximate the cardinal B-spline basis functions using neural networks.

As has been shown in [6, 106, 109], a neural network can approximate B-spline basis functions, and the error decreases exponentially with the number of layers.

Lemma 6.8 (Lemma 11 in [6]) *Let $M_{m,k,\boldsymbol{s}}$ be the B-spline of order m with scale 2^{-k} in each dimension and position $\boldsymbol{s} \in \mathbb{R}^d$: $M_{m,k,\boldsymbol{s}}(\boldsymbol{x}) := M_m(2^k(\boldsymbol{x} - \boldsymbol{s}))$, M_m is defined in (5.2). There exists a neural network with d-dimensional input and one output, with width $w_{d,m} = O(dm)$ and depth $L \lesssim \log(C_{14}/\epsilon)$ for some constant C_{14} that depends only on m and d, approximates the B spline basis function $M_{m,k,\boldsymbol{s}}(\boldsymbol{x}) := M_m(2^k(\boldsymbol{x} - \boldsymbol{s}))$. This neural network, denoted as $\tilde{M}_{m,k,\boldsymbol{s}}(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^d$, satisfy*

- $|\tilde{M}_{m,k,\boldsymbol{s}}(\boldsymbol{x}) - M_{m,k,\boldsymbol{s}}(\boldsymbol{x})| \leq \epsilon$, *if* $0 \leq 2^k(x_i - s_i) \leq m + 1, \forall i \in [d]$,
- $\tilde{M}_{m,k,\boldsymbol{s}}(\boldsymbol{x}) = 0$, *otherwise.*
- *The total square norm of the weights is bounded by $2^{2k/L} C_{15} dmL$ for some universal constant C_{15}.*

Use a ResNeXt to Approximate the target function.

Using the results above, the target function can be(approximately) decomposed as

$$\sum_{i=1}^{C_{\mathcal{M}}} \sum_{j=1}^{P} a_{i,k_j,\boldsymbol{s}_j} M_{m,k_j,\boldsymbol{s}_j} \circ \phi_i \times \mathbf{1}(x \in B(\boldsymbol{c}_i, r)). \tag{6.5}$$

We first demonstrate that a ReLU neural network can approximate

$$y \times \mathbf{1}(x \in B_{r,i})$$

where $\tilde{\times}$ satisfy that $y\tilde{\times}1 = y$ for all y, and $y\tilde{\times}\tilde{x} = 0$ if any of x or y is 0, and the soft indicator function $\tilde{\mathbf{1}}(x \in B_{r,i})$ satisfy $\tilde{\mathbf{1}}(x \in B_{r,i}) = 1$ when $x \in B_{r,i}$, and $\tilde{\mathbf{1}}(x \in B_{r,i}) = 0$ when $x \notin B_{r+\Delta,i}$. The detail is deferred to Section 6.6.2.

Then, we show that it is possible to construct a ResNeXt with $MN = P, W_{out} = [0, \ldots, 0, 1]$, such that each building block takes the input from the first $D+1$-th channels and accummulates the output to the $D + 2$-th channel. The k-th building block (the position of the block does not matter) approximates

$$a_{i,k_j,\boldsymbol{s}_j} M_{m,k_j,\boldsymbol{s}_j} \circ \phi_i \times \mathbf{1}(x \in B(\boldsymbol{c}_i, r))$$

where $i = ceiling(k/N), j = rem(k, N)$. Each building block has width $C_1(md + D)$ and depth L, where $0 < \alpha < min(m, m - 1 + 1/p)$, where a sub-block with width D and depth $L-1$ approximates the chart selection, a sub-block with width md and depth $L - 1$ approximates the B-spline function, and the last layer approximates the multiply function. The norm of this block is bounded by

$$\sum_{\ell=1}^{L} \|\mathbf{W}_\ell^{(i,j)}\|_F^2 \leq O(2^{2k/L} dmL + DL). \tag{6.6}$$

Making use of the 1-homogeneous property of ReLU layers, we can further scale all the weights in the residual blocks by $\frac{1}{\tilde{B}}$, and scale W_{out} by $\tilde{B}^L$ such that the new network we construct satisfy the constraint in (6.1). See Section 6.6.3 for the detail.

6.4.2 Estimation error

We first prove the covering number of an overparameterized ResNeXt with norm-constraint as in Lemma 6.9, then compute the critical radius of this function class using the covering number as in Corollary 6.16. The critical radius can be used to bound the estimation error for 1-Lipschitz loss as in Theorem 6.17, or for MSE using self-bounding trick as in Theorem 6.18. The proof is deferred to Section 6.6.7.

Lemma 6.9 *Consider a neural network defined in Section 6.3. Let the last layer of this neural network is a single linear layer with norm $\|W_{out}\|_F^2 \leq B_{out}$. Let the input of this neural network satisfy $\|x\|_2 \leq 1, \forall x$, and is concatenated with 1 before feeding into this neural network so that part of the weight plays the role of the bias. The covering number of this neural network is bounded by*

$$\log \mathcal{N}(\cdot, \delta) \lesssim w^2 L B_{res}^{\frac{1}{1-2/L}} K^{\frac{2-2/L}{1-2/L}} \left(B_{out}^{1/2} \exp((KB_{res}/L)^{L/2})\right)^{\frac{2/L}{1-2/L}} \delta^{-\frac{2/L}{1-2/L}} \tag{6.7}$$

where the logarithmic term is omitted, $K = 1$ when the feedforward neural network is the building blocks and K is the size of the convolution kernel when the convolution neural network is the building blocks.

Proof: Using AM-GM inequality, From Proposition 6.19, Proposition 6.21 and Proposition 6.22, if any residual block is removed, the perturbation to the output is no more than

$$(KB_m/L)^{L/2} B_{out}^{1/2} \exp((KB_{res}/L)^{L/2})$$

where B_m is the total norm of parameters in this block, $K = 1$ when the feedforward neural network is the building blocks and K is the size of the convolution kernel when the convolution neural network is the building blocks. Because of that, the residual blocks can be divided into two kinds depending on the norm of the weights $B_m < \epsilon$

("small blocks") and $B_m \geq \epsilon$ ("large blocks"). If all the "small blocks" are removed, the perturbation to the output for any input $\|x\|_2 \leq 1$ is no more than

$$\begin{aligned}
&\exp((KB_{res}/L)^{L/2})B_{out}^{1/2} \sum_{m:B_m<\epsilon} (KB_m/L)^{L/2} \\
&\leq \exp((KB_{res}/L)^{L/2})B_{out}^{1/2} \sum_{m:B_m<\epsilon} (KB_m/L)(K\epsilon/L)^{L/2-1} \\
&\leq \exp((KB_{res}/L)^{L/2})K^{L/2}B_{res}B_{out}^{1/2}(\epsilon/L)^{L/2-1}/L
\end{aligned}$$

Choosing $\epsilon = L\left(\frac{\delta L}{2\exp((B_{\rm res}/L)^{L/2})K^{L/2}B_{\rm res}B_{\rm out}^{1/2}}\right)^{\frac{1}{L/2-1}}$, the perturbation above is no more than $\delta/2$. The covering number can be determined by the number of the "large blocks" in the neural network, which is no more than B/ϵ.

Taking our choice of ϵ into Proposition 6.15 and noting that for any block, $B_{\rm in}L_{\rm post} \leq B_{\rm out}^{1/2}\exp((KB_{\rm res}/L)^{L/2})$ finishes the proof, where $B_{\rm in}$ is the upper bound of the input to this block as defines in Proposition 6.15, and $L_{\rm post}$ is the Lipschitze parameter of all the layers following the block.

Taking our choise of ϵ into Proposition 6.15 and **??** finishes the proof.

■

Remark 6.1 *The proof of Lemma 6.9 shows that under weight decay, the building blocks in a ResNet or ResNeXt is sparse, i.e. only a finite number of blocks contribute non-trivially to the network even though the model can be overparameterized. This explains why a ResNet or ResNeXt can generalize well despite overparameterization, and provide a new prospective in explaining why residual connections improves the performance of deep neural networks.*

6.5 Discussion

In this paper, we study the approximation and estimation error of a ResNet and ResNeXt. We show that with property weight decay, the blocks in a ResNet or ResNeXt converges to a sparse representation, so the covering number of a ResNet and ResNeXt does not depend only on the total norm of the parameters and not on the number of residual blocks, which allows an overparameterized neural network to generalize. Suppose that the target function is supported on a smooth manifold, the estimation error of ResNet and ResNeXt depends only weakly on the ambient dimension of the target function, which shows that these models do not suffer from the curse of dimensionality, thus can adapt to functions on a smooth manifold.

sectionIntroduction to Besov space and smooth manifold For the ease of the readers, in this section, we provide detailed definition of the Besov space and smooth manifold.

6.5.1 Smooth manifold

Definition 6.2 (Chart) *A chart on $\mathcal{M}$ is a pair (U, ϕ) such that $U \subset \mathcal{M}$ is open and $\phi : U \mapsto \mathbb{R}^d$, where ϕ is a homeomorphism (i.e., bijective, ϕ and ϕ^{-1} are both continuous).*

In a chart (U, ϕ), U is called a coordinate neighborhood and ϕ is a coordinate system on U. Essentially, a chart is a local coordinate system on $\mathcal{M}$. A collection of charts which covers $\mathcal{M}$ is called an atlas of $\mathcal{M}$.

Definition 6.3 (C^k Atlas) *A C^k atlas for $\mathcal{M}$ is a collection of charts $\{(U_\alpha, \phi_\alpha)\}_{\alpha \in \mathcal{A}}$ which satisfies $\bigcup_{\alpha \in \mathcal{A}} U_\alpha = \mathcal{M}$, and are pairwise C^k compatible:*

$$\phi_\alpha \circ \phi_\beta^{-1} : \phi_\beta(U_\alpha \cap U_\beta) \to \phi_\alpha(U_\alpha \cap U_\beta) \quad \textit{and} \quad \phi_\beta \circ \phi_\alpha^{-1} : \phi_\alpha(U_\alpha \cap U_\beta) \to \phi_\beta(U_\alpha \cap U_\beta)$$

are both C^k for any $\alpha, \beta \in \mathcal{A}$. An atlas is called finite if it contains finitely many charts.

Definition 6.4 (Smooth Manifold) *A smooth manifold is a manifold $\mathcal{M}$ together with a C^∞ atlas.*

Classical examples of smooth manifolds are the Euclidean space, the torus and the unit sphere. Furthermore, we define C^s functions on a smooth manifold $\mathcal{M}$ as follows:

Definition 6.5 (C^s functions on $\mathcal{M}$) *Let $\mathcal{M}$ be a smooth manifold and $f : \mathcal{M} \to \mathbb{R}$ be a function on $\mathcal{M}$. A function $f : \mathcal{M} \to \mathbb{R}$ is C^s if for any chart (U, ϕ) on $\mathcal{M}$, the composition $f \circ \phi^{-1} : \phi(U) \to \mathbb{R}$ is a continuously differentiable up to order s.*

We next define the C^∞ partition of unity which is an important tool for the study of functions on manifolds.

Definition 6.6 (Partition of Unity, Definition 13.4 in [150]) *A C^∞ partition of unity on a manifold $\mathcal{M}$ is a collection of C^∞ functions $\{\rho_\alpha\}_{\alpha\in\mathcal{A}}$ with $\rho_\alpha : \mathcal{M} \to [0, 1]$ such that for any $x \in \mathcal{M}$,*

1. *there is a neighbourhood of x where only a finite number of the functions in $\{\rho_\alpha\}_{\alpha\in\mathcal{A}}$ are nonzero;*
2. $\sum_{\alpha\in\mathcal{A}} \rho_\alpha(x) = 1.$

An open cover of a manifold $\mathcal{M}$ is called locally finite if every $x \in \mathcal{M}$ has a neighbourhood which intersects with a finite number of sets in the cover. The following proposition shows that a C^∞ partition of unity for a smooth manifold always exists.

Proposition 6.10 (Existence of a C^∞ partition of unity, Theorem 13.7 in [150]) *Let $\{U_\alpha\}_{\alpha\in\mathcal{A}}$ be a locally finite cover of a smooth manifold $\mathcal{M}$. Then there is a C^∞ partition of unity $\{\rho_\alpha\}_{\alpha=1}^\infty$ where every ρ_α has a compact support such that* $\mathrm{supp}(\rho_\alpha) \subset U_\alpha$.

Let $\{(U_\alpha, \phi_\alpha)\}_{\alpha\in\mathcal{A}}$ be a C^∞ atlas of $\mathcal{M}$. Proposition 6.10 guarantees the existence of a partition of unity $\{\rho_\alpha\}_{\alpha\in\mathcal{A}}$ such that ρ_α is supported on U_α.

To characterize the curvature of a manifold, we adopt the geometric concept, reach.

Definition 6.7 (Reach [151, 152]) *Denote*

$$G = \left\{ x \in \mathbb{R}^D : \exists p \neq q \in \mathcal{M} \text{ such that } \|x-p\|_2 = \|x-q\|_2 \} = \inf_{y\in\mathcal{M}} \|x-y\|_2 \right\}.$$

as the set of points that have at least two nearest neighbors on $\mathcal{M}$. The closure of G is called the medial axis of $\mathcal{M}$. Then the reach of $\mathcal{M}$ is defined as

$$\tau = \inf_{x\in\mathcal{M}} \inf_{y\in G} \|x-y\|_2.$$

Reach has a simple geometrical interpretation: for every point $x \in \mathcal{M}$, the radius of the osculating circle is at least τ. A large reach for $\mathcal{M}$ indicates that the manifold changes slowly, as illustrated in Figure 6.2.

Slow Change: Large τ Rapid Change: Small τ

Figure 6.2: Illustration of manifolds with large and small reach.

6.5.2 Besov functions on a smooth manifold

We next define Besov function spaces on the smooth manifold $\mathcal{M}$, which generalizes more elementary function spaces such as the Sobolev and Hölder spaces.

The definition of Besov class can be found in Section 5.3.2.

We define $B^{\alpha}_{p,q}$ functions on $\mathcal{M}$.

Definition 6.8 ($B^{\alpha}_{p,q}$ Functions on $\mathcal{M}$ [153, 154]) *Let $\mathcal{M}$ be a compact smooth manifold of dimension d. Let $\{(U_i, \phi_i)\}_{i=1}^{C_{\mathcal{M}}}$ be a finite atlas on $\mathcal{M}$ and $\{\rho_i\}_{i=1}^{C_{\mathcal{M}}}$ be a partition*

of unity on $\mathcal{M}$ *such that* $\text{supp}(\rho_i) \subset U_i$. *A function* $f : \mathcal{M} \to \mathbb{R}$ *is in* $B^{\alpha}_{p,q}(\mathcal{M})$ *if*

$$\|f\|_{B^{\alpha}_{p,q}(\mathcal{M})} := \sum_{i=1}^{C_{\mathcal{M}}} \|(f\rho_i) \circ \phi_i^{-1}\|_{B^{\alpha}_{p,q}(\mathbb{R}^d)} < \infty. \tag{6.8}$$

Since ρ_i is supported on U_i, the function $(f\rho_i) \circ \phi_i^{-1}$ is supported on $\phi(U_i)$. We can extend $(f\rho_i) \circ \phi_i^{-1}$ from $\phi(U_i)$ to $\mathbb{R}^d$ by setting the function to be 0 on $\mathbb{R}^d \setminus \phi(U_i)$. The extended function lies in the Besov space $B^s_{p,q}(\mathbb{R}^d)$ [154, Chapter 7].

6.6 Proof of technical results

6.6.1 Locally approximate the decomposed functions using cardinal B-spline basis functions.

In this section, we provide the proof to Proposition 6.7 From the definition of $B^{\alpha}_{p,q}(\mathcal{M})$, and applying Proposition 6.6, there exists a decomposition of f^* as

$$f^* = \sum_{i=1}^{C_{\mathcal{M}}} (f_i) = \sum_{i=1}^{C_{\mathcal{M}}} (f_i \circ \phi_i^{-1}) \circ \phi_i \times \mathbf{1}_{U_i}$$

where $f_i := f^* \cdot \rho_i$, ρ_i satisfy the condition in Definition 6.6, and $f_i \circ \phi_i^{-1} \in B^{\alpha}_{p,q}$. Using Proposition 5.8, for any i, one can approximate $f_i \circ \phi_i^{-1}$ with $\bar{f}_i$:

$$\bar{f}_i = \sum_{j=1}^{P} a_{i,k_j,\boldsymbol{s}_j} M_{m,k_j,\boldsymbol{s}_j}$$

such that $\|f_i \circ \phi_i^{-1}\|_{\infty} \leq C_1 M^{-\alpha/d}$, and the coefficients satisfy $\|\{2^{k_j} a_{k_j,\boldsymbol{s}_j}\}_j\|_p \leq C_{13} \|f_i \circ \phi_i^{-1}\|_{B^{\alpha}_{p,q}}$. Define

$$\bar{f} = \sum_{i=1}^{C_{\mathcal{M}}} \bar{f}_i \circ \phi_i \times \mathbf{1}_{U_i}$$

one can verify that $\|f - \tilde{f}\|_\infty \leq C_{12} C_{\mathcal{M}} N^{-\alpha/d}$. On the other hand, using triangular inequality (and padding the vectors with 0),

$$\|\{2^{k_j} a_{i,k_j,\boldsymbol{s}_j}\}_{i,j}\|_p \leq \sum_{i=1}^{C_{\mathcal{M}}} \|\{2^{k_j} a_{i,k_j,\boldsymbol{s}_j}\}_j\|_p \leq \sum_{i=1}^{C_{\mathcal{M}}} C_{13} \|f_i \circ \phi_i^{-1}\|_{B^\alpha_{p,q}} = C_{13} \|f^*\|_{B^\alpha_{p,q}(\mathcal{M})},$$

which finishes the proof.

6.6.2 Neural network for chart selection

In this section, we demonstrate that a feedforward neural network can approximate the chart selection function $z \times \mathbf{1}(\boldsymbol{x} \in B(\boldsymbol{c}_i, r))$, and it is error-free as long as $z = 0$ when $r < d(\boldsymbol{x}, \boldsymbol{c}_i) < R$. We start by proving the following supporting lemma:

Proposition 6.11 *Fix some constant $B > 0$. For any $\boldsymbol{x}, \boldsymbol{c} \in \mathrm{R}^D$ satisfying $|x_i| \leq B$ and $|c_i| \leq B$ for $i = 1, \ldots, D$, there exists an L-layer neural network $\tilde{d}(\boldsymbol{x}; \boldsymbol{c})$ with width $w = O(d)$ that approximates $d^2(\boldsymbol{x}; \boldsymbol{c}) = \sum_{i=1}^D (x_i - c_i)^2$ such that $|\tilde{d^2}(\boldsymbol{x}; \boldsymbol{c}) - d^2(\boldsymbol{x}; \boldsymbol{c})| \leq 8DB^2 \exp(-C_{16} L)$ with an absolute constant $C_{16} > 0$ when $d(\boldsymbol{x}; \boldsymbol{c}) < \tau$, and $\tilde{d^2}(\boldsymbol{x}; \boldsymbol{c}) \geq \tau^2$ when $d(\boldsymbol{x}; \boldsymbol{c}) \geq \tau$, and the norm of the neural network is bounded by*

$$\sum_{\ell=1}^{L} \|W_\ell\|_F^2 + \|b_\ell\|_2^2 \leq C_{17} DL$$

Proof: The proof is given by construction. By Proposition 2 in Yarotsky(2017), the function $f(x) = x^2$ on the segment $[0, 2B]$ can be approximated with any error $\epsilon > 0$ by a ReLU network g having depth and the number of neurons and weight parameters no more than $c \log(4B^2/\epsilon)$ with an absolute constant c. The width of the network g is an absolute constant. We also consider a single layer ReLU neural network $h(t) = \mathrm{ReLU}(t) - \mathrm{ReLU}(-t)$, which is equal to the absolute value of the input.

Now we consider a neural network $G(\boldsymbol{x}; \boldsymbol{c}) = \sum_{i=1}^D g \circ h(x_i - c_i)$. Then for any

$\boldsymbol{x}, \boldsymbol{c} \in \mathrm{R}^D$ satisfying $|x_i| \leq B$ and $|c_i| \leq B$ for $i = 1, \ldots, D$, we have

$$\begin{aligned} |G(\boldsymbol{x}; \boldsymbol{c}) - d^2(\boldsymbol{x}; \boldsymbol{c})| &\leq \left| \sum_{i=1}^{D} g \circ h(x_i - c_i) - \sum_{i=1}^{D} (x_i - c_i)^2 \right| \\ &\leq \sum_{i=1}^{D} \left| g \circ h(x_i - c_i) - (x_i - c_i)^2 \right| \\ &\leq D\epsilon. \end{aligned}$$

Moreover, define another neural network

$$\begin{aligned} F(\boldsymbol{x}; \boldsymbol{c}) &= -\mathrm{ReLU}(\tau^2 - D\epsilon - G(\boldsymbol{x}; \boldsymbol{c})) + \tau^2 \\ &= \begin{cases} G(\boldsymbol{x}; \boldsymbol{c}) + D\epsilon & \text{if } G(\boldsymbol{x}; \boldsymbol{c}) < \tau^2 - D\epsilon, \\ \tau^2 & \text{if } G(\boldsymbol{x}; \boldsymbol{c}) \geq \tau^2 - D\epsilon, \end{cases} \end{aligned}$$

which has depth and the number of neurons no more than $c' \log(4B^2/\epsilon)$ with an absolute constant c'. The weight parameters of G are upper bounded by $\max\{\tau^2, D\epsilon, c \log(4B^2/\epsilon)\}$ and the width of G is $O(D)$.

If $d^2(\boldsymbol{x}; \boldsymbol{c}) < \tau^2$, we have

$$\begin{aligned} |F(\boldsymbol{x}; \boldsymbol{c}) - d^2(\boldsymbol{x}; \boldsymbol{c})| &= |-\mathrm{ReLU}(\tau^2 - D\epsilon - G(\boldsymbol{x}; \boldsymbol{c})) + \tau^2 - d^2(\boldsymbol{x}; \boldsymbol{c})| \\ &= \begin{cases} |G(\boldsymbol{x}; \boldsymbol{c}) - d^2(\boldsymbol{x}; \boldsymbol{c}) + D\epsilon| & \text{if } G(\boldsymbol{x}; \boldsymbol{c}) < \tau^2 - D\epsilon, \\ \tau^2 - d^2(\boldsymbol{x}; \boldsymbol{c}) & \text{if } G(\boldsymbol{x}; \boldsymbol{c}) \geq \tau^2 - D\epsilon. \end{cases} \end{aligned}$$

For the first case when $G(\boldsymbol{x}; \boldsymbol{c}) < \tau^2 - D\epsilon$, $|F(\boldsymbol{x}; \boldsymbol{c}) - d^2(\boldsymbol{x}; \boldsymbol{c})| \leq 2D\epsilon$ since $d^2(\boldsymbol{x}; \boldsymbol{c})$ can be approximated by $G(\boldsymbol{x}; \boldsymbol{c})$ up to an error ϵ. For the second case when $G(\boldsymbol{x}; \boldsymbol{c}) \geq \tau^2 - D\epsilon$, we have $d^2(\boldsymbol{x}; \boldsymbol{c}) \geq G(\boldsymbol{x}; \boldsymbol{c}) - D\epsilon \geq \tau^2 - 2D\epsilon$ and . Thereby we also have $|F(\boldsymbol{x}; \boldsymbol{c}) - d^2(\boldsymbol{x}; \boldsymbol{c})| \leq 2D\epsilon$.

If $d^2(\boldsymbol{x};\boldsymbol{c}) \geq \tau^2$ instead, we will obtain $G(\boldsymbol{x};\boldsymbol{c}) \geq d^2(\boldsymbol{x};\boldsymbol{c}) - D\epsilon \geq \tau^2 - D\epsilon$. This gives that $F(\boldsymbol{x};\boldsymbol{c}) = \tau^2$ in this case.

Finally, we take $\epsilon = 4B^2\exp(-L/c')$. Then $F(\boldsymbol{x};\boldsymbol{c})$ is an L-layer neural network with $O(L)$ neurons. The weight parameters of G are upper bounded by $\max\{\tau^2, 4DB^2\exp(-L/c'), cL/c'\}$ and the width of G is $O(D)$. Moreover, $F(\boldsymbol{x};\boldsymbol{c})$ satisfies $|F(\boldsymbol{x};\boldsymbol{c}) - d^2(\boldsymbol{x};\boldsymbol{c})| < 8DB^2\exp(-L/c')$ if $d^2(\boldsymbol{x};\boldsymbol{c}) \leq \tau^2$ and $F(\boldsymbol{x};\boldsymbol{c}) = \tau^2$ if $d^2(\boldsymbol{x};\boldsymbol{c}) \geq \tau^2$. ■

Proposition 6.12 *There exists a single layer ReLU neural network that approximates* $\tilde{\times}$*, such that for all* $0 \leq x \leq C, y \in \{0,1\}$*,* $x\tilde{\times}y = x$ *when* $y = 1$*, and* $x\tilde{\times}y = 0$ *when either* $x = 0$ *or* $y = 0$*.*

Proof: Consider a single layer neural network $g(x,y) := A_2\mathrm{ReLU}(A_1(x,y)^\top)$ with no bias, where

$$A_1 = \begin{bmatrix} -\frac{1}{C} & 1 \\ 0 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -C \\ C \end{bmatrix}.$$

Then we can rewrite the neural network g as $g(x,y) = -C\mathrm{ReLU}(-x/C+y)+C\mathrm{ReLU}(y)$. If $y = 1$, we will have $g(x,y) = -C\mathrm{ReLU}(-x/C+1)+C = x$, since $x \leq C$. If $y = 0$, we will have $g(x,y) = -C\mathrm{ReLU}(-x/C) = 0$, since $x \geq 0$. Thereby we can conclude the proof. ■

By adding a single linear layer

$$y = \frac{1}{R-r-2\Delta}(\mathrm{ReLU}(R-\Delta-x) - \mathrm{ReLU}(r+\Delta-x))$$

after the one shown in Proposition 6.11, where $\Delta = 8DB^2\exp(-CL)$ denotes the error in Proposition 6.11, one can approximate the indicator function $\mathbf{1}(\boldsymbol{x} \in B(\boldsymbol{c}_i, r))$ such that it is error-free when $d(\boldsymbol{x},\boldsymbol{c}_i) \leq r$ or $\geq R$. Choosing $R \leq \tau/2, r < R - 2\Delta$, and combining with Proposition 6.12, the proof is finished. Considering that f_i is locally supported on

$B(\boldsymbol{c}_i, r)$ for all i by our method of construction, the chart selection part does not incur any error in the output.

6.6.3 Constructing the neural network to Approximate the target function

In this section, we provide the proof to Theorem 6.1. Combining Lemma 6.8, Proposition 6.11 and Proposition 6.12, by putting the neural network in Lemma 6.8 and Proposition 6.11 in parallel and adding the one in Proposition 6.12 after them, one can construct a feedforward neural network with bias with depth L, width $w = O(d) + O(D) = O(d)$, that approximates $M_{m,k_j,\boldsymbol{s}_j}(\boldsymbol{x}) \times \mathbf{1}(\boldsymbol{x} \in B(\boldsymbol{c}_i, r))$ for any i, j.

To construct a ResNeXt that approximates f_0, we follow the method in Oono et al. [108], Liu et al. [109]. Specifically, let the neural network constructed above has parameter $\tilde{\mathbf{W}}_1^{(i,j)}, \tilde{\boldsymbol{b}}_1^{(i,j)}, \ldots, \tilde{\mathbf{W}}_L^{(i,j)}, \boldsymbol{b}_L^{(i,j)}$ in each layer, one can construct a feedforward block without bias as

$$\mathbf{W}_1^{(i,j)} = \begin{bmatrix} \tilde{\mathbf{W}}_1^{(i,j)} & \tilde{\boldsymbol{b}}_1^{(i,j)} & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{W}_\ell^{(i,j)} = \begin{bmatrix} \tilde{\mathbf{W}}_\ell^{(i,j)} & \tilde{\boldsymbol{b}}_\ell^{(i,j)} \\ 0 & 1 \end{bmatrix} \quad \mathbf{W}_L^{(i,j)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \tilde{\mathbf{W}}_L^{(i,j)} & \tilde{\boldsymbol{b}}_L^{(i,j)} \end{bmatrix}.$$

Remind that the input is padded with $[1, 0]^T$ before feeding into the neural network, the above construction provide an equivalent representation to the neural network including the bias, and route the output to the last channel. From Lemma 6.8, it can be seen that the total square norm of this block is bounded by (6.6).

To approximate our decomposition of the target function as in (6.5), we only need to scale the all the weights in this block with $|a_{i,k_j,\boldsymbol{s}_j}|^{1/L}$, setting the sign of the weight in the last layer as $\text{sign}(a_{i,k_j,\boldsymbol{s}_j})$, and place $C_{\mathcal{M}}P$ number of these blocks in a ResNeXt.

Since this block always output 0 in the first $D+1$ channels, the order and the placement of the feedforward blocks does not change the output. The last fully connected layer can be simply set to

$$\mathbf{W}_{out} = [0, \dots, 0, 1], b_{out} = 0.$$

Combining Proposition 5.8 and Lemma 5.15, the norm of this ResNeXt we construct satisfy

$$\begin{aligned}
\bar{B}_{res} &\leq \sum_{i=1}^{C_{\mathcal{M}}} \sum_{j=1}^{P} a_{i,k_j,\boldsymbol{s}_j}^{2/L} (2^{2k/L} C_{15} dmL + C_{17} DL) \\
&\leq \sum_{i=1}^{C_{\mathcal{M}}} \sum_{j=1}^{P} (2^k a_{i,k_j,\boldsymbol{s}_j})^{2/L} (C_{15} dmL + C_{17} DL) \\
&\leq (C_{\mathcal{M}} P)^{1-2/(pL)} \|\{2^k a_{i,k_j,\boldsymbol{s}_j}\}\|_p^{2/L} (C_{15} dmL + C_{17} DL) \\
&\leq (C_{13} C_f)^{2/L} (C_{\mathcal{M}} P)^{1-2/(pL)} (C_{15} dmL + C_{17} DL), \\
\bar{B}_{out} &\leq 1.
\end{aligned}$$

By scaling all the weights in the residual blocks by $\bar{B}_{res}^{-1/2}$, and scaling the output layer by $\bar{B}_{res}^{L/2}$, the network that satisfy (6.1) can be constructed.

Notice that the chart selection part does not introduce error by our way of construction, we only need to sum over the error in Section 6.4.1 and Section 6.4.1, and notice that for any $\boldsymbol{x}$, for any linear projection ϕ_i, the number of B-spline basis functions $M_{m,k,\boldsymbol{s}}$ that is nonzero on $\boldsymbol{x}$ is no more than $m^d \log P$, the approximation error of the constructed neural network can be proved.

6.6.4 Constructing a convolution neural network to approximate the target function

In this section, we prove that any feedforward neural network can be realized by a convolution neural network with similar size and norm of parameters. The proof is

similar to Theorem 5 in [108].

Lemma 6.13 *For any feedforward neural network with depth L, width w, input dimension h and output dimension h', for any kernel size $K > 1$, there exists a convolution neural network with depth $L' = L + L_0 - 1$, where $L_0 = \lceil \frac{h-1}{K-1} \rceil$ number of channels $w' = 4w$, and the first dimension of the output equals the output of the feedforward neural network for all inputs, and the norm of the convolution neural network is bounded as*

$$\sum_{\ell=1}^{L'} \|\mathbf{W}'_\ell\|_F^2 \leq 4 \sum_{\ell=1}^{L} \|\mathbf{W}_\ell\|_F^2 + 4wL_0$$

where $W_1 \in \mathbb{R}^{w\times h}; \mathbf{W}_\ell \in \mathbb{R}^{w\times w}, \ell = 2, \ldots, L-1; \mathbf{W}_L \in \mathbb{R}^{h\times w}$ are the weights in the feedforward neural network, and $\mathbf{W}'_1 \in \mathbb{R}^{K\times w\times h}, \mathbf{W}'_\ell \in \mathbb{R}^{K\times w\times w}, \ell = 2, \ldots, L'-1; \mathbf{W}_L \in \mathbb{R}^{K\times h\times w}$ are the weights in the convolution neural network.

Proof: We follow the same method as Oono et al. [108] to construct the CNN that is equivalent to the feedforward neural network. By combining Oono et al. [108] lemma 1 and lemma 2, for any linear transformation, one can construct a convolution neural network with at most $L_0 = \lceil \frac{h-1}{K-1} \rceil$ convolution layers and 4 channels, such that the first dimension in the output equals the linear transformation, and the norm of all the weights is no more than

$$\sum_{\ell=1}^{L_0} 4\|\mathbf{W}'_\ell\|_F^2 \leq \|\mathbf{W}\|_F^2 + 4L_0, \tag{6.9}$$

where $\mathbf{W}$ is the weight of the linear transformation. Putting w number of such convolution neural networks in parallel, a convolution neural network with L_0 layers and $4w$ channels can be constructed to implement the first layer in the feedforward neural network.

To implement the remaining layers, one choose the convolution kernel $\mathbf{W}'_{\ell+L_0-1)}[:, i, j] = [0, \ldots, \mathbf{W}[i, j], \ldots, 0], \forall 1 \leq i, j \leq w$, and pad the remaining parts with 0, such

that this convolution layer is equivalent to the linear layer applied on the dimension of channels. Noticing that this conversion does not change the norm of the parameters in each layer. Adding both sides of (6.9) by the norm of the $2-L$-th layer finishes the proof. ■

6.6.5 Covering number of a neural network block

Proposition 6.14 *If the input to a ReLU neural network is bounded by $\|x\|_2 \leq B_{in}$, the covering number of the ReLU neural network defined in Proposition 6.19 is bounded by*

$$\mathcal{N}(\mathcal{F}_{NN}, \delta, \|\cdot\|_2) \leq \left(\frac{(B/L)^{L/2} wL}{\delta}\right)^{w^2 L}$$

Proof: Similar to Proposition 6.19, we only consider the case $\|W_\ell\|_F \leq \sqrt{B/L}$. For any $1 \leq \ell \leq L$, for any $W_1, \ldots W_{\ell-1}, W_\ell, W'_\ell, W_{\ell+1}, \ldots W_L$ that satisfy the above constraint and $\|W_\ell - W'_\ell\|_F \leq \epsilon$, define $g(\ldots; W_1, \ldots W_L)$ as the neural network with parameters $W_1, \ldots W_L$, we can see

$$\begin{aligned}
&\|g(x; W_1, \ldots W_{\ell-1}, W_\ell, W_{\ell+1}, \ldots W_L) - g(x; W_1, \ldots W_{\ell-1}, W_\ell, W_{\ell+1}, \ldots W_L)\|_2 \\
&\leq (B/L)^{(L-\ell)/2} \|W_\ell - W'_\ell\|_2 \|ReLU(W_{\ell-1} \ldots ReLU(W_1(x)))\|_2 \\
&\leq (B/L)^{(L-1)/2} B_{in} \epsilon.
\end{aligned}$$

Choosing $\epsilon = \frac{\delta}{L(B/L)^{(L-1)/2}}$, the above inequality is no larger than δ/L. Taking the sum over ℓ, we can see that for any $W_1, W'_1, \ldots, W_L, W'_L$ such that $\|W_\ell - W'_\ell\|_F \leq \epsilon$,

$$\|g(x; W_1, \ldots W_L) - g(x; W'_1, \ldots W'_L))\|_2 \quad \leq \delta.$$

Finally, observe that the covering number of W_ℓ is bounded by

$$\mathcal{N}(\{W : \|W\|_F \leq B\}, \epsilon, \|\cdot\|_F) \leq \left(\frac{2Bw}{\epsilon}\right)^{w^2} \tag{6.10}$$

Substituting B and ϵ and taking the product over ℓ finishes the proof. ■

Proposition 6.15 *If the input to a ReLU neural network is bounded by $\|\boldsymbol{x}\|_2 \leq B_{\text{in}}$, the covering number of the ReLU neural network defined in Proposition 6.19 is bounded by*

$$\mathcal{N}(\mathcal{F}_{NN}, \delta, \|\cdot\|_2) \leq \left(\frac{B_{\text{in}}(B/L)^{L/2}wL}{\delta}\right)^{w^2L}.$$

Proof: Similar to Proposition 6.19, we only consider the case $\|W_\ell\|_F \leq \sqrt{B/L}$. For any $1 \leq \ell \leq L$, for any $W_1, \ldots W_{\ell-1}, W_\ell, W'_\ell, W_{\ell+1}, \ldots W_L$ that satisfy the above constraint and $\|W_\ell - W'_\ell\|_F \leq \epsilon$, define $g(\ldots; W_1, \ldots W_L)$ as the neural network with parameters $W_1, \ldots W_L$, we can see

$$\begin{aligned}
&\|g(\boldsymbol{x}; W_1, \ldots W_{\ell-1}, W_\ell, W_{\ell+1}, \ldots W_L) - g(\boldsymbol{x}; W_1, \ldots W_{\ell-1}, W_\ell, W_{\ell+1}, \ldots W_L)\|_2 \\
&\leq (B/L)^{(L-\ell)/2}\|W_\ell - W'_\ell\|_2\|ReLU(W_{\ell-1} \ldots ReLU(W_1(\boldsymbol{x})))\|_2 \\
&\leq (B/L)^{(L-1)/2}B_{\text{in}}\epsilon.
\end{aligned}$$

Choosing $\epsilon = \frac{\delta}{L(B/L)^{(L-1)/2}}$, the above inequality is no larger than δ/L. Taking the sum over ℓ, we can see that for any $W_1, W'_1, \ldots, W_L, W'_L$ such that $\|W_\ell - W'_\ell\|_F \leq \epsilon$,

$$\|g(\boldsymbol{x}; W_1, \ldots W_L) - g(\boldsymbol{x}; W'_1, \ldots W'_L))\|_2 \leq \delta.$$

Finally, observe that the covering number of W_ℓ is bounded by

$$\mathcal{N}(\{W : \|W\|_F \leq B\}, \epsilon, \|\cdot\|_F) \leq \left(\frac{2Bw}{\epsilon}\right)^{w^2}. \tag{6.11}$$

Substituting B and ϵ and taking the product over ℓ finishes the proof. ■

6.6.6 Proof of Lemma 6.9: covering number of ResNet and ResNeXt

From Proposition 6.19, Proposition 6.21 and Proposition 6.22, if any residual block is removed, the perturbation to the output is no more than $(KB_m/L)^{L/2}B_{out}^{1/2}\exp((KB_{res}/L)^{L/2})$ where B_m is the total norm of parameters in this block, $K = 1$ when the feedforward neural network is the backbone and K is the size of the convolution kernel when the convolution neural network is the backbone. If all the blocks with norm no more than ϵ is removed, the perturbation is no more than

$$\begin{aligned}
&\exp((KB_{res}/L)^{L/2})B_{out}^{1/2}\sum_{m:B_m<\epsilon}(KB_m/L)^{L/2}\\
&\leq \exp((KB_{res}/L)^{L/2})B_{out}^{1/2}\sum_{m:B_m<\epsilon}(KB_m/L)(K\epsilon/L)^{L/2-1}\\
&\leq \exp((KB_{res}/L)^{L/2})K^{L/2}B_{res}B_{out}^{1/2}(\epsilon/L)^{L/2-1}/L
\end{aligned}$$

Choosing $\epsilon = L\left(\frac{\delta L}{2\exp((B_{res}/L)^{L/2})K^{L/2}B_{res}B_{out}^{1/2}}\right)^{\frac{1}{L/2-1}}$, the perturbation above is no more than $\delta/2$. In this case, the remaining number of blocks is no more than B_{res}/ϵ. Combining this with Proposition 6.15 and **??** finishes the proof.

6.6.7 Proof of Theorem 6.3

For 1-Lipschitz loss, the proof is a direct application of Theorem 14.20 in Wainwright [155]. Define $\tilde{f} = \arg\min_f \mathbb{E}_{\mathcal{D}}[\text{loss}(f)]$. any function class $\partial\mathcal{F}$ that is star-shaped around

$\tilde{f}$, the empirical risk minimizer $\hat{f} = \arg\min_{f\in\mathcal{F}} \text{loss}_n(f)$ satisfy

$$\mathbb{E}_{\mathcal{D}}[\text{loss}(\hat{f})] \le \mathbb{E}_{\mathcal{D}}[\text{loss}(\tilde{f})] + 10\delta_n(2+\delta_n) \tag{6.12}$$

with probability at least $1 - c_1\exp(-c_2 n\delta_n^2)$ for any δ_n that satisfy (6.16), where c_1, c_2 are universal constants.

The function of neural networks is not star-shaped, but can be covered by a star-shaped function class. Specifically, let $\{f - \tilde{f} : f \in \mathcal{F}^{\text{Conv}}\} \subset \{f_1 - f_2 : f_1, f_2 \in \mathcal{F}^{\text{Conv}}\} := \partial\mathcal{F}$. Any function in $\partial\mathcal{F}$ can be represented using a ResNeXt: one can put two neural networks of the same structure in parallel, adjusting the sign of parameters in one of the neural networks and summing up the result, which increases M, B_{res} and B_{out} by a factor of 2. This only increases the log covering number in (5.10) by a factor of constant (remind that $B_{\text{res}} = O(1)$ by assumption).

Taking the log covering number of the ResNeXt (6.7), the sufficient condition for the above inequality is

$$\begin{aligned} &n^{-1/2} w L^{1/2} B_{res}^{\frac{1}{2-4/L}} K^{\frac{1-1/L}{1-2/L}} \left(B_{out}^{1/2}\exp((KB_{res}/L)^{L/2})\right)^{\frac{1/L}{1-2/L}} \delta_n^{\frac{1-3/L}{1-2/L}} \lesssim \frac{\delta_n^2}{4\sigma}, \\ &\delta_n \gtrsim K(w^2L)^{\frac{1-2/L}{2-2/L}} B_{res}^{\frac{1}{2-2/L}} \left(B_{out}^{1/2}\exp((KB_{res}/L)^{L/2})\right)^{\frac{1/L}{1-1/L}} n^{-\frac{1-2/L}{2-2/L}} \sigma^{\frac{1-2/L}{1-1/L}}, \end{aligned} \tag{6.13}$$

where $\lesssim$ hides the logarithmic term. Finally, from Theorem 6.1, the minimum width of each subnetwork is $w = O(D + dm)$. Because loss is 1-Lipschitz, we have

$$\text{loss}(f) \le \text{loss}(\tilde{f}) + \|f - \tilde{f}\|_\infty.$$

Choosing

$$P = O\Big(\frac{K^{-\frac{2}{L-2}}(D+dm)^{\frac{3L-4}{L-2}} L^{\frac{3L-2}{L-2}}}{n}\Big)^{-\frac{1-2/L}{2\alpha/d(1-1/L)+1-2/pL}}$$

and taking it into Theorem 6.1 and Theorem 6.2 finishes the proof.

For MSE loss, the proof depends on Theorem 6.17 and Theorem 6.18, which infers that $\|\hat{f} - f_0\|_n^2 \lesssim \|f^* - f_0\|_n^2 + \delta_n^2 + \sigma^2/n$ both in expectation and with high probability. Taking δ_n in (6.13) into it finishes the proof.

6.6.8 Lower bound of error

Without the loss of generalization, assume that $\frac{\partial L(y)}{\partial y} \geq 0.5$ for $-\epsilon \leq y \leq \epsilon$. Define the function space

$$\mathcal{F} = \left\{ f = \sum_{j_1,\ldots,j_d=1}^{s} \pm \frac{\epsilon}{s^\alpha} \times M^{(m)}((\boldsymbol{x} - \boldsymbol{j})/s) \right\}, \tag{6.14}$$

where $M^{(m)}$ denotes the Cardinal B-spline basis function that is supported on $(0,1)^d, \boldsymbol{j} = [j_1, \ldots, j_d]$. The support of each B-spline basis function splits the space into s^d number of blocks, where the target function in each block has two choices (positive or negative), so the total number of different functions in this function class is $|\mathcal{F}| = 2^{s^d}$. Using Dũng [136, Theorm 2.2], we can see that for any $f \in \mathcal{F}$,

$$\|f\|_{B^\alpha_{p,q}} \leq \frac{\epsilon}{s^\alpha} s^{\alpha - d/p} s^{d/p} = \epsilon.$$

For a fixed $f_0 \in \mathcal{F}$, let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a set of noisy observations with $y_i = f(x_i) + \epsilon_i, \epsilon_i \sim SubGaussian(0, \sigma^2 I)$. Further assume that $\boldsymbol{x}_i$ are evenly distributed in $(0,1)^d$ such that in all regions as defined in (6.14), the number of samples is $n_{\boldsymbol{j}} := O(n/s^d)$. Using Le Cam's inequality, we get that in any region, any estimator θ satisfy

$$\sup_{f_0 \in \mathcal{F}} \mathbb{E}_{\mathcal{D}}[\|\theta(\mathcal{D}) - f_0\|_{\boldsymbol{j}}] \geq \frac{C_m \epsilon}{16 s^\alpha}$$

as long as $(\frac{\epsilon}{\sigma s^\alpha})^2 \lesssim \frac{s^d}{n}$, where $\|\cdot\|_{\boldsymbol{j}} := \frac{1}{n_{\boldsymbol{i}}} \sum_{s(\boldsymbol{x}-\boldsymbol{j})\in[0,1]^d} |f(\boldsymbol{x})|$ denotes the norm defined in the block indexed by $\boldsymbol{i}$, C_m is a constant that depends only on m. Choosing $s = O(n^{\frac{1}{2\alpha+d}})$, we get

$$\sup_{f_0\in\mathcal{F}} \mathbb{E}_{\mathcal{D}}[\|\theta(\mathcal{D}) - f_0\|_{\boldsymbol{j}}] \geq n^{-\frac{\alpha}{2\alpha+d}}$$

Observing $\frac{1}{n}\sum_{i=1}^n L(\hat{(}f(x_i))) \geq 0.5\sum_{i=1}^n |f(x_i) - f_0(x_i)| \eqsim \frac{1}{s^d}\sum_{\boldsymbol{j}\in[s]^d} \|\hat{f} - f_0\|_{\boldsymbol{j}}$ finishes the proof.

6.6.9 Supporting theorem

Corollary 6.16 (Corollary 13.7 and Corollary 14.3 in Wainwright [155]) *Let*

$$\mathcal{G}_n(\delta,\mathcal{F}) = \mathbb{E}_{w_i}\left[\sup_{g\in\mathcal{F},\|g\|_n\leq\delta}\left|\frac{1}{n}\sum_{i=1}^n w_i g(\boldsymbol{x}_i)\right|\right], \mathcal{R}_n(\delta,\mathcal{F}) = \mathbb{E}_{\epsilon_i}\left[\sup_{g\in\mathcal{F},\|g\|_n\leq\delta}\left|\frac{1}{n}\sum_{i=1}^n \epsilon_i g(\boldsymbol{x}_i)\right|\right],$$

denotes the local Gaussian complexity and local Rademacher complexity respectively, where $w_i \sim \mathcal{N}(0,1)$ are the i.i.d. Gaussian random variables, and $\epsilon_i \sim$ uniform$\{-1,1\}$ are the Rademacher random variables. Suppose that the function class $\mathcal{F}$ is star-shaped, for any $\sigma > 0$, any $\delta \in (0,\sigma]$ such that

$$\frac{16}{\sqrt{n}}\int_{\delta_n^2/4\sigma}^{\delta_n} \sqrt{\log \mathcal{N}(\mathcal{F},\mu,\|\cdot\|_\infty)}d\mu \leq \frac{\delta_n^2}{4\sigma}.$$

satisfies

$$\mathcal{G}_n(\delta,\mathcal{F}) \leq \frac{\delta^2}{2\sigma}. \tag{6.15}$$

Furthermore, if $\mathcal{F}$ is uniformly bounded by b, i.e. $\forall f\in\mathcal{F}, \boldsymbol{x}|f(\boldsymbol{x})| \leq b$ any $\delta > 0$ such that

$$\frac{64}{\sqrt{n}}\int_{\delta_n^2/2b4\sigma}^{\delta_n} \sqrt{\log \mathcal{N}(\mathcal{F},\mu,\|\cdot\|_\infty)}d\mu \leq \frac{\delta_n^2}{b}.$$

satisfies

$$\mathcal{R}_n(\delta, \mathcal{F}) \leq \frac{\delta^2}{b}. \tag{6.16}$$

Theorem 6.17 (Theorem 14.1 in Wainwright [155]) *Given a star-shaped and b-uniformly bounded function class $\mathcal{F}$, let δ_n be any positive solution of the inequality*

$$\bar{\mathcal{R}}_n(\delta, \mathcal{F}) \leq \frac{\delta^2}{b},$$

where

$$\bar{\mathcal{R}}_n(\delta, \mathcal{F}) = \mathbb{E}_x \left[\sup_{f in \mathcal{F}, \|f\|_2 \leq \delta} \sup \left| \frac{1}{n} \sum_{i=1}^{n} \epsilon_i f(x_i) \right| \right]$$

denotes the localized population Rademacher complexity, then for any $t \geq \delta_n$, we have

$$\|f\|_2^2 \leq 2\|f\|_n^2 + t^2 \quad \text{for all } f \in \mathcal{F}$$

with probability at least $1 - c_1 \exp(-c_2 n t^2 / b^2)$.

Theorem 6.18 (Theorem 13.13 in Wainwright [155]) *Let δ_n be any positive solution satisfying the inequality (6.15) for the function class $\partial\mathcal{F} = \{f_1 - f_2 | f_1, f_2 \in \mathcal{F}\}$. There are universal positive constants (c_0, c_1, c_2) such that for any $t \geq \delta_n$, with probability greater than $1 - c_1 \exp(-c_2 n t \delta_n / \sigma^2)$, the nonpara-metric least-squares estimate $\hat{f}$ satisfies the bound*

$$\|\hat{f} - f_0\|_n^2 \leq \frac{1+\gamma}{1-\gamma} \|f^* - f_0\|_n^2 + \frac{c_0}{\gamma(1-\gamma) t \delta_n}$$

for any $\gamma \in (0, 1)$.

Proposition 6.19 *An L-layer ReLU neural network with no bias and bounded norm*

$$\sum_{\ell=1}^{L} \|\mathbf{W}_\ell\|_F^2 \leq B.$$

is Lipschitz continuous with Lipschitz constant $(B/L)^{L/2}$

Proof: Notice that ReLU function is 1-homogeneous, similar to Proposition 4 in [6], for any neural network there exists an equivalent model satisfying $\|\mathbf{W}_\ell\|_F = \|\mathbf{W}_{\ell'}\|_F$ for any ℓ, ℓ', and its total norm of parameters is no larger than the original model. Because of that, it suffices to consider the neural network satisfying $\|\mathbf{W}_\ell\|_F \leq \sqrt{B/L}$ for all ℓ. The Lipschitz constant of such linear layer is $\|\mathbf{W}_\ell\|\|_2 \leq \|\mathbf{W}_\ell\|\|_F \leq \sqrt{B/L}$, and the Lipschitz constant of ReLU layer is 1. Taking the product over all layers finishes the proof. ■

Proposition 6.20 *An L-layer ReLU convolution neural network with convolution kernel size K, no bias and bounded norm*

$$\sum_{\ell=1}^{L} \|\mathbf{W}_\ell\|_F^2 \leq B.$$

is Lipschitz continuous with Lipschitz constant $(KB/L)^{L/2}$

This proposition can be proved by taking Proposition 6.23 into the proof of Proposition 6.19.

Proposition 6.21 *Let* $f = f_{post} \circ (1 + f_{NN} + f_{other}) \circ f_{pre}$ *be a ResNeXt, where* $1 + f_{NN} + f_{other}$ *denotes a residual block,* f_{pre} *and* f_{post} *denotes the part of the neural network before and after this residual block, respectively.* f_{NN} *denotes one of the feedforward block in this residual block and* f_{other} *denotes the other residual blocks. Assume* $f_{pre}, f_{NN}, f_{post}$ *are Lipschitz continuous with Lipschitz constant* $L_{pre}, L_{NN}, L_{post}$ *respectively. Let the input*

be x, if the residual block is removed, the perturbation to the output is no more than $L_{pre}L_{NN}L_{post}\|x\|$

Proof:

$$
\begin{aligned}
&|f_{post} \circ (1 + f_{NN} + f_{other}) \circ f_{pre}(x) - f_{post} \circ (1 + f_{other}) \circ f_{pre}(x)| \\
&\leq L_{post}|(1 + f_{NN} + f_{other}) \circ f_{pre}(x) - (1 + f_{other}) \circ f_{pre}(x)| \\
&= L_{post}|f_{NN} \circ f_{pre}(x)| \\
&\leq L_{pre}L_{NN}L_{post}\|x\|
\end{aligned}
$$

■

Proposition 6.22 *The neural network defined in Lemma 6.9 with arbitrary number of blocks has Lipschitz constant $\exp((KB_{res}/L)^{L/2})$, where $K = 1$ when the feedforward neural network is the backbone and K is the size of the convolution kernel when the convolution neural network is the backbone.*

Proof: Note that the m-th block in the neural network defined in Lemma 6.9 can be represented as $y = f_m(x; \omega_m) + x$, where f_m is an L-layer feedforward neural network with no bias. By Proposition 6.19 and Proposition 6.20, such block is Lipschitz continuous with Lipschitz constant $1 + (KB_m/L)^{L/2}$, where the weight parameters of the m-th block satisfy that $\sum_{\ell=1}^{L} \|W_\ell^{(m)}\|_F^2 \leq B_m$ and $\sum_{m=1}^{M} B_m \leq B_{res}$.

Since the neural network defined in Lemma 6.9 is a composition of M blocks, it is Lipschitz with Lipschitz constant L_{res}. We have

$$
L_{res} \leq \prod_{m=1}^{M} \left(1 + \left(\frac{KB_m}{L}\right)^{L/2}\right) \leq \exp\left(\sum_{m=1}^{M} \left(\frac{KB_m}{L}\right)^{L/2}\right),
$$

where we use the inequality $1 + z \leq \exp(x)$ for any $x \in \mathrm{R}$. Furthermore, notice that $\sum_{m=1}^{M}(KB_m/L)^{L/2}$ is convex with respect to $(B_1, B_2, \ldots, B_M)$ when $L > 2$. Since

$\sum_{m=1}^{M} B_m \leq B_{res}$ and $B_m \geq 0$, then we have $\sum_{m=1}^{M}(KB_m/L)^{L/2} \leq (KB_{res}/L)^{L/2}$ by convexity. Therefore, we obtain that $L_{res} \leq \exp((KB_{res}/L)^{L/2})$. ■

Proposition 6.23 *For any* $\boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{w} \in \mathbb{R}^K, K \leq d$, $\|\text{Conv}(\boldsymbol{x}, \boldsymbol{w})\|_2 \leq \sqrt{K}\|\boldsymbol{x}\|_2\|\boldsymbol{w}\|_2$.

Proof: For simplicity, denote $x_i = 0$ for $i \leq 0$ or $i > d$.

$$\begin{aligned}\|\text{Conv}(\boldsymbol{x}, \boldsymbol{w})\|_2^2 &= \textstyle\sum_{i=1}^{d}\langle \boldsymbol{x}[i - \frac{K-1}{2} : i + \frac{K-1}{2}], \boldsymbol{w}\rangle^2 \\ &\leq \textstyle\sum_{i=1}^{d} \|\boldsymbol{x}[i - \frac{K-1}{2} : i + \frac{K-1}{2}]\|_2^2\|\boldsymbol{w}\|_2^2 \\ &\leq K\|\boldsymbol{x}\|_2^2\|\boldsymbol{w}\|_2^2\end{aligned}$$

where the second line comes from Cauchy-Schwarz inequality, the third line comes by expanding $\|\boldsymbol{x}[i - \frac{K-1}{2} : i + \frac{K-1}{2}]\|_2^2$ by definition and observing that each element in $\boldsymbol{x}$ appears at most K times. ■

Bibliography

[1] K. Zhang, X. Zhang, and Z. Zhang, *Tucker tensor decomposition on fpga*, in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2019.

[2] C. Cui, K. Zhang, T. Daulbaev, J. Gusak, I. Oseledets, and Z. Zhang, *Active subspace of neural networks: Structural analysis and universal attacks*, *SIAM Journal on Mathematics of Data Science* **2** (2020), no. 4 1096–1122.

[3] K. Zhang, C. Hawkins, X. Zhang, C. Hao, and Z. Zhang, *On-fpga training with ultra memory reduction: A low-precision tensor method*, in *ICLR Workshop on Hardware Aware Efficient Training*, 2021.

[4] K. Zhang, C. Hawkins, and Z. Zhang, *General-purpose bayesian tensor learning with automatic rank determination and uncertainty quantification*, *Frontiers in Artificial Intelligence* **4** (2022) 668353.

[5] K. Zhang, M. Yin, and Y.-X. Wang, *Why quantization improves generalization: Ntk of binary weight neural networks*, *arXiv preprint arXiv:2206.05916* (2022).

[6] K. Zhang and Y.-X. Wang, *Deep learning meets nonparametric regression: Are weight-decayed dnns locally adaptive?*, *arXiv preprint arXiv:2204.09664* (2022).

[7] Y. LeCun, J. S. Denker, and S. A. Solla, *Optimal brain damage*, in *NIPS*, pp. 598–605, 1990.

[8] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, *Structured Bayesian pruning via log-normal multiplicative noise*, in *NIPS*, pp. 6775–6784, 2017.

[9] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, *arXiv:1510.00149* (2015).

[10] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, *Incremental network quantization: Towards lossless cnns with low-precision weights*, *arXiv:1702.03044* (2017).

[11] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, *arXiv:1503.02531* (2015).

[12] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*, in *ICASSP*, pp. 6655–6659, 2013.

[13] J. Xue, J. Li, and Y. Gong, *Restructuring of deep neural network acoustic models with singular value decomposition.*, in *Interspeech*, pp. 2365–2369, 2013.

[14] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, *Compression of deep convolutional neural networks for fast and low power mobile applications*, *arXiv:1511.06530* (2015).

[15] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, *Speeding-up convolutional neural networks using fine-tuned cp-decomposition*, *arXiv preprint arXiv:1412.6553* (2014).

[16] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, *Tensorizing neural networks*, in *Advances in neural information processing systems*, pp. 442–450, 2015.

[17] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, *Ultimate tensorization: compressing convolutional and fc layers alike*, *arXiv preprint arXiv:1611.03214* (2016).

[18] C. Cui, C. Hawkins, and Z. Zhang, *Tensor methods for generating compact uncertainty quantification and deep learning models*, *arXiv preprint arXiv:1908.07699* (2019).

[19] R. M. Neal, *Bayesian training of backpropagation networks by the hybrid monte carlo method*, tech. rep., Citeseer, 1992.

[20] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, *Hybrid monte carlo*, *Physics letters B* **195** (1987), no. 2 216–222.

[21] T. Chen, E. Fox, and C. Guestrin, *Stochastic gradient hamiltonian monte carlo*, in *International conference on machine learning*, pp. 1683–1691, 2014.

[22] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, *Journal of Mathematics and Physics* **6** (1927), no. 1-4 164–189.

[23] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, *Psychometrika* **31** (1966), no. 3 279–311.

[24] I. V. Oseledets, *Tensor-train decomposition*, *SIAM Journal on Scientific Computing* **33** (2011), no. 5 2295–2317.

[25] Q. Zhao, L. Zhang, and A. Cichocki, *Bayesian cp factorization of incomplete tensors with automatic rank determination*, *IEEE transactions on pattern analysis and machine intelligence* **37** (2015), no. 9 1751–1763.

[26] C. Hawkins and Z. Zhang, *Robust factorization and completion of streaming tensor data via variational bayesian inference*, *arXiv preprint arXiv:1809.01265* (2018).

[27] N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven, *Bayesian sampling using stochastic gradient thermostats*, in *Advances in neural information processing systems*, pp. 3203–3211, 2014.

[28] Q. Zhao, L. Zhang, and A. Cichocki, *Bayesian sparse tucker models for dimension reduction and tensor completion*, *arXiv preprint arXiv:1505.02343* (2015).

[29] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, *arXiv preprint arXiv:1708.07747* (2017).

[30] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, *Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks*, *IEEE journal of solid-state circuits* **52** (2016), no. 1 127–138.

[31] Y. Chen, K. Zhang, C. Gong, C. Hao, X. Zhang, T. Li, and D. Chen, *T-dla: An open-source deep learning accelerator for ternarized dnn models on embedded fpga*, in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 13–18, IEEE, 2019.

[32] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, *Optimizing fpga-based accelerator design for deep convolutional neural networks*, in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pp. 161–170, 2015.

[33] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, *Fpga/dnn co-design: An efficient design methodology for 1ot intelligence on the edge*, in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[34] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, *Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics*, in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 40–50, 2020.

[35] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in NLP*, in *Proc. Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019.

[36] S. Teerapittayanon, B. McDanel, and H.-T. Kung, *Distributed deep neural networks over the cloud, the edge and end devices*, in *Intl. Conf. Distributed Computing Systems*, pp. 328–339, 2017.

[37] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Quantized neural networks: Training neural networks with low precision weights and activations*, *The Journal of Machine Learning Research* **18** (2017), no. 1 6869–6898.

[38] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, *et. al.*, *Flexpoint: An adaptive numerical format for efficient training of deep neural networks*, in *NIPS*, pp. 1742–1752, 2017.

[39] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, *Deep learning with limited numerical precision*, in *International Conference on Machine Learning*, pp. 1737–1746, 2015.

[40] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, *Ultra-low precision 4-bit training of deep neural networks*, *NIPS* **33** (2020).

[41] M. Courbariaux, Y. Bengio, and J.-P. David, *Binaryconnect: Training deep neural networks with binary weights during propagations*, in *Advances in neural information processing systems*, pp. 3123–3131, 2015.

[42] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, *Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1*, *arXiv preprint arXiv:1602.02830* (2016).

[43] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, *SIAM review* **51** (2009), no. 3 455–500.

[44] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, *TIE: energy-efficient tensor train-based inference engine for deep neural network*, in *ISCA*, pp. 264–278, 2019.

[45] P. Zhen, B. Liu, Y. Cheng, H.-B. Chen, and H. Yu, *Fast video facial expression recognition by deeply tensor-compressed lstm neural network on mobile device*, in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pp. 298–300, 2019.

[46] H. Huang, L. Ni, K. Wang, Y. Wang, and H. Yu, *A highly parallel and energy efficient three-dimensional multilayer cmos-rram accelerator for tensorized neural network*, *IEEE Trans. on Nanotechnology* **17** (2017), no. 4 645–656.

[47] G. G. Calvi, A. Moniri, M. Mahfouz, Q. Zhao, and D. P. Mandic, *Compression and interpretability of deep neural networks via tucker tensor layer*, *arXiv:1903.06133* (2019).

[48] V. Khrulkov, O. Hrinchuk, L. Mirvakhabova, and I. Oseledets, *Tensorized embedding layers for efficient model compression*, *arXiv:1901.10787* (2019).

[49] M. Zhou, Y. Liu, Z. Long, L. Chen, and C. Zhu, *Tensor rank learning in cp decomposition via convolutional neural network*, *Signal Processing: Image Communication* **73** (2019) 12–21.

[50] G. G. Calvi, A. Moniri, M. Mahfouz, Z. Yu, Q. Zhao, and D. P. Mandic, *Tucker tensor layer in fully connected neural networks*, *arXiv preprint arXiv:1903.06133* (2019).

[51] M. Yin, S. Liao, X.-Y. Liu, X. Wang, and B. Yuan, *Compressing recurrent neural networks using hierarchical tucker tensor decomposition*, *arXiv preprint arXiv:2005.04366* (2020).

[52] A. Tjandra, S. Sakti, and S. Nakamura, *Compressing recurrent neural network with tensor train*, in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 4451–4458, IEEE, 2017.

[53] B. Recht, M. Fazel, and P. A. Parrilo, *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, *SIAM review* **52** (2010), no. 3 471–501.

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

[55] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein, *Training quantized nets: A deeper understanding*, in *Advances in Neural Information Processing Systems*, pp. 5811–5821, 2017.

[56] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, *arXiv preprint arXiv:1308.3432* (2013).

[57] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, *Understanding straight-through estimator in training activation quantized neural nets*, *arXiv preprint arXiv:1903.05662* (2019).

[58] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, *arXiv preprint arXiv:1708.07747* (2017).

[59] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, *Learnability and the vapnik-chervonenkis dimension*, *Journal of the ACM (JACM)* **36** (1989), no. 4 929–965.

[60] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, *Understanding deep learning requires rethinking generalization*, *arXiv preprint arXiv:1611.03530* (2016).

[61] F. He and D. Tao, *Recent advances in deep learning theory*, *arXiv preprint arXiv:2012.10931* (2020).

[62] E. Weinan, J. Han, and Q. Li, *A mean-field optimal control formulation of deep learning*, *Research in the Mathematical Sciences* **6** (2019), no. 1 1–41.

[63] F. He, B. Wang, and D. Tao, *Piecewise linear activations substantially shape the loss surfaces of neural networks*, *arXiv preprint arXiv:2003.12236* (2020).

[64] D. Li, T. Ding, and R. Sun, *Over-parameterized deep neural networks have no strict local minima for any continuous activations*, *arXiv preprint arXiv:1812.11039* (2018).

[65] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, *The loss surfaces of multilayer networks*, in *Artificial intelligence and statistics*, pp. 192–204, PMLR, 2015.

[66] Z. Allen-Zhu, Y. Li, and Y. Liang, *Learning and generalization in overparameterized neural networks, going beyond two layers*, *arXiv preprint arXiv:1811.04918* (2018).

[67] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang, *Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks*, *arXiv preprint arXiv:1901.08584* (2019).

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

[69] F. Bach, *Breaking the curse of dimensionality with convex neural networks*, *The Journal of Machine Learning Research* **18** (2017), no. 1 629–681.

[70] A. Bietti and J. Mairal, *On the inductive bias of neural tangent kernels*, in *Advances in Neural Information Processing Systems*, pp. 12893–12904, 2019.

[71] A. Geifman, A. Yadav, Y. Kasten, M. Galun, D. Jacobs, and R. Basri, *On the similarity between the laplace and neural tangent kernels*, *arXiv preprint arXiv:2007.01580* (2020).

[72] L. Chen and S. Xu, *Deep neural tangent kernel and laplace kernel have the same rkhs*, *arXiv preprint arXiv:2009.10683* (2020).

[73] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Binarized neural networks*, in *Proceedings of the 30th international conference on neural information processing systems*, pp. 4114–4122, Citeseer, 2016.

[74] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, *Fast neural networks without multipliers*, *IEEE transactions on Neural Networks* **4** (1993), no. 1 53–62.

[75] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, *Pruning and quantization for deep neural network acceleration: A survey*, *Neurocomputing* **461** (2021) 370–403.

[76] T. Chu, Q. Luo, J. Yang, and X. Huang, *Mixed-precision quantized neural networks with progressively decreasing bitwidth*, *Pattern Recognition* **111** (2021) 107647.

[77] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks*, in *European conference on computer vision*, pp. 525–542, Springer, 2016.

[78] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, *Ternary neural networks for resource-efficient ai applications*, in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2547–2554, IEEE, 2017.

[79] Y. Ding, J. Liu, J. Xiong, and Y. Shi, *On the universal approximability and complexity bounds of quantized relu neural networks*, *arXiv preprint arXiv:1802.03646* (2018).

[80] B. Bordelon, A. Canatar, and C. Pehlevan, *Spectrum dependent learning curves in kernel regression and wide neural networks*, *arXiv preprint arXiv:2002.02561* (2020).

[81] J. B. Simon, M. Dickens, and M. R. DeWeese, *Neural tangent kernel eigenvalues accurately predict generalization*, *arXiv preprint arXiv:2110.03922* (2021).

[82] D. Soudry and Y. Carmon, *No bad local minima: Data independent training error guarantees for multilayer neural networks*, *arXiv preprint arXiv:1605.08361* (2016).

[83] S. S. Du, X. Zhai, B. Poczos, and A. Singh, *Gradient descent provably optimizes over-parameterized neural networks*, *arXiv preprint arXiv:1810.02054* (2018).

[84] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, *Gradient descent finds global minima of deep neural networks*, in *International Conference on Machine Learning*, pp. 1675–1685, PMLR, 2019.

[85] Y. Li and Y. Liang, *Learning overparameterized neural networks via stochastic gradient descent on structured data*, *arXiv preprint arXiv:1808.01204* (2018).

[86] L. Chizat, E. Oyallon, and F. Bach, *On lazy training in differentiable programming*, *arXiv preprint arXiv:1812.07956* (2018).

[87] Y. Dong, R. Ni, J. Li, Y. Chen, J. Zhu, and H. Su, *Learning accurate low-bit deep neural networks with stochastic quantization*, *arXiv preprint arXiv:1708.01001* (2017).

[88] H. Q. Minh, P. Niyogi, and Y. Yao, *Mercer's theorem, feature maps, and smoothing*, in *International Conference on Computational Learning Theory*, pp. 154–168, Springer, 2006.

[89] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et. al.*, *Gradient-based learning applied to document recognition*, *Proceedings of the IEEE* **86** (1998), no. 11 2278–2324.

[90] S. Arora, S. S. Du, Z. Li, R. Salakhutdinov, R. Wang, and D. Yu, *Harnessing the power of infinitely wide deep nets on small-data tasks*, *arXiv preprint arXiv:1910.01663* (2019).

[91] E. A. Nadaraya, *On estimating regression*, *Theory of Probability & Its Applications* **9** (1964), no. 1 141–142.

[92] C. De Boor, C. De Boor, E.-U. Mathématicien, C. De Boor, and C. De Boor, *A practical guide to splines*, vol. 27. Springer-Verlag New York, 1978.

[93] G. Wahba, *Spline models for observational data*, vol. 59. Siam, 1990.

[94] D. L. Donoho, I. M. Johnstone, *et. al.*, *Minimax estimation via wavelet shrinkage*, *The annals of Statistics* **26** (1998), no. 3 879–921.

[95] S. Mallat, *A wavelet tour of signal processing*. Elsevier, 1999.

[96] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[97] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT Press, 2006.

[98] E. Mammen and S. van de Geer, *Locally adaptive regression splines*, *The Annals of Statistics* **25** (1997), no. 1 387–413.

[99] R. J. Tibshirani, *Adaptive piecewise polynomial estimation via trend filtering*, *The Annals of Statistics* **42** (2014), no. 1 285–323.

[100] Y.-X. Wang, A. Smola, and R. Tibshirani, *The falling factorial basis and its statistical applications*, in *International Conference on Machine Learning*, pp. 730–738, PMLR, 2014.

[101] D. Baby and Y.-X. Wang, *Online forecasting of total-variation-bounded sequences*, in *Neural Information Processing Systems (NeurIPS)*, 2019.

[102] D. Baby and Y.-X. Wang, *Adaptive online estimation of piecewise polynomial trends*, *Neural Information Processing Systems (NeurIPS)* (2020).

[103] M. Belkin, S. Ma, and S. Mandal, *To understand deep learning we need to understand kernel learning*, in *International Conference on Machine Learning*, pp. 541–549, PMLR, 2018.

[104] 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*, pp. 8141–8150, 2019.

[105] D. L. Donoho, R. C. Liu, and B. MacGibbon, *Minimax risk over hyperrectangles, and implications*, *The Annals of Statistics* (1990) 1416–1437.

[106] T. Suzuki, *Adaptivity of deep relu network for learning in besov and mixed smooth besov spaces: optimal rate and curse of dimensionality*, *arXiv preprint arXiv:1810.08033* (2018).

[107] R. Parhi and R. D. Nowak, *Banach space representer theorems for neural networks and ridge splines.*, *J. Mach. Learn. Res.* **22** (2021) 43–1.

[108] K. Oono and T. Suzuki, *Approximation and non-parametric estimation of resnet-type convolutional neural networks*, in *International conference on machine learning*, pp. 4922–4931, PMLR, 2019.

[109] H. Liu, M. Chen, T. Zhao, and W. Liao, *Besov function approximation and binary classification on low-dimensional manifolds using convolutional residual networks*, in *International Conference on Machine Learning*, pp. 6770–6780, PMLR, 2021.

[110] B. D. Haeffele and R. Vidal, *Global optimality in neural network training*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7331–7339, 2017.

[111] T. Ergen and M. Pilanci, *Path regularization: A convexity and sparsity inducing regularization for parallel relu networks*, *arXiv preprint arXiv:2110.09548* (2021).

[112] H. Zhang, J. Shao, and R. Salakhutdinov, *Deep neural networks with multi-branch architectures are intrinsically less non-convex*, in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1099–1109, PMLR, 2019.

[113] T. Ergen and M. Pilanci, *Global optimality beyond two layers: Training deep relu networks via convex programs*, in *International Conference on Machine Learning*, pp. 2993–3003, PMLR, 2021.

[114] T. Ergen and M. Pilanci, *Revealing the structure of deep neural networks via convex duality*, in *International Conference on Machine Learning*, pp. 3004–3014, PMLR, 2021.

[115] S. Zagoruyko and N. Komodakis, *Wide residual networks*, *arXiv preprint arXiv:1605.07146* (2016).

[116] A. Veit, M. J. Wilber, and S. Belongie, *Residual networks behave like ensembles of relatively shallow networks*, *Advances in neural information processing systems* **29** (2016) 550–558.

[117] Y. Yao, L. Rosasco, and A. Caponnetto, *On early stopping in gradient descent learning*, *Constructive Approximation* **26** (2007), no. 2 289–315.

[118] S. Wager, S. Wang, and P. Liang, *Dropout training as adaptive regularization*, *arXiv preprint arXiv:1307.1493* (2013).

[119] R. Parhi and R. D. Nowak, *Near-minimax optimal estimation with shallow relu neural networks*, *arXiv preprint arXiv:2109.08844* (2021).

[120] J. Schmidt-Hieber, *Nonparametric regression using deep neural networks with relu activation function*, *The Annals of Statistics* **48** (2020), no. 4 1875–1897.

[121] A. Lewkowycz, Y. Bahri, E. Dyer, J. Sohl-Dickstein, and G. Gur-Ari, *The large learning rate phase of deep learning: the catapult mechanism*, *arXiv preprint arXiv:2003.02218* (2020).

[122] Z. Allen-Zhu and Y. Li, *What can resnet learn efficiently, going beyond kernels?*, *Advances in Neural Information Processing Systems* **32** (2019).

[123] B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari, *When do neural networks outperform kernel methods?*, *Advances in Neural Information Processing Systems* **33** (2020) 14820–14830.

[124] R. Parhi and R. D. Nowak, *What kinds of functions do deep neural networks learn? insights from variational spline theory*, *arXiv preprint arXiv:2105.03361* (2021).

[125] B. Neyshabur, R. Tomioka, and N. Srebro, *In search of the real inductive bias: On the role of implicit regularization in deep learning*, *arXiv preprint arXiv:1412.6614* (2014).

[126] P. Savarese, I. Evron, D. Soudry, and N. Srebro, *How do infinite width bounded norm networks look in function space?*, in *Conference on Learning Theory*, pp. 2667–2690, PMLR, 2019.

[127] G. Ongie, R. Willett, D. Soudry, and N. Srebro, *A function space view of bounded norm infinite width relu nets: The multivariate case*, in *International Conference on Learning Representations*, 2019.

[128] T. Ergen and M. Pilanci, *Convex geometry and duality of over-parameterized neural networks*, *Journal of machine learning research* (2021).

[129] M. Pilanci and T. Ergen, *Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks*, in *International Conference on Machine Learning*, pp. 7695–7705, PMLR, 2020.

[130] N. Srebro, J. D. Rennie, and T. S. Jaakkola, *Maximum-margin matrix factorization.*, in *NIPS*, vol. 17, pp. 1329–1336, Citeseer, 2004.

[131] R. J. Tibshirani, *Equivalences between sparse models and neural networks*, .

[132] T. Ergen and M. Pilanci, *Implicit convex regularizers of cnn architectures: Convex optimization of two-and three-layer networks in polynomial time*, *arXiv preprint arXiv:2006.14798* (2020).

[133] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, *Mathematics of control, signals and systems* **2** (1989), no. 4 303–314.

[134] A. R. Barron, *Approximation and estimation bounds for artificial neural networks*, *Machine learning* **14** (1994), no. 1 115–133.

[135] D. Yarotsky, *Error bounds for approximations with deep relu networks*, *Neural Networks* **94** (2017) 103–114.

[136] D. Dũng, *Optimal adaptive sampling recovery*, *Advances in Computational Mathematics* **34** (2011), no. 1 1–41.

[137] V. Sadhanala, Y.-X. Wang, A. J. Hu, and R. J. Tibshirani, *Multivariate trend filtering for lattice data*, *arXiv preprint arXiv:2112.14758* (2021).

[138] R. A. DeVore and G. G. Lorentz, *Constructive approximation*, vol. 303. Springer Science & Business Media, 1993.

[139] S. Burer and R. D. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, *Mathematical Programming* **95** (2003), no. 2 329–357.

[140] R. J. Tibshirani, *Degrees of freedom and model search*, *Statistica Sinica* (2015) 1265–1296.

[141] R. J. Tibshirani. Personal communication, Jan. 24,, 2022.

[142] D. Hsu, S. M. Kakade, and T. Zhang, *An analysis of random design linear regression*, *arXiv preprint arXiv:1106.2363* (2011).

[143] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, *Deep double descent: Where bigger models and more data hurt*, *Journal of Statistical Mechanics: Theory and Experiment* **2021** (2021), no. 12 124003.

[144] C. K. Chui and H. N. Mhaskar, *Deep nets for local manifold learning*, *Frontiers in Applied Mathematics and Statistics* **4** (2018) 12.

[145] U. Shaham, A. Cloninger, and R. R. Coifman, *Provable approximation properties for deep neural networks*, *Applied and Computational Harmonic Analysis* **44** (2018), no. 3 537–557.

[146] J. Schmidt-Hieber, *Deep relu network approximation of functions on a manifold*, *arXiv preprint arXiv:1908.00695* (2019).

[147] M. Chen, H. Jiang, W. Liao, and T. Zhao, *Nonparametric regression on low-dimensional manifolds using deep relu networks: Function approximation and statistical recovery*, *Information and Inference: A Journal of the IMA* **11** (2022), no. 4 1203–1253.

[148] 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*, pp. 770–778, 2016.

[149] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[150] L. W. Tu, *Manifolds*, in *An Introduction to Manifolds*, pp. 47–83. Springer, 2011.

[151] H. Federer, *Curvature measures*, *Transactions of the American Mathematical Society* **93** (1959), no. 3 418–491.

[152] P. Niyogi, S. Smale, and S. Weinberger, *Finding the homology of submanifolds with high confidence from random samples*, *Discrete & Computational Geometry* **39** (2008) 419–441.

[153] D. Geller and I. Z. Pesenson, *Band-limited localized parseval frames and besov spaces on compact homogeneous manifolds*, *Journal of Geometric Analysis* **21** (2011), no. 2 334–371.

[154] H. Tribel, *Theory of function space ii*, *Monographs in Mathematics* **78** (1992).

[155] M. J. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019.