Title
Structure-Informed Neural Network Architecture in Regression Applications

Author
Zhang, Jiefu

Publication Date
2021

Peer reviewed|Thesis/dissertation

Structure-Informed Neural Network Architecture in Regression Applications

by

Jiefu Zhang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Lin Lin, Chair
Professor Jon Wilkening
Professor Per-Olof Persson

Fall 2021

Structure-Informed Neural Network Architecture in Regression Applications

Abstract

Structure-Informed Neural Network Architecture in Regression Applications

by

Jiefu Zhang

Doctor of Philosophy in Applied Mathematics

University of California, Berkeley

Professor Lin Lin, Chair

This dissertation concerns the importance of the structure-informed neural network architecture in several regression applications. The word structure-informed neural network architecture refers to the neural network architecture that has some built-in properties motivated by the structure of the problem. For example, if the target function that the neural network is approximating has the permutation symmetry, instead of implementing a basic feed-forward neural network to explore the entire function space, we can build a permutation symmetric neural network architecture to only explore a much smaller space of permutation invariant functions. For supervised learning, there are multiple methods to achieve better numerical results when training a neural network for a regression application problem, including but not limited to: improved optimization techniques, increased training sample size, and superior neural network architecture. In this dissertation we investigate and observe the importance of the superior neural network architecture which incorporates the information hidden in the application problem, and then showcase a quantum chemistry application where such superior neural network architecture that incorporates various kinds of symmetries achieves powerful numerical results. Finally we discuss how one kind of structure, permutation symmetry, can be built into the neural network.

After the introduction of the fundamentals in chapter 1, in chapter 2 we investigate the importance of the structure-informed neural network architecture on a toy problem: use a neural network to approximate the mapping $\mathbf{x} \mapsto \sum_{i=1}^{n} x_i^2$. We observe that the role the structure-informed neural network architecture plays in this scenario is irreplaceable. For instance, it will take significantly more training data samples for a neural network architecture that does not take the problem structure into consideration

to match the performance of a neural network that does. The training tricks and heuristics, including using various optimizer or applying regularization, cannot easily close the gap of the performance either. In chapter 3 we look at one particular real world quantum chemistry application and observe how the superior neural network architecture leads to the positive numerical results. The application problem is to use a neural network to predict the electron density in an electronic system, with the input being atomic configuration (e.g. positions of some water molecules). There are several symmetries hidden in the problem. For example, interchanging two identical atoms should not affect the electron density in the system. By building the translation symmetry, rotation symmetry, and permutation symmetry into the neural network architecture, we are able to predict the electron density accurately for multiple 1D and 3D systems. In chapter 4 we focus on a specific type of structure that can be built into the neural network architecture, which is permutation symmetry. We summarize the existing approaches of incorporating the permutation invariant and equivariant symmetry into the neural network architecture and offer proofs of the validity of the approximation ansatz tailored for permutation symmetry.

Contents

Acknowledgments

First I would like to express my gratitude to my advisor Lin Lin. Lin's extensive help guides me through my PhD years and without his help, none of the work in this dissertation would have been possible .

I want to extend my sincere thanks to my collaborators for the work covered in this dissertation: Leonardo Zepeda-Núñez, Yixiao Chen, Linfeng Zhang, Weile Jia, Yuan Yao, Jiequn Han, Yingzhou Li, and Jianfeng Lu with whom I feel very honored and fortunate to have the opportunity to work with.

Among those who have offered me great support during my graduate study, I would like to thank Vicky Lee, Isabel Seneca for all the logistics assistance. I would like to thank Lin Lin, Per-Olof Persson, and Jon Wilkening, for being the members in both my qualification exam committee and my dissertation committee. I would like to thank Martin Head-Gordon for being in my qualification exam committee.

I would like to express my thanks to my parents, my mother Rong Zhou and my father Lixin Zhang, for their endless love and support. Finally, I would like to express my appreciation to my wife Qinyi Zhu, who I met during the graduate study and became the most important person in my life.

Chapter 1

Introduction

The artificial neural network has achieved great success in real world applications including speech recognition [46], computer vision [59], drug discovery [68], genomics [63], etc. In the past decade, many studies aim at replicating the success of the neural network in the field of scientific computing because of the advantage the neural network has for high dimensional problems and the efficiency in computing first derivatives of the neural network by the process of back propagation. One such successful application is to use the neural network ansatz to approximate the solutions to high dimensional partial differential equations [41]. Unlike the popular classification problems on common data sets such as MNIST, CIFAR10 and ImageNet, where the output label is categorical, the scientific computing problems usually have a continuous output label because the goal is to approximate some target continuous functions with a neural network. The process of fitting a neural network model to a target function in scientific computing problems can be regarded as a regression application, and these regression applications will be our main focus in this dissertation.

While there have been a number of noteworthy architectures arising from classification applications, including but not limited to Convolutional Neural Network (CNN) [62] for image classification, and Recurrent Neural Network (RNN) [81] for speech recognition, the importance of the neural network architecture informed by the structure of the problem is just starting to be revealed for the regression applications. This dissertation concerns the structure-informed architecture in several regression applications which can be found in scientific computing. In this chapter we provide the preliminary materials to set up the foundation for the remaining discussion in this dissertation.

1.1 Basics of an artificial neural network

An Artificial Neural Network (ANN) [71] can be viewed as a map from inputs to outputs. We start with the feed-forward neural network, and we use Fig. 1.1 as a sample feed-forward neural network. Here the leftmost layer corresponds to the input layer, and in this case the input is 3 dimensional. The rightmost layer corresponds to the output layer, and in this case the output is 1 dimensional. The two layers in between are known as the hidden layers, and each circle in the hidden layers represents a neuron. The evaluation of the neural network output based on a given input can be carried out by a forward propagation from left to right. The mathematical formula for computing the numbers in the next layer $l+1$ based on the current layer l is

$$\mathbf{x}^{(l+1)} = \sigma(W^{(l,l+1)}\mathbf{x}^{(l)} + \mathbf{b}^{(l+1)})$$

Let m denote the number of neurons in the layer $l+1$, and n denote the number of neurons in the layer l, then $\mathbf{x}^{(l+1)}$ is an m-dimensional vector, and $\mathbf{x}^{(l)}$ is an n-dimensional vector. The $m \times n$ weight matrix $W^{(l,l+1)}$, and the n-dimensional bias vector $\mathbf{b}^{(l+1)}$ are all parameters that can be considered to be fixed when the forward propagation occurs. The $\sigma(\cdot)$ function is known as the activation function. The role the activation function plays is crucial especially when the target function that the neural network is approximating is non-linear. Notice that for a particular entry in the weight matrix, say the entry at row p and column q, the associated one dimensional scalar equation will be

$$\mathbf{x}_p^{(l+1)} = \sigma(W_{pq}^{(l,l+1)}\mathbf{x}_q^{(l)} + \mathbf{b}_p^{(l+1)})$$

This equation evaluates the pth neuron in the layer $l+1$ based on the value of the qth neuron in the layer l, and thus in a picture like Fig. 1.1, each line segment connecting two neurons can be regarded as a parameter in the corresponding weight matrix.

The power of the feed-forward neural network lies in two aspects. First, the neural network is remarkably expressive. The universal approximation theorem [21] states that a feed-forward neural network with one hidden layer, arbitrary width, and sigmoid activation function can approximate any well-behaved function. Since then the universal approximation theorem has been extended to more cases [20, 48, 52, 73] (fixed width and arbitrary depth, or with other activation functions, etc.), and these results are the theoretical proofs of the expressiveness of the neural network. Second, for a neural network, the derivatives of the output with respect to any intermediate variable are easy to compute due to back propagation [37]. This allows a quick derivative computation for gradient related optimization algorithms, and leads to a

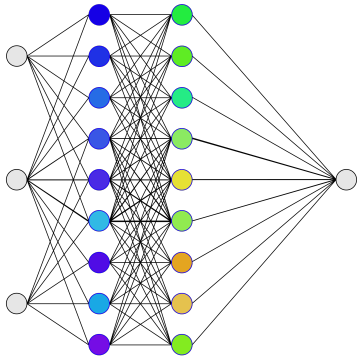


Figure 1.1: An example of a feed-forward neural network

quick update for all parameters in a neural network during the optimization process. The optimization procedure will be introduced in the next section.

1.2 Basics of supervised learning

In the previous section we have introduced the basic neural network and the related notations. In this section, we introduce the learning scheme that finds the best parameters (weight matrices and biases) so that the resulting neural network can accurately represent the target function we wish to learn.

In this dissertation we focus on the supervised learning where each data point has both the input label and the output label. The collection of data can be split into two parts: a training data set and a test data set. The neural network will be trained using the training data set, and its performance will be evaluated in the end using the test data set. The performance can be measured using a loss function. As an example and also the most widely used loss function, the formula of the mean squared loss is given by

$$l(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y}_i - f_{\mathrm{NN}}(\mathbf{x}_i))^2$$

For simplicity we assume our output dimension (dimension of $\mathbf{y}_i$) to be 1, and we have N data points in total. $f_{\mathrm{NN}}(\cdot)$ is the current neural network of which we want

to evaluate the performance (meaning that all parameters in the neural network are fixed in the loss evaluation process). The loss, which is the output of the loss function for a given neural network and a fixed data set, can be regarded as a measure of the accumulated errors. The optimization goal for the supervised learning training process is to minimize the loss for the training data set by updating the parameters in the neural network.

The training of the neural network is done using an optimization scheme. For simplicity we will only discuss the gradient descent algorithm in this section. For a given batch of training data, we can use the loss function to measure the performance of the current neural network. The optimization goal is to minimize the loss function, which is a measure of the errors. The back propagation allows efficient computation of the partial derivative of the loss function with respect to each individual neural network parameter for a given batch of training data with the input output pair $(\mathbf{x}, \mathbf{y})$. Once all derivatives are computed, the parameters can then be updated using the formula

$$w_{new} = w_{old} - \frac{\partial l}{\partial w}(\mathbf{x}, \mathbf{y})$$

Here w stands for a scalar parameter from either the weight matrices or the bias vectors. We can repeat this parameter update process by iterating through batches of the training data again and again, and we expect to see a decreasing trend of the training loss (error on the training data) as the iteration number increases. However, minimal training loss is not necessarily a good thing because the neural network may overfit the training data set. The difference between the training loss and the test loss is called the generalization gap. Despite how small the training loss is, a large generalization gap implies that the positive performance of the neural network cannot be generalized, and thus we consider the performance of this neural network to be negative. In reality, we can use the test loss to determine the sweet spot between underfitting and overfitting and select the best trained neural network model.

There are many useful tricks and heuristics that can improve the optimization process, including but not limited to, adaptive optimizer [53], pruning which reduces the parameters without affecting the performance [11], and regularization [88]. Here we introduce the regularization in more details to prepare for the discussion in chapter 2. The motivation behind the regularization is the issue that the magnitude of the parameters in the neural network becomes uncontrollably large after multiple rounds of gradient update. The large magnitude may lead to many numerical stability issues and in many real world applications the large magnitude itself is counter-intuitive. The regularization technique adds a term in the loss function that penalizes the large

magnitude of the parameters. For example, the L1 regularization may have the loss function in the form

$$l(\mathbf{x}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - f_{\mathrm{NN}}(\mathbf{x}_i))^2 + \lambda \sum_{\text{all parameters } w} |w|$$

The regularization constant λ controls how much we wish to penalize the large magnitude of the parameters and is an adjustable hyper-parameter. By minimizing this modified loss function, the magnitude of the parameters will stay relatively stable. The regularized neural network usually displays a smaller generalization gap and can achieve better performance on the test data set.

1.3 Structure-informed neural network

In this dissertation we use the word structure-informed neural network to refer to a neural network that incorporates the structure of the problem into its architecture. It has been shown in various literature that the choice of the neural network architecture to reduce the number of parameters is of paramount importance for the quality of the approximation [44, 45, 73], and often the most effective way to reduce the number of parameters without sacrificing the desired model expressiveness is to utilize the structure of the target function. Below we briefly introduce the two scenarios covered in this dissertation. We then highlight how the structure-informed neural network takes advantage of the information given in the problem in these two scenarios.

One of the simplest examples of the structure-informed neural network is the neural network we build in chapter 2 when approximating the target function $\mathbf{x} \mapsto \sum_{i=1}^{n} x_i^2$, where the input is an n-dimensional vector, and the output is the squared norm of this input vector. We observe that the target function has the following structure: The scalar mapping $t \mapsto t^2$ is applied to each entry of the input vector, and then the sum of the outputs from these scalar mappings give the desired squared norm of the input vector. Using this piece of information, we can design a neural network with the architecture pictured in Fig. 1.2, where the small feed-forward neural networks (2 hidden layers and 3 neurons in each layer in the figure) attached to each input entry share identical parameters and are approximating the scalar mapping $t \mapsto t^2$. The structure-informed architecture greatly reduces the burden of the training procedure because the neural network now only needs to learn the scalar mapping instead of approximating a function whose domain is $\mathbb{R}^n$. More details can be found in chapter 2.

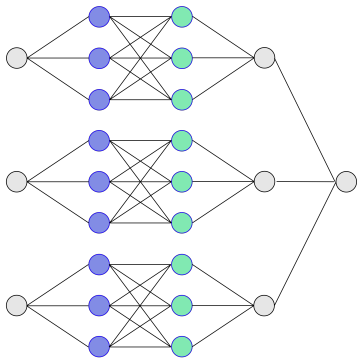


Figure 1.2: Structure-informed neural network for approximating the squared norm function $\mathbf{x} \mapsto \sum_{i=1}^{n} x_i^2$. Here the input dimension of vector $\mathbf{x}$ is 3.

A more complicated example is the quantum chemistry application discussed in chapter 3. The task is to learn the electron density based on the atomic configuration input. For example, if we input the positions of two hydrogen atoms in 3D space and one location $\mathbf{r}$ in 3D space to the neural network, the output will be a scalar representing the electron density (can be regarded as the probability of finding an electron) at $\mathbf{r}$ for this hydrogen system. If we repeat the evaluation process for all grid locations, we will obtain a heat map characterizing the electron density in the 3D space. When designing the neural network for this application, we incorporate the structure of this problem from two directions. First is on the electron density (output) side. Since the electron density is highly localized in the neighborhood of the atoms and resembles a normal distribution, we only use the neural network and parameters to represent the coefficients in front of the exponential function and on the exponents. Second is on the atomic configuration (input) side. The translation symmetry, rotation symmetry, and permutation symmetry of the atomic configuration can all be observed in this application, so we build the neural network with a carefully designed architecture so that the output of the neural network is invariant with respect to these symmetries. By utilizing the structure of the problem, our resulting neural network model is able to produce snapshots of electron density for various 3D systems that are usually very difficult to compute. More mathematical details can be found in chapter 3.

During these years experimenting with various neural networks, my observation is that there is no free lunch in neural network training. If the structure of the problem is

not built into the neural network architecture, the cost in the optimization procedure may be huge and usually we may not even be able to fix the issue by increasing the budget. By spending more effort in constructing a structure-informed neural network architecture, we will have a better parameter landscape for the optimization procedure to explore, and thus have more hope in finding the optimal neural network. For example, the successful implementation of neural networks in the past decade [82, 75, 18, 90, 91, 42] tackling molecular dynamics simulation (similar to the application problem in chapter 3) all have one kind or another of a cleverly designed architecture that incorporates certain symmetries. The overall goal of this dissertation is to display the advantage of the structure-informed neural network architecture in regression applications.

1.4 Permutation symmetry

In this section we introduce one particular type of structure that can be taken advantage of when building a neural network architecture, the permutation symmetry. In the past decade there have been multiple success stories of neural networks in representing high-dimensional permutation invariant/equivariant functions with remarkable accuracy and efficiency in various applications, see, e.g., [10, 90, 103, 105] for interatomic potential energy, [79, 80] for 3D classification and segmentation of point sets, and [34, 106] for solutions of partial differential equations.

We begin by defining the permutation invariant mapping. A function $f : (\mathbb{R}^d)^N \to \mathbb{R}$ is permutation invariant if

$$f(\mathbf{x}_{\sigma(1)}, \ldots, _{\sigma(N)}) = f(\mathbf{x}_1, \ldots, \mathbf{x}_N), \tag{1.1}$$

for any permutation $\sigma \in S(N)$, and elements $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$. Note that the permutation is only applied to the outer indices $1, \ldots, N$, but not the Cartesian indices $1, \ldots, d$ for each $\mathbf{x}_i$. A good example to consider is a system with 10 identical atoms in a 3D space. Here we consider the input being the collection of atom positions with $N = 10$ and $d = 3$. The system will be the same if we interchange two atom positions, but will be different if we interchange the xy coordinates of one particular atom. A closely related concept we will introduce next is the permutation equivariant mapping, which is of the form $Y : (\mathbb{R}^d)^N \to (\mathbb{R}^{\tilde{d}})^N$ that satisfies

$$Y_i(\mathbf{x}_{\sigma(1)}, \ldots, \mathbf{x}_{\sigma(N)}) = Y_{\sigma(i)}(\mathbf{x}_1, \ldots, \mathbf{x}_N), \qquad i = 1, \ldots, N \tag{1.2}$$

for any permutation $\sigma \in S(N)$, and $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$. Here each component $Y_i \in \mathbb{R}^{\tilde{d}}$, and $\tilde{d}$ can be different from d.

There are multiple ways to build the permutation symmetry into the architecture of the neural network, which can be seen in the works mentioned at the beginning of this section, and also the neural network model defined in chapter 3. One of the most important theoretical results that enables these neural network designs is mentioned in [98, Theorem 2]. The theorem states that a function f is permutation invariant if and only if it satisfies the decomposition

$$f(\mathbf{x}) = \phi(\sum_{j=1}^{N} g(\mathbf{x}_j))$$

for some functions $g : \mathbb{R}^d \to \mathbb{R}^M$ and $\phi : \mathbb{R}^M \to \mathbb{R}$. Here M is a parameter depending on the construction. The most straightforward usage of this theorem for building a permutation invariant neural network architecture is to now build two separate neural networks to approximate ϕ and g instead of the original single neural network that approximates f. Although two functions now need to be approximated compared with one, we no longer have the dependence on N for the target functions that the neural network models are approximating, and this may greatly reduce the number of necessary parameters, especially when N is large.

Our main contribution in this dissertation regarding the permutation symmetry is centered around the theorem mentioned above. The proofs in [98] work for the $d = 1$ case, but cannot be easily generalized to $d > 1$ case. In this dissertation we give two different proofs of the universality of the ansatz proposed in [98], both with explicit error bounds. The first proof is based on the Ryser formula [84] for permanents, and the second is based on the partition of the state space. The results justify the usage of this ansatz for applications where $d > 1$.

1.5 Organization of the dissertation

In chapter 2 we design a deceivingly simple task: use neural network to approximate the mapping $\mathbf{x} \mapsto \sum_{i=1}^{n} x_i^2$ through supervised learning. Given the knowledge of the separable structure of the function, a sparse network architecture can be designed to represent the function accurately, or even exactly. When such structural information is not available and we may only use a dense network architecture, the optimization

procedure to find the sparse network embedded in the dense network is similar to finding the needle in a haystack, with a fixed number of samples of the function. We demonstrate that the cost (measured by sample complexity) of finding the needle is directly related to the Barron norm of the function. While only a small number of samples is needed to train a sparse network, the dense network trained with the same number of samples exhibits large test loss and a large generalization gap. In order to control the size of the generalization gap, we find that the use of explicit regularization becomes increasingly more important as d increases. The numerically observed sample complexity with explicit regularization scales as $\mathcal{O}(d^{2.5})$, which is in fact better than the theoretically predicted sample complexity that scales as $\mathcal{O}(d^4)$. Without explicit regularization (also called implicit regularization), the numerically observed sample complexity is significantly higher and is close to $\mathcal{O}(d^{4.5})$.

In chapter 3 we look at a successful implementation of a neural network with built-in symmetries in a quantum chemistry application. We leverage the neural network structure inspired by Deep Potential to effectively represent the mapping from the atomic configuration to the electron density in Kohn-Sham density function theory (KS-DFT). By directly targeting at the self-consistent electron density, we demonstrate that the adapted network architecture, called the Deep Density, can effectively represent the self-consistent electron density as the linear combination of contributions from many local clusters. The network is constructed to satisfy the translation, rotation, and permutation symmetries, and is designed to be transferable to different system sizes. We demonstrate that using a relatively small number of training snapshots, with each snapshot containing a modest amount of data-points, Deep Density achieves excellent performance for one-dimensional insulating and metallic systems, as well as systems with mixed insulating and metallic characters. We also demonstrate its performance for real three-dimensional systems, including small organic molecules, as well as extended systems such as water (up to 512 molecules) and aluminum (up to 256 atoms).

In chapter 4 we focus on a certain type of symmetry widely observed in quantum chemistry, which is permutation invariance and equivariance. Besides summarizing the existing results, we give two different proofs of the universality of the ansatz for permutation invariant functions, both with explicit error bounds. The first proof is based on the Ryser formula for permanents, and the second is based on the partition of the state space.

Please note that Chapter 2 is based on [101], Chapter 3 is based on [100], and Chapter 4 is based on [43].

Chapter 2

A Toy Problem

2.1 Introduction

Machine learning and, in particular, deep learning methods have revolutionized numerous fields such as speech recognition [46], computer vision [59], drug discovery [68], genomics [63], etc. The foundation of deep learning is the universal approximation theorem [20, 48, 52, 73], which allows neural networks (NN) to approximate a large class of functions arbitrarily well, given a sufficient large number of degrees of freedom. In practice, however, the number of degrees of freedom is often limited by the computational power, thus the choice of the architecture to reduce the number of degrees of freedom is of paramount importance for the quality of the approximation [44, 45, 73]. Empirically, NN models have been shown to be surprisingly efficient in finding good local, and sometimes global, optima when using an overparameterized model, e.g. training a sparse teacher network is less efficient than training a dense, overparametrized student network [67]. It has been argued that the energy landscape of an overparameterized model may be benign, and in certain situations all local minima become indeed global minima [32, 51, 60, 92]. Furthermore, starting from an overparameterized model, observations such as the lottery ticket hypothesis [30, 31, 66] states that with proper initializations, it is possible to identify the "winning tickets", i.e. a sparse subnetwork with accuracy comparable to the original dense network.

We point out that many of the aforementioned studies focus on image classification problems using common data sets such as MNIST, CIFAR10 and ImageNet, with or without the presence of noise. However, in scientific computing, the setup of the problem can be very different: usually, we are interested in using NN models to parameterize a smooth, high-dimensional function accurately, and often without

artificial noise. Within this context, the results mentioned above naturally raise the following questions:

(1) How important is it to select the optimal architecture? In other words, does it matter whether one uses an overparameterized model?

(2) If there is a sparse subnetwork that is as accurate as the overparameterized network, can the training procedure automatically identify the subnetwork? In other words, what is the cost of finding the needle (sparse subnetwork) in a haystack (overparameterized network)?

(3) If (2) is possible, how does the training procedure (such as the use of regularization) play a role?

This chapter presents a case study of these questions in terms of a deceivingly simple task: given $\mathbf{x} \in [-1, 1]^d$ drawn from a certain probability distribution and a target accuracy ϵ, learn the square of its 2-norm, i.e. the function

$$\widetilde{f}^*(\mathbf{x}) := \sum_{i=1}^{d} x_i^2. \tag{2.1}$$

More specifically, for a given neural network model $f(\mathbf{x}, \theta)$, where θ denotes the parameters in the model, and for a given loss function, such as the quadratic loss $\ell(y, y') = \frac{1}{2}(y - y')^2$, our goal is to find θ such that the population loss

$$L(\theta) = \mathbb{E}_{\mathbf{x}}[\ell(f(\mathbf{x}; \theta), \widetilde{f}^*(\mathbf{x}))] \leq \epsilon.$$

Note that $\widetilde{f}^*(\mathbf{x}) \sim \mathcal{O}(d)$, so we consider the scaled target function [1] in order to normalize the output

$$f^*(\mathbf{x}) = \frac{1}{d}\widetilde{f}^*(\mathbf{x}). \tag{2.2}$$

However, as in many scientific computing applications, the magnitude of the quantity of interest indeed grows with respect to the dimension, and our interest here is to approximate the original function $\widetilde{f}^*(\mathbf{x})$ to ϵ accuracy. Using a quadratic loss the population loss needed for approximating f^* becomes ϵ/d^2. In other words, if each component of $\mathbf{x}$ is chosen randomly, then by the law of large number $f^*(\mathbf{x})$ converges to a constant $\mathbb{E}[x^2]$ as $d \to \infty$. So the ϵ/d^2 target accuracy means that it is the deviation from such a mean value that we are interested in.

[1] Correspondingly $\widetilde{f}^*$ will be called the original target function, or the unscaled target function.

If we are allowed to use the mapping $x \mapsto x^2$ as an activation function, we would apply this function to each component and sum up the results, ensuing that the representation will be exact. Therefore, we exclude such an activation function, and only use standard activation functions such as ReLU or sigmoid, which requires only $\mathcal{O}(\log(1/\epsilon))$ neurons to reach accuracy ϵ [97].

We can build a network leveraging the separability of Eq. (2.1). In particular, we can use a small network to approximately represent the scalar mapping $x \mapsto x^2$, and sum up the results from all components. The weights for the neural network of each component are shared, so the number of parameters is independent of d. The network will be called the local network (LN) below. However, if we do not have the *a priori* structural information that the target function is separable, we need to use a dense or fully connected neural network, which is referred to as the global network (GN). Fig. 2.1 sketches the structure of LN and GN. Note that LN can be naturally embedded into GN as a subnetwork by deleting certain edges. Therefore the *optimal* performance of GN should be at least as good as that of LN.

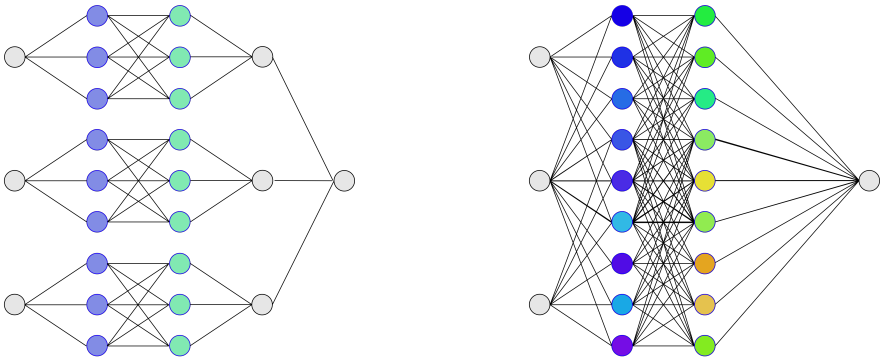


Figure 2.1: The architecture of the local network is on the left, and the architecture of the global network is on the right. Here $d = 3$ and the number of channels $\alpha = 3$.

Throughout this chapter we assume that the number of neurons in the LN is large enough so that the scalar mapping $x \mapsto x^2$ is learned very accurately (with test error less than e.g. 10^{-5}), and the structure GN is then obtained by connecting all the remaining edges. In addition to the architectural bias, we are also concerned with the sample complexity about GN, i.e. the number of independent samples of $f(\mathbf{x})$ as

the training data to reach a certain target accuracy (i.e. generalization error). Our results can be summarized as follows.

1. Using the same number of samples, LN can perform significantly better than GN. This shows that the global energy landscape of GN cannot be very simple, and the desired LN subnetwork cannot be easily identified.

2. If we embed a converged LN into a GN, add small perturbations to the weights of GN, and start the training procedure, LN can still outperform GN. This shows that the local energy landscape of GN may not be simple either.

3. In order to use GN to achieve performance that is comparable to LN, we need a significantly larger number of samples. The number of samples needed to reach certain target accuracy increases with respect to the dimension as $\mathcal{O}(d^\gamma)$ up to logarithmic factors. From *a priori* error analysis, we have $\gamma = 4$.

4. The numerical scaling of the sample complexity with respect to d depends on the regularizations. In particular, when proper regularization (ℓ^1, ℓ^2, or path norm regularization [26]) is used, the numerically observes sample complexity is around $\mathcal{O}(d^{2.5})$, which behaves better than the theoretically predicted worst case complexity, which scales as $\mathcal{O}(d^4)$. On the other hand, when implicit regularization (i.e. early stopping [96]) is used, we observe $n \sim \mathcal{O}(d^{4.5})$, i.e. the sample complexity using implicit regularization is significantly larger than that with explicit regularization. The early stopping criteria we use in this chapter is that: after T (1000 in default) epochs, we find an optimal $t^* \leq T$ where the validation error is minimized. Furthermore, the trained weight matrix obtained with explicit regularization is approximately a sparse matrix, while the weight matrix obtained with early stopping is observed to be a dense matrix.

Related works: In order to properly describe the sample complexity to reach certain target accuracy, we need to have *a priori* error estimate (a.k.a. worst-case error), and/or *a posteriori* error estimate (a.k.a. instance-based error) of the generalization error. For two-layer neural networks, such estimates have been recently established [5, 26] for a large class of functions called Barron functions [7, 54]. The estimates have also been recently extended to deep networks based on the ResNet structure [45, 25]. We obtain our theoretical estimate of the sample complexity with respect to d by applying results of [26] to $f(\mathbf{x})$. For a sufficiently wide two-layer neural network, [4] studied the generalization error together with the gradient descent dynamics when using a polynomial activation function. The problem of "finding the needle in a haystack" by comparing the performance of fully-connected networks

(FCN) and convolutional neural networks (CNN) was also recently considered by [22] for image recognition problems. It was found that there are rare basins in the space of fully-connected networks associated with very small generalization errors, which can be accessed only with prior information from CNN. This corroborates our finding for learning $f(\mathbf{x})$ here. Note that the separable structure in the target function can also be viewed from the perspective of permutation symmetry. Therefore our study also supports the argument for recognizing the importance of preserving symmetries when designing neural network architectures, which has been observed by numerous examples in physics based machine learning applications [98, 104]. Our result also corroborates the recent study [76], which questioned the prediction power of theoretical generalization error bound rates for overparameterized deep networks trained without explicit regularization.

Organization

In Section 2.2 we provide the theoretical foundations for the two-layer networks, including *a priori* and *a posteriori* bounds on the generalization gap. In Section 2.3 we use the theory developed in Section 2.2 to find the bounds for the generalization error for the squared norm. Section 2.4 provides the numerical experiments, followed by discussion in Section 2.5.

2.2 Generalization error of two-layer networks

In this section, we briefly describe the concept of the Barron norm, and the generalization error for two-layer neural networks. We refer readers to [26, 27] for more details. Let the domain of interest be $\Omega = [-1, 1]^d$. We assume the magnitude of the target function is already normalized to be $\mathcal{O}(1)$, e.g. the scaled function $f^*(\mathbf{x})$ in Eq. (2.2). Then for any $y' \sim \mathcal{O}(1)$, both the magnitude and the Lipschitz constant of the square loss function $\ell(\cdot, y')$ are of $\mathcal{O}(1)$.

Barron norm

We say that a function $f : \Omega \to \mathbb{R}$ can be represented by a two-layer NN if

$$f(\mathbf{x}; \theta) = \sum_{k=1}^{m} a_k \sigma\left(\mathbf{w}_k^\top \mathbf{x}\right). \tag{2.3}$$

Here $\mathbf{w}_k \in \mathbb{R}^d$, $\theta := \{(a_k, \mathbf{w}_k)\}_{k=0}^m$ represents all the parameters in the network, and $\sigma(\cdot)$ is an scale-invariant activation function such as ReLU. The scale invariance

implies

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \|\mathbf{w}\|_1 \, \sigma(\hat{\mathbf{w}}^\top \mathbf{x}), \quad \hat{\mathbf{w}} = \mathbf{w}/\|\mathbf{w}\|_1 . \tag{2.4}$$

Therefore we may, without loss of generality, absorb the magnitude $\|\mathbf{w}\|_1$ into the scalar a, and assume $\|\mathbf{w}\|_1 = 1$.

The training set is composed of n i.i.d. samples $\mathcal{S} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$. To distinguish the indices, we use $\mathbf{x}^{(i)}$ to denote the i-th sample of the vector ($1 \le i \le n$), and use $x_j^{(i)}$ to denote the j-th component of the vector $\mathbf{x}^{(i)}$ ($1 \le j \le d$).

Our goal is to minimize the population loss

$$L(\theta) = \mathbb{E}_{\mathbf{x}}[\ell(f(\mathbf{x}; \theta), y)],$$

through the minimization of the training loss

$$\hat{L}_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(f\left(\mathbf{x}^{(i)}; \theta\right), y^{(i)}\right).$$

For a realization of parameters θ, the generalization gap is defined as $\left|L(\theta) - \hat{L}_n(\theta)\right|$.

A function f represented by a two-layer neural network is a special case of the Barron function, which admits the following integral representation

$$f(\mathbf{x}) = \int_{S^d} a(\mathbf{w}) \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) \, \mathrm{d}\pi(\mathbf{w}), \tag{2.5}$$

where π is a probability distribution over $S^d := \{\mathbf{w} | \, \|\mathbf{w}\|_1 = 1\}$, and $a(\cdot)$ is a scalar function. In particular, when we choose $\pi(\mathbf{w}) := \sum_{k=1}^m \delta(\mathbf{w} - \mathbf{w}_k)$ to be a discrete measure and define $a_k = a(\mathbf{w}_k)$, we recover the standard two-layer network in Eq. (2.3).

Definition 1 (Barron norm and Barron space)**.** *Let f be a Barron function. Denote by Θ_f all the possible representations of f, i.e.*

$$\Theta_f = \{(a, \pi) | f(\mathbf{x}) = \int_{S^d} a(\mathbf{w}) \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) \, \mathrm{d}\pi(\mathbf{w})\}.$$

Then the Barron-p norm is defined by

$$\gamma_p(f) := \inf_{(a,\pi) \in \Theta_f} \left(\int_{S^d} |a(\mathbf{w})|^p \, \mathrm{d}\pi(\mathbf{w}) \right)^{1/p}. \tag{2.6}$$

We may then define the Barron space as

$$\mathcal{B}_p(\Omega) = \{f : \gamma_p(f) < \infty\}. \tag{2.7}$$

Unlike the standard L^p norms, Proposition 2 shows a remarkable result, which is that all Barron norms are equivalent ([27, Proposition 2.1]).

Proposition 2 (Equivalence of Barron norms)**.** *For any function $f \in \mathcal{B}_1(\Omega)$,*

$$\gamma_1(f) = \gamma_p(f), \quad 1 \le p \le \infty. \tag{2.8}$$

To see why this can be the case, let us consider the two-layer NN in Eq. (2.3), and assume $\|\mathbf{w}_k\|_1 = 1$ for all k. By Hölder's inequality $\gamma_p(f) \le \gamma_q(f)$ when $1 \le p < q \le \infty$. In particular, $\gamma_1(f) \le \gamma_\infty(f)$, and we only need to show $\gamma_\infty(f) \le \gamma_1(f)$. Since π should be a discrete measure, let (a, π) be the minimizer for γ_1 as in Eq. (2.6). Then (recall $a_k := a(\mathbf{w}_k)$)

$$\gamma_1(f) = \sum_{k=1}^m |a_k| .$$

However, we may define a new distribution

$$\widetilde{\pi}(\mathbf{w}) = \sum_{k=1}^m \frac{1}{\gamma_1(f)} |a_k| \, \delta(\mathbf{w} - \mathbf{w}_k),$$

then

$$\int_{S^d} a(\mathbf{w}) \, \mathrm{d}\pi(\mathbf{w}) = \sum_{k=1}^m |a_k| = \gamma_1(f) \sum_{k=1}^m \frac{1}{\gamma_1(f)} |a_k| = \gamma_1(f) \int_{S^d} \mathrm{d}\widetilde{\pi}(\mathbf{w}).$$

In other words, $(\widetilde{a}, \widetilde{\pi}) \in \Theta_f$, where $\widetilde{a}_k = \gamma_1(f)$ is a constant. By definition of Eq. (2.6),

$$\gamma_\infty(f) \le \gamma_1(f) \int_{S^d} \mathrm{d}\widetilde{\pi}(\mathbf{w}) = \gamma_1(f).$$

This proves $\gamma_\infty(f) = \gamma_1(f)$. The same principle can be generalized to prove Proposition 2 for general Barron functions [27].

Proposition 2 shows that the definition of the Barron space $\mathcal{B}_p(\Omega) = \mathcal{B}_1(\Omega)$ is independent of p. Therefore we may drop the subscript p and write $\mathcal{B}_p(\Omega)$ as $\mathcal{B}(\Omega)$. Similarly we denote by $\|f\|_{\mathcal{B}} := \gamma_1(f)$ as the Barron norm.

Error bound

For a given parameter set θ defining Eq. (2.3), the Barron norm is closely related to the path norm

$$\|\theta\|_{\mathcal{P}} := \sum_{k=1}^m |a_k| . \tag{2.9}$$

According to Eq. (2.6) we immediately have $\|f\|_{\mathcal{B}} \leq \|\theta\|_{\mathcal{P}}$. The path norm can be used to obtain the following *a posteriori* error estimate for the generalization gap for any choice of θ.

Theorem 3 (*A posteriori* error estimate [26])**.** *For any choice of parameter θ, for any $\delta > 0$, and with probability at least $1 - \delta$ over the choice of the training set $\mathcal{S}$,*

$$\left|L(\theta) - \hat{L}_n(\theta)\right| \lesssim \sqrt{\frac{\ln(d)}{n}} \left(\|\theta\|_{\mathcal{P}} + 1\right) + \sqrt{\frac{\ln((\|\theta\|_{\mathcal{P}} + 1)^2/\delta)}{n}}.$$

For a given two-layer network, the path norm can be computed directly. Hence in order to reduce the generalization error to ϵ, the number of samples needed is approximately $\mathcal{O}(\|\theta\|_{\mathcal{P}}^2/\epsilon^2)$, which is a practically useful bound. However, in order to estimate the scaling of n with respect to the dimension d for a particular function, we need to replace the $\|\theta\|_{\mathcal{P}}$ by some measure of the complexity associated with f^* itself, such as the Barron norm. There indeed exists a two-layer network with a bounded path norm, so that the population loss is small. This is given in Proposition 4 ([26, Proposition 2.1]).

Proposition 4. *For any $f \in \mathcal{B}(\Omega)$, there exists a two layer neural network $f(\mathbf{x}; \tilde{\theta})$ of width m with $\|\tilde{\theta}\|_{\mathcal{P}} \leq 2\|f^*\|_{\mathcal{B}}$, such that*

$$L(\widetilde{\theta}) \lesssim \frac{\|f^*\|_{\mathcal{B}}}{m}. \tag{2.10}$$

Eq. (2.10) characterizes the approximation error due to the use of a neural network of finite width, which decays as m^{-1}. Proposition 4 states that it is in principle possible to reduce the population loss while keeping the path norm being bounded. However, numerical results (both previous works and our own results here) indicate that when minimizing with respect to the training loss directly (when early stopping is used, this is also called the implicit regularization), the path norm associated with the optimizer can be very large. According to Theorem 3, the resulting generalization error bound can be large as well.

A key result connecting the *a priori* and *a posteriori* error analysis is to impose stronger requirements of the training procedure. Instead of minimizing with respect to the training loss $\hat{L}_n(\theta)$ directly, [26] proposes to minimize with respect to the following regularized loss function

$$J_\lambda(\theta) := \hat{L}_n(\theta) + \lambda\|\theta\|_{\mathcal{P}}, \tag{2.11}$$

where $\lambda > 0$ is a penalty parameter. The corresponding minimizer is defined as

$$\hat{\theta}_{n,\lambda} = \arg\min_{\theta} J_\lambda(\theta).$$

The benefit of minimizing with respect to Eq. (2.11) is that the path norm is penalized explicitly in the objective function, which allows us to control both the path norm and the generalization error.

Theorem 5 (*A priori* error estimate [26])**.** *Assume that the target function $f^* \in \mathcal{B}(\Omega)$, and $\lambda \geq \lambda_n = 4\sqrt{2\ln(2d)/n}$. Then for any $\delta > 0$ and with probability at least $1-\delta$ over the choice of the training set $\mathcal{S}$*

$$L(\hat{\theta}_{n,\lambda}) \lesssim \frac{\|f^*\|_{\mathcal{B}}^2}{m} + \lambda(\|f^*\|_{\mathcal{B}} + 1) + \frac{1}{\sqrt{n}}\left(\|f^*\|_{\mathcal{B}} + \sqrt{\ln(n/\delta)}\right). \tag{2.12}$$

The path norm of the parameter satisfies

$$\left\|\hat{\theta}_{n,\lambda}\right\|_{\mathcal{P}} \lesssim \frac{\|f^*\|_{\mathcal{B}}^2}{\lambda m} + \|f^*\|_{\mathcal{B}} + \sqrt{\ln(1/\delta)}. \tag{2.13}$$

For a fixed target function f^*, the contribution to the error in Eq. (2.12) can be interpreted as follows: the first term is the approximation error, determined by the width of the network. The third is determined by the sample complexity which is proportional to $n^{-\frac{1}{2}}$. The second term is present due to the need of balancing the loss and the path norm in the objective function (2.11). If λ is too large, then the regularized loss function is too far away from the training loss, and the error bound becomes large. On the other hand, Theorem 5 requires λ should be at least $\sim n^{-\frac{1}{2}}$. This can also be seen from Eq. (2.13) that if λ is too small, the path norm becomes unbounded. Therefore to balance the two factors we should choose the regularization parameter as $\lambda \sim n^{-\frac{1}{2}}$.

2.3 Generalization error for squared norm

In this section we apply the generalization error bound in Section 2.2 to study the scaling of the sample complexity with respect to the dimension d for the function in Eq. (2.1). The two-layer network in Eq. (2.3) can be viewed as a particular realization of GN. According to Theorem 3 and Theorem 5, we need to estimate the Barron norm $\|f^*\|_{\mathcal{B}}$.

For a given function, the Barron norm is often difficult to compute due to the minimization with respect to all possible (a, π). Instead we may compute the spectral

norm. For a given function $f \in C(\Omega)$, let F be an extension of f to $\mathbb{R}^d$, denoted by $F|_\Omega = f$. Define the Fourier transform

$$\hat{F}(\mathbf{k}) = \frac{1}{2\pi} \int_{\mathbb{R}^d} e^{-i\mathbf{k}\cdot\mathbf{x}} F(\mathbf{x}) \, \mathrm{d}\mathbf{x}.$$

Then spectral norm of f is defined as

$$\|f\|_s = \inf_{F|_\Omega = f} \int_{\mathbb{R}^d} \|\mathbf{k}\|_1^2 |\hat{F}(\mathbf{k})| \, \mathrm{d}\mathbf{k}. \tag{2.14}$$

Note that the infimum is taken over all possible extensions F. Then the Barron norm can be bounded by the spectral norm as [27, Theorem 2]

$$\|f\|_{\mathcal{B}} \leq 2 \|f\|_s + 2\|\nabla f(0)\|_1 + 2|f(0)|. \tag{2.15}$$

Therefore we may obtain an upper bound of the Barron norm via the spectral norm.

Let us now consider $f(\mathbf{x})$ in (2.2), which satisfies $\|\nabla f(0)\|_1 = |f(0)| = 0$. To evaluate the spectral norm, we consider the one-dimensional version $g(x) = x^2$. Consider any C^2 extension of g to $\mathbb{R}$, denoted by G, which satisfies $\int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k < \infty$. Then by definition $\gamma_s(g) \leq \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k < \infty$.

Now consider the extension $F(\mathbf{x}) = \frac{1}{d} \sum_{i=1}^d G(x_i)$, and $\hat{F}(\mathbf{k}) = \frac{1}{d} \sum_{i=1}^d \hat{G}(k_i) \prod_{j \neq i} \delta(k_j)$. Here $\delta(\cdot)$ is the Dirac-δ function. Then

$$\int_{\mathbb{R}^d} \|\mathbf{k}\|_1^2 |\hat{F}(\mathbf{k})| \, \mathrm{d}\mathbf{k} = \frac{1}{d} \sum_{i=1}^d \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k = \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k,$$

which gives

$$\|f\|_s \leq \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k. \tag{2.16}$$

Combined with Eq. (2.15), we find that $\|f\|_{\mathcal{B}} \sim \mathcal{O}(1)$. Thus according to Proposition 4 we expect that the path norm of the regularized solution satisfies $\|\hat{\theta}\|_{\mathcal{P}} \sim \mathcal{O}(1)$. Hence the leading term of the generalization gap is

$$\|\theta\|_{\mathcal{P}} \sqrt{\frac{\ln(d)}{n}} \sim \sqrt{\frac{\ln(d)}{n}}. \tag{2.17}$$

This scaling seems quite favorable, as n only needs to grow as $\ln(d)$. However, recall the discussion in Section 2.1 that the target generalization error should be ϵ/d^2, this means that the required sample complexity (up to logarithmic factors) is

$$n \sim \mathcal{O}(d^4/\epsilon^2). \tag{2.18}$$

According to Theorem 5, the network also needs to be wide enough as $m \sim \mathcal{O}(d^2/\epsilon)$. $\lambda \sim \mathcal{O}(n^{-\frac{1}{2}}) = \mathcal{O}(\epsilon/d^2)$. In particular, the sample complexity increases very rapidly with respect to the dimension d in order to approximate the unscaled function $\widetilde{f}^*(\mathbf{x})$.

The analysis of the local network (LN) is essentially applying a two-layer network $g(x;\theta)$ to the scalar mapping $g(x) = x^2$. Since $g(x) = x^2$ is a special case of $f^*(\mathbf{x}) = \frac{1}{d}\sum_{i=1}^d x_i^2$ with $d = 1$, by Eq. (2.18) and setting $d = 1$, the sample complexity for the scalar mapping should be $n \sim \mathcal{O}(1/\epsilon^2)$. Let $z_i = x_i^2 - g(x_i;\theta)$, and assume that the error from each component are of mean zero and independent, i.e.

$$\mathbb{E}(z_i z_j) = 0, \quad i \neq j. \tag{2.19}$$

Then the generalization error is simply

$$\begin{aligned} L(\hat{\theta}) =& \frac{1}{2}\mathbb{E}\left(\frac{1}{d}\sum_{i=1}^d x_i^2 - \frac{1}{d}\sum_{i=1}^d g(x_i;\theta)\right)^2, \\ =& \frac{1}{2d^2}\mathbb{E}\left(\sum_{i=1}^d z_i^2 + 2\sum_{i\neq j} z_i z_j\right), \\ \approx& \frac{1}{2d}\mathbb{E}\left(z_i^2\right) \sim \frac{1}{d}. \end{aligned}$$

Therefore the generalization error of LN can be $\mathcal{O}(d^{-1})$ smaller than that of GN. Note that the condition (2.19) is crucial for the d^{-1} factor. In fact if the errors from all components are correlated, there may be no gain at all in terms of the asymptotic scaling with respect to d!

However, our numerical results in Section 2.4 demonstrate that the performance of LN can be significant better than GN by a very large margin. Therefore there is still gap in terms of the theoretical understanding of the performance of LN.

2.4 Numerical results

In this section we describe in detail the numerical experiments along with the analysis of the empirical results.

Our first goal is to study the dependence of the generalization or test error with respect to the architectural bias. In section 2.4, we showcase the superior performance of the local network, which is built using structural information about the underlying

problem, compared to the the global network, in which no information is used.

Our second goal is to study the impact of the explicit regularization (versus implicit regularization) when training the global network to approximate the unscaled target function. In section 2.4, we characterize the scaling of the test loss with respect to the input dimension, and the scaling of the test loss with respect to the number of samples with/without the explicit regularization. The explicit regularization improves both rates according to the numerical experiments.

Data generation and loss function

For all the numerical experiments, the dimension of the input denoted by d, ranges from 4 to 60 with step size 4. For each d, we generate 10^6 samples by first sampling $10^6 \times d$ numbers uniformly from the interval $[-1, 1]$ and then organizing the resulting data as a matrix of dimensions $10^6 \times d$. We then compute $y = \sum_{i=1}^{d} x_i^2$ for each row. With this setup, the domain of the target function is restricted to $\Omega = [-1, 1]^d$.

We also generate data for the sum of quartic and the cosine terms, namely $y = \sum_{i=1}^{d} x_i^4$ or $y = \sum_{i=1}^{d} \cos x_i$. These datasets are used in section 2.4 to showcase the importance of the architectural bias for target functions other than $y = \sum_{i=1}^{d} x_i^2$.

For the sake of reproducibility, all the experiments described below use the data generated with a fixed seed. In this section we use n to denote the number of the training samples and N_{sample} for the number of total samples (training, validation, test all combined). For experiments that involve $N_{\text{sample}} \leq 10^6$ samples, we extract the first N_{sample} rows from the total 10^6 rows of data.

To simplify the training procedure, we enforce the permutation symmetry of the inputs by including the sorting procedure while preprocessing the input data. We point out that this permutation symmetry can be directly embedded in the network [99].

On one hand, we have done our analysis in section 2.2 and 2.3 with respect to the scaled target function $f^*(\mathbf{x})$ in Eq. (2.2), so the neural networks we build in this section aim to learn the scaled target function. Thus, the mean squared error loss for training data when optimizing the networks are defined as

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(f_{NN}(\mathbf{x}^{(i)}, \theta) - f^*(\mathbf{x}^{(i)}) \right)^2 \tag{2.20}$$

Here n denotes the number of training data, f_{NN} denotes the function represented by the network, and $\mathbf{x}^{(i)}$ represents ith row in the dataset. The test loss can be calculated in the same way using the test data instead of the training data. On the other hand, we are interested in the performance of the models in the original scale, so the reported training/test loss are scaled by d^2 to represent the mean squared error of approximating the original unscaled target function:

$$\mathrm{MSE}_{\text{originial}} = \frac{1}{n}\sum_{i=1}^{n}\left(d \cdot f_{NN}(\mathbf{x}^{(i)}, \theta) - \widetilde{f}^{*}(\mathbf{x}^{(i)})\right)^2 \tag{2.21}$$

Architectural bias

In this section we showcase the empirical effects of the architectural bias on the test or generalization errors. To illustrate this effect, in a slightly more generality, we consider as target functions: $\widetilde{f}^{*}(\mathbf{x}) := \sum_{i=1}^{d} x_i^2$, $\sum_{i=1}^{d} x_i^4$, and $\sum_{i=1}^{d} \cos x_i$, respectively. We use three different network architectures to approximate each target function: global network, the local network, and the locally connected network (LCN), which is similar to the local network, but whose weights are not longer shared.

These three architectures are closely related. In the global (or dense) network, each layer is represented by a weight matrix that is an arbitrary dense matrix (see Figs. 2.1 and 2.2). In the locally connected network we have the same matrix but constrained to be block diagonal. Finally, in the local network we further constraint the weight matrix to be both block diagonal and block circulant, thus implying that the block in the diagonal are the same (usually called weight-sharing in the machine learning community). This observation allows us to embed a local network into a locally connected one, and then into a global one. This embedding is performed by simply copying the corresponding entries of the weights matrices at each layer and filling the rest with zeros as shown in Fig. 2.2.

Following [48] all three networks can approximate the target function to arbitrary accuracy with just one hidden layer: the result is straightforward for the global network by the universal approximation; for the locally connected and local networks, the result stems from applying the universal approximation to each coordinate, thus approximating the scalar component function $g : \mathbb{R} \to \mathbb{R}$ ($g(x) = x^2$ in the square case, x^4 in the quartic case and $\cos x$ in the cosine case). In practice, however, we find that with two hidden layers the networks are easier to train, all the hyperparameters being equal.

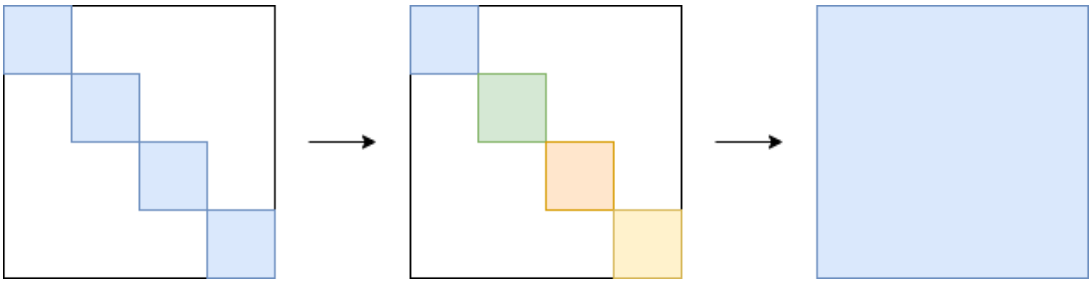


Figure 2.2: The relation of the weight matrices after embedding weight matrices for all three networks (between the first hidden layer and the second hidden layer with input dimension $d = 4$) into the weight matrix for the global network. The arrows mean that the matrices on the left hand side are subsets of the matrices on the right hand side. The blocks in the first weight matrix have the same color to illustrate the weight-sharing. Matrix elements in the white regions are zeros.

Following these considerations, for all the numerical experiments we fix the number of hidden layers to be two. In addition, we define α as the number of nodes per input node, which for the local networks coincides with the number of channels. Thus we have $d \cdot \alpha$ nodes in the hidden layers for the global network.

The networks are implemented[2] with Keras [19], using Tensorflow [1] as the back-end. Within the Keras framework, we use dense layers, locally connected layers and one-dimensional convolutional layers to implement the global network, the locally connected network, and the local network, respectively (see Algs. **??**, **??**, and **??**). We add a lambda layer that divides the output by d at the end of the networks to learn the scaled function $f^*(\mathbf{x})$ in Eq. (2.2). For both the locally connected and convolutional layers, we use stride and window size both equal to one, and α the number of channels. We point out that α can be considered as the number of nodes in the hidden layers for the block component that approximates the component function g as shown in Fig. 2.1. In this way, the function represented by the local network has the structure $\frac{1}{d}\sum_{i=1}^{d} g(x_i, \theta)$ enforced by the architecture, where $g(\cdot, \theta)$ is the neural network block that approximates component function g.

[2]Detailed implementation can be found at `https://github.com/jfetsmas/NormSquareLearning.git`

```
layerInput       = Input(shape=(d,))
layerHidden1     = Dense(DenseNodes, activation='relu',
                         use_bias=True)(layerInput)
layerHidden2     = Dense(DenseNodes, activation='relu',
                         use_bias=True)(layerHidden1)
layerOutput_pre  = Dense(1, activation='linear',
                         use_bias=False)(layerHidden2)
layerOutput      = Lambda(lambda x: x / d)(layerOutput_pre)
model            = Model(inputs=layerInput, outputs=layerOutput)
```

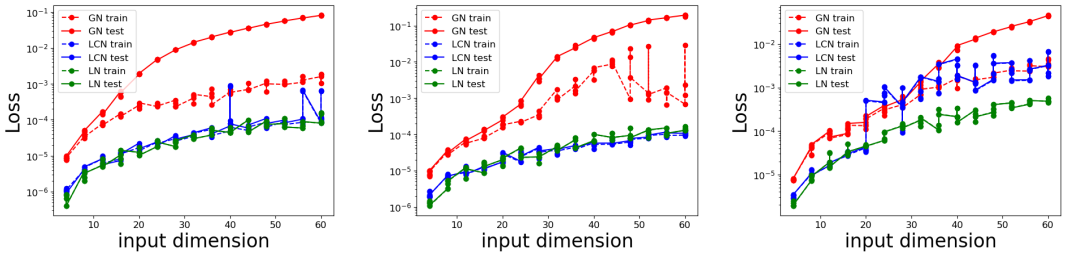
```
layerInput   = Input(shape=(d,1))
layerHidden1 = LocallyConnected1D(alpha, 1, strides=1,
               activation='relu', use_bias=True)(layerInput)
layerHidden2 = LocallyConnected1D(alpha, 1, strides=1,
               activation='relu', use_bias=True)(layerHidden1)
layerOutput  = LocallyConnected1D(1, 1, strides=1,
               activation='linear', use_bias=False)(layerHidden2)
Sum          = Lambda(lambda x: K.sum(x, axis=1), name='sum')
layerSum_pre = Sum(layerOutput)
layerSum     = Lambda(lambda x: x / d)(layerSum_pre)
model        = Model(inputs=layerInput, outputs=layerSum)
```

```
layerInput   = Input(shape=(d,1))
layerHidden1 = Conv1D(alpha, 1, strides=1, activation='relu',
               use_bias=True)(layerInput)
layerHidden2 = Conv1D(alpha, 1, strides=1, activation='relu',
               use_bias=True)(layerHidden1)
layerOutput  = Conv1D(1, 1, strides=1, activation='linear',
               use_bias=False)(layerHidden2)
Sum          = Lambda(lambda x: K.sum(x, axis=1), name='sum')
layerSum_pre = Sum(layerOutput)
layerSum     = Lambda(lambda x: x / d)(layerSum_pre)
model        = Model(inputs=layerInput, outputs=layerSum)
```

In the experiments, we set the number of channels $\alpha = 50$ and we use ReLU as the nonlinear activation function. The models are permutation invariant due to a sorting

procedure used to pre-process the data. In all experiments we use the default weight initializer (glorot uniform initializer) and the Adam optimizer with starting learning rate = 0.01 and other default parameters [53].

We split the data into training, validation, and test datasets. Among the data for a single numerical experiment, 64% is training data, 16% is validation data, and 20% is test data. We use batch size = (number of training and validation samples) / 100 for all experiments in this subsection, and thus each epoch contains 80 iterations. During the training process, we evaluate the loss on the validation set every epoch and we keep the model with the lowest validation loss, which is then reported.



(a) target function $y = \sum_{i=1}^{d} x_i^2$ (b) target function $y = \sum_{i=1}^{d} x_i^4$ (c) target function $y = \sum_{i=1}^{d} \cos x_i$

Figure 2.3: Comparison of rescaled average mean square error for the three architectures. GN stands for global network, LCN for locally connected network, and LN for local network.

For each value of the input dimension, we run four experiments with the same configuration, but with a different random seed for the optimizer. All experiments in this subsection use a dataset of size $N_{\text{sample}} = 10^5$ (training, validation, and test data all combined).

Fig. 2.3 depicts the behavior of the test and training losses for the three networks as the input dimension increases. The losses are computed using the mean squared error in the original scale as in Eq. (2.21). For all three target functions, the local network significantly outperforms the global network, especially for large input dimension. LN and LCN shows comparable performance for the quadratic and quartic functions. The training error of LCN is larger for the cosine function, but there is no noticeable generalization gap. We expect that the performance of LCN can be further improved through further hyperparameter tuning. In all three cases, the global network exhibits

a large generalization gap, whereas this gap is almost non-existent for the local network. These results clearly indicates the influence of the architecture in the accuracy of the approximation and the generalization gap.

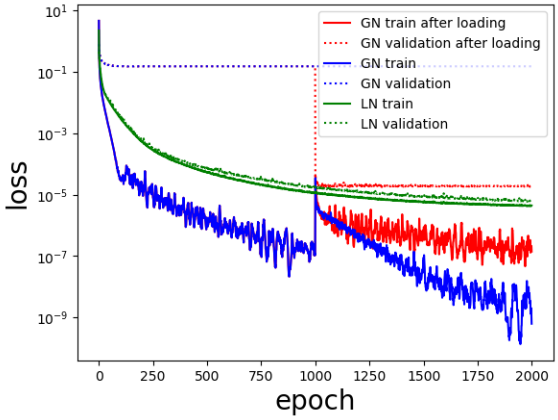


Figure 2.4: Evolution of the mean squared error with respect to the number of epochs for the different networks. Blue lines are the trajectories of the training and validation loss of the global network; green lines are for the local network; Red lines are the trajectories after loading the local network weights into the global network at epoch 1000.

As depicted in Fig. 2.2, we know that the local networks are indeed a subset of the global ones, and given the high-level of accuracy obtained by the local network we can infer that the global network should have enough representation power to approximate the target solution accurately. Following this logic, the large gap can be then attributed to the optimization procedure, which is unable to identify a suitable approximation to the local network ("needle") among all overparameterized dense networks ("haystack").

A natural question is: can the optimization procedure find a better solution starting from a good initial guess? To study this, we train a local network until a local minimum is achieved, and we transfer this minimum to its corresponding global network, and we use it as an initial guess for the training procedure. In particular, we fix $d = 20$, and the number of data at 1000. We train both the local and the global networks for 1000 epochs, do the weight transfer, and then resume training for another 1000 epochs. To obtain more stable training curves, we add decay of 0.03 in the argument of the Adam optimizer. From Fig. 2.4 we can observe that by properly

initializing the global network, we are able to significantly reduce the test loss, and to drastically bridge the generalization gap (red lines starting at epoch 1000). Although the training error of the global network is consistently lower than that of the local network (solid blue and red lines, compared with solid green line), the generalization gap (the gap between solid and dashed for the blue and red lines, compared with the gap for the green lines) of the global network remains noticeably larger compared with the local network.

The impact of the explicit regularization

In this subsection, we explore the relation between the test loss, the number of samples and the input dimension. The calculations in section 2.3 suggests that the leading term of the generalization gap should satisfy Eq. (2.17). For the global network, the generalization gap is sufficiently large[3] that we can regard the test loss to be approximately equal to the generalization error. Thus, we expect the test loss of the global network to be bounded by Eq. (2.17) (multiplied by d^2 for the unscaled function). We test the tightness of the bound by conducting the following two experiments:

1. we fix the sample size and investigate the relation between the test loss and the input dimension; and

2. we fix the input dimension d and obtain the rate of growth of the test loss with respect to the sample size.

To summarize our numerical observations, for the first experiment, despite numerical errors and the fact that we only use the leading term in Theorem 3, we observe that the rate with respect to the input dimension suggested by Eq. (2.17) regarding the input dimension is close to optimal when using the explicit regularization. For the second experiment however, the rates with respect to the number of samples obtained in numerical experiments are around $\mathcal{O}(N_{\text{sample}}^{-0.8})$ without explicit regularization, and $\mathcal{O}(N_{\text{sample}}^{-1.0})$ with explicit regularization. Hence the convergence rate with respect to the number of samples is faster than the theoretical predicted worst case rate as $\mathcal{O}(N_{\text{sample}}^{-0.5})$.

Loss vs. input dimension without explicit regularization
Here we fix the sample size to 10^5. For each d ranging from 4 to 60, we repeat the training procedure four times with the same set of hyperparameters (the default values

[3]Given that the training loss is usually one or more magnitude smaller than the test loss.

have been discussed in section 2.4). In Fig. 2.5(a), we display the training loss and test loss with respect to the input dimension in log-log scale. The losses are computed using the mean squared error in the original scale as in Eq. (2.21). The slope in Fig. 2.5(a) indicates that the generalization error of the approximation to the original target function $\widetilde{f}^*(\mathbf{x})$ as in Eq. (2.1) grows as $d^{3.6}$, and thus the generalization error of the approximation to the scaled $f^*(\mathbf{x})$ as in Eq. (2.2) grows as $d^{1.6}$. The empirical rate $d^{1.6}$ is a lot larger than logarithmic growth predicted in the bound in Eq. (2.17).

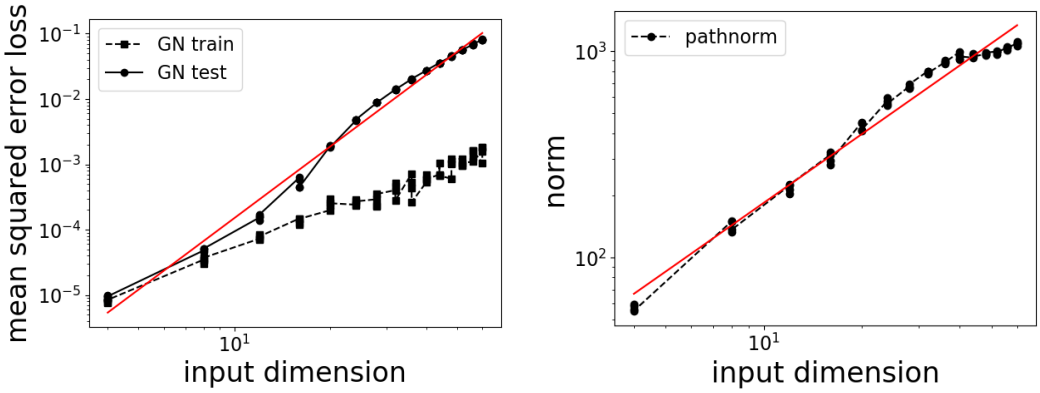
In particular, Eq. (2.17) indicates that in order to bound the generalization error, the path norm needs to be $\sim \mathcal{O}(1)$. However, the boundedness of the path norm is only proven in the context of explicit regularization. Without such an explicit regularization term (also called the "implicit regularization"), there is no *a priori* guarantee that the path norm can remain bounded as the input dimension increases.

We remark that to compute the path norm for our three-layer network, we need to modify the formula for two-layer network. Let us denote the weight matrices of the three layers to be $\mathbf{w}^1, \mathbf{w}^2, \mathbf{w}^3$. With the width of hidden layer for the global network being $d \cdot \alpha$ as shown in section 2.4, the size of the weight matrices are $d \cdot \alpha \times d, d \cdot \alpha \times d \cdot \alpha, 1 \times d \cdot \alpha$ respectively. In the three-layer case, the path norm can be calculated using the formula:

$$\|\theta\|_{\mathcal{P}} := \frac{1}{d} \sum_{i=1}^{d} \sum_{j,k=1}^{d\alpha} |\mathbf{w}_{ij}^1||\mathbf{w}_{jk}^2||\mathbf{w}_k^3| \tag{2.22}$$

The $\frac{1}{d}$ factor is due to the last layer in Alg. **??**, which divides the output by d. Using the updated formula we plot the path norm with respect to the input dimension in Fig. 2.5(b). From the figure we can observe that the path norm grows as $\sim d^{1.1}$, thus violating the $\mathcal{O}(1)$ assumption for Eq. (2.17). Notice that in this scenario, although Eq. (2.17) does not apply, the bound in Theorem 3 provides a fairly good estimate of the growth of the generalization error. The leading term in the bound given by Theorem 3 grows as $\|\theta\|_{\mathcal{P}}\sqrt{\ln(d)}$. With the path norm $\sim d^{1.1}$, and $\sqrt{\ln(d)}$ empirically behaving like a fractional power of d for small d, the product has a rate similar to the rate of the test loss observed, which is $\sim d^{3.6}$ (after taking into account the d^2 scaling factor).

Loss vs. input dimension with explicit regularization
To demonstrate that explicit regularization indeed reduces the path norm, test loss and generalization error, we implement three types of regularization schemes, and study the growth of the errors with respect to the input dimension. Let the number of



(a) the relationship between the generalization error and the input dimension. Slope of the best fitted line (red) is 3.638

(b) the relationship between the path norm and the input dimension. Slope of the best fitted line (red) is 1.107

Figure 2.5: Experiments without regularization

trainable parameters in the neural network be N_{par}. Denote the trainable parameters by $\{\theta_i\}_{i=1}^{N_{\text{par}}}$ and regularization constant by λ. We can summarize the implementation of the three regularization as

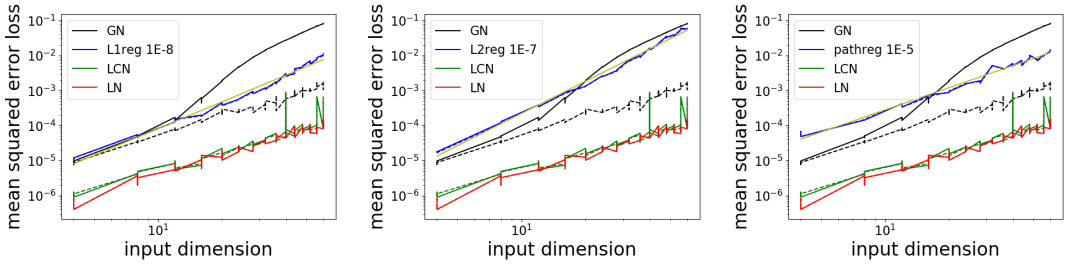
- L1 regularization, minimizing MSE+$\lambda \sum_{i=1}^{N_{\text{par}}} |\theta_i|$
- L2 regularization, minimizing MSE+$\lambda \sum_{i=1}^{N_{\text{par}}} \theta_i^2$
- path norm regularization, minimizing MSE+$\lambda \|\theta\|_{\mathcal{P}}$, where $\|\theta\|_{\mathcal{P}}$ is calculated by Eq. (2.22)

The MSE is computed as in Eq. (2.20). To implement the three regularization schemes, we use the kernel regularizer in Keras in tensorflow 1.7.0 for the L1 and the L2 regularization (the regularization is only on the weight matrices, but not on the bias), and we use tensorflow 2.0 to implement the path norm regularization. We choose the regularization constant λ (fixed with respect to d) empirically so that the global network achieves the best test loss. The regularization constants we select in Fig. 2.6 are 10^{-8} for L1 regularization, 10^{-7} for the L2 regularization, and 10^{-5} for the path norm regularization.

In addition to the test/train loss vs. the input dimension for the regularization experiments, we also display our previous results for GN, LN, and LCN as a reference

in Fig. 2.6

Recall that the growth rate of the test loss for the global network without regularization (shown in Fig. 2.5(a)) is around 3.638, all three regularization helps reduce down the rate in Fig. 2.6. Path norm regularization in Fig. 2.6(c) exhibits the best growth rate, despite that the test loss for small d is slightly larger. We plot the path norm vs. the input dimension in Fig. 2.7 and we indeed observe that the path norm is $\mathcal{O}(1)$ as desired. As a matter of fact, the path norm even decreases slightly as d increases, and this is qualitatively different from the behavior of implicit regularization. In addition, with the explicit regularization and thus bounded path norm, Eq. (2.17) gives a tight estimate of the rate of growth.



(a) Loss vs. input dimension for GN with L1 regularization. Slope of the best fitted line (yellow): 2.527

(b) Loss vs. input dimension for GN with L2 regularization. Slope of the best fitted line (yellow): 3.051

(c) Loss vs. input dimension for GN with path norm regularization. Slope of the best fitted line (yellow): 2.098

Figure 2.6: Experiments with regularization, solid lines are test loss and dashed lines are training loss.

Since the weight matrices for the LN embedded GN are block diagonal, and hence are sparse matrices. We will look at the first weight matrices of the models under the following scenario:

- GN without regularization,
- GN with L1 regularization,
- GN with L2 regularization,
- GN with path norm regularization, and
- GN with optimal LN weights embedded.

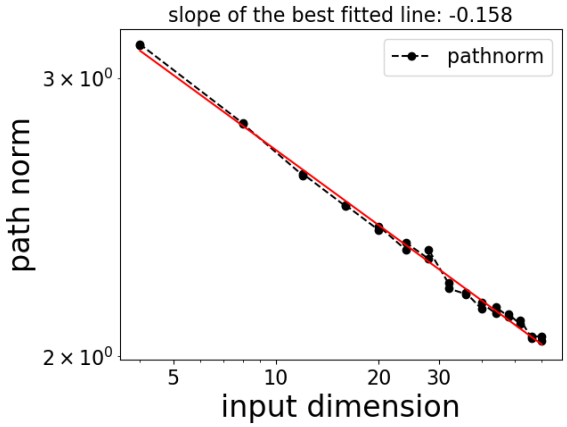


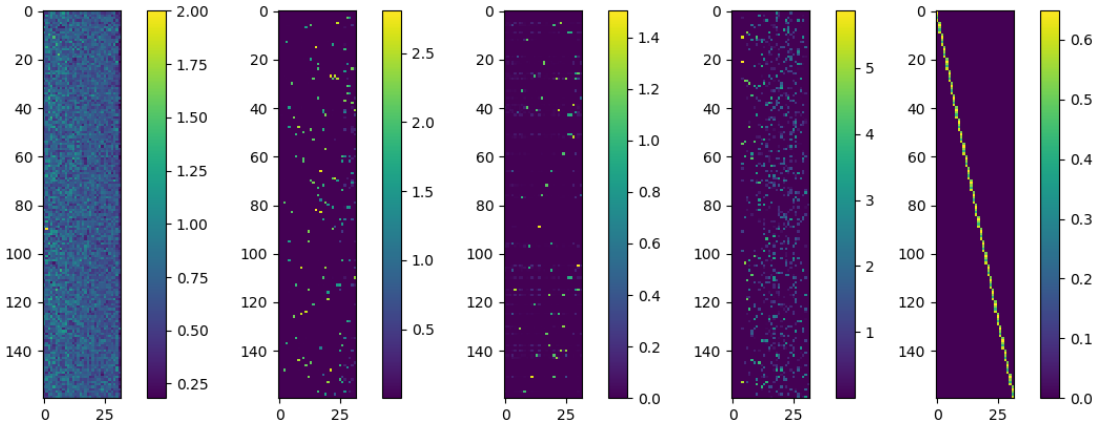
Figure 2.7: the relationship between the path norm and the input dimension for the path norm regularization experiment with $\lambda = 10^{-5}$. Slope of the best fitted line (red): -0.158. This trend shows that the path norm almost stays constant when input dimension increases and displays how powerful the explicit regularization is.

In Fig. 2.8, we display the first weight matrices for input dimension $d = 32$ for the five trained models. Since the weight matrix is from the input nodes to the first hidden layer, the dimension is 1600×32, where 1600 is obtained from input dimension (32) multiplied by number of nodes per input node (50). To improve visibility, we display the maximum absolute value among the 10 adjacent cells so that the first dimension is reduced by a factor of 10. We can see from the picture that L1 regularization, L2 regularization, and path norm regularization all lead to a much sparser weight matrix compared with the global network without regularization.

Loss vs. sample size without explicit regularization
For the experiment regarding the rate of growth of the test loss with respect to the sample size, we fix $d = 40$ (the choice is arbitrary), and we vary the sample size from 10^3 to 10^6. Since the sample size varies, we consider the following two choices of the batch size:

1. fix the ratio and let batch size be (number of training and validation samples) / 100, this choice ensures same number of iterations per epoch as the sample size increases.

2. fix the batch size to 80 as sample size varies. The number of iterations increases as the sample size increases.

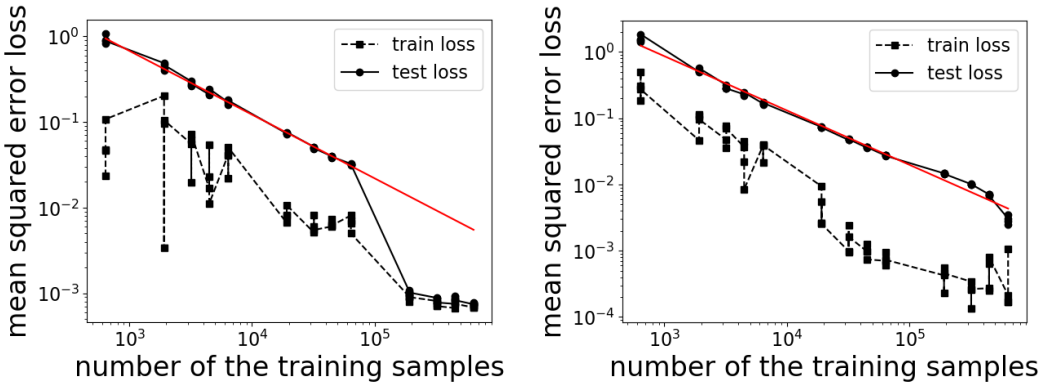


(a) GN w/o reg (b) GN L1 reg (c) GN L2 reg (d) GN path norm reg (e) loaded LN

Figure 2.8: The sparsity pattern of the weight matrices between the first and second layers after training.

The other hyperparameters are the same as the setup in section 2.4. We run 4 training procedures for each number of samples and we report the resulting test loss (early stopping still applies) versus the number of samples, which are summarized in Fig. 2.9. In Fig. 2.9(a), with batch size fixed at 80, the performance changes when the number of training samples exceed 10^5. Since we focus on the generalization error in this chapter (so we need the existence of generalization gap), and also the time cost of a training procedure drastically increases when the small batch size is applied to a large sample size (number of iterations per epoch, computed by training sample size / batch size, is large), we focus on the segment where the number of training samples is below 10^5 and the generalization gap is visible. Following Theorem 3, the generalization error decreases asymptotically as $\mathcal{O}(N_{\text{sample}}^{-0.5})$, but numerically we observe different exponents in both figures, which is around -0.8. The fixed batch size experiment in Fig. 2.9(a) returns a similar rate to the fixed ratio experiment in Fig. 2.9(b), therefore we may keep using the default setup (batch size with fixed ratio) for the rest of the studies in this subsection.

To confirm that the rate is consistent for other input dimension values, we repeat the same procedure (with batch size obtained by the fixed ratio) for $d = 60$. The trend and rate are reported in Fig. 2.10, and indeed the rate is still around -0.8, different from the theoretical rate -0.5.



(a) Loss vs. sample size for GN with fixed batch size. Slope of the best fitted line for test loss with an obvious generalization gap (red): -0.745

(b) Loss vs. sample size for GN with batch size given by a fixed ratio. Slope of the best fitted line (red) is -0.820.

Figure 2.9: the relationship between the test loss and the sample size with two choices of batch size.

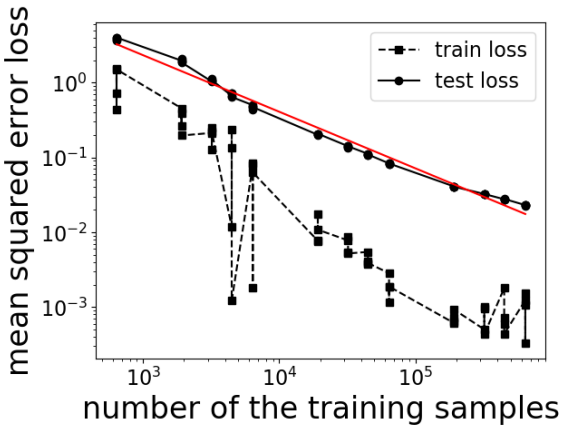
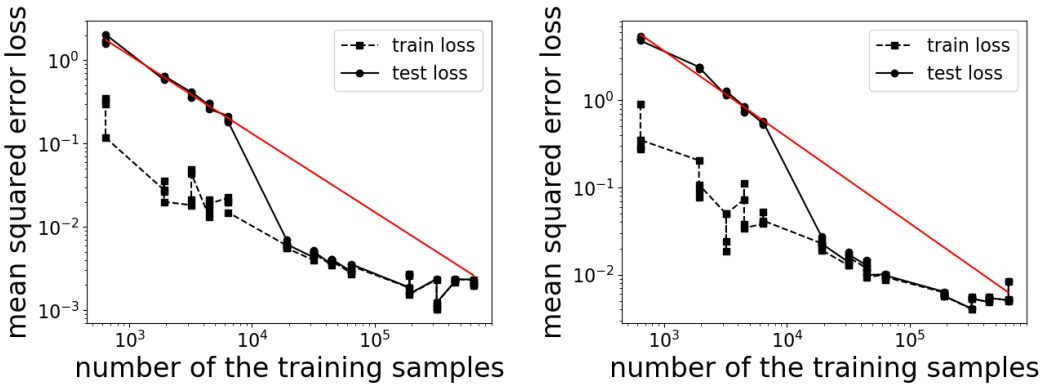


Figure 2.10: Loss vs. sample size for GN for $d = 60$. Slope of the best fitted line (red) is -0.756

Loss vs. sample size with explicit regularization
We test the impact of explicit regularization on the rate of growth of the test loss with respect to the sample size by adding L1 regularization to all the weight matrices. The setup is the same as the global network with L1 regularization experiment displayed in Fig. 2.6(a), except that instead of fixing $N_{\text{sample}} = 10^5$ and letting d varies, we fix $d = 40$, $d = 60$ and let N_{sample} vary from 10^3 to 10^6. We fix the regularization constant at 10^{-8} as in Fig. 2.6(a). The rates of the segment where there is a visible generalization gap are shown in Fig. 2.11. We observe that as N_{sample} increases to certain point (around 10^4), the fixed regularization constant makes the regularization term more significant in the loss, resulting in nearly vanishing generalization gap. Since our focus is on generalization error inspired by the theory in section 2.2 and 2.3, we report the rate in the segment with visible generalization gap. Notice that the rate in the L1 regularization is around -1.0, and hence the decay of the test loss with respect to the number of samples is faster than that without explicit regularization, which is around -0.8.



(a) Loss vs. sample size for GN with L1 regularization for $d = 40$. Slope of the best fitted line for test loss with an obvious generalization gap (red): -0.943

(b) Loss vs. sample size for GN with L1 regularization for $d = 60$. Slope of the best fitted line for test loss with an obvious generalization gap (red): -0.986

Figure 2.11: the relationship between the test loss and the sample size for the global network with L1 regularization.

Empirical sample complexity with and without explicit regularization
Eq. (2.18) gives a theoretical prediction of the scaling of growth of number of samples with respect to the input dimension in order to maintain a predetermined level of test

loss. We find this relation hard to verify directly because: 1) The input dimension, and the number of samples are usually chosen among finitely many values, so the grid may not be fine enough to find the (n, d) combination that gives the predetermined test loss; 2) The test loss varies even with same hyperparamter setting, so it is rather difficult to make sure that the test loss stays at a constant value especially given that the loss of the scaled function can be very small ($10^{-6} \sim 10^{-4}$). However, we can compute the numerical sample complexity if we assume that the generalization gap is in the function form $\frac{d^{\beta_1}}{n^{\beta_2}}$ as in Eq.(2.17). Recall that theoretical prediction gives $\beta_1 = 2$ because the loss for the original function needs to be scaled by d^2 as noted in Eq.(2.20), and $\beta_2 = 0.5$. Then the sample complexity with respect to d can be characterized by a rate γ as $n \sim \mathcal{O}(d^\gamma)$, where $\gamma = \frac{\beta_1}{\beta_2} = 4$.

The rate we obtained for the implicit regularization is $\beta_1 = 3.6$ as in Fig. 2.5(a), and $\beta_2 = 0.8$ as in Fig. 2.9(b), and thus the sample complexity rate is approximately $\gamma = 4.5$, i.e. $n \sim \mathcal{O}(d^{4.5})$. This is close to the theoretically predicted rate, but this largely benefited from the observation that $\beta_2 = 0.8 > 0.5$. The rate we obtained for the explicit regularization (L1 with regularization constant 10^{-8} to be precise) is $\beta_1 = 2.5$ as in Fig. 2.6(a), and $\beta_2 = 1.0$ as in Fig. 2.11, and thus the sample complexity rate is approximately $\gamma = 2.5$, i.e. $n \sim \mathcal{O}(d^{2.5})$, which is significantly better than the theoretically predicted rate. We do point out that this is a very rough estimate because of the assumed function form and the fact that the β_1 rate in the L1 regularization in Fig. 2.6(a) is not based on the generalization gap but the test loss (the generalization gap is very small). Overall, the explicit regularization helps improve the rate in both the test loss vs. the input dimension, and the test loss vs. the number of training samples, so the advantage of the explicit regularization is convincing.

2.5 Conclusion

Despite the fact that an overparameterized neural network architecture can represent a large class of functions, such representation power can come at the cost of a large sample complexity. This is particularly relevant in many scientific machine learning applications, as the required accuracy (in the form of a regression problem) is high, and the training data can be difficult to obtain. Therefore a number of recent works have focused on domain-specific neural network architectures aiming at reducing the number of parameters, "retreating" from the overparameterized regime.

This chapter gives an unambiguous, and minimal working example illustrating why

this makes sense. Even for the seemingly simple task of computing the sum of squares of d numbers in a compact domain (i.e. learning the square of a 2-norm), a general purpose dense neural network struggles to find a highly accurate approximation of the function even for relatively low d (tens to hundreds). In particular, the sample complexity of an empirically optimally tuned and explicitly regularized dense network is $\mathcal{O}(d^{2.5})$. This behaves better than the sample complexity from *a priori* error bound, which is $\mathcal{O}(d^4)$. The origin of such improvement deserves study in the future. The sample complexity of an empirically optimally tuned and implicitly regularized dense network is close to $\mathcal{O}(d^{4.5})$, and hence can be prohibitively expensive as d becomes large.

When we choose a proper architecture, such as the local network or the locally connected network, the generalization error still grows with respect to d. However, we find that the generalization gap is nearly invisible with explicit or implicit regularization, and the test loss is orders of magnitude smaller than that of the global (dense) network. Given that the sample complexity can asymptotically scale as the square of the test loss (assuming training error is negligible), the practical savings due to the use of a local network is vital to the success of the neural network.

From a theoretical perspective, we remark that existing error analysis based on the Rademacher complexity-type cannot yet explain why the prefactor of the local network should be lower by orders of magnitude compared to the global (dense) network. Our results illustrate that implicit regularization as early stopping may not give the optimal generalization error rate in practice, although in theory it can be shown to achieve the same optimal rates as L^2 regularization in reproducing kernel Hilbert spaces [96, 94]. Given that implicit regularization is still a prevailing regularization method used in practical applications, better theoretical understanding of the behavior of early stopping regularization in neural networks, and methods to improve the performance of such an implicit regularization in practice are also needed.

Chapter 3

Deep Density

3.1 Introduction

Kohn-Sham density function theory (KS-DFT) [56] is the most widely-used electronic structure theory because the electron density completely determines the ground state [47] and the thermal [72] properties of a quantum many-body system. The goal of KS-DFT is to obtain a mapping of the atomic configuration to the electron density, denoted by $\varrho(\mathbf{r}, \{\mathbf{R}_I\}_{I=1}^{N_a})$. Here $\mathbf{R}_I$ is the position of the I-th nuclei, $\mathbf{r}$ is the electronic position, and N_a is the number of atoms. Notwithstanding its enormous success, KS-DFT still suffers from two significant challenges. The first challenge is its computational cost; the cost of KS-DFT calculations typically scales cubically with respect to the system size, and hence the calculations can be expensive for large systems. Despite the availability and development of linear scaling methods [35, 14], they are only applicable to treating insulating systems with relatively large energy gaps. The second, and more fundamental, challenge is its accuracy, which cannot be systematically improved due to the limitation of the available exchange-correlation functionals [70].

The recent surge in applications of machine learning methods to scientific computing problems provides an alternative route for revisiting both problems. If one can find a more effective mapping for ϱ using, e.g., a neural network, we can bypass the solution of the Kohn-Sham equations, and directly obtain the self-consistent electron density for a given atomic configuration. This may drastically reduce the computational time, particularly for large systems. We may further use such a network to encode the electron density obtained from theories that are more accurate than standard KS-DFT, such as the density matrix renormalization group [95] or the coupled-cluster theory [8]. Based on such considerations, the representation of the electron density

has received much attention in the past few years [15, 39, 83, 13, 29, 2, 16]. These approaches are typically based on carefully hand-crafted descriptors that encode the atomic configuration, a projection of ϱ onto a basis, and a machine learning algorithm mapping the descriptors to the coefficients of the projection. For example, in [15] the authors adopted a descriptor using a fictitious smooth potential centered at each nuclei, which is then mapped to a basis for the density (either a Fourier basis or a Kernel PCA (KPCA) basis [89]) using a ridge-kernel regression algorithm (a more detailed exposition can be found in [13]). This approach is then extended in [38] to improve its transferability by using the Gaussian process regression (GPR) framework introduced in [39]. In a nutshell, in [38] the authors replace the global representation of the density by a localized basis around each nuclei, and they use a regularized ridge-regression algorithm. This approach was further improved in [29] by developing better basis to represent the electron density, which allow them to further increase the transferability of their approach.

Another approach is to treat the problem as an image processing task. In [83] authors recast the map from the external potential to the electron density, as an end-to-end maps between two images, which is learnt using deep convolutional inverse graphics networks (DCIGNs) [61].

An approach related to the one presented in this chapter can be found in [2] where authors use a *electron-centered* approach to compute the electron density. To compute the value of the electron density at a given point in space, the algorithm computes a list of the closest atoms, and uses theirs positions and species as features, which are then fed to either an off-the-shelf multilayered fully-connected network, or a random forest. This approach is also explored in [16], where the authors build rotationally invariant tensor and vector fingerprints, which are averaged using Gaussian weights, and then fed into a multilayered fully-connected network.

The problem of using machine learning methods to represent the electron density is closely related to the problem of finding the interatomic potential and corresponding force field for molecular dynamics simulation [10, 9, 82, 75, 18, 90, 91, 42]. In particular, the recently developed Deep Potential scheme [103, 105] has been successful in describing with high fidelity various finite-size and extended systems, including organic molecules, metals, semiconductors, and insulators.

In this chapter, we leverage the construction of the Deep Potential to build a neural network representation for ϱ. In contrast with aforementioned related approaches, we learn the descriptor on the fly, and instead of learning the mapping to the coefficients of a basis, we evaluate the total electron density directly in a point-wise manner. The total electron density is decomposed into a linear combination of N_a components, with the I-th component describing the contribution to the electron density from the I-th atom and its neighbors. Each component is constructed to locally satisfy the

necessary translation, rotation, and permutation symmetries. The number of atoms involved in each component does not scale with respect to the global system size, and hence the cost for evaluating ϱ scales linearly with respect to the system size.

By targeting the self-consistent electron density, we demonstrate that Deep Density can take advantage of the screening effect to effectively represent ϱ for both insulating and metallic systems, and thus can bypass the solution of the nonlinear Kohn-Sham equations. The resulting algorithm is not tied to a given discretization scheme nor a particular choice of basis, and it can be transferred to systems of larger sizes. The algorithm can also be implemented in an embarrassingly parallel fashion: one can evaluate different points of the density given by different configuration completely independently. The total cost of evaluating the density scales linearly with the number of atoms in the system.

We demonstrate that Deep Density can accurately predict the electron density and exhibits excellent transferability properties for one-dimensional model systems of insulating, metallic, as well as mixed insulating-metallic characters. We also report the performance of our model for three-dimensional real systems, including small molecules (C_2H_6, C_4H_{10}), as well as condensed matter systems, including water (up to 512 water molecules), and aluminum (up to 256 aluminum atoms).

Organization

The rest of this chapter is organized as follows. In Section 3.2 we provide the framework of KS-DFT, and the map we seek to approximate. In Section 3.3 we provide the architecture for the neural network. In particular, we provide a succinct review of the techniques in [102] and we explain how to implement the physical ansatz and the symmetry requirements. In Section 3.4 we provide the numerical examples showcasing the accuracy and transferability of the algorithm. In particular, we provide the electron density for 1D models (more details in Section 3.5) and realistic 3D systems (more details in Section 3.6) and we compare them against the electron density produced by classical KS-DFT computations. We show that it is possible to train these models to single precision in 1D and within three digits in 3D, and that the accuracy is maintained even for test systems an order of magnitude larger. In Section 3.7 we provide several comments about the current work and we point to several future directions of research. Additional details for the numerical experiments are given in the appendices.

3.2 Preliminaries

For a system with N_e electrons at a given atomic configuration $\{\mathbf{R}_I\}_{I=1}^{N_a}$ in a d-dimensional space (i.e. $\mathbf{r}, \mathbf{R}_I \in \mathbb{R}^d$), KS-DFT solves the following nonlinear eigenvalue problem (spin omitted for simplicity)

$$H[\rho; \{\mathbf{R}_I\}]\psi_i = \varepsilon_i \psi_i, \quad i = 1, ..., N_e, \tag{3.1}$$

$$\int \psi_i^*(\mathbf{r})\psi_j(\mathbf{r})\,\mathrm{d}\mathbf{r} = \delta_{ij}, \quad \rho(\mathbf{r}) = \sum_{i=1}^{N_e} |\psi_i(\mathbf{r})|^2 . \tag{3.2}$$

The Kohn-Sham Hamiltonian is

$$H[\rho; \{\mathbf{R}_I\}] := -\frac{1}{2}\Delta_{\mathbf{r}} + V_{\text{ion}}(\mathbf{r}; \{\mathbf{R}_I\}) + V_{\text{hxc}}[\mathbf{r}; \rho]. \tag{3.3}$$

Here V_{ion} characterizes the interaction between electrons and nuclei, and it does not depend on the electron density. V_{hxc} is called the Hartree-exchange-correlation potential, which includes the mean-field electron-electron interaction, as well as the contribution from the exchange-correlation energy. The eigenvalues $\{\varepsilon_i\}_{i=1}^{N_e}$ are real and ordered non-decreasingly, so $\{\psi_i\}_{i=1}^{N_e}$ correspond to the eigenfunctions with lowest N_e eigenvalues. Due to the ρ-dependence of V_{hxc}, the Kohn-Sham equations needs to be solved self-consistently till convergence. We refer to [70, 65] for more details of numerical solutions of KS-DFT. For the purpose of this paper, we are interested in learning the mapping

$$\varrho : \{\mathbf{r}\} \cup \{\mathbf{R}_I\}_{I=1}^{N_a} \mapsto \rho(\mathbf{r}), \tag{3.4}$$

which is also denoted as $\varrho(\mathbf{r}, \{\mathbf{R}_I\}_{I=1}^{N_a}) = \rho(\mathbf{r})$.

In KS-DFT, we need to distinguish between the external potential $V_{\text{ion}}(\mathbf{r}; \{\mathbf{R}_I\})$, and the effective potential

$$V_{\text{eff}}(\mathbf{r}) = V_{\text{ion}}(\mathbf{r}; \{\mathbf{R}_I\}) + V_{\text{hxc}}[\mathbf{r}; \rho]. \tag{3.5}$$

We refer to the mapping from V_{eff} to ρ as the linearized Kohn-Sham map. The evaluation of the Kohn-Sham equations requires (partially) diagonalizing the Hamiltonian $H_{\text{eff}} := -\frac{1}{2}\Delta_{\mathbf{r}} + V_{\text{eff}}(\mathbf{r})$, after proper discretization. Correspondingly we refer to the mapping from V_{ion} to ρ as the self-consistent Kohn-Sham map, of which the evaluation requires solving the Kohn-Sham equations self-consistently.

For insulating systems with a positive energy gap (i.e. $\varepsilon_{N_e+1} - \varepsilon_{N_e} > 0$), it is known that the dependence of electron density at position $\mathbf{r}$ with respect to the potential at position $\mathbf{r}'$ decays exponentially with respect to $|\mathbf{r} - \mathbf{r}'|$. This is often

referred to as the "nearsightedness" principle of electrons [55, 78]. More specifically, the Fréchet derivative

$$\chi_0(\mathbf{r}, \mathbf{r}') = \frac{\delta\rho(\mathbf{r})}{\delta V_{\text{eff}}(\mathbf{r}')}, \tag{3.6}$$

which is also called the irreducible polarizability operator in physics literature, satisfies the decay property $\chi_0(\mathbf{r}, \mathbf{r}') \sim e^{-c|\mathbf{r}-\mathbf{r}'|}$ for some constant $c > 0$. The nearsightedness principle allows the design of linear scaling methods [35, 14], which effectively truncate the global domain into many small domains to solve KS-DFT.

For metallic systems with a zero or very small energy gap, $\chi_0(\mathbf{r}, \mathbf{r}')$ only decays algebraically as $|\mathbf{r} - \mathbf{r}'| \to \infty$. Hence the nearsightedness principle does not hold anymore, and linear scaling methods become either very expensive (with a large truncation radius) or inaccurate (with a small truncation radius). On the other hand, even for metallic systems, the reducible polarizability operator, defined as

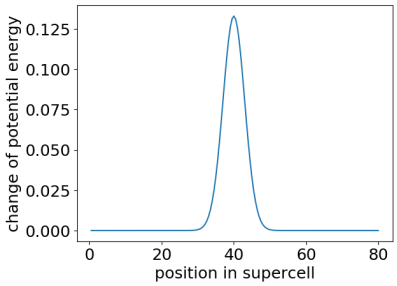
$$\chi(\mathbf{r}, \mathbf{r}') = \frac{\delta\rho(\mathbf{r})}{\delta V_{\text{ion}}(\mathbf{r}')}, \tag{3.7}$$

can be much more localized compared to χ_0. This is known as the screening effect [28, 36].

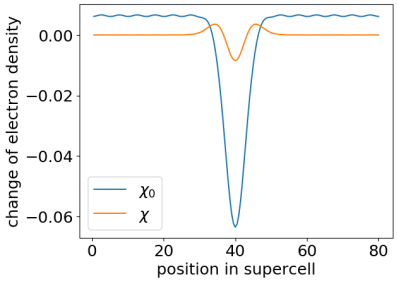
To illustrate the screening effect, we consider a one-dimensional periodic metallic system with 8 atoms. The details of the setup will be discussed in Section 3.4 and Section 3.5. We introduce a localized and exponentially decaying perturbation of the potential δV_{ion} as

$$\delta V_{\text{ion}}(r) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{1}{2\sigma_0^2}(r - \mu_0)^2\right), \tag{3.8}$$

where we choose $\sigma_0 = 3.0$, and $\mu_0 = 40$ is the center of the supercell. Fig. 3.1(a) displays the profile of δV_{ion}. Fig. 3.1(b) shows that for a metallic system, the induced electron density obtained from the linearized Kohn-Sham map, which can be approximately computed as $\chi_0 \delta V_{\text{ion}}$, has a large magnitude, and a delocalized and oscillatory tail. On the other hand, due to the screening effect, the magnitude of the induced electron density obtained from the self-consistent Kohn-Sham map, which can be approximately computed as $\chi \delta V_{\text{ion}}$, has a much smaller magnitude. Its tail is much shorter and smoother, and the support of $\chi \delta V_{\text{ion}}$ is localized around that of δV_{ion}. The screening effect indicates that Deep Density should be able to leverage such an additional level of localization which is not present in standard linear scaling solvers, and a relatively small truncation radius may be possible even for metallic systems (see the architecture in Section 3.3).



(a) Perturbation δV_{ion}



(b) Comparison of $\chi\delta V_{\text{ion}}$ and $\chi_0\delta V_{\text{ion}}$

Figure 3.1: Illustration of the screening effect for a one-dimensional model metallic system.

3.3 Network architecture

Locality

Standard methods for solving KS-DFT can only be used to evaluate the linearized Kohn-Sham map, which is then used to obtain the self-consistent electron density via self-consistent field iterations. In contrast, our goal is to *directly* learn the self-consistent electron density, or the mapping $\varrho(\mathbf{r}, \{\mathbf{R}_I\}_{I=1}^{N_a})$, which already takes into account the screening effect. This allows us to construct neural networks that can be decomposed into the linear combination of local components for both insulating and metallic systems.

As illustrated in Fig. 3.2, we partition the electron density as

$$\varrho(\mathbf{r}, \{\mathbf{R}_I\}_{I=1}^{N_a}) = \sum_{I=1}^{N_a} \varrho^I(\mathbf{r}, \mathcal{R}^I). \tag{3.9}$$

Here ϱ^I characterizes the contribution to the electron density at $\mathbf{r}$ from the neighborhood of the I-th atom. The locality is imposed by building an interaction list $\mathring{\mathcal{I}}_{R_c}(I)$, defined as the set of indices J such that $|\mathbf{R}_J - \mathbf{R}_I| < R_c$. Following this notation, $\mathcal{R}^I$ denotes the set of atomic positions

$$\mathcal{R}^I = \{\mathbf{R}_J, J \in \mathring{\mathcal{I}}_{R_c}(I)\}.$$

We denote by $s(I)$ the species index (i.e. the atomic number) of the I-th atom. We may also treat the electron as a special "atom" equipped with index $J = 0$. For

simplicity we define $s(0) = 0, \mathbf{R}_0 = \mathbf{r}$, and then define the extended interaction list as the index set

$$\mathcal{I}_{R_c}(I) = \{0\} \cup \mathring{\mathcal{I}}_{R_c}(I). \tag{3.10}$$

In other words, the electron (formally) belongs to $\mathcal{I}_{R_c}(I)$ for every atom I.

The density ϱ^I can be constructed as

$$\varrho^I(\mathbf{r}, \mathcal{R}^I) = \mathcal{N}^{\mathtt{lin},I}(\mathbf{r}, \mathcal{R}^I) e^{C_{s(I)}\left(|\mathbf{r}-\mathbf{R}_I| - D_{s(I)}\right)^2 + \mathcal{E}^I(\mathbf{r}, \mathcal{R}_I)}. \tag{3.11}$$

We assume $\mathcal{N}^{\mathtt{lin},I}$ takes the form

$$\mathcal{N}^{\mathtt{lin},I}_{s(I)}(\mathbf{r}, \mathcal{R}^I) = \mathcal{N}^I(\mathbf{r}, \mathcal{R}_I) + A_{s(I)} |\mathbf{r} - \mathbf{R}_I| + B_{s(I)}. \tag{3.12}$$

The constants $A_{s(I)}, B_{s(I)}, C_{s(I)}, D_{s(I)}$ are trainable parameters and only depend on the species of the I-th atom.

The ansatz is built such that the electron density decays exponentially with the distance of the electron to the center of the cluster, i.e., the I-th atom. The rate of decay is given by $C_{s(I)}$ (which is constrained to be negative), and $D_{s(I)}$ accounts for the possibility that the bulk of the electron density is not centered at the atom I. In addition, due to possible issues with the pseudopotential in the KS-DFT computation, the electron density can be very small in the neighborhood of the nuclei. In order to fit this the nonlinear correction, $\mathcal{E}^I(\mathbf{r}, \mathcal{R}_I)$ would need to have a large negative value. Thus, in order to bypass this issue we multiply the exponential with a linear term to capture this behavior. Moreover, to capture abrupt changes on the density we add $\mathcal{N}^I$, a non-linear correction, to this term.

In a nutshell, the exponential term mimics the behavior of the local density far from the center of the cluster, whereas $\mathcal{N}^{\mathtt{lin},I}$ captures the behavior close to center of atom I. $\mathcal{N}^I, \mathcal{E}^I$ are neural networks to be introduced later. Clearly we may absorb the constants $A_{s(I)}, B_{s(I)}, C_{s(I)}, D_{s(I)}$ into the neural networks as well. In practice, we found that separating out such terms can reduce the training time and the test error of the network. In addition, numerically we observed that the derivative associated to the constant may widely differ from the rest of the neural network, introducing them explicitly allows us to re-scale the gradient if necessary, thus accelerating the optimization.

Symmetry

When one considers modeling the physics at different scales, it is often crucial to preserve symmetry properties. Additionally, from a learning perspective, symmetries can also significantly improve the efficiency of training in terms of the required number

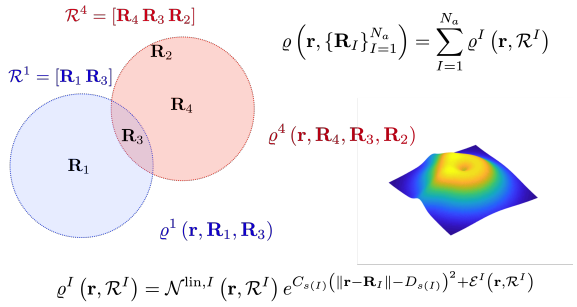


Figure 3.2: Illustration of the local interaction lists and local electron densities.

of parameters, the number of training steps, and the amount of training data. In this regard, as a mapping from electronic and atomic positions to the value of the electron density, $\varrho(\mathbf{r}, \{\mathbf{R}_I\}_{I=1}^{N_a})$ should be invariant under permutation of indices of identical atoms, and under a collective translation or rotation of the electron and atomic positions. As mentioned in the introduction, the problem can also be considered as finding a map from atomic positions to a electron density field represented by electron positions or other basis sets. One disadvantage is that the dimension of the output may be large: the output consists of values of the density on a list of grid points, or the coefficients corresponding to a basis set. When the output is a list of grid points, it can also be difficult to satisfy symmetry requirements. Here we follow the first approach, and we find it is much easier to define a neural network representation with a single output. Finally, to combine the locality and the symmetry requirements, we guarantee that the values of $\mathcal{N}^I$ and $\mathcal{E}^I$ in Deep Density are invariant with respect to the following symmetry operations:

1. Permutation: relabeling of indices of the same atom species in the index set $\mathcal{I}_{R_c}(I)$.

2. Translation: uniformly shifting $\mathbf{R}_I$ and the positions of all particles in $\mathcal{I}_{R_c}(I)$ by a vector.

3. Rotation: rotating all particles in $\mathcal{I}_{R_c}(I)$ around $\mathbf{R}_I$.

In order to reduce the number of parameters, all neural networks involved should share the same set of parameters if the species of the particles involved are the same.

We now illustrate the treatment of $\mathcal{N}^I$; the procedure for $\mathcal{E}^I$ is analogous. In order to satisfy permutation symmetry, we adopt a variant of the representation in [98],

which states that any permutation invariant function $f : \mathbb{R}^N \to \mathbb{R}$ can be represented in the form

$$f(\mathbf{x}) = f(x_1, \ldots, x_N) = \mathcal{F}\left(\sum_{i=1}^{N} h(x_i)\right), \tag{3.13}$$

where $h : \mathbb{R} \to \mathbb{R}^M$ maps each coordinate x_i into an M-dimensional feature space, and $\mathcal{F} : \mathbb{R}^M \to \mathbb{R}$ combines the features to obtain the value f. Both $\mathcal{F}$ and h can are represented by neural networks in what follows.

In this work, we follow the construction in [105], and decompose $\mathcal{N}^I$ as

$$\mathcal{N}^I(\mathbf{r}, \mathcal{R}^I) = \mathcal{F}_{s(I)} \circ \mathcal{D}^I(\mathbf{r}, \mathcal{R}^I). \tag{3.14}$$

Here $\mathcal{F}_{s(I)} : \mathbb{R}^{M\times M} \to \mathbb{R}$ is called a fitting network that only depends on the species index $s(I)$. $\mathcal{D}^I(\mathbf{r}, \mathcal{R}^I)$ is called a descriptor network, and $\mathcal{D}^I(\mathbf{r}, \mathcal{R}^I) \in \mathbb{R}^{M\times M}$ is a matrix of the following form

$$\mathcal{D}^I(\mathbf{r}, \mathcal{R}^I) = \left(\sum_{J\in\mathcal{I}_{R_c}(I)} h_{s(I),s(J)}(\mathbf{R}_J - \mathbf{R}_I)\right)^{\top} \left(\sum_{J\in\mathcal{I}_{R_c}(I)} h_{s(I),s(J)}(\mathbf{R}_J - \mathbf{R}_I)\right). \tag{3.15}$$

Here $h_{s(I),s(J)} : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}\times M}$ is a feature mapping, which only depends on the species of the particle I and J (recall that the electron is labeled with $J = 0$ with species index 0). Then $\mathcal{D}^I \in \mathbb{R}^{M\times M}$ is a symmetric matrix that naturally satisfies the permutation and translation symmetries. As will be demonstrated below, it satisfies the rotation symmetry as well.

Now we demonstrate the construction of the mapping $h_{s(I),s(J)}$. Define $R_{JI} = |\mathbf{R}_J - \mathbf{R}_I|$, and we would like to require h to depend smoothly on $\mathbf{R}_J$ moves in and out of the index set $\mathcal{I}_{R_c}(I)$. In other words, h should continuously vanish as R_{JI} approaches R_c. We may introduce a cutoff function

$$\phi(R) = \begin{cases} \frac{1}{R+\delta}, & 0 \le R \le R_{cs} \\ \frac{1}{R+\delta}\left\{\frac{1}{2}\cos\left[\pi\frac{(R-R_{cs})}{(R_c-R_{cs})}\right] + \frac{1}{2}\right\}, & R_{cs} < R < R_c \\ 0, & R \ge R_c \end{cases} \tag{3.16}$$

where $0 < R_{cs} < R_c$. Note that $\phi \in C^1(\mathbb{R}^+ \cup \{0\})$ for any $\delta > 0$.

Then we may define the generalized coordinate as

$$\mathbf{d}_J^I = \begin{bmatrix} \phi(R_{JI}) \\ \frac{\phi(R_{JI})}{R_{JI}}(\mathbf{R}_J - \mathbf{R}_I) \end{bmatrix} \in \mathbb{R}^{d+1}, \quad J \in \mathring{\mathcal{I}}_{R_c}(I). \tag{3.17}$$

Here $\tilde{d} := d + 1$ is the dimension of the generalized coordinate. In principle we may use a different set of generalized coordinates for electrons. For simplicity, in

this work we apply the same definition as in (3.17) to the electron, with the same truncation radius R_c. Therefore, effectively, the electron at position $\mathbf{r}$ only belongs to the extended interaction lists of its neighboring atoms.

We require h to depend only on the generalized coordinates as

$$h_{s(I),s(J)}(\mathbf{R}_J - \mathbf{R}_I) = \mathbf{d}_J^I \left[g_{s(I),s(J)} \left((\mathbf{d}_J^I)_1 \right) \right]^\top \in \mathbb{R}^{\tilde{d} \times M}. \tag{3.18}$$

Here $g_{s(I),s(J)} : \mathbb{R} \to \mathbb{R}^M$ is a neural network that only depends on the first component of $\mathbf{d}_J^I$ (*i.e.* the radial information R_{JI}), and only depends on the species of the particles I, J. We call $g_{s(I),s(J)}$ an embedding network. Combining Eq. (3.15) and (3.18), we have

$$\mathcal{D}^I(\mathbf{r}, \mathcal{R}^I) = \sum_{J,J' \in \mathcal{I}_{R_c}(I)} \left[g_{s(I),s(J)} \left((\mathbf{d}_J^I)_1 \right) \right] \left[(\mathbf{d}_J^I)^\top (\mathbf{d}_{J'}^I) \right] \left[g_{s(I),s(J')} \left((\mathbf{d}_{J'}^I)_1 \right) \right]^\top. \tag{3.19}$$

Both the radial information $(\mathbf{d}_J^I)_1$ and the inner product $(\mathbf{d}_J^I)^\top (\mathbf{d}_{J'}^I)$ satisfy the rotation symmetry. Therefore the descriptor $\mathcal{D}^I$ is invariant to permutation, rotation, and translation symmetry operations.

Fig. 3.3 provides a simplified illustration for computing $\mathcal{D}^I$, where the matrix $\mathbf{g}^I$ encodes $\left(g_{s(I),s(J)} \right)^\top$ for $J \in \mathcal{I}_{R_c}(I)$ and

$$\mathbf{h}^I = \sum_{J \in \mathcal{I}_{R_c}(I)} h_{s(I),s(J)}(\mathbf{R}_J - \mathbf{R}_I) = \sum_{J \in \mathcal{I}_{R_c}(I)} \mathbf{d}_J^I \left[g_{s(I),s(J)} \left((\mathbf{d}_J^I)_1 \right) \right]^\top = \mathbf{d}^I \mathbf{g}^I. \tag{3.20}$$

We use a ResNet [45] architecture using dense layers to construct $\mathcal{F}_{s(I)}$, while the feature mapping $g_{s(I),s(J)}$ is a dense feed-forward neural network with a few layers.

To better illustrate the symmetry-preserving procedure adopted by Deep Density, we mention that a widely adopted idea has been the one pioneered by Behler and Parrinello [10], which replaces the descriptor $\mathcal{D}^I(\mathbf{r}, \mathcal{R}^I)$ by a list of so-called radial and angular symmetry functions. The radial symmetry functions take the form

$$G_m^{I(r)} = \sum_J e^{-\eta_m (R_{JI} - R_m^s)^2} \phi\left(R_{JI}\right), \tag{3.21}$$

while the angular symmetry functions take the form

$$G_n^{I(a)} = 2^{1-\delta_n} \sum_{JK} (1 + \lambda_n \cos\theta_{IJK})^{\delta_n} e^{-\eta_n (R_{JI}^2 + R_{KI}^2 + R_{JK}^2)} \phi\left(R_{JI}\right) \phi\left(R_{KI}\right) \phi\left(R_{JK}\right). \tag{3.22}$$

A predetermined list of parameters $\{\eta_m, R_m^s\}$ or $\{\delta_n, \lambda_n, \eta_n\}$ has to be given to kick-off the training process. These parameters are very different from the parameters

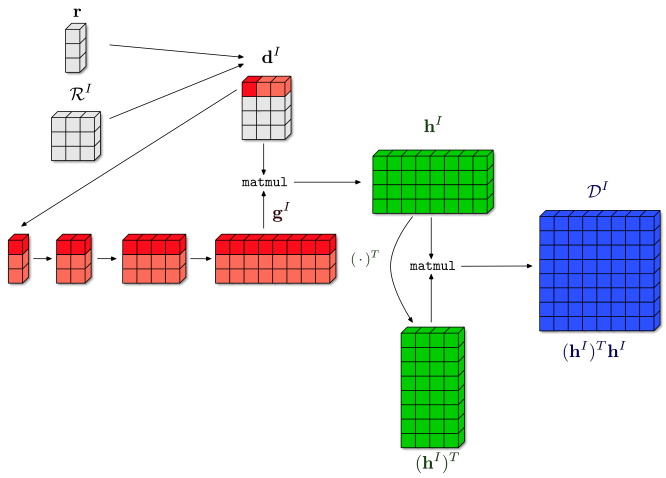


Figure 3.3: Schematic illustration of the computation of $\mathcal{D}^I$, where $n(I) = 3$, and there is only one atomic species (besides the electron represented in dark red).

in a neural network model. A moment-based first order optimization scheme, like Adam [53], could easily optimize the latter, but it could be much harder to optimize simultaneously the former and the latter without human intervention. This inspires us to adopt both an embedding network and a fitting network to automatically discover the proper "symmetry functions" on the fly during the training process, making Deep Density an end-to-end model.

3.4 Numerical examples

In this section we report the performance of Deep Density for one-dimensional model problems, as well as three dimensional real systems. The details of the setup as well as the choice of hyperparameters can be found in section 3.5 and 3.6, respectively. In all calculations the test error is measured in terms of the relative ℓ^1/ℓ^2 error, defined as:

$$\texttt{err}_{\ell^1} := \frac{\sum_i |\varrho(\mathbf{r}_i, \{\mathbf{R}_I\}) - \varrho_{NN}(\mathbf{r}_i, \{\mathbf{R}_I\})|}{\sum_i |\varrho(\mathbf{r}_i, \{\mathbf{R}_I\})|}, \tag{3.23}$$

$$\texttt{err}_{\ell^2} := \frac{\left(\sum_i \left[\varrho(\mathbf{r}_i, \{\mathbf{R}_I\}) - \varrho_{NN}(\mathbf{r}_i, \{\mathbf{R}_I\})\right]^2\right)^{\frac{1}{2}}}{\left(\sum_i \left[\varrho(\mathbf{r}_i, \{\mathbf{R}_I\})\right]^2\right)^{\frac{1}{2}}}. \tag{3.24}$$

Here the index i is taken over all the discretization points, ϱ is the electron density computed using Kohn-Sham solvers, and ϱ_{NN} is the approximation given by the neural network. In particular, the relative ℓ^1 error approximates the following quantity

$$\frac{\int |\varrho(\mathbf{r}, \{\mathbf{R}_I\}) - \varrho_{NN}(\mathbf{r}, \{\mathbf{R}_I\})| \, \mathrm{d}\mathbf{r}}{N_e} \tag{3.25}$$

which is the same error metric used by e.g. [38].

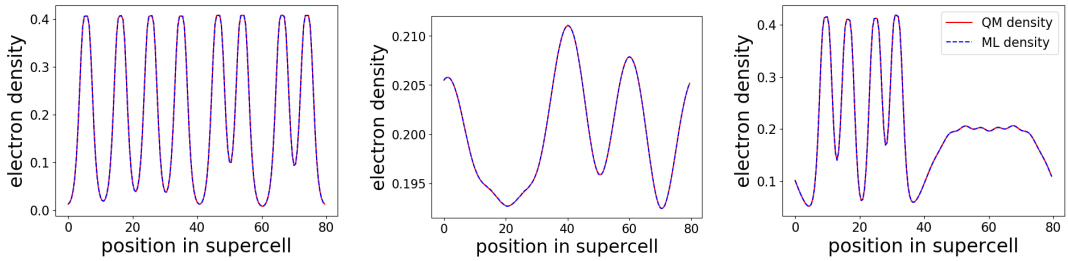
One dimensional systems

In this section we study three model systems in 1D with different characters: insulating, metallic, and mixed metallic-insulating systems. Previous study indicates that when the system is metallic or has mixed metallic-insulating characters, the self-consistent field iteration can be very difficult to converge (without a proper preconditioner) due to the small energy gaps and the associated charge sloshing behavior [64]. The details of the setup can be found in section 3.5.

We first consider a small supercell consisting of 8 atoms initially separated by 10 a.u. At the beginning of the *ab initio* molecular dynamics simulation, we perturb each of the 8 atoms by a uniform random number in $[-3, 3]$ a.u., and then let the systems evolve for 30000 time steps. In order to reduce the correlation among the snapshots and the amount of training time, we down-sample the trajectory for the first 8000 time steps by a factor 80, and we take the resulting first 100 snapshots as the training snapshots. The same procedure is applied to the validation snapshots for the next 400 time steps. We then use these 100 training snapshots and 5 validation snapshots to train the network.

We test the trained model by comparing the predicted density and the density obtained from the KS-DFT calculation, using a snapshot which is part of the original training set at time step 2019 (before the down-sampling) in Fig. 3.4, as well as a snapshot that is far outside the training set at time step 29000 in Fig. 3.5. In both cases, the error of the electron density is very small, and is $0.01\% \sim 0.43\%$ measured by the relative ℓ^1 norm.

Our architecture constructs a local density ϱ^I for each atom as in Eq. (3.9). This enables us to use the trained model to predict the electron density of a larger system. We test the transferability of our model by loading parameters trained using the 8-atom systems into the model that predicts the electron density for the 32-atom systems. For the mixed metallic-insulating system, the first 8 atoms are insulator-like and the latter 24 atoms are metal-like. Our model achieves excellent transferability, and the error is $0.01\% \sim 0.57\%$, as shown in Fig. 3.6.

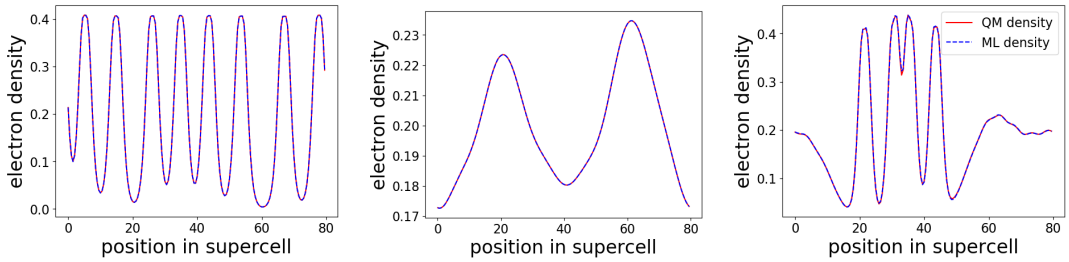


(a) the insulating system, $\texttt{err}_{\ell^1}$ = 6.27E-04

(b) the metallic system, $\texttt{err}_{\ell^1}$ = 1.21E-04

(c) the mixed metallic-insulating system, $\texttt{err}_{\ell^1}$ = 5.42E-04

Figure 3.4: Comparison of the electron density at time step 2019.



(a) the insulator system, $\texttt{err}_{\ell^1}$ = 8.12E-04

(b) the metallic system, $\texttt{err}_{\ell^1}$ = 1.15E-04

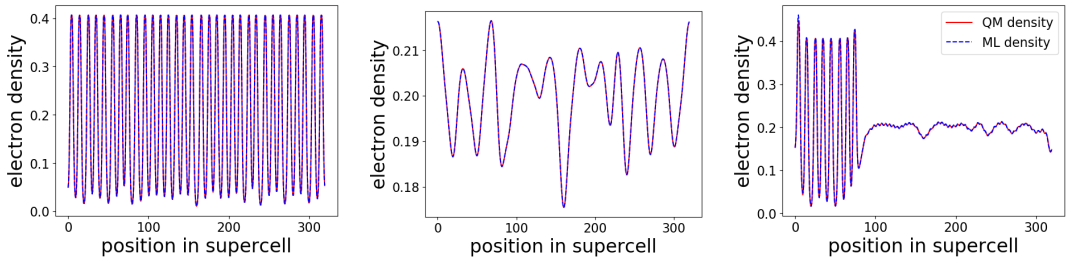
(c) the mixed metallic-insulating system, $\texttt{err}_{\ell^1}$ = 4.30E-03

Figure 3.5: Comparison of the electron density at time step 29000.

Three dimensional systems

For three-dimensional molecular and condensed matter systems, we present the following three test sets:

- organic molecules: a single ethane molecule (C_2H_6), a single isobutane molecule and a single n-butane molecule (C_4H_{10}).
- water: a set of systems composed of $32, 64, 128, 256$, and 512 water molecules in the liquid phase at T=300K.
- aluminum: a set systems composed of $32, 108$, and 256 aluminum atoms formed



(a) the insulator system, $\texttt{err}_{\ell^1}$ = 6.86E-04;

(b) the metallic system, $\texttt{err}_{\ell^1}$ = 1.49E-04;

(c) the mixed metallic-insulating system, $\texttt{err}_{\ell^1}$ = 5.71E-03.

Figure 3.6: Transferability of the one-dimensional model, which is trained using for a system with 8 atoms and tested on a system with 32 atoms.

initially by $2 \times 2 \times 2$, $3 \times 3 \times 3$, and $4 \times 4 \times 4$ face-center cubic (fcc) unit cells, and at temperatures 300K, 600K, and 900K, respectively.

For organic molecules, the configurations are collected from the dataset provided in [17]. We take the first 101 (uncorrelated) snapshots of ethane, n-butane and isobutane from the dataset. For each molecule, we use 100 snapshots for training and one for testing.

For water and aluminum, the configurations are obtained using the DeePMD-kit package [93]. For each case, the atomic configurations are uniformly sampled from long molecular dynamics trajectories at different temperatures and different system sizes. For all simulations we perform NpT simulations at the standard pressure p=1 bar with a time step of 1 fs. The potential energy models used in the simulation are obtained with the DP-GEN scheme [102] using *ab initio* data. We take one snapshot in every 1000 time steps from the trajectory to reduce the correlation among configurations. The data sets are divided as follows: The training set was a randomly selected subset of 80 snapshots from the 100 snapshots for the smallest system. The test set was composed of the remaining 20 snapshots for the smallest systems in addition to the snapshots of the larger ones.

For all systems, we compute the electron density for each snapshot using PWDFT (which is based on planewaves and is an independent module of the DGDFT package [50]). We use the Perdew-Burke-Ernzerhof (PBE) exchange-correlation functional [77], and the SG15 Optimized Norm-Conserving Vanderbilt (ONCV) pseudopotentials [40, 87]. Other details of the setup of test systems and the training hyperparameters are given in section 3.6.

Small Molecules

The kinetic energy cut-off is set to 30 a.u. for both C_2H_6 and C_4H_{10}. The total number of grid points for each system is 1,906,624. For the embedding networks in Eq. (3.18), we use a dense linear network with three layers, containing $\{5, 10, 20\}$ nodes respectively. For each fitting network we used a ResNet [45] with 3 dense layers containing 50 nodes per layer, where the skip connections are weighted by a trainable coefficient. The cutoff is the same for the 3 molecules $R_c = 4\mathring{A}$. We trained the network for a few times with different random seeds and picked the one with the smallest generalization error.

Fig. 3.7 shows the slice with the largest error for a snapshot in the test set for both molecules. The relative test errors for the molecules are shown in Table 3.1. The relative ℓ^1 error in [38] for ethane and butane are 1.14% and 1.19%, respectively. Therefore our error is 6.1 and 3.8 times smaller, respectively. Finally Fig. 3.10 (section 3.6) summarizes the distribution of the prediction error for the three molecules. We observe that the distribution remains exceptionally close to the diagonal, thus indicating a very low error.

Molecule \error	$\texttt{err}_{\ell^2}$	$\texttt{err}_{\ell^1}$
ethane	0.172%	0.186%
isobutane	0.194%	0.222%
butane	0.289%	0.314%

Table 3.1: Relative error of the testing samples for different molecules.

Another way to assess the accuracy of Deep Density is to evaluate quantities derived from the electron density, such as the dipole moment $D = \int \rho(\mathbf{r})\mathbf{r}\,\mathrm{d}\mathbf{r}$, as well as $Q = \int \rho(\mathbf{r})\,|\mathbf{r}|^2\,\mathrm{d}\mathbf{r}$ which is related to the quadrupole moment. We report ℓ^2 errors of D and absolute errors of Q for these three molecules using the density predicted by Deep Density in Table 3.2 and 3.3, respectively. For all three molecules the dipole moment is very close to zero, and we find that the main error is due to that the density from Deep Density may not preserve the total charge. Hence we post-processed it using a multiplicative normalization to make the total charge equals to the exact value and to reduce the error. For the Q value, the reference values for ethane, isobutane, and butane ranges from $30 \sim 60$ au, so the relative error ranges from $0.02\% \sim 0.9\%$, which is similar to the error measured by the ℓ^1 and ℓ^2 metric for the electron density.

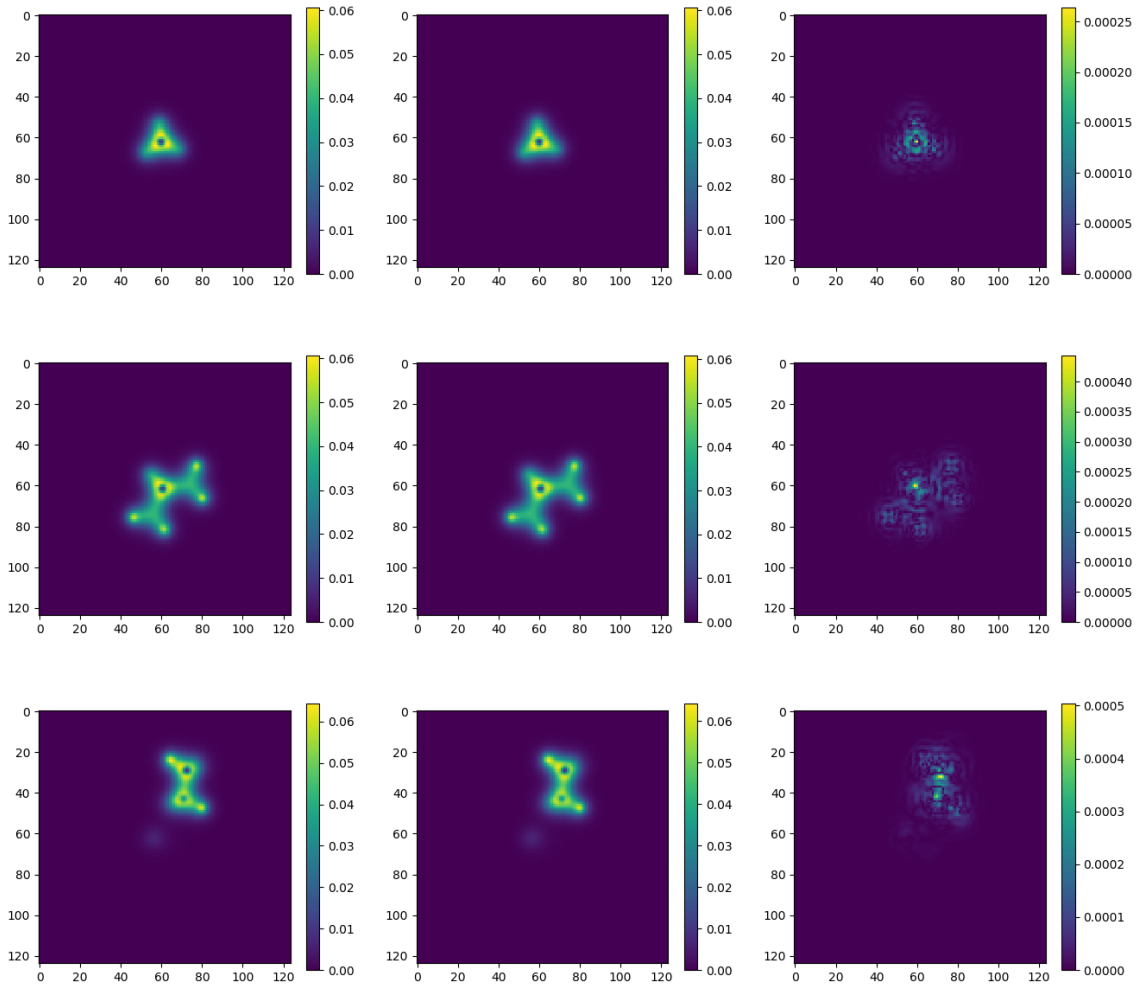


Figure 3.7: (left column) Slice of a snapshot of the electron density containing the largest point-wise error , (center column) slice of the density computed using the network, (right column) absolute error. Rows starting from the top : results for ethane, isobutane, and butane.

Molecule \error	$\texttt{err}_{train}$	$\texttt{err}_{train}$ norm	$\texttt{err}_{test}$	$\texttt{err}_{test}$ norm
ethane	4.70e-3	1.12e-4	4.25e-2	3.20e-3
isobutane	6.88e-2	2.57e-2	8.52e-2	4.63e-2
butane	1.50e-2	8.94e-5	2.29e-2	7.61e-3

Table 3.2: ℓ^2 Error of the dipole moment D for a snapshot in the training and test sets. Results before and after the multiplicative normalization are both included. Values are given in atomic unit.

Molecule \error	$\texttt{err}_{train}$	$\texttt{err}_{train}$ norm	$\texttt{err}_{test}$	$\texttt{err}_{test}$ norm
ethane	9.34e-2	8.24e-2	5.27e-2	7.30e-3
isobutane	1.90e-1	5.99e-2	6.06e-1	2.05e-1
butane	5.44e-1	2.58e-1	4.75e-1	3.34e-1

Table 3.3: Absolute Error of the value of Q for a snapshot in the training and test sets. Values are given in atomic unit.

$\mathbf{H_2O}$

For water the kinetic energy cut-off is set to 40 a.u. Both the training and test systems consist of 32 water molecules. For the embedding networks in Eq. (3.18), we use a dense linear network with four layers, each one containing $\{5, 10, 20, 40\}$ nodes respectively. The ResNet fitting network uses 5 dense layers and 50 nodes per layer, where the skip connections are weighted by a trainable coefficient. The cutoff radius is $R_c = 3.5\mathring{A}$. We trained the network a few times changing the random seed and we picked the one with the smallest generalization error. The error for a test snapshot with 32 atoms is showcased in Fig. 3.8, where we provide the slice containing the largest point-wise error.

Next we test the transferability of our model using systems with different sizes, which consists of 64, 128, 256 and 512 water molecules, respectively. The largest system has a total number of $20,213,648$ grid points (see Table 3.4 for the number of grid points of the other systems). The relative ℓ^2 and ℓ^1 errors are summarized in Table 3.4, where we can observe that inference error remains almost constant across different systems, which is around 0.5% for the ℓ^2 relative error (and 1.0% for the ℓ^1 relative error). Finally Fig. 3.10 (section 3.6) summarizes the distribution of the prediction error as we increase the system size. We can observe that the distribution remains very well concentrated within the diagonal thus indicating a very low error.

$N_{\texttt{mol}}(H_2O)\backslash$error	$\texttt{err}_{\ell^2}$	$\texttt{err}_{\ell^1}$	time [s]	$N_{\texttt{grid-points}}$
32	0.606%	1.080%	2.78e-2	108^3
64	0.612%	1.021%	5.70e-2	136^3
128	0.503%	0.891%	9.84e-2	168^3
256	0.520%	0.903%	1.86e-1	216^3
512	0.528%	0.921%	3.66e-1	272^3

Table 3.4: Error of the testing sample in both ℓ^2 and ℓ^1 norms, the wall-clock time of inference for 32 grid points of the electron density, and the total number of grid points inferred, for systems with different number of water molecules.

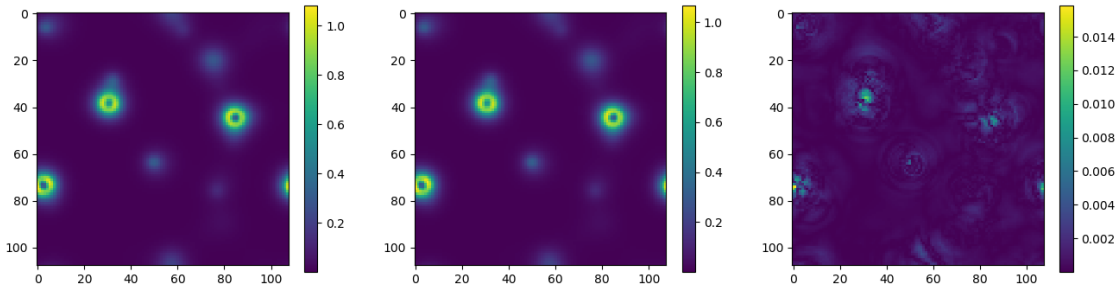


Figure 3.8: (left) slice of a snapshot of the electron density with 32 water molecules, (center) slice of the density computed using the network, (right) slice containing the largest point-wise absolute error for the snapshot.

Fig. 3.11 (Section 3.6) shows the generalization error measured by the slice with the largest error for a snapshot in the test set. It demonstrates that the Deep Density network, learned using a small sized system, has excellent transferability to large systems. The magnitude of the relative error agrees with that in Table 3.4. In this case, we observe that the error is mostly concentrated on high-gradient portions of ρ near the nuclei.

Finally, we present the scaling of the inference time with respect to $N_{\texttt{mol}}$, the number of molecules in the input. We measure the average inference time for a batch of 32 grid points with respect to systems of different sizes, and the results are summarized in Table 3.4. We can clearly observe that the time grows linearly with $N_{\texttt{mol}}$, albeit with a relatively large pre-constant. We find that the bottleneck

for inferring the electron density is the computation of the interaction lists for each configuration and each point of evaluation, which is linear in the number of atoms. This cost can be amortized by reusing the same interaction list for nearby evaluation points, which would drastically reduce the pre-constant.

In addition, Table 3.4 shows that the number of grid points for inference grows linearly with $N_{\texttt{mol}}$. Hence when using Deep Density to predict the electron density for every grid-point, the total complexity scales quadratically with respect to $N_{\texttt{mol}}$. This super-linear complexity is due to that the current implementation (3.9) includes the electron coordinate in the interaction list of *all* nuclei. Thus for a given grid point to be inferred, we sum the contributions over *all* atoms. Given the fast decay of the ansatz in (3.11), we can expect that the contribution of most atoms would be negligible for a large system. Thus we can truncate the sum to include atoms only within a certain radius, and the overall computational complexity would be reduced to linear with respect to $N_{\texttt{mol}}$.

Aluminum

For the aluminum systems, we follow the same training pipeline as for the water case, and the kinetic energy cutoff is 20 a.u., and the largest system has total number of 1,906,624 grid points. The feature and fitting networks are chosen in a similar fashion to the water case. However, we used a larger truncation radius $R_c = 6\mathring{A}$, and the initialization of constants in (3.11) was modified by using a larger truncation radius in order to start with a more uniform initial density .

The training stage was performed using a system with 32 atoms as explained in Section 3.6. In Fig. 3.13 (Section 3.6) we demonstrate that the trained model is able to efficiently recover the peaks concentrated at the center of each nuclei, which accounts mainly for the electron density associated to semi-core orbitals. Note that the ONCV pseudopotential treats all the 2s and 2p orbitals as semi-core electrons. As a result, the electron density in Al has sharp peaks and relatively large magnitudes. We test the transferability following the same procedure as for the water system. In Fig. 3.13 we compare the electron density provided by the network and those from KS-DFT calculations for configurations containing 108 and 256 aluminum atoms. For both cases we observe a relative ℓ^2 error below 2.5% (see Table 3.7). This is larger compared to that of the water system. From Fig. 3.13 and Table 3.7 we observe that the errors also grow with respect to the system size. This may be due to the quality of the training data generated by PWDFT, which only uses the Γ-point sampling of the Brillouin zone, and the system size is relatively small. To verify this, we train the network with just 4 snapshots of the $3 \times 3 \times 3$ configuration. The relative test ℓ^2 and ℓ^1 error can be improved to 0.5% and 1.2%, respectively, for a $3 \times 3 \times 3$ configuration

as shown in Fig. 3.14. In addition, even though the absolute error of water and Al systems are comparable to each other, further inspection of Figs. 3.11 and 3.13 reveals some qualitative difference between the two systems: the errors from the Al systems appear to be much more spatially delocalized, and hence it is possible that the errors are mainly contributed by the valence electrons rather than the semi-core electrons. To verify this, we tested our method with another data set, which uses the same configurations, but with the density generated by the Vienna *ab initio* simulation package (VASP, version 5.4.4) [58], which do not treat electrons at 2s and 2p orbitals as semi-core electrons. The results are reported in Section 3.6.

3.5 Numerical results for 1D systems

We use a 1D reduced Hartree-Fock model similar to the one presented in [64]. This simplified model depends nonlinearly on the electron density ρ only through the Hartree interaction, and does not include the exchange-correlation functional. However, it can still qualitatively reproduce certain phenomena in 3D, such as the difference between insulating and metallic systems, and the screening effect shown in Section 3.2. The Hamiltonian in our 1D model is given by (we still use the notations $\mathbf{r}$ and $\mathbf{R}$ though this is a one-dimensional system)

$$H[\rho, \{\mathbf{R}_I\}_{I=1}^{N_a}] = -\frac{1}{2}\frac{d^2}{d\mathbf{r}^2} + V_{\text{hxc}}[\mathbf{r}; \rho] + V_{\text{ion}}[\mathbf{r}; \{\mathbf{R}_I\}_{I=1}^{N_a}], \tag{3.26}$$

$$= -\frac{1}{2}\frac{d^2}{d\mathbf{r}^2} + \int K(\mathbf{r}, \mathbf{r}')(\rho(\mathbf{r}') + m(\mathbf{r}'; \{\mathbf{R}_I\}_{I=1}^{N_a}))\, \mathrm{d}\mathbf{r}'. \tag{3.27}$$

Here we use a pseudopotential to represent the electron-ion interaction, and the total pseudo charge density is given by

$$m(\mathbf{r}; \{\mathbf{R}_I\}_{I=1}^{N_a}) = \sum_{I=1}^{N_a} -\frac{Z_I}{\sqrt{2\pi\sigma_I^2}} \exp\left(-\frac{1}{2\sigma_I^2}(\mathbf{r} - \mathbf{R}_I)^2\right). \tag{3.28}$$

Here Z_I represents the charge of I-th nucleus, and σ_I represents the width of the nuclei potential within the pseudopotential theory. σ_I is tuned so that I-th nucleus can qualitative behave as a metal or as an insulator. Since the standard Coulomb interaction diverges in 1D, we employ the Yukawa kernel

$$K(\mathbf{r}, \mathbf{r}') = \frac{2\pi}{\kappa\epsilon_0} e^{\kappa|\mathbf{r}-\mathbf{r}'|}, \tag{3.29}$$

where the parameters $\kappa = 0.01$ and $\epsilon_0 = 10.0$ are fixed constants throughout the experiments. Our units here are arbitrary, but will be referred to as the atomic unit (a.u.) for simplicity.

The Kohn-Sham equations are solved using the standard self-consistent field iteration [70] method. In particular, we use Anderson mixing [3] of the potential with mix dimension 10.

We study three types of systems: the insulating system, the metallic system, and the mixed metallic-insulating system. Fig. 3.9 displays the occupied eigenvalues and the first ten unoccupied eigenvalues for all the three systems. In particular, when the system is metallic or has a mixed metallic-insulating character, the self-consistent field iteration can be very difficult to converge due to the small energy gaps and charge sloshing behavior [64].

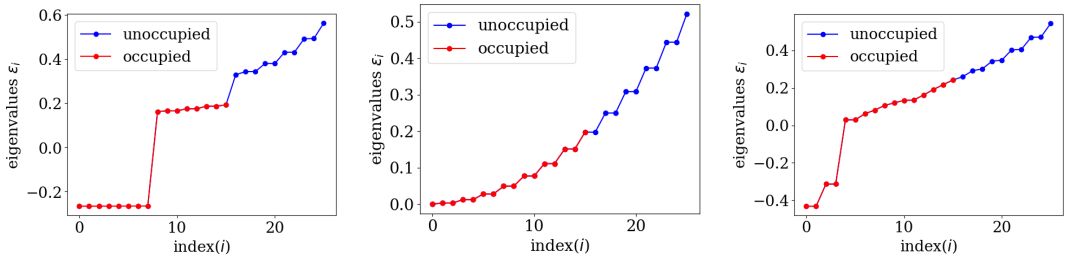


Figure 3.9: Eigenvalues for (a) the insulating system, $\sigma_I = 1.0$ for all I; (b) the metallic system, $\sigma = 6.0$ for all I; (c) the mixed metallic-insulating system, $\sigma_I = 1.0$ for $1 \leq I \leq 4$ and $\sigma_I = 6.0$ for $5 \leq I \leq 8$.

We consider a periodic system that includes 8 atoms in the unit cell, with 2 electrons per site, i.e. $Z_I = 2$ for $I = 1, ..., 8$. The atoms are 10 a.u. apart, located at $5, 15, 25, \cdots, 75$. The size of the supercell in this case is thus 80 a.u. As mentioned above, by adjusting σ_I we can obtain different qualitative behaviors. On the one hand, when $\sigma_I = 1.0$, the model qualitatively behaves as an insulating system with an energy gap of 0.136. On the other hand, if $\sigma_I = 6.0$, then the model qualitatively behaves as a metallic system and its energy gap is 5.5×10^{-8} (i.e. the system is gapless). To test the ability of the proposed architecture to deal with interactions between atoms of different species, we also introduce a mixed metallic-insulating system, which is obtained by setting $\sigma_I = 1.0$ for $I = 1, 2, 3, 4$, and $\sigma_I = 6.0$ for $I = 5, 6, 7, 8$. The energy gap in this case equals to 0.018. Fig. 3.9 displays the occupied eigenvalues and the first ten unoccupied eigenvalues for all the three systems.

For the 1D problem, our implementation is purely based on `python` and `tensorflow`. Furthermore, 1D model does not involve the rotational degrees of freedom. This allows us to simplify the network structure in Section 3.3 as below.

N_{near}	1	2	3	4	5	6	7
Insulator	1.91E-08	4.02E-08	2.26E-08	5.02E-08	1.38E-07	1.39E-07	1.13E-07
Metal	9.11E-10	1.08E-09	1.04E-09	1.10E-09	3.00E-09	7.65E-09	3.53E-09

Table 3.5: Validation error (MSE) for single type system with different values of N_{near}

For simplicity of implementation, we introduce a parameter N_{near} instead of cutoff R_c, so the index set $\mathcal{I}_{N_{\text{near}}}(I)$ is decided by choosing the indices of the nearest N_{near} atoms. The model is constructed using the same ansatz as Eq. (3.11) where $\mathcal{N}^I$ and $\mathcal{E}^I$ are neural networks. However, the construction of the input to these networks, which are the descriptors $\mathcal{D}^I(\mathbf{r}, \mathcal{R}^I)$ defined in Section 3.3 as Eq. (3.15), is much simpler. To define the descriptors, we start with $\mathbf{d}_J^I = [R_{JI}, \frac{1}{R_{JI}}(\mathbf{R}_J - \mathbf{R}_I)]$ for $J \in \mathcal{I}_{N_{\text{near}}}(I)$ (distance information and direction information). Since we follow the form in Eq. (3.17) with $\mathbf{R}_J - \mathbf{R}_I$ reduced to one dimension, each $\mathbf{d}_J^I$ is in $\mathbb{R}^2$. The electron information $\mathbf{d}_0^I \in \mathbb{R}^2$ is fed to the descriptor directly, whereas the atom information $\mathbf{d}_J^I, J \neq 0$, is passed to the function $g_{s(I),s(J)}$ before being fed to the descriptor. For the insulating system and the metallic system, we only have one such function g, whereas for the mixed metallic-insulating system, we have four $g_{s(I),s(J)}$ networks because each of $s(I), s(J)$ can be one of the two types. Given that rotation symmetry in 1D is trivial, the descriptor $\mathcal{D}^I$ is formed simply by concatenating the electron information, $\mathbf{d}_0^I$, and atom information, $g_{s(I),s(J)}(\mathbf{d}_J^I)$. The output of g is of M dimension and there are N_{near} number of nearby atoms, so descriptors are $\mathcal{D}^I \in \mathbb{R}^{2+MN_{\text{near}}}$.

To treat the mixed metal-insulator system with two types of atoms, we implement a control flow so that at run time, the model knows which $g_{s(I),s(J)}$ to apply based on species of atom I and J. For simplicity we incorporate the information of the species as follows. Let range of $s(J)$ be $\{1, 2\}$. We encode the two atom types as vectors $\mathbf{v}_1 = [1, 0]^T, \mathbf{v}_2 = [0, 1]^T$. For a fixed center atom I, and adjacent atom J, we pass $\mathbf{d}_J^I$ to $g_{s(I),s}$ for both $s = 1, 2$ and then calculate the output

$$g_{s(I),s(J)}(\mathbf{d}_J^I) = \left(\mathbf{v}_{s(J)}^T \mathbf{v}_1\right) g_{s(I),1}(\mathbf{d}_J^I) + \left(\mathbf{v}_{s(J)}^T \mathbf{v}_2\right) g_{s(I),2}(\mathbf{d}_J^I).$$

The training and test data sets are generated through molecular dynamics simulations. We use the Verlet algorithm [33] for the time propagation, where the forces are computed using the Hellmann-Feynman formula. At each time step we store the atomic configuration, $\{\mathbf{R}_I\}_{I=1}^8$, and the corresponding self-converged electron density, ρ, generated from the KS-DFT computation. For all three systems, we use the first 8000 snapshots for training and the next 400 snapshots for validation. In order to

N_{sample}	100	200	400
Two-atom-type	1.87E-07	6.94E-08	4.40E-08

Table 3.6: Validation error (MSE) for two-atom-type system with increasing training samples

reduce the correlation of the shots and the amount of training time, we down-sample the training snapshots by a factor 80, i.e., we take 100 evenly time-spaced snapshots. The same procedure is applied to the validation snapshots. We then use these 100 training snapshots and 5 validation snapshots to train the network. For the mixed metallic-insulating system, the number of trainable parameters increases because of the four $g_{s(I),s(J)}$ networks, so we reduce the down-sampling factor to have more training snapshots (e.g. down sample the first 8000 snapshots by a factor of 20 to obtain 400 training snapshots). We also find that if we only use 100 training snapshots, the relative ℓ^1 error can increase and be higher than 1%. This indicates that the mixed insulating-metallic system is indeed more difficult and requires a larger number of training samples.

The training is performed using standard Adam optimizer [53] and a mean squared error loss. Given the simplicity and small scale of the problem we visit all the points at each snapshot, in contrast with the 3D training that will require importance sampling for efficiency consideration. The network was trained for 400 epochs, the model with lowest validation loss was saved. For each hyperparameter setting (Fixed N_{layer}, N_{nodes}, N_{near}), we run 5 experiments and report the one with lowest validation error. The validation errors are measured using mean squared error (MSE), namely

$$\frac{1}{N_{\text{validation}}} \sum_{j=1}^{N_{\text{validation}}} \left(\varrho(\mathbf{r}_j, \{R_I\}) - \varrho_{NN}(\mathbf{r}_j, \{R_I\})\right)^2 . \tag{3.30}$$

In Table 3.5, we observe that the validation loss reaches well below 1E-06. Another observation is that the network is relatively insensitive to the hyperparameter N_{near} here, even when the system is gapless. Thus we fix $N_{\text{near}} = 2$ for the mixed metallic-insulating system model for simplicity.

In Table 3.6, N_{sample} is the number of snapshots in training, so the real amount of training data is the number of snapshots multiplied by the number of grid points for the 1D electron density. The validation loss reaches below 1E-07 once we increase the number of snapshots to 200.

3.6 Numerical results for 3D systems

Simulation parameters

The parameters in the ansatz are initialized after a precomputation step that depends on each setup. This precomputation involves the following steps: selecting one atom for each species, sampling the density within a small radius of that atom, and computing the parameters $A_{s(I)}, B_{s(I)}, C_{s(I)}$, and $D_{s(I)}$ that best fits the sampled density, without the neural networks, using standard quasi-Newton optimization methods. The purpose of this precomputation step is to help the optimization find a suitable minimum. The weights in the Neural Network are initialized using a normalized Gaussian distribution. The objective function is the mean squared loss.

For the training stage we use the Nadam optimizer [53] with an exponential scheduling, in which for every 20000 iterations we decrease the learning rate by a factor 0.95, and we initialize the learning rate as 0.003. At each iteration we draw $n_{\mathtt{s}} = 64$ samples from the snapshots. The training is scheduled as follows: we train the network for a million iterations using only 5 snapshots, then we train the network for another million iterations using 20 snapshots and finally we train the network for two million iteration using 80 snapshots. The remaining 20 snapshots were used for testing throughout the training.

At each iteration $n_{\mathtt{s}}$ samples are extracted from the training data. Each sample represents a pixel of the images shown in Fig. 3.11. For the water system we have 80 training snapshots and each snapshot contain around 1.24E6 pixels, totalling roughly 100 million data points. For the aluminum system we have the same amount of training snapshots but each snapshot contain around 2.564E5 pixels, totalling roughly 21 million data points. For the small organic molecules we have roughly 125 million data point for each system. Thus we need to visit them judiciously in order to be efficient. Given that different systems may have very different localization properties we use a sampling strategy based on the norm of the density at each pixel. In particular, during the training stage the samples are drawn following the distribution $|\rho(\mathbf{r})|^{\alpha}$, where the value of α is tuned for each setup, in order to avoid visiting small values of the densities too often, thus improving the efficiency of training. In particular we used 1/2 for the small organic molecules, 6/5 for the water systems, and 1 for the aluminum systems.

We estimate the test error by comparing the result given by the network against the test snapshots in the small system, and we estimate the transferability of the algorithm by comparing the electron density generated by the trained model for the larger systems versus the one computed using PWDFT.

The computation of the electron density were performed at the NERSC cluster

Cori, which is comprised of 2,388 dual socket nodes with 32 cores and 256 GB of RAM, whereas, the training of the models and inference steps were performed in a 16 core machine used with 64 GB of RAM and a Tesla V100 GPU with 16GB memory.

Additional plots of the organic molecules and water systems

In addition to the figures in the main text, we include Fig. 3.10, which depicts the performance of Deep Density for the different small molecules and different water systems. Fig. 3.10 represent a scatter plot in logarithmic scale of the value of the predicted density and the density computed with PWDFT for the same configuration and sampling points. We can observe that for the former the error are almost negligible, and the later the errors are higher but are still very small.

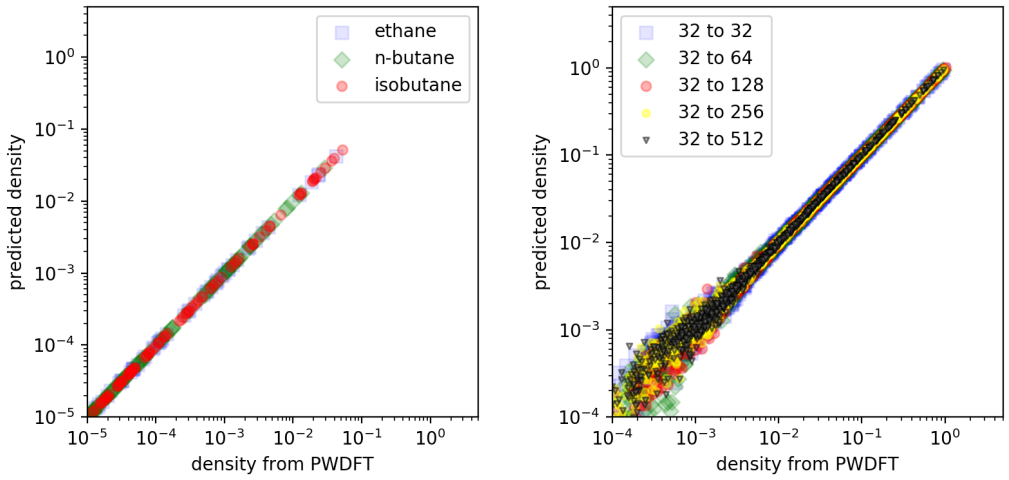


Figure 3.10: (left) scatter plot of the predicted and test densities for the different small organic molecules, (right) scatter plot of the predicted and test densities for different water systems.

Additional results for aluminum

Our VASP calculation also used the PBE exchange-correlation functional, but with the projector augmented wave method (PAW) [12] to handle the core electrons. In particular, the 2s and 2p orbitals are treated as core electrons, and hence there are only 3 valence electrons per atom. The kinetic energy cutoff for the plane wave

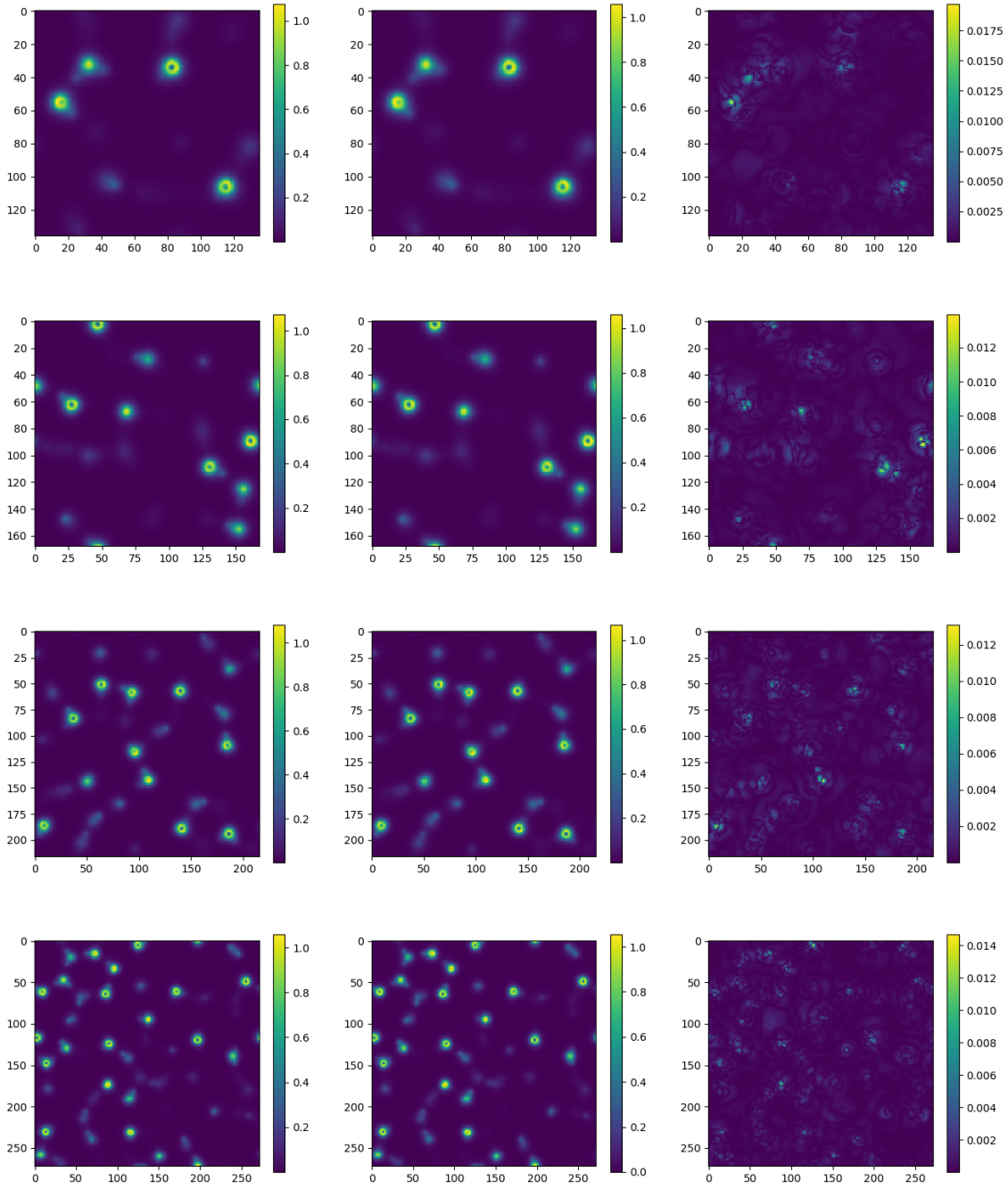


Figure 3.11: (left column) slice of a snapshot of the electron density, (center column) slice of the density computed using the network, (right column) slice of the absolute error with the highest point-wise absolute error. rows starting from the top : results for the system containing 64, 128, 256 and 512 water molecules

expansion is set to 600 eV, and the Brillouin zone is sampled with the Monkhorst-Pack mesh [74] at the spacing $h_k = 0.08\ \mathring{A}^{-1}$. The order 1 Methfessel-Paxton smearing method with $\sigma = 2900$ K is adopted. The self-consistent field (SCF) iteration stops when the total energy and band structure energy differences between two consecutive steps are smaller than 10^{-6} eV. In this case the density contains only the contribution of valence electrons.

We used the same training pipeline as before. We perform training using a number of snapshots for a system containing $2 \times 2 \times 2$ unit cells, and test the network for a number of systems with $2 \times 2 \times 2$, $3 \times 3 \times 3$ and $4 \times 4 \times 4$ unit cells, respectively. The scatter plot in Fig. 3.12 suggests that the test error for the aluminum system is indeed much larger. Fig. 3.15 shows the test error using the density generated with VASP. The error is largely delocalized, which confirms the previous study with PWDFT that most error originates from valence electrons. In addition, from Table 3.7, we observe that the generalization error still grows, albeit slightly slower, with respect to the system size, thanks to the refined Brillouin zone sampling when generating the training data set.

N_a (Al)n error	$\texttt{err}_{\ell^2}$ PWDFT	$\texttt{err}_{\ell^1}$ PWDFT	$\texttt{err}_{\ell^2}$ VASP	$\texttt{err}_{\ell^1}$ VASP
32	0.504%	1.400%	2.614%	2.015%
108	1.512%	3.937%	3.040%	2.529%
256	2.244%	5.801%	4.847%	4.515%

Table 3.7: Error of the testing samples for different number of atoms for Al. The data are generated using PWDFT (with semicore electrons) and VASP (without semicore electrons) respectively.

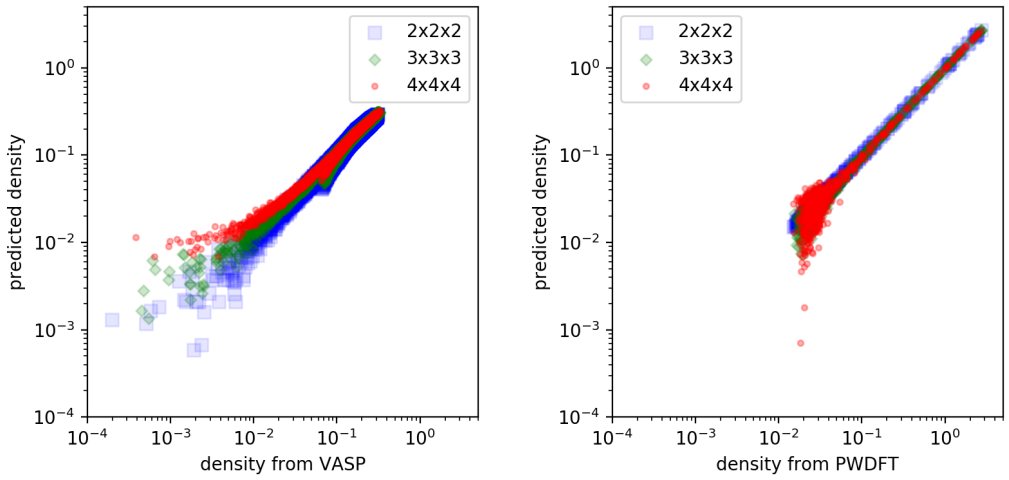


Figure 3.12: (left) scatter plot of the predicted and test densities generated by VASP for the different aluminum systems, (right) scatter plot of the predicted and test densities generated by PWDFT for the different aluminum systems. The magnitudes of the density from PWDFT are higher due to the inclusion of semi-core electrons.

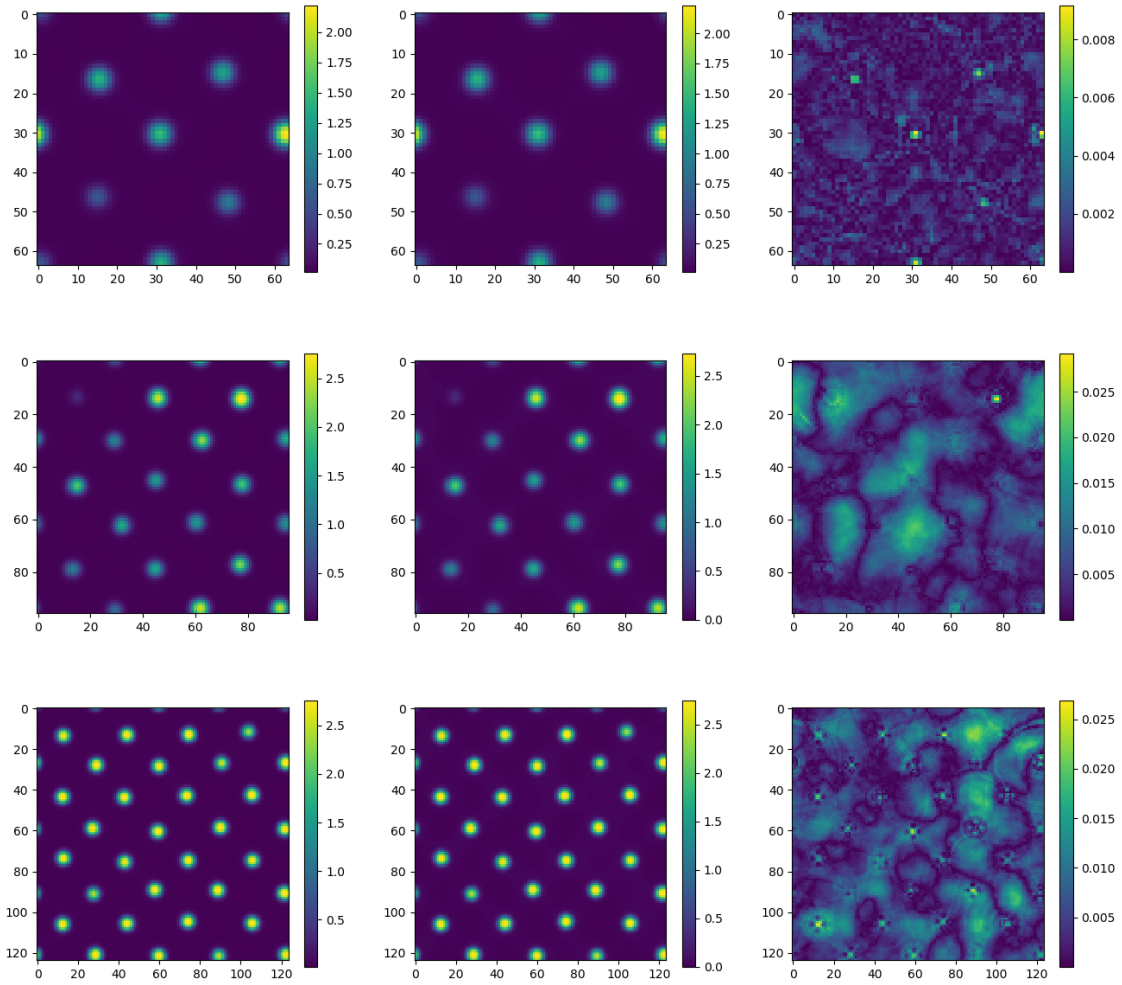


Figure 3.13: (left column) slice of a snapshot of the electron density, (center column) slice of the density computed using the network, (right column) slice of the absolute error with the highest point-wise absolute error. Rows starting from the top : results for the system containing 32, 108, and 256 aluminum atoms following $2 \times 2 \times 2$, $3 \times 3 \times 3$, and $4 \times 4 \times 4$ configurations respectively. The calculations are performed using PWDFT.

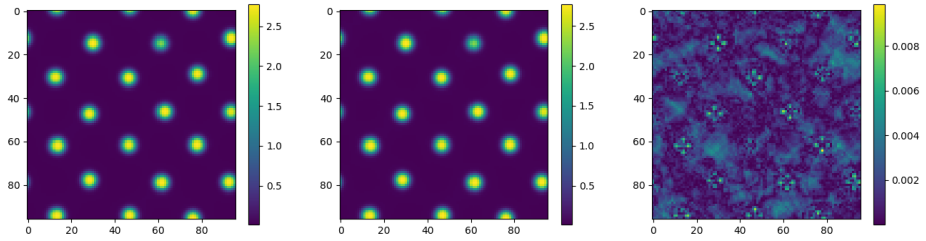


Figure 3.14: (left) slice of the snapshot produced by computing the electron density of 108 aluminum atoms in a $3 \times 3 \times 3$ configuration, (center) slice of the density computed using the network which was re-trained using 4 snapshots of the $3 \times 3 \times 3$ configuration, (right) slice of the absolute error with the highest point-wise absolute error. The calculations are performed using PWDFT.

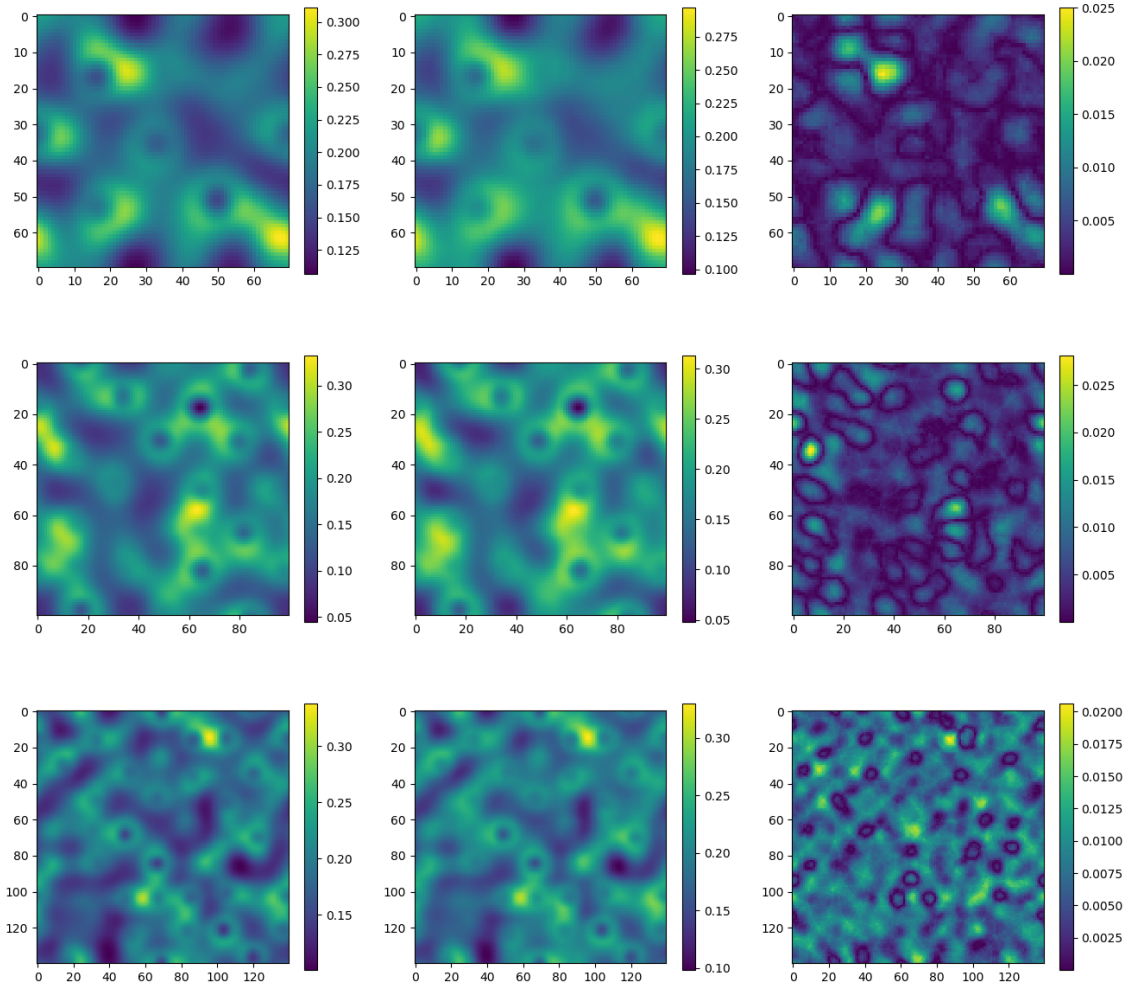


Figure 3.15: (left column) slice of a snapshot of the electron density from VASP, (center column) slice of the density computed using the network, (right column) slice of absolute error containing the largest point-wise error. Rows starting from the top : results for the system containing 32, 108, and 256 aluminum atoms following $2 \times 2 \times 2$, $3 \times 3 \times 3$, and $4 \times 4 \times 4$ configurations respectively.

3.7 Conclusion

Leveraging the success of the recently developed Deep Potential, we propose the Deep Density method to use machine learning to bypass the solution of the Kohn-Sham equations, and to obtain the self-consistent electron density in the context of *ab initio* molecular dynamics simulation. We demonstrate that the localization principle not only holds for insulating systems, but at least to some extent is also valid for metallic systems due to screening effects. Numerical results in one-dimensional systems and small molecular systems demonstrate that our construction can be very accurate, using a relatively small number of training samples. Our model can also be used to predict the electron density in the condensed phase, and can achieve excellent transferability for systems with up to 512 water molecules.

We envisage to accelerate the current algorithm. The complexity for the point-wise evaluation of the electron density is linear with respect to the systems size. However, a simple modification of the proposed approach can lead to a point-wise time evaluation that is independent of the systems size, thus producing a linear scaling algorithm. Another line of work is to improve the efficiency of training and prediction. In the current implementation, the descriptors are computed on CPUs, and this becomes a bottleneck when the electron density on millions of data points or more need to be evaluated. We expect that by employing a GPU based implementation and by computing the electron density at different grid points in an embarrassingly parallel fashion, the efficiency can be greatly improved. We expect that these improvements would make Deep Density to be a useful tool for the analysis and prediction of electronic structures.

Chapter 4

Permutation Symmetry

4.1 Introduction

In this chapter we study one type of symmetry that is ubiquitous in scientific computing applications: Permutation symmetry. The way to build the permutation symmetry into the architecture of the neural network is partially discussed in [98] by utilizing the universal approximation representation. Here we explore and extends further around this universal approximations of symmetric functions. A function $f : (\mathbb{R}^d)^N \to \mathbb{R}$ is **(totally) symmetric** if

$$f(\mathbf{x}_{\sigma(1)}, \ldots, \mathbf{x}_{\sigma(N)}) = f(\mathbf{x}_1, \ldots, \mathbf{x}_N), \tag{4.1.1}$$

for any permutation $\sigma \in S(N)$, and elements $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$. Note that the permutation is only applied to the outer indices $1, \ldots, N$ (also referred to as the "particle" indices later), but not the Cartesian indices $1, \ldots, d$ for each $\mathbf{x}_i$. In other words, f is not totally symmetric when viewed as a function on $\mathbb{R}^{d\times N}$. A totally symmetric function is also called a permutation invariant function. A closely related concept is the permutation equivariant mapping, which is of the form $Y : (\mathbb{R}^d)^N \to (\mathbb{R}^{\tilde{d}})^N$ that satisfies

$$Y_i(\mathbf{x}_{\sigma(1)}, \ldots, \mathbf{x}_{\sigma(N)}) = Y_{\sigma(i)}(\mathbf{x}_1, \ldots, \mathbf{x}_N), \qquad i = 1, \ldots, N \tag{4.1.2}$$

for any permutation $\sigma \in S(N)$, and $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$. Here each component $Y_i \in \mathbb{R}^{\tilde{d}}$, and $\tilde{d}$ can be different from d.

Perhaps the most important example of totally symmetric functions is the wavefunction of identical particles in quantum mechanics. The indistinguishability of identical particles implies that their wavefunctions should be either totally symmetric or totally anti-symmetric upon exchanging the variables associated with any two of

the particles, corresponding to two categories of particles: bosons and fermions. The former can share quantum states, giving rise to, e.g., the celebrated Bose-Einstein condensate; while the latter cannot share quantum states as described by the famous Pauli exclusion principle. Such exchange/permutation symmetry also arises from other applications than identical particles in quantum mechanics, mostly in the form of symmetric functions. For instance, in chemistry and materials science, the interatomic potential energy should be invariant under the permutation of the atoms of the same chemical species. Another example is in computer vision, where the classification of point clouds should not depend on the ordering of points.

The dimension of symmetric functions is usually large in practice because it is proportional to the number of considered elements. This means that in computation the notorious difficulty of "curse of dimensionality" is often encountered when dealing with such functions. Recent years have witnessed compelling success of neural networks in representing high-dimensional symmetric functions with great accuracy and efficiency, see, e.g., [10, 90, 103, 105] for interatomic potential energy, [79, 80] for 3D classification and segmentation of point sets, and [34, 106] for solutions of partial differential equations.

Despite the empirical success of neural networks approximating symmetric functions, theoretical understanding of these approximations is still limited. There are numerous results (see e.g. [7, 21, 49]) concerning the universal approximation of general continuous functions on compact domains. Nevertheless, if the target function is symmetric, it is much less investigated whether one can achieve the universal approximation with a class of functions with the same symmetry constraints. Explicitly guaranteeing the symmetric property of an ansatz is often mandatory. From a machine learning perspective, symmetries can also significantly reduce the number of effective degrees of freedom, improve the efficiency of training, and enhance the generalizability of the model (see e.g. [24]). However, one needs to first make sure that the function class is still universal and sufficiently expressive.

The universal approximations of symmetric functions were partially studied in [98]. However, as will be illustrated in later sections, the proof of [98] only holds for the case when $d = 1$. Moreover, there is no error estimation provided for the proposed approximation. The more recent work [86] considered the universal approximation of permutation invariant functions and equivariant mappings for $d = 1$ as well. The work [85] gives a generalization bound that is improved by a factor of $\sqrt{N!}$ with the introduction of the quotient feature space, but the result only applies for $d = 1$. By respecting the permutation symmetry, the resulting neural network involves much fewer parameters than the corresponding dense neural networks. The recent works [23, 6] on polynomial approximation of symmetric functions display a reduction of the curse of dimensionality when the target function has low-order many-body expansions

or sparse polynomial approximates. The work [107] studies learning symmetric functions from a different perspective by treating symmetric functions (of any size) as functions over probability measures. But such a setting is not always applicable to practical problems.

In this chapter we aim to study the universal approximation of general symmetric functions for any $d \geq 1$. We now summarize the main result of this chapter: For the symmetric function, we give two different proofs of the universality of the ansatz proposed in [98], both with explicit error bounds. The first proof is based on the Ryser formula [84] for permanents, and the second is based on the partition of the state space, as elaborated in Section 4.2 and Section 4.4, respectively. We summarize below in a theorem the ansatz we proved with the universal approximation for symmetric functions. The approximation rate only relies on a weak condition that the gradient is uniformly bounded. We conclude in Section 4.5 with some practical considerations and future directions for further investigation. The proofs of Theorem 1 are given in Section 4.4.

Theorem 1 (Approximation to symmetric functions)**.** *Let $f : \Omega^N \to \mathbb{R}$ be a continuously differentiable, totally symmetric function, where Ω is a compact subset of $\mathbb{R}^d$. Let $0 < \epsilon < \|\nabla f\|_2 \sqrt{Nd} N^{-\frac{1}{d}}$, here $\|\nabla f\|_2 = \max_X \|\nabla f(X)\|_2$. Then there exist $g : \mathbb{R}^d \to \mathbb{R}^M$, $\phi : \mathbb{R}^M \to \mathbb{R}$, such that for any $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in \Omega^N$,*

$$\left| f(X) - \phi\left(\sum_{j=1}^{N} g(\mathbf{x}_j) \right) \right| \leq \epsilon,$$

where M, the number of feature variables, satisfies the bound

$$M \leq 2^N (\|\nabla f\|_2^2 N d)^{Nd/2} / (\epsilon^{Nd} N!). \tag{4.1.3}$$

We remark that for the bounds in the theorem above, $\|\nabla f\|_2$ may also scale with respect to N. For the examples listed below, we assume the domain for the function to be a hypercube $\Omega_N = [0,1]^N$. In the symmetric case, if f is in the simple form $f(X) = \sum_{i=1}^N g(\mathbf{x}_i)$, $\|\nabla f\|_2$ will scale as $\mathcal{O}(\sqrt{N})$. The simple example above has $\mathcal{O}(\sqrt{N})$ scaling, but the exact scaling will depend on the choice of function f.

4.2 Universal approximation for $d = 1$

Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, with $\mathbf{x}_i \in \Omega := [0,1]^d$. Consider a totally symmetric function $f(X)$. It is proved in [98] that when $d = 1$ (therefore $\Omega = [0,1]$), the following

universal approximation representation holds

$$f(X) = \phi\left(\sum_{j=1}^{N} g(x_j)\right) \tag{4.2.1}$$

for continuous functions $g : \mathbb{R} \to \mathbb{R}^M$, and $\phi : \mathbb{R}^M \to \mathbb{R}$. For completeness we briefly recall the proof.

Let $\mathcal{X} = \{(x_1, \ldots, x_N) \in \Omega^N = \mathbb{R}^N : x_1 \leq x_2 \leq \cdots \leq x_N\}$. Define the mapping $E : \mathcal{X} \to \mathcal{Z} \subset \mathbb{R}^{N+1}$, with each component function defined as

$$z_q = E_q(X) := \sum_{n=1}^{N} (x_n)^q, \quad q = 0, 1, \ldots, N.$$

The mapping E is a homeomorphism between $\mathcal{X}$ and its image in $\mathbb{R}^{N+1}$ [98]. Hence, if we let $g : \mathbb{R} \to \mathbb{R}^{N+1}, x \mapsto [1, x, x^2, \ldots, x^N]$ and $\phi : \mathbb{R}^{N+1} \to \mathbb{R}, Z \mapsto f(E^{-1}(Z))$, then we have

$$f(X) = \phi\left(\sum_{j=1}^{N} g(x_j)\right).$$

Here the number of feature variables is $M = N + 1$ by construction. The main difficulty associated with this construction is that the mapping E^{-1} can be arbitrarily complex to be approximated in practice. In fact the construction is similar in flavor to the Kolmogorov-Arnold representation theorem [57], which provides a universal representation for multivariable continuous functions, but without any *a priori* guarantee of the accuracy with respect to the number of parameters.

In order to generalize to the case $d > 1$, the proof of [98, Theorem 9] in fact suggested an alternative proof for the case $d = 1$ as follows. Using the Stone-Weierstrass theorem, a totally symmetric function can be approximated by a polynomial. After symmetrization, this polynomial becomes a totally symmetric polynomial. By the fundamental theorem of symmetric polynomials [69], any symmetric polynomial can be represented by a polynomial of elementary symmetric polynomials. In other words, for any symmetric polynomial P, we have

$$P(X) = Q(e_1(X), \ldots, e_N(X)),$$

where Q is some polynomial, and the elementary polynomials e_k are defined as

$$e_k(X) = \sum_{1 \leq j_1 < j_2 < \cdots < j_k \leq N} x_{j_1} x_{j_2} \cdots x_{j_k}.$$

Using the Newton-Girard formula, an elementary symmetric polynomial can be represented with power sums by

$$e_k(X) = \frac{1}{k!}\begin{vmatrix} E_1(X) & 1 & 0 & 0 & \cdots & 0 \\ E_2(X) & E_1(X) & 2 & 0 & \cdots & 0 \\ E_3(X) & E_2(X) & E_1(X) & 3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ E_{k-1}(X) & E_{k-2}(X) & E_{k-3}(X) & E_{k-4}(X) & \cdots & k-1 \\ E_k(X) & E_{k-1}(X) & E_{k-2}(X) & E_{k-3}(X) & \cdots & E_1(X) \end{vmatrix}. \tag{4.2.2}$$

Now let g be the same function defined in the previous proof, and define ϕ in terms of Q and the determinant computation. We can obtain a polynomial approximation in the form of

$$f(x) = P(X) + \epsilon = \phi\left(\sum_{j=1}^{N} g(x_j)\right) + \epsilon.$$

Here the error ϵ is due to the Stone-Weierstrass approximation. Letting $\epsilon \to 0$ we obtain the desired representation.

However, it is in fact not straightforward to extend the two proofs above to the case $d > 1$. For the first proof, we can not define an ordered set $\mathcal{X}$ when each $\mathbf{x}_i \in \mathbb{R}^d$ to define the homeomorphism E. For the second proof, in the case $d > 1$ a monomial (before symmetrization) takes the form

$$\prod_{i=1}^{N}\prod_{\alpha=1}^{d} \mathbf{x}_{i,\alpha}^{\gamma_{i,\alpha}}, \quad \gamma_{i,\alpha} \in \mathbb{N}.$$

Note that $f(X)$ is only symmetric with respect to the particle index i, but not the component index α. Hence the symmetrized monomial is *not* a totally symmetric function with respect to all variables. Therefore the fundamental theorem of symmetric polynomials does not apply.

4.3 Proof 1 for universal approximation for $d \geq 1$

In this section we prove that the representation (4.2.1) indeed holds for any $d \geq 1$, and therefore we complete the proof of [98]. For technical reasons to be illustrated below, and without loss of generality, we shift the domain Ω and assume $\mathbf{x}_i \in \Omega := [1, 2]^d$. Following the Stone-Weierstrass theorem and after symmetrization, $f(X)$ can be approximated by a symmetric polynomial. Every symmetric polynomial can be

written as the linear combination of symmetrized monomials of the form

$$\sum_{\sigma\in S(N)}\prod_{i=1}^{N}\prod_{\alpha=1}^{d}\mathbf{x}_{\sigma(i),\alpha}^{\gamma_{i,\alpha}}:=\sum_{\sigma\in S(N)}\prod_{i=1}^{N}f_i(\mathbf{x}_{\sigma(i)}):=\mathrm{perm}([f_i(\mathbf{x}_j)]).$$

Here $f_i(\mathbf{x}_j)=\prod_{\alpha=1}^{d}\mathbf{x}_{j,\alpha}^{\gamma_{i,\alpha}}$, $[f_i(\mathbf{x}_j)]$ denotes an $N\times N$ matrix whole entry in the i-th row and j-column is $f_i(\mathbf{x}_j)$, and $\mathrm{perm}(A)$ stands for the permanent of square matrix A.

Following the Ryser formula [84] for representing a permanent (noting that permanent is invariant under transposition), we have

$$\begin{aligned}\mathrm{perm}([f_i(\mathbf{x}_j)]) =&(-1)^N\sum_{S\subseteq\{1,\dots,N\}}(-1)^{|S|}\prod_{i=1}^{N}\sum_{j\in S}f_j(\mathbf{x}_i)\\ =&(-1)^N\sum_{S\subseteq\{1,\dots,N\}}(-1)^{|S|}e^{\sum_{i=1}^{N}\log\left(\sum_{j\in S}f_j(\mathbf{x}_i)\right)}.\end{aligned}$$

Here we have used that $f_j(\mathbf{x})>0$ for all $\mathbf{x}\in\Omega$. Now we write down the approximation using a symmetric polynomial, which is a linear combination of L symmetrized monomials

$$\begin{aligned}f(X)-\epsilon=P(X)=&\sum_{l=1}^{L}c^{(l)}\mathrm{perm}([f_i^{(l)}(\mathbf{x}_j)])\\ =&(-1)^N\sum_{l=1}^{L}c^{(l)}\sum_{S\subseteq\{1,\dots,N\}}(-1)^{|S|}e^{\sum_{i=1}^{N}\log\left(\sum_{j\in S}f_j^{(l)}(\mathbf{x}_i)\right)}\end{aligned}$$

Define $g:\mathbb{R}^d\to\mathbb{R}^{L2^N}$ with each component function

$$g_S^{(l)}(\mathbf{x})=\log\left(\sum_{j\in S}f_j^{(l)}(\mathbf{x})\right).$$

Then we define $\phi:\mathbb{R}^{L2^N}\to\mathbb{R}$ given by

$$\phi(Y)=(-1)^N\sum_{l=1}^{L}c^{(l)}\sum_{S\subseteq\{1,\dots,N\}}(-1)^{|S|}e^{Y_S^{(l)}},$$

where $Y_S^{(l)}$ is the S-th component of $g^{(l)}$. We now have an approximation of the target totally symmetric function in the desired form

$$f(X)=\phi\left(\sum_{j=1}^{N}g(\mathbf{x}_j)\right)+\epsilon,$$

and we finish the proof. Here the number of feature variables is $M = L2^N$, where L is the number of symmetrized monomials used in the approximation.

4.4 Proof 2 for universal approximation for $d \geq 1$

In this section, we prove Theorem 1 for any $d \geq 1$. In particular, our proof is more explicit and does not rely on the Stone-Weierstrass theorem. The main idea is to partition the space into a lattice and use piecewise-constant functions to approximate the target permutation invariant function.

Again without loss of generality we assume $\mathbf{x}_i \in [0,1]^d := \Omega$. We then partition the domain Ω into a lattice $\mathbb{L}$ with grid size δ along each direction. Due to symmetry, we can assign a lexicographical order $\preceq$ to all lattice points $\mathbf{z}_i \in \mathbb{L}$. That is, $\mathbf{z}_1 \preceq \mathbf{z}_2$ if $\mathbf{z}_{1,\alpha} < \mathbf{z}_{2,\alpha}$ for the first α where $\mathbf{z}_{1,i}$ and $\mathbf{z}_{2,i}$ differs, $\alpha = 1, 2, \cdots, d$. We define the tensor product of the N copies of the lattice $\mathbb{L}$ as $\mathbb{L}^N$, and a wedge of $\mathbb{L}^N$ is defined accordingly as

$$\bigwedge^N \mathbb{L} := \{Z = (\mathbf{z}_1, \ldots, \mathbf{z}_N) | \mathbf{z}_1 \preceq \mathbf{z}_2 \cdots \preceq \mathbf{z}_N\}.$$

For each $Z \in \bigwedge^N \mathbb{L}$, a corresponding union of boxes in Ω^N can be written as

$$B^{Z;\delta} = \bigcup_{\sigma \in S(N)} \{X \mid \mathbf{x}_i = \mathbf{z}_{\sigma(i)} + \delta \mathbf{u}_i, \quad \mathbf{u}_i \in [0,1]^d\}.$$

By construction, the piecewise-constant approximation to the target permutation invariant function is then

$$f(X) = \sum_{Z \in \bigwedge^N \mathbb{L}} f(Z) \mathbb{1}_{B^{Z;\delta}}(X) + \epsilon.$$

We assume that the derivative $\nabla_X f(X)$ is uniformly bounded for $X \in \Omega^N$ and denote the bound in 2-norm by $\|\nabla f\|_2$. The maximal distance between two points in a box is bounded by the length of the longest diagonal, which is $\delta\sqrt{Nd}$. Hence the approximation error satisfies $|\epsilon| \leq \|\nabla f\|_2 \delta \sqrt{Nd}$, which is obtained by applying the mean value theorem and the Cauchy-Schwarz inequality. Note that the indicator

function $\mathbb{1}_{B^{Z;\delta}}(X)$ is permutation invariant and can be rewritten as

$$\begin{aligned}\mathbb{1}_{B^{Z;\delta}}(X) =& \frac{1}{C_Z}\sum_{\sigma\in S(N)} \mathbb{1}_{\{X|\mathbf{x}_i=\mathbf{z}_{\sigma(i)}+\delta\mathbf{u}_i,\mathbf{u}_i\in[0,1]^d,1\leq i\leq N\}}(X)\\ =& \frac{1}{C_Z}\sum_{\sigma\in S(N)}\prod_{i=1}^{N}\mathbb{1}_{\{\mathbf{x}_i=\mathbf{z}_{\sigma(i)}+\delta\mathbf{u}_i,\mathbf{u}_i\in[0,1]^d\}}(\mathbf{x}_i)\\ =& \frac{1}{C_Z}\sum_{\sigma\in S(N)}\prod_{i=1}^{N}\mathbb{1}_{\{\mathbf{x}_{\sigma(i)}=\mathbf{z}_i+\delta\mathbf{u}_i,\mathbf{u}_i\in[0,1]^d\}}(\mathbf{x}_{\sigma(i)}) = \frac{1}{C_Z}\mathrm{perm}([f_i^Z(\mathbf{x}_j)]),\end{aligned}$$

where $f_i^Z(\mathbf{x}) = \mathbb{1}_{\{\mathbf{x}|\mathbf{x}=\mathbf{z}_i+\delta\mathbf{u},\mathbf{u}\in[0,1]^d\}}(\mathbf{x})$. The constant C_Z takes care of repetition that can happen depending on Z. When all elements in Z are distinct, the box X lives in only corresponds to one permutation, so $C_Z = 1$ in this case. If say $\mathbf{z}_1 = \mathbf{z}_2$ and all other elements distinct, then the box X lives in has two corresponding permutations that differ by a swapping of the first two elements. In this case, $C_Z = 2$ will account for the arising repetition. Next we apply the Ryser formula to the permanent,

$$\mathbb{1}_{B^{Z;\delta}}(X) = \frac{1}{C_Z}\mathrm{perm}([f_i^Z(\mathbf{x}_j)]) = \frac{(-1)^N}{C_Z}\sum_{S\subseteq\{1,\ldots,N\}}(-1)^{|S|}e^{\sum_{i=1}^N\log\left(\sum_{j\in S}f_j^Z(\mathbf{x}_i)\right)}.$$

We can now define $g:\mathbb{R}^d\to\mathbb{R}^{|S(N)|\times|\bigwedge^N\mathbb{L}|}$ where each component function is given by

$$g_S^Z(\mathbf{x}) = \log\left(\sum_{j\in S}f_j^Z(\mathbf{x})\right)$$

and we define $\phi:\mathbb{R}^{|S(N)|\times|\bigwedge^N\mathbb{L}|}\to\mathbb{R}$ as

$$\phi(Y) = \sum_{Z\in\bigwedge^N\mathbb{L}}\frac{(-1)^N f(Z)}{C_Z}\sum_{S\subseteq\{1,\ldots,N\}}(-1)^{|S|}e^{Y_S^Z}. \tag{4.4.1}$$

Since f_j^Z's are indicator functions we naturally have $\sum_{j\in S}f_j^Z(\mathbf{x})\geq 0$. In the case when $\sum_{j\in S}f_j^Z(\mathbf{x}) = 0$, $g_S^Z(\mathbf{x}) = -\infty$. In this case, $e^{g_S^Z(\mathbf{x})} = 0$, and therefore its contribution to ϕ vanishes as desired. In summary, we arrive at the universal approximation

$$f(X) = \phi\left(\sum_{j=1}^N g(\mathbf{x}_j)\right) + \epsilon. \tag{4.4.2}$$

Due to the explicit tabulation strategy, the number of terms needed in the approximation (4.4.2) can be counted as follows. The number of points in $\bigwedge^N\mathbb{L}$ is

$\mathcal{O}((\delta^{-Nd})/N!)$, where $N!$ comes from the lexicographic ordering. Note that formally as $N \to \infty$, $(\delta^{-Nd})/N! \sim (\delta^{-d}/N)^N$ can vanish for fixed δ. However, this means that the number of elements has exceeded the number of grid points in $\mathbb{L}$ and is unreasonable. So we should at least have $\delta \lesssim N^{-\frac{1}{d}}$. In order to obtain an ϵ-close approximation of $f(X)$, we require our error bound $\|\nabla f\|_2 \delta \sqrt{Nd} \sim \epsilon$. When $\epsilon \leq \|\nabla f\|_2 \sqrt{Nd} N^{-\frac{1}{d}}$, we can choose $\delta = \frac{\epsilon}{\|\nabla f\|_2}(Nd)^{-\frac{1}{2}}$ so that both $\delta \lesssim N^{-\frac{1}{d}}$ and $\|\nabla f\|_2 \delta \sqrt{Nd} \sim \epsilon$ are fulfilled, and the number of points in $\bigwedge^N \mathbb{L}$ becomes $(\|\nabla f\|_2^2 Nd)^{Nd/2}/(\epsilon^{Nd} N!)$ because $\delta^{-Nd} = (\|\nabla f\|_2^2 Nd)^{Nd/2}/\epsilon^{Nd}$. For each Z, the number of terms to be summed over in Eq. (4.4.1) is $|S(N)| = 2^N$. Therefore in order to obtain an ϵ-approximation, the number of feature variables is given by Eq. (4.1.3). This proves Theorem 1.

This is of course a very pessimistic bound, and we will discuss on the practical implications for designing neural network architectures in Section 4.5. We remark that one may expect that following the same tabulation strategy, we may also provide a quantitative bound for ϕ constructed by the homeomorphism mapping E^{-1} as discussed in Section 4.2. However, the difference is that our bound only relies on the smoothness of the original function f and hence the bound for δ. On the other hand, the mapping E^{-1} and hence ϕ can be arbitrarily pathological, and therefore it is not even clear how to obtain a double-exponential type of bound as discussed above. We also remark that if the indicator functions $\mathbb{1}_{B^{Z;\delta}}(X)$ and $\mathbb{1}_{\{\mathbf{x}|\mathbf{x}=\mathbf{z}_i+\delta\mathbf{u},\mathbf{u}\in[0,1]^d\}}(\mathbf{x})$ in the proof are replaced by proper smooth cutoff functions with respect to corresponding domains, the ansatz in Eq. (4.4.2) can be continuous to accommodate the applications that require continuity. For example, an interatomic potential energy should be continuous to guarantee the total energy is conserved during molecular dynamics simulations.

4.5 Conclusion

In this chapter we study the universal approximation for symmetric functions. Following the line of learning theory, there are many questions open. For instance, the impact of symmetry on the generalization error remains unclear. This requires in-depth understanding of the suitable function class for symmetric functions, such as some adapted Barron space [27]. Note that a recent work [85] investigates the approximation and generalization bound of permutation invariant deep neural networks in the general case $d \geq 1$, however with two limitations. The first is a rather strong assumption that the target function is Lipschitz with respect to the ℓ_∞ norm (but not the usual Euclidean norm). This can be a severe limitation as the dimension (both N and d) increases. Indeed, under the same Lipschitz assumption of [85], the number of feature variables M in our Theorem 1 can be improved accordingly to

$\mathcal{O}\left(2^N/(\epsilon^{Nd}N!)\right)$. The second limitation is that the proposed ansatz in [85] introduces *sorting layers* to represent the sorting procedure at the first step. The sorting procedure will bring discontinuity, which leads to serious problems in some scientific applications, such as interatomic potential energy in molecular dynamics simulations.

Bibliography

[1] Martın Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning.” In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] John M. Alred et al. “Machine learning electron density in sulfur crosslinked carbon nanotubes”. In: *Composites Science and Technology* 166 (2018). Carbon nanotube composites for structural applications, pp. 3–9. ISSN: 0266-3538. DOI: https://doi.org/10.1016/j.compscitech.2018.03.035. URL: http://www.sciencedirect.com/science/article/pii/S0266353817330300.

[3] D. G. Anderson. “Iterative Procedures for Nonlinear Integral Equations”. In: *J. ACM* 12.4 (Oct. 1965), pp. 547–560. ISSN: 0004-5411. DOI: 10.1145/321296.321305. URL: http://doi.acm.org/10.1145/321296.321305.

[4] Sanjeev Arora et al. “Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks”. In: (). eprint: 1901.08584v1.

[5] Francis Bach. “Breaking the curse of dimensionality with convex neural networks”. In: *J. Mach. Learn. Res.* 18.1 (2017), pp. 629–681.

[6] Markus Bachmayr, Geneviève Dusson, and Christoph Ortner. “Polynomial Approximation of Symmetric Functions”. In: *arXiv preprint arXiv:2109.14771* (2021).

[7] Andrew R Barron. “Universal approximation bounds for superpositions of a sigmoidal function”. In: *IEEE Trans. Inform. Theory* 39 (1993), pp. 930–945.

[8] Rodney J Bartlett and Monika Musiał. “Coupled-cluster theory in quantum chemistry”. In: *Rev. Mod. Phys.* 79.1 (2007), p. 291.

[9] Albert P Bartók et al. “Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons”. In: *Physical Review Letters* 104.13 (2010), p. 136403.

[10] J. Behler and M. Parrinello. "Generalized neural-network representation of high-dimensional potential-energy surfaces". In: *Phys. Rev. Lett.* 98.14 (2007), p. 146401.

[11] Davis Blalock et al. *What is the State of Neural Network Pruning?* 2020. arXiv: 2003.03033 [cs.LG].

[12] P. E. Blöchl. "Projector augmented-wave method". In: *Phys. Rev. B* 50 (1994), p. 17953.

[13] Mihail Bogojeski et al. "Efficient prediction of 3D electron densities using machine learning". In: *arXiv:1811.06255* (2018).

[14] D. R. Bowler and T. Miyazaki. "$O(N)$ methods in electronic structure calculations". In: *Rep. Prog. Phys.* 75 (2012), p. 036503.

[15] Felix Brockherde et al. "Bypassing the Kohn-Sham equations with machine learning". In: *Nature Commun.* 8.1 (2017), p. 872.

[16] A. Chandrasekaran et al. "Solving the electronic structure problem with machine learning". In: *npj Computational Materials* 5.1 (2019), p. 22. DOI: 10.1038/s41524-019-0162-7. URL: https://doi.org/10.1038/s41524-019-0162-7.

[17] L. Cheng et al. *Thermalized (350K) QM7b, GDB-13, water, and short alkane quantum chemistry dataset including MOB-ML features*. eng. 2019. DOI: 10.22002/d1.1177. URL: https://data.caltech.edu/records/1177.

[18] S. Chmiela et al. "Machine learning of accurate energy-conserving molecular force fields". In: *Science Advances* 3.5 (2017), e1603015.

[19] François Chollet et al. *Keras*. https://keras.io. 2015.

[20] Nadav Cohen, Or Sharir, and Amnon Shashua. "On the expressive power of deep learning: A tensor analysis". In: *Conference on Learning Theory*. 2016, pp. 698–728.

[21] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314.

[22] Stephane d'Ascoli et al. "Finding the Needle in the Haystack with Convolutions: on the benefits of architectural bias". In: (). eprint: 1906.06766v1.

[23] Genevieve Dusson et al. "Atomic cluster expansion: Completeness, efficiency and stability". In: *arXiv preprint arXiv:1911.03550* (2019).

[24] Weinan E, Jiequn Han, and Linfeng Zhang. "Machine-learning-assisted modeling". In: *Physics Today* 74.7 (July 2021), pp. 36–41.

[25] Weinan E, Chao Ma, and Qingcan Wang. "A Priori Estimates of the Population Risk for Residual Networks". In: *arXiv:1903.02154* (2019).

[26] Weinan E, Chao Ma, and Lei Wu. "A Priori Estimates for Two-layer Neural Networks". In: *arXiv: 1810.06397* (2018), pp. 1–14.

[27] Weinan E, Chao Ma, and Lei Wu. "Barron Spaces and the Compositional Function Spaces for Neural Network Models". In: *arXiv:1906.08039* (2019).

[28] Adolfo G Eguiluz. "Self-consistent static-density-response function of a metal surface in density-functional theory". In: *Phys. Rev. B* 31.6 (1985), p. 3303.

[29] Alberto Fabrizio et al. "Electron density learning of non-covalent systems". In: *Chem Sci.* (2019).

[30] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations (ICLR)* (2019). arXiv preprint arXiv:1803.03635.

[31] Jonathan Frankle et al. "The Lottery Ticket Hypothesis at Scale". In: *arXiv preprint arXiv:1903.01611* (2019).

[32] C. Daniel Freeman and Joan Bruna. "Topology and geometry of half-rectified network optimization". In: *ICLR*. arXiv preprint arXiv:1611.01540. 2017.

[33] D. Frenkel and B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, 2002.

[34] Maximilien Germain et al. "DeepSets and their derivative networks for solving symmetric PDEs". In: *arXiv preprint arXiv:2103.00838* (2021).

[35] S. Goedecker. "Linear scaling electronic structure methods". In: *Rev. Mod. Phys.* 71 (1999), pp. 1085–1123.

[36] X. Gonze and C. Lee. "Dynamical matrices, Born effective charges, dielectric permittivity tensors, and interatomic force constants from density-functional perturbation theory". In: *Phys. Rev. B* 55 (1997), p. 10355.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[38] A. Grisafi et al. "Transferable Machine-Learning Model of the Electron Density". In: *ACS Central Science* 5.1 (2018), pp. 57–64.

[39] Andrea Grisafi et al. "Symmetry-adapted machine learning for tensorial properties of atomistic systems". In: *Physical Review Letters* 120.3 (2018), p. 036002.

[40] D. R. Hamann. "Optimized norm-conserving Vanderbilt pseudopotentials". In: *Phys. Rev. B* 88 (2013), p. 085117.

[41] J. Han, A. Jentzen, and W. E. "Solving high-dimensional partial differential equations using deep learning". In: *Proceedings of the National Academy of Sciences* 115.34 (2018), pp. 8505–8510.

[42] J. Han et al. "Deep Potential: a general representation of a many-body potential energy surface". In: *Comms. Comp. Phys.* 23.3 (2018), pp. 629–639.

[43] Jiequn Han et al. "Universal approximation of symmetric and anti-symmetric functions". In: *arXiv preprint arXiv:1912.01765* (2019).

[44] Kaiming He and Jian Sun. "Convolutional neural networks at constrained time cost". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 5353–5360.

[45] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[46] G. Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2205597.

[47] Pierre Hohenberg and Walter Kohn. "Inhomogeneous electron gas". In: *Physical review* 136.3B (1964), B864.

[48] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T.

[49] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[50] W. Hu, L. Lin, and C. Yang. "DGDFT: A massively parallel method for large scale density functional theory calculations". In: *J. Chem. Phys.* 143 (2015), p. 124110.

[51] Kenji Kawaguchi. "Deep learning without poor local minima". In: *arXiv preprint arXiv:1605.07110*. 2016.

[52] V. Khrulkov, A. Novikov, and I. Oseledets. "Expressive power of recurrent neural networks". In: *arXiv:1711.00811* (2017).

[53] D. Kingma and J Ba. "Adam: a method for stochastic optimization". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. May 2015.

[54] Jason M. Klusowski and Andrew R. Barron. "Risk Bounds for High-dimensional Ridge Function Combinations Including Neural Networks". In: (). eprint: 1607.01434v4.

[55] W. Kohn. "Density Functional and Density Matrix Method Scaling Linearly with the Number of Atoms". In: *Phys. Rev. Lett.* 76 (1996), pp. 3168–3171.

[56] Walter Kohn and Lu Jeu Sham. "Self-consistent equations including exchange and correlation effects". In: *Physical review* 140.4A (1965), A1133.

[57] A. N. Kolmogorov. "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition". In: *Doklady Akademii Nauk*. Vol. 114. 5. 1957, pp. 953–956.

[58] G. Kresse and J. Furthmüller. "Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set". In: *Phys. Rev. B* 54 (1996), pp. 11169–11186.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[60] Rohith Kuditipudi et al. "Explaining Landscape Connectivity of Low-cost Solutions for Multilayer Net". In: *NeurIPS*. arXiv preprint arXiv:161. 2019.

[61] Tejas D Kulkarni et al. "Deep Convolutional Inverse Graphics Network". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2539–2547. URL: http://papers.nips.cc/paper/5851-deep-convolutional-inverse-graphics-network.pdf.

[62] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), p. 436.

[63] Michael K. K. Leung et al. "Deep learning of the tissue-regulated splicing code". In: *Bioinformatics* 30.12 (2014), pp. i121–i129. DOI: 10.1093/bioinformatics/btu277.

[64] L. Lin and C. Yang. "Elliptic preconditioner for accelerating self consistent field iteration in Kohn-Sham density functional theory". In: *SIAM J. Sci. Comp.* 35 (2013), S277–S298.

[65] Lin Lin, Jianfeng Lu, and Lexing Ying. "Numerical methods for Kohn–Sham density functional theory". In: *Acta Numer.* 28 (2019), pp. 405–539.

[66] Zhuang Liu et al. "Rethinking the Value of Network Pruning". In: *ICLR*. arXiv preprint arXiv:1810.05270. 2019.

[67] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. "On the Computational Efficiency of Training Neural Networks". In: *Advances in Neural Information Processing Systems* (2014). arXiv preprint arXiv:1410.1141.

[68] Junshui Ma et al. "Deep Neural Nets as a Method for Quantitative Structure-Activity Relationships". In: *Journal of Chemical Information and Modeling* 55.2 (2015), pp. 263–274. DOI: 10.1021/ci500747n.

[69] Ian Grant Macdonald. *Symmetric functions and Hall polynomials*. Oxford Univ. Pr, 1998.

[70] R. Martin. *Electronic Structure: Basic Theory and Practical Methods*. Cambridge Univ. Pr., 2008.

[71] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[72] N.D. Mermin. "Thermal properties of the inhomogeneous electron gas". In: *Phys. Rev.* 137 (1965), A1441.

[73] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. "Learning Functions: When Is Deep Better Than Shallow". In: *arXiv preprint arXiv:1603.00988* (2016).

[74] H. J. Monkhorst and J. D. Pack. "Special points for Brillouin-zone integrations". In: *Phys. Rev. B* 13.12 (1976), p. 5188.

[75] G. Montavon et al. "Machine learning of molecular electronic properties in chemical compound space". In: *New Journal of Physics* 15.9 (2013), p. 095003.

[76] Vaishnavh Nagarajan and J Zico Kolter. "Uniform convergence may be unable to explain generalization in deep learning". In: *Advances in Neural Information Processing Systems*. 2019, pp. 11611–11622.

[77] J. P. Perdew, K. Burke, and M. Ernzerhof. "Generalized gradient approximation made simple". In: *Phys. Rev. Lett.* 77 (1996), pp. 3865–3868.

[78] E. Prodan and W. Kohn. "Nearsightedness of electronic matter". In: *Proc. Natl. Acad. Sci.* 102 (2005), pp. 11635–11638.

[79] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3D classification and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 652–660.

[80] Charles Ruizhongtai Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5099–5108.

[81] David E. Rumelhart and James L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[82] M. Rupp et al. "Fast and accurate modeling of molecular atomization energies with machine learning". In: *Phys. Rev. Lett.* 108.5 (2012), p. 058301.

[83] Kevin Ryczko, David A Strubbe, and Isaac Tamblyn. "Deep learning and density-functional theory". In: *Physical Review A* 100.2 (2019), p. 022512.

[84] Herbert John Ryser. *Combinatorial Mathematics*. Vol. 14. The Carus Mathematical Monographs. Mathematical Association of America, 1963.

[85] Akiyoshi Sannai and Masaaki Imaizumi. "Improved Generalization Bound of Permutation Invariant Deep Neural Networks". In: *arXiv preprint arXiv:1910.06552* (2019).

[86] Akiyoshi Sannai, Yuuki Takai, and Matthieu Cordonnier. "Universal approximations of permutation invariant/equivariant functions by deep neural networks". In: *arXiv preprint arXiv:1903.01939* (2019).

[87] M. Schlipf and F. Gygi. "Optimization algorithm for the generation of ONCV pseudopotentials". In: *Comput. Phys. Commun.* 196 (2015), pp. 36–44.

[88] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003.

[89] B. Scholkopf et al. "Input space versus feature space in kernel-based methods". In: *IEEE Transactions on Neural Networks* 10.5 (1999), pp. 1000–1017.

[90] K. Schütt et al. "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions". In: *Advances in Neural Information Processing Systems*. 2017, pp. 992–1002.

[91] J. S. Smith, O. Isayev, and A. E. Roitberg. "ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost". In: *Chemical Science* 8.4 (2017), pp. 3192–3203.

[92] L. Venturi, A. S. Bandeira, and Joan Bruna. "Spurious valleys in two-layer neural network optimization landscapes". In: *arXiv preprint arXiv:1802.06384*. 2018.

[93] H Wang et al. "DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics". In: *Computer Physics Communications* 228 (2018), pp. 178–184. ISSN: 0010-4655.

[94] Yuting Wei, Fanny Yang, and Martin J. Wainwright. "Early stopping for kernel boosting algorithms: A general analysis with localized complexities". In: *NIPS*. arXiv preprint arXiv:1707.01543. 2017.

[95] S. R. White. "Density matrix formulation for quantum renormalization groups". In: *Phys. Rev. Lett.* 69.19 (1992), p. 2863.

[96] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. "On Early Stopping in Gradient Descent Learning". In: *Constructive Approximation* 26 (Aug. 2007), pp. 289–315. DOI: 10.1007/s00365-006-0663-2.

[97] Dmitry Yarotsky. "Error bounds for approximations with deep ReLU networks". In: *Neural Networks* 94 (2017), pp. 103–114.

[98] Manzil Zaheer et al. "Deep sets". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3391–3401.

[99] Manzil Zaheer et al. "Deep sets". In: *Advances in neural information processing systems*. 2017, pp. 3391–3401.

[100] Leonardo Zepeda-Núñez et al. "Deep Density: Circumventing the Kohn-Sham equations via symmetry preserving neural networks". In: *Journal of Computational Physics* 443 (June 2021), p. 110523. DOI: 10.1016/j.jcp.2021.110523.

[101] Jiefu Zhang et al. "Learning the Mapping

$$\mathbf{x} \mapsto \sum_{i=1}^{d} x_i^2$$

: the Cost of Finding the Needle in a Haystack". In: *Communications on Applied Mathematics and Computation* 3 (Aug. 2020). DOI: 10.1007/s42967-020-00078-2.

[102] L. Zhang et al. "Active learning of uniformly accurate interatomic potentials for materials simulation". In: *Phys. Rev. Materials* 3.2 (2019), p. 023804.

[103] L. Zhang et al. "Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics". In: *Phys. Rev. Lett.* 120 (2018), p. 143001.

[104] Linfeng Zhang et al. "DeePCG: constructing coarse-grained models via deep neural networks". In: *arXiv preprint arXiv:1802.08549* (2018).

[105] Linfeng Zhang et al. "End-to-end Symmetry Preserving Inter-atomic Potential Energy Model for Finite and Extended Systems". In: *Advances in Neural Information Processing Systems 31*. 2018, pp. 4436–4446.

[106] Xu-Hui Zhou, Jiequn Han, and Heng Xiao. "Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids". In: *Computer Methods in Applied Mechanics and Engineering* 388 (2022), p. 114211.

[107] Aaron Zweig and Joan Bruna. "A Functional Perspective on Learning Symmetric Functions with Neural Networks". In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139. PMLR, 18–24 Jul 2021, pp. 13023–13032.