# Approximation Power of Deep Neural Networks An explanatory mathematical survey

Mohammad Motamed<sup>\*1</sup>

<sup>1</sup>Department of Mathematics and Statistics, University of New Mexico, Albuquerque, USA

Dedicated to Raúl Tempone

## Abstract

The goal of this survey is to present an explanatory review of the approximation properties of deep neural networks. Specifically, we aim at understanding how and why deep neural networks outperform other classical linear and nonlinear approximation methods. This survey consists of three chapters. In Chapter 1 we review the key ideas and concepts underlying deep networks and their compositional nonlinear structure. We formalize the neural network problem by formulating it as an optimization problem when solving regression and classification problems. We briefly discuss the stochastic gradient descent algorithm and the back-propagation formulas used in solving the optimization problem and address a few issues related to the performance of neural networks, including the choice of activation functions, cost functions, overfitting issues, and regularization. In Chapter 2 we shift our focus to the approximation theory of neural networks. We start with an introduction to the concept of density in polynomial approximation and in particular study the Stone-Weierstrass theorem for real-valued continuous functions. Then, within the framework of linear approximation, we review a few classical results on the density and convergence rate of feedforward networks, followed by more recent developments on the complexity of deep networks in approximating Sobolev functions. In Chapter 3, utilizing nonlinear approximation theory, we further elaborate on the power of depth and approximation superiority of deep ReLU networks over other classical methods of nonlinear approximation.

*keywords:* artificial neural networks, approximation theory, power of depth, mathematics of deep learning

**Intended audience.** The material in this survey should be accessible to undergraduate and graduate students and researchers in mathematics, statistics, computer science, and engineering, and to all those who wish to obtain a "deeper" understanding of "deep" neural networks.

<sup>\*</sup>motamed@unm.edu

# Contents

Ρ	rologue: What this survey is and is not about	4
С	hapter: 1 Neural networks: formalization and key concepts	<b>5</b>
1	What is a neural network?	<b>5</b>
<b>2</b>	What is the use of a neural network?	7
3	Network training is an optimization problem	7
4	Solving the optimization problem4.1Gradient descent4.2Backpropagation4.3Mini-batch stochastic GD and backpropagation	<b>8</b> 8 9 12
5	Activation functions5.1 Desirable properties of activation functions5.2 Popular choices of activation functions	<b>13</b> 13 14
6	Loss functions	16
7	Overfitting and regularization7.1Adding penalty terms7.2Early stopping7.3Dropout	17 18 19 19
8	Validation and hyper-parameter tuning	20
9	Procedure summary	<b>21</b>
С	hapter: 2 Deep neural networks and linear approximation	22
10	Introduction	22
11	Density in polynomial approximation11.1 Uniform convergence11.2 Weierstrass approximation theorem11.3 Bernstein polynomials11.4 A constructive proof of Weierstrass theorem11.5 Stone-Weierstrass theorem	<ul> <li>23</li> <li>24</li> <li>24</li> <li>26</li> <li>27</li> </ul>
12	Density of two-layer networks12.1 Two-layer feedforward networks12.2 Pinkus theorem	<b>28</b> 28 29

	12.3 Proof sketch of Pinkus theorem 29
13	Convergence rate of approximation by two-layer networks3213.1 Target functions of interest3313.2 Sobolev spaces34
	13.3 A typical convergence rate result       33         13.4 A short discussion on curse of dimension       37
<b>14</b>	Complexity of deep ReLU networks 39
	14.1 Target functions of interest
	14.2 Main complexity result
	14.3 Proof strategy
	14.4 Averaged Taylor polynomial approximation         4
	14.5 Partition of unity $\ldots \ldots $
	14.6 Construction of a global polynomial approximant
	14.7 Expressive power of ReLU networks: an intuitive example
	14.8 Connected and standard ReLU networks
	14.9 Approximating product of two numbers by ReLU networks
	14.10ReLU networks and the product of their output components
	14.11ReLU networks and the partition of unity
	14.12Approximating global polynomials by ReLU networks
	14.13Proof of Complexity Theorem
	14.14Further reading and nonlinear approximation $\ldots \ldots \ldots$
Cł	hapter: 3 Deep neural networks and nonlinear approximation 63
15	Introduction 65
16	Standard ReLU networks and free knot linear splines 65
	16.1 Space of standard ReLU networks
	16.2 Space of free knot linear splines
	16.3 A comparison between one-layer ReLU networks and free knot linear splines 6'
	16.4 A comparison between deep ReLU networks and free knot linear splines 68
17	ReLU networks are at least as expressive as free knot linear splines 68
	17.1 Special ReLU networks
	17.2 Special ReLU networks that realize continuous piecewise linear functions $\dots$ 7
	17.3 Proof of Theorem 1
18	The power of depth 75
-	18.1 Additive and compositional properties of ReLU networks
	18.2 A large class of functions with self-similarity

# Prologue: What this survey is and is not about

Three major questions about deep neural networks include:

- 1. Approximation theory/property: Given a target function space and a neural network architecture, what is the best the network can do in approximating the target function? Are there function spaces in which certain neural network architectures can outperform other methods of approximation?
- 2. Learning process: Given a neural network architecture and a data set, how to efficiently train a generalizable network? How/when/why optimization techniques (such as stochastic gradient descent or random sampling) work?
- 3. Optimal experimental design: How to choose/sample and generate data? How to achieve/construct the best (or close to best) approximation with the least amount of data and work?

This survey focuses only on the first question and does not address/discuss the second and third questions. The main goal of this survey is to present an explanatory review of the approximation properties of deep (feedforward) neural networks.

The survey consists of three chapters. In Chapter 1 we review the key ideas and concepts underlying deep networks and their compositional nonlinear structure. We formalize the neural network problem by formulating it as an optimization problem when solving regression and classification problems. We briefly discuss the stochastic gradient descent algorithm and the back-propagation formulas used in solving the optimization problem and address a few issues related to the performance of neural networks, including the choice of activation functions, cost functions, overfitting issues, and regularization. In Chapter 2 we focus on the approximation theory of neural networks. We start with an introduction to the concept of density in polynomial approximation and in particular study the Stone-Weierstrass theorem for real-valued continuous functions. Then, within the framework of linear approximation, we review a few classical results on the density and convergence rate of feedforward networks, followed by more recent developments on the complexity of deep networks in approximating Sobolev functions. Finally, in Chapter 3, utilizing nonlinear approximation theory, we further elaborate on the power of depth and approximation superiority of deep ReLU networks over other classical methods of nonlinear approximation.

## Chapter 1: Neural networks: formalization and key concepts

In this chapter we introduce the key ideas and concepts underlying artificial neural networks in the context of supervised learning and with application to solving regression and classification problems. We start with the compositional nonlinear structure of networks and formulate the network problem as an optimization problem. We will further discuss the (stochastic) gradient descent algorithm and will derive the backpropagation formulas used in solving the optimization problem. Finally, we will briefly address a few topics related to the performance of neural networks, including the choice of activation functions, cost functions, overfitting issues, and regularization.

## 1 What is a neural network?

A neural network (NN) is a map

$$\mathbf{f}_{\boldsymbol{\theta}}: \mathbb{R}^{n_{\mathrm{in}}} \to \mathbb{R}^{n_{\mathrm{out}}},$$

with a particular compositional structure; see (1)-(2)-(3) below. Here,  $n_{\text{in}} \in \mathbb{N}$  and  $n_{\text{out}} \in \mathbb{N}$ are the dimension of the input and output spaces, respectively, and  $\boldsymbol{\theta} \in \mathbb{R}^{n_{\theta}}$  is the vector of network parameters. The network map is formed by the composition of  $L \geq 1$  maps

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{f}^L \circ \cdots \circ \mathbf{f}^1(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^{n_{\text{in}}}, \tag{1}$$

where each individual map  $\mathbf{f}^{\ell} : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_{\ell}}$ , with  $\ell = 1, \ldots, L$ , and  $n_0 = n_{\text{in}}$ , and  $n_L = n_{\text{out}}$ , is given by the component-wise application of a nonlinear activation function  $\sigma_{\ell}$  to a multidimensional linear (or affine) transformation

$$\mathbf{f}^{\ell}(\mathbf{z}) = \sigma_{\ell}(W^{\ell}\,\mathbf{z} + \mathbf{b}^{\ell}), \qquad \mathbf{z} \in \mathbb{R}^{n_{\ell-1}}, \quad W^{\ell} \in \mathbb{R}^{n_{\ell} \times n_{\ell-1}}, \quad \mathbf{b}^{\ell} \in \mathbb{R}^{n_{\ell}}, \quad \ell = 1, \dots, L.$$
(2)

The parameters  $W^{\ell}$  and  $\mathbf{b}^{\ell}$  that define the linear map transformation at level (or layer)  $\ell$  are referred to as the weights (or edge weights) and biases (or node weights) of the  $\ell$ -th level, respectively. The parameter vector  $\boldsymbol{\theta}$  then consists of all weights and biases:

$$\boldsymbol{\theta} = \{ (W^{\ell}, \mathbf{b}^{\ell}) \}_{\ell=1}^{L}.$$
(3)

The total number of network parameters  $n_{\theta}$  can then be obtained in terms of the number of neurons and layers, given by  $n_{\theta} = \sum_{\ell=1}^{L} n_{\ell} (n_{\ell-1} + 1)$ .

A neural network with a given set of activation functions is uniquely determined by its parameter vector  $\boldsymbol{\theta}$ . The manapeter vector is *tuned* during a process referred to as *training* so that  $\mathbf{f}_{\boldsymbol{\theta}}$  does what it is supposed to do; we will discuss the training process in Sections 3-4.

The compositional structure (1)-(2) can be represented by an artificial neural network, with one input layer consisting of  $n_0 = n_{\rm in}$  neurons, one output layer consisting of  $n_L = n_{\rm out}$ neurons, and  $L - 1 \in \mathbb{N}$  hidden layers consisting of  $n_1, \ldots, n_{L-1} \in \mathbb{N}$  neurons, respectively. Each layer represents an individual map (2). We refer to the number of layers, neurons, and their corresponding activation functions as the "architecture" of the network. Note that different layers of a network may have different number of neurons and may use different activation functions. Figure 1 shows a graph representation of the network, where each node represents a neuron, and each edge connecting two nodes represents a multiplication by a scalar weight. The input neurons (or nodes) take the  $n_{\rm in}$  components of the independent variable vector  $\mathbf{x} = (x_1, \ldots, x_{n_{\rm in}})$ , and the output neurons produce the  $n_{\rm out}$  components of  $\mathbf{f}_{\boldsymbol{\theta}} = (f_{\boldsymbol{\theta},1}, \ldots, f_{\boldsymbol{\theta},n_{\rm out}})$ .



Figure 1: Graph representation of a feed-forward network with L = 3 layers (2 hidden layers and 1 output layer).

The number of layers L in a network determines the "depth" of the network: the larger L, the deeper the network. The number of neurons  $n_{\ell}$  in each layer  $\ell$  determines the "width" of that layer and hence the width of the network: the larger the number of neurons in each layer, the "wider" the network. We can increase the number of a network's parameters, and hence make it more complex, by either making it wider or deeper or both. The larger the number of a network's structure.

Let us fix a few notations:

- $W_{j,k}^{\ell}$  = weight of the connection from neuron k in layer  $\ell 1$  to neuron j in layer  $\ell$
- +  $b_j^\ell$  = bias of neuron j in layer  $\ell$
- $a_j^{\ell}$  = the output of neuron j in layer  $\ell$  (i.e. after applying activation), defined as  $a_j^{\ell} = \sigma_{\ell} \Big( \sum_k W_{j,k}^{\ell} a_k^{\ell-1} + b_j^{\ell} \Big),$  sum is taken over all neurons k in layer  $\ell - 1$
- $z_j^{\ell}$  = the input of neuron j in layer  $\ell$  (i.e. before applying activation), defined as  $z_j^{\ell} = \sum_k W_{j,k}^{\ell} a_k^{\ell-1} + b_j^{\ell}, \qquad \text{sum is taken over all neurons } k \text{ in layer } \ell 1.$

Note: we have  $a_j^\ell = \sigma_\ell(z_j^\ell)$ .

• In vector-matrix form, if we introduce the following matrices and vectors,

$$W^{\ell} = (W_{j,k}^{\ell}) \in \mathbb{R}^{n_{\ell} \times n_{\ell-1}},$$
  

$$\mathbf{b}^{\ell} = (b_j^{\ell}) \in \mathbb{R}^{n_{\ell}},$$
  

$$\mathbf{a}^{\ell} = (a_j^{\ell}) \in \mathbb{R}^{n_{\ell}},$$
  

$$\mathbf{z}^{\ell} = (z_j^{\ell}) \in \mathbb{R}^{n_{\ell}},$$

then we can write

$$\mathbf{z}^{\ell} = W^{\ell} \, \mathbf{a}^{\ell-1} + \mathbf{b}^{\ell}, \qquad \mathbf{a}^{\ell} = \sigma_{\ell}(\mathbf{z}^{\ell}).$$

As an example, a simple network with  $n_{in} = n_{out} = 1$  and L = 2 layers (1 hidden layer and 1 output layer) with  $n_1 = 1$  neuron in the hidden layer reads

$$f_{\theta}(x) = f_2(f_1(x)) = \sigma_2(W_{1,1}^2(\sigma_1(W_{1,1}^1x + b_1^1)) + b_1^2), \qquad x \in \mathbb{R}$$

- The first layer  $f_1$  takes the input x, linearly transforms it into  $W_{1,1}^1 x + b_1^1$ , and applies an activation function  $\sigma_1$  to return  $\sigma_1(W_{1,1}^1 x + b_1^1)$ .
- The second (output) layer  $f_2$  takes the output of the first layer  $\sigma_1(W_{1,1}^1x + b_1^1)$ , linearly transforms it into  $W_{1,1}^2\sigma_1(W_{1,1}^1x + b_1^1) + b_1^2$ , and applies an activation function  $\sigma_2$  to return  $\sigma_2(W_{1,1}^2(\sigma_1(W_{1,1}^1x + b_1^1)) + b_1^1)$ .
- If we collect the weights and biases in a parameter vector as  $\boldsymbol{\theta} = (W_{1,1}^1, W_{1,1}^2, b_1^1, b_1^2) = (\theta_1, \theta_2, \theta_3, \theta_4)$ , then we have

$$f_{\theta}(x) = \sigma_2(\theta_2(\sigma_1(\theta_1 x + \theta_3)) + \theta_4).$$

This is an interesting non-linear function with four parameters.

## 2 What is the use of a neural network?

Neural Networks (in supervised learning) are today widely used for solving two types of problems: 1) multivariate regression, and 2) classification.

**Regression:** In multivariate regression, for a given set of input-output data

$$\{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}_{m=1}^n \in \mathbb{R}^{n_{\text{in}}} \times \mathbb{R}^{n_{\text{out}}},$$

we want to construct a multivariate map  $\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$  such that  $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}) \approx \mathbf{y}^{(m)}$ , without overfitting; we will discuss overfitting in Section 7.

**Classification.** In classification the data are labeled into  $n_{\text{out}}$  categories  $c_1, \ldots, c_{n_{\text{out}}}$ . Given an input set of data  $\{\mathbf{x}^{(m)}\}_{m=1}^n \in \mathbb{R}^{n_{\text{in}}}$  with their corresponding categories, say  $\{c^{(m)}\}_{m=1}^n$ where  $c^{(m)} \in \{c_1, \ldots, c_{n_{\text{out}}}\}$ , we want to construct a map  $\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^{n_{\text{in}}} \to [0, 1]^{n_{\text{out}}}$  that returns the posterior probabilities of category membership for any observed pattern  $\mathbf{x} \in \mathbb{R}^{n_{\text{in}}}$ 

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) \approx (\operatorname{Prob}(c_1|\mathbf{x}), \dots, \operatorname{Prob}(c_{n_{\text{out}}}|\mathbf{x})).$$

## 3 Network training is an optimization problem

Network training is the process of finding (or adjusting) the parameters of a network. Given n pairs of input-output training data points  $\{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}_{m=1}^{n}$ , our goal is to *train* a network with pre-assigned architecture that *learns* the data. That is, we want to find a parametric map  $\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$ , i.e. to find a set of network parameters  $\boldsymbol{\theta} = \{(W^{\ell}, \mathbf{b}^{\ell})\}_{\ell=1}^{L}$ , such that

 $\mathbf{f}_{\theta}(\mathbf{x})$  well approximates the target  $\mathbf{y}$ . This is done by minimizing a prescribed loss (or cost or misfit or risk) function, denoted by C, that measures the "distance" between  $\mathbf{f}_{\theta}(\mathbf{x})$  and  $\mathbf{y}$  for the training data set. Training can therefore be formulated as an optimization problem:

Find 
$$\boldsymbol{\theta}$$
 as the solution of  $\arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{m=1}^{n} C_m(\boldsymbol{\theta}), \qquad C_m(\boldsymbol{\theta}) = C(\mathbf{y}^{(m)}, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(m)})).$  (4)

Here,  $C(\mathbf{a}, \mathbf{b})$  is a cost function that measures the distance between vectors  $\mathbf{a}$  and  $\mathbf{b}$ . For instance, a typical cost function is the quadratic cost function  $C(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}||_2^2$ , obtained by squaring the  $L^2$  norm of differences.

We note that training is usually referred to as the process of finding  $\boldsymbol{\theta}$ , given training data, network architecture, and the cost function. However, in practice the selection of training data (if not given) and network architecture and cost function may also be considered as parts of the training process.

## 4 Solving the optimization problem

The optimization problem (4) is often solved by a gradient-based method, such as stochastic gradient descent [26, 16] or Adam [17]. In these methods the gradient of the cost function with respect to the network parameters is usually computed by the chain rule using a differentiation technique known as *back propagation* [27]. Refer to the review paper [4] for more details.

### 4.1 Gradient descent

Gradient descent (GD), or steepest descent, is an iterative method in optimization. Given a data batch  $\{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}_{m=1}^{n}$  (in multivariate regression problems) and a fixed network architecture with unknown parameters  $\boldsymbol{\theta}$ , we want to find  $\boldsymbol{\theta}$  that minimizes an "empirical" risk

$$R_n(\boldsymbol{\theta}) := \frac{1}{n} \sum_{m=1}^n C_m(\boldsymbol{\theta}), \qquad C_m(\boldsymbol{\theta}) = C(\mathbf{y}^{(m)}, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(m)})).$$

For example we may consider the quadratic cost

$$C_m(\boldsymbol{\theta}) = \frac{1}{2} ||\mathbf{y}^{(m)} - \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(m)}))||_2^2 = \frac{1}{2} \sum_{j=1}^{n_{\text{out}}} (y_j^{(m)} - a_j^L(\boldsymbol{\theta}))^2.$$
(5)

where  $a_j^L$  is the output of *j*-th neuron at layer *L* (last layer) of the network, with input  $\mathbf{x}^{(m)}$ .

GD finds the minimum of  $R_n(\boldsymbol{\theta})$  through an iterative process as follows. We start with an initial guess  $\boldsymbol{\theta}^{(0)}$  for the minimizer of  $R_n(\boldsymbol{\theta})$  and consecutively update it using the gradient of the empirical risk. Specifically, suppose that  $\boldsymbol{\theta}^{(k)}$  is the set of parameters at iteration level  $k \geq 0$ . We compute the gradient of the empirical risk function with respect to  $\boldsymbol{\theta}$  at  $\boldsymbol{\theta}^{(k)}$ , and then update the parameter set by moving in the negative direction of the gradient:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \, \nabla_{\boldsymbol{\theta}} R_n(\boldsymbol{\theta}^{(k)}), \qquad k = 0, 1, 2, \dots .$$
(6)

Here,  $\eta > 0$  is the "learning rate". It is considered as a "hyper-parameter" to be either selected in advance as a fixed value or tuned via a validation process; see Section 8. As we

update  $\boldsymbol{\theta}^{(k)}$ , we monitor the value of  $R_n(\boldsymbol{\theta}^{(k)})$  that is to be decreasing as k increases. We continue iterations until we reach a level, say K, at which  $R_n(\boldsymbol{\theta}^{(K)})$  is small enough (below a desired tolerance). If  $R_n(\boldsymbol{\theta}^{(k)})$  is decreasing slowly or even worse if it is increasing, we will need to consider a different learning rate  $\eta$ .

## 4.2 Backpropagation

Backpropagation is the main algorithm that computes the gradient of the empirical risk  $\nabla_{\boldsymbol{\theta}} R_n(\boldsymbol{\theta}^{(k)})$  used in GD formula (6). Recall that the parameter set  $\boldsymbol{\theta}$  consists of two sets of parameters: the weights  $\{W^{\ell}\}_{\ell=1}^{L}$  and the biases  $\{\mathbf{b}^{\ell}\}_{\ell=1}^{L}$ . Given a set of parameters  $\boldsymbol{\theta}^{(k)}$  at iteration level  $k \geq 0$ , for every single training set  $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})$ , the backpropagation algorithm computes  $\nabla_{W^{\ell}} C_m(\boldsymbol{\theta}^{(k)})$  and  $\nabla_{\mathbf{b}^{\ell}} C_m(\boldsymbol{\theta}^{(k)})$  for all  $\ell = 1, \ldots, L$ . We may repeat this for all  $m = 1, \ldots, n$ , i.e. all training data points, and then we take the sample average to compute the gradient of the empirical risk:

$$\nabla_{W^{\ell}} R_n(\boldsymbol{\theta}^{(k)}) = \frac{1}{n} \sum_{m=1}^n \nabla_{W^{\ell}} C_m(\boldsymbol{\theta}^{(k)}), \qquad \nabla_{\mathbf{b}^{\ell}} R_n(\boldsymbol{\theta}^{(k)}) = \frac{1}{n} \sum_{m=1}^n \nabla_{\mathbf{b}^{\ell}} C_m(\boldsymbol{\theta}^{(k)}). \tag{7}$$

**Remark 1.** In practice, we "approximate" the sample averages (7) by randomly selecting a small batch of  $n_b \ll n$  data points; see the mini-batch stochastic GD in Section 4.3.

We now discuss the backpropagation algorithm, in four steps, for computing the gradient of  $C_m(\boldsymbol{\theta}^{(k)})$ , i.e.  $\nabla_{W^{\ell}}C_m(\boldsymbol{\theta}^{(k)})$  and  $\nabla_{\mathbf{b}^{\ell}}C_m(\boldsymbol{\theta}^{(k)})$ , for any fixed m. We will drop the dependence on  $\boldsymbol{\theta}^{(k)}$  for the ease of reading.

**Step 1.** We first introduce a useful quantity  $\delta_i^{\ell}$  that will simplify the derivation:

$$\delta_j^{\ell} := \frac{\partial C_m}{\partial z_j^{\ell}}, \qquad j = 1, \dots, n_{\ell}, \qquad \ell = 1, \dots, L, \qquad \text{in vector form:} \quad \boldsymbol{\delta}^{\ell} = (\delta_j^{\ell}) \in \mathbb{R}^{n_{\ell}}.$$

This quantity represents the "sensitivity" of the cost function  $C_m$  to change with respect to the *j*-th neuron in layer  $\ell$ .

**Step 2.** We compute  $\delta_i^L$  (at the last level  $\ell = L$ ) by the chain rule, noting that  $a_i^L = \sigma_L(z_i^L)$ :

$$\delta_j^L = \frac{\partial C_m}{\partial z_j^L} = \frac{\partial C_m}{\partial a_j^L} \, \sigma_L'(z_j^L).$$

The first term  $\partial C_m / \partial a_j^L$  can be computed exactly. For instance for the quadratic cost (5) we have  $\partial C_m / \partial a_j^L = y_j^{(m)} - a_j^L$ . We also have analytic expression for  $\sigma'_L$ , i.e. the derivative of the activation function at the last level. Finally,  $z_j^L$  can be computed by a forward sweep, given  $\boldsymbol{\theta}^{(k)}$ . In vector form we can write

$$\boldsymbol{\delta}^{L} = \nabla_{\mathbf{a}^{L}} C_{m} \odot \sigma_{L}'(\mathbf{z}^{L}),$$

where  $\odot$  denotes component-wise multiplication.

**Step 3.** We compute  $\delta_j^{\ell}$  for  $\ell = L - 1, L - 2, ..., 1$ , and hence the name "backpropagation", as follows. Using the chain rule, we first write

$$\delta_j^{\ell} = \frac{\partial C_m}{\partial z_j^{\ell}} = \sum_{i=1}^{n_{\ell+1}} \frac{\partial C_m}{\partial z_i^{\ell+1}} \frac{\partial z_i^{\ell+1}}{\partial z_j^{\ell}} = \sum_{i=1}^{n_{\ell+1}} \delta_i^{\ell+1} \frac{\partial z_i^{\ell+1}}{\partial z_j^{\ell}}.$$

This formula expresses  $\boldsymbol{\delta}^{\ell}$  in terms of  $\boldsymbol{\delta}^{\ell+1}$ . Here, the sum is taken over all neurons at layer  $\ell + 1$ . In order to compute  $\partial z_i^{\ell+1} / \partial z_j^{\ell}$ , we write

$$z_i^{\ell+1} = \sum_{j=1}^{n_\ell} W_{i,j}^{\ell+1} \, a_j^\ell + b_i^{\ell+1} = \sum_{j=1}^{n_\ell} W_{i,j}^{\ell+1} \, \sigma_\ell(z_j^\ell) + b_i^{\ell+1}$$

Here, the sum is taken over all neurons in layer  $\ell$ . Note that we have used  $a_j^{\ell} = \sigma_{\ell}(z_j^{\ell})$ , i.e. the output of *j*-th neuron at layer  $\ell$  is obtained by the application of  $\sigma_{\ell}$  to the input of *j*-th neuron at layer  $\ell$ . From the last equality, we get

$$\frac{\partial z_i^{\ell+1}}{\partial z_j^\ell} = W_{i,j}^{\ell+1} \, \sigma_\ell'(z_j^\ell).$$

Hence we obtain

$$\delta_j^{\ell} = \sum_{i=1}^{n_{\ell+1}} \delta_i^{\ell+1} \, W_{i,j}^{\ell+1} \, \sigma_\ell'(z_j^{\ell}),$$

where, all weights  $W_{i,j}^{\ell+1}$  are available via the given  $\boldsymbol{\theta}^{(k)}$ , and all neuron inputs  $z_j^{\ell}$  are computed by the forward sweep. In vector form we can write

$$\boldsymbol{\delta}^{\ell} = \left( W^{\ell+1^{\top}} \boldsymbol{\delta}^{\ell+1} \right) \odot \sigma'_{\ell}(\mathbf{z}^{\ell}).$$

**Step 4.** After computing all sensitivity ratios  $\delta^1, \ldots, \delta^L$ , we can compute the gradients:

$$\frac{\partial C_m}{\partial b_j^\ell} = \frac{\partial C_m}{\partial z_j^\ell} \frac{\partial z_j^\ell}{\partial b_j^\ell} = \frac{\partial C_m}{\partial z_j^\ell} = \delta_j^\ell,$$

noting that  $\partial z_j^{\ell} / \partial b_j^{\ell} = 1$ . In vector form we have

$$\nabla_{\mathbf{b}^{\ell}} C_m = \boldsymbol{\delta}^{\ell} \in \mathbb{R}^{n_{\ell}}, \qquad \ell = 1, \dots, L.$$

Similarly, we can write

$$\frac{\partial C_m}{\partial W_{j,k}^\ell} = \frac{\partial C_m}{\partial z_j^\ell} \frac{\partial z_j^\ell}{\partial W_{j,k}^\ell} = \delta_j^\ell \, a_k^{\ell-1},$$

noting that  $\partial z_j^{\ell} / \partial W_{j,k}^{\ell} = a_k^{\ell-1}$ . In vector form we have

$$\nabla_{W^{\ell}} C_m = \boldsymbol{\delta}^{\ell} \mathbf{a}^{\ell-1^{\top}} \in \mathbb{R}^{n_{\ell} \times n_{\ell-1}}, \qquad \ell = 1, \dots, L.$$

The pseudocode for backpropagation is given in Function 1. It is to be noted that the backward movement in the algorithm is a natural consequence of the fact that the loss is a

Function 1: Backpropagation algorithm

function  $\{\nabla_{\mathbf{b}^{\ell}} C_m, \nabla_{W^{\ell}} C_m\}_{\ell=1}^L = \text{BACKPRO}(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, \{(W^{\ell}, \mathbf{b}^{\ell})\}_{\ell=1}^L, \{\sigma_{\ell}\}_{\ell=1}^L, \ "C_m")$ 

1: Set  $\mathbf{a}^0 = \mathbf{x}^{(m)}$  for the input layer (we may call it layer zero)

- 2: Forward pass: for each  $\ell = 1, ..., L$  compute  $\mathbf{z}^{\ell} = W^{\ell} \mathbf{a}^{\ell-1} + \mathbf{b}^{\ell}$  and  $\mathbf{a}^{\ell} = \sigma_{\ell}(\mathbf{z}^{\ell})$ .
- 3: Loss: Using the given function for loss " $C_m$ " and  $\mathbf{y}^{(m)}$  and  $\mathbf{a}^L$ , compute  $\nabla_{\mathbf{a}^L} C_m$ .
- 4: **Output sensitivity:** Compute  $\boldsymbol{\delta}^{L} = \nabla_{\mathbf{a}^{L}} C_{m} \odot \sigma'_{L}(\mathbf{z}^{L})$ .
- 5: **Backpropagate:** for each  $\ell = L 1, \dots, 1$  compute  $\boldsymbol{\delta}^{\ell} = (W^{\ell+1^{\top}} \boldsymbol{\delta}^{\ell+1}) \odot \sigma'_{\ell}(\mathbf{z}^{\ell}).$
- 6: **Outputs:** for each  $\ell = L, ..., 1$  (together with steps 4 and 5) compute the gradients

$$\nabla_{\mathbf{b}^{\ell}} C_m = \boldsymbol{\delta}^{\ell}, \qquad \nabla_{W^{\ell}} C_m = \boldsymbol{\delta}^{\ell} \mathbf{a}^{\ell-1^{+}}.$$

function of network's output. Using the chain rule, we need to move backward to compute all gradients.

It is also easy to derive the computational complexity (or cost) of the backpropagation algorithm, i.e. the number of floating point operations needed to compute the gradients for a given data point at a given set of parameters. The main portion of cost is due to steps 2 (forward pass) and step 5 (backward pass) of the algorithm, which depends on the number of layers and neurons in each layer, and the cost of computing activation functions. Assuming that each activation or its derivative applied to a scalar requires N operations, the number of operations in steps 2 and 5 are  $\sum_{\ell=1}^{L} n_{\ell}(n_{\ell-1}+N+1)$  and  $\sum_{\ell=1}^{L} n_{\ell}(n_{\ell+1}+N+1)$ , respectively. If we further assume that the number of neurons in each layer is a fixed number W, and that N is also of the order of W, then the total cost of the algorithm will be  $\mathcal{O}(LW^2)$ .

Efficiency of backpropagation. In order to better appreciate backpropagation, we compare it with an alternative, simple approach for computing the gradient: numerical differentiation. Suppose we want to compute  $\partial C_m / \partial W_{j,k}^{\ell}$  for one single weight. Suppose that this weight takes the *i*-th place in the parameter vector  $\boldsymbol{\theta}$ , i.e  $W_{j,k}^{\ell} = \boldsymbol{\theta}_i$ . Suppose we employ a first-order accurate numerical differentiation and approximate the derivative

$$\frac{\partial C_m}{\partial W_{ik}^{\ell}} = \frac{\partial C_m(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} \approx \frac{C_m(\boldsymbol{\theta} + h \, \mathbf{e}_i) - C_m(\boldsymbol{\theta})}{h},$$

where h > 0 is a small number, and  $\mathbf{e}_i$  is the vector with one component 1 and all other components 0, where the 1 is in the *i*-th place. Although this approach is simple and easy to implement (much simpler than the algebra involved in backpropagation), it is ridiculously more expensive than backpropagation. Each single derivative involves two evaluations of the loss function  $C_m$ , one at  $\boldsymbol{\theta}$  and one at a slightly different point  $\boldsymbol{\theta} + h \mathbf{e}_i$ . Assuming the network has  $n_{\theta}$  parameters, we would need  $n_{\theta} + 1$  evaluations of  $C_m$  to compute all  $n_{\theta}$  derivatives, and this requires  $n_{\theta} + 1$  forward passes through the network (for each training data point). Compare this huge cost with the cost of backpropagation where we simultaneously compute all derivatives by just one forward pass followed by one backward pass. Assuming the cost of a backward pass is comparable to the cost of a forward pass, the cost of backpropagation is hence roughly proportional to the cost of two forward passes, while the cost of numerical differentiation is proportional to the cost of  $n_{\theta} + 1 \gg 2$  forward passes.

#### 4.3 Mini-batch stochastic GD and backpropagation

The backpropagation algorithm discussed above needs to be combined with an optimization algorithm, e.g. GD or stochastic gradient descent (SGD). In order to compute the gradient of the empirical risk  $\nabla_{\boldsymbol{\theta}} R_n(\boldsymbol{\theta}^{(k)})$  used in GD formula (6), one approach is to apply the backpropagation algorithm *n* times, each for one training data point. Then one can take the sample average of gradients for all *n* data points to obtain the gradient of the empirical risk by (7). This leads to the standard GD:

$$W^{\ell(k+1)} = W^{\ell(k)} - \eta \,\nabla_{W^{\ell}} R_n(\boldsymbol{\theta}^{(k)}), \qquad k = 0, 1, \dots, K-1, \qquad \ell = 1, \dots, L.$$
(8)

$$\mathbf{b}^{\ell(k+1)} = \mathbf{b}^{\ell(k)} - \eta \, \nabla_{\mathbf{b}^{\ell}} R_n(\boldsymbol{\theta}^{(k)}), \qquad k = 0, 1, \dots, K-1, \qquad \ell = 1, \dots, L.$$
(9)

In (mini-batch) SGD, we randomly select a mini-batch of  $n_b \ll n$  training data points and apply a gradient descent step based on that mini-batch:

- Divide the set of n training data points into  $\lfloor n/n_b \rfloor$  mini-batches of size  $n_b \ll n$ ;
- Loop over all mini-batches containing  $n_b$  data points (note that the last batch may have fewer or larger number of points than  $n_b$ );
  - For each training data point in the mini-batch, i.e. for  $m = 1, ..., n_b$ , call the backpropagation algorithm (BACKPRO) and compute  $\{\nabla_{\mathbf{b}^\ell} C_m, \nabla_{W^\ell} C_m\}_{\ell=1}^L$ ;
  - Approximate the gradients by taking the sample average of all  $n_b$  gradients:

$$\nabla_{W^{\ell}} R_n(\boldsymbol{\theta}^{(k)}) \approx \frac{1}{n_b} \sum_{m=1}^{n_b} \nabla_{W^{\ell}} C_m(\boldsymbol{\theta}^{(k)}), \qquad \nabla_{\mathbf{b}^{\ell}} R_n(\boldsymbol{\theta}^{(k)}) \approx \frac{1}{n_b} \sum_{m=1}^{n_b} \nabla_{\mathbf{b}^{\ell}} C_m(\boldsymbol{\theta}^{(k)});$$

• GD step: update the weights and biases according to the rules (8)-(9), using the approximate gradients;

The above procedure is referred to as one-*epoch* SGD, that is, one run of SGD over all n data points with one particular mini-batch selection. In practice, we repeat the above process for multiple epochs of training. This would require an outer loop that goes through  $n_e$  epochs, where for each epoch we randomly shuffle the training data points before dividing them into mini-batches.

Note that to implement the full SGD (either standard or multiple-epoch mini-batch) we will need to start with an initial guess for the parameters, say  $W^{\ell^{(0)}}$  and  $\mathbf{b}^{\ell^{(0)}}$  for  $\ell = 1, \ldots, L$ . This is usually done by pseudo-random generation of numbers.

**Remark 2.** (Total number of SGD iterations) Each mini-batch will form one iteration of SGD. One epoch of SGD involves  $\lfloor n/n_b \rfloor$  mini-batches and hence  $\lfloor n/n_b \rfloor$  iterations. The total number of SGD iterations is therefore  $K = n_e \lfloor n/n_b \rfloor$ .

## 5 Activation functions

The most important feature of activation functions  $\sigma$  (we drop the subscript  $\ell$  for simplicity) is their ability to add general, arbitrary "nonlinearity" into networks. They enable a network to learn complex patters in the data, in both classification and regression problems. It is to be noted that by just stacking multiple linear layers without  $\sigma$  (i.e. with  $\sigma$  being the identity function) we can generate polynomial nonlinearity. But polynomials are not general and complex enough to capture complex patterns and model complex functions.

Another feature of activation functions  $\sigma$  is their ability to limit and control a neuron's output, if needed. Without  $\sigma$  (i.e. with  $\sigma$  being the identity function) the value w x + b can become very large, especially in deep networks, leading to computational issues. Moreover, in most cases, the network output needs to be restricted to a certain limit (e.g a positive number or a value between 0 and 1, etc.). In such cases, the activation of the output layer plays an important role in enforcing the limit.

## 5.1 Desirable properties of activation functions

Bside the above two features, an activation function should have a few desirable properties, listed below.

Efficient computation. Activation functions are applied multiple times ( $\sum_{\ell=1}^{L} n_{\ell}$  times). In deep networks they may be applied millions of times. Hence they should be computationally cheap to calculate, e.g. involving just a few (maybe 1 or 2) operations.

**Differentiability.** Activation functions need to be (almost everywhere) differentiable for computing the gradients of the loss function.

Avoiding vanishing gradients. The backpropagation algorithm in networks with multiple layers involves multiple applications of the chain rule. Each time a  $\sigma'$  gets multiplied by another one, and we have  $\delta^{\ell} \sim (\sigma')^{L+1-\ell}$ ; see the final formulas in Steps 2-3 in Section 4.2. If the values of  $\sigma'$  at all layers is between 0 and 1, then  $\delta^{\ell}$  and hence the value of gradients at initial layers (with small  $\ell$ ) becomes very small. Consequently the weights and biases of those initial layers would learn very slowly. Similarly, when  $\sigma'_{\ell}$  at a layer  $\ell$  is close to zero (e.g. when  $\sigma$  is flat), then  $\delta^{\ell}$  and hence the value of gradients at the layer will become close to zero. Consequently, the weights and biases of that layer would also learn very slowly. Another situation is when  $\sigma_{\ell-1}$  is close to zero. In this case, the gradient with respect to weights at layer  $\ell$  will be close to zero and hence those weights will learn slowly. In all above cases, the gradients become very small and the network learns very slowly. This problem is known as the *vanishing gradient problem*. We would like to have an activation function that does not shift the gradient towards zero, e.g. an increasing function whose derivative is positive and almost never gets close to zero.

## 5.2 Popular choices of activation functions

Different layers of a network may use different activation functions. In the output layer (i.e. the last layer L) of networks, activation functions are often selected based on the type of problem that we try to solve. Two common choices include:

• Identity function:

$$\sigma_L(x) = x$$

This is usually used in regression problems. It is very fast to compute since it does not involve any operation. It is also differentiable with derivative equal to 1 and does not suffer from the vanishing gradient problem.

• Softmax (or normalized exponential):

$$\sigma_L(x_1, \dots, x_{n_L}) = (p_1, \dots, p_{n_L}), \qquad p_j = \frac{\exp(x_j)}{\sum_{j=1}^{n_L} \exp(x_j)}, \qquad j = 1, \dots, n_L.$$

This is usually used in classification problems. It converts the real-valued output of networks into a set of pseudo-probabilities  $p_1, \ldots, p_{n_L}$  with  $\sum_{j=1}^{n_L} p_j = 1$ . Each probability  $p_j$  is an approximation of the posterior probability  $\operatorname{Prob}(c_j|\operatorname{data})$ , where  $c_j$  denotes the occurrence of the *i*-th class or category. Eventually, we will choose the class with largest probability. Its computation is more expensive than the identity function, involving exponentiation and addition operators. It is differentiable and does not suffer from the vanishing gradient problem.

Two classical activation functions for hidden layers  $\ell = 1, \ldots, L - 1$  include:

- Sigmoid function  $\sigma_{\ell}(x) = 1/(1 + \exp(-x))$
- Hyperbolic tangent  $\sigma_{\ell}(x) = \tanh(x)$

These two functions are rather expensive to compute and suffer from the vanishing gradient problem. Note that both functions are rather flat in a wide range of their domains. These two functions are hence no longer used in practice.

More practical types of activation functions that are widely used today include the rectified linear unit (ReLU) family.

• ReLU function:  $\sigma_{\ell}(x) = \max(0, x)$ ; see Figure 3 (top).

This is the most commonly used activation function today. It is cheap and easy to compute and does not cause the vanishing gradient problem. Its (weak) derivative is 0 when x < 0 and 1 when  $x \ge 0$ ; see the notion of weak derivatives in Chapter 2.

It also features a property known as "dying ReLU": since it returns zero for negative inputs, it causes some neurons to remain inactive (or dead). This in turn results in "model sparsity", which is often desirable. Intuitively, a biological neural network is sparse: among billions of neurons in a human brain, only a portion of them fire (i.e.



Figure 2: Sigmoid and hyperbolic tangent activation functions.

are active) at a time for a particular task. For example, in a classification problem, there may be a set of neurons that can identify apples, which obviously should not be activated if the image displays a car. But beside the resemblance to biological networks, sparsity in artificial networks have two advantages. First, sparse networks are concise, parsimonious models that often have better predictive power and less overfitting and noise. Second, sparse networks are faster to compute than dense networks, as they involve fewer number of operations.

The downside of a dying ReLU is that since its slope in the negative range is zero, once a neuron gets negative, it does not learn anything and it may not recover at all, i.e. the neuron may become permanently dead and hence useless. Recall that a single step in SGD involves multiple data points. If for some of them the input to a neuron is not negative, we can still get a slope out of ReLU. The dying problem may also occur when the learning rate is too high, as it would amount to large variations in the weights, turning a positive input into a negative value with a zero ReLU slope.

• Leaky ReLU function:  $\sigma_{\ell}(x) = \max(\alpha x, x)$ , with  $\alpha \in (0, 1)$  fixed.

This activation function prevents the dying ReLU problem to some extent: it has a small slope in the negative range. The parameter  $\alpha$  is usually set to 0.01-0.05. Note that if we set  $\alpha = 0$  then we get ReLU, and if we set  $\alpha = 1$  then we get the identity function; see Figure 3 (middle).

- Parametric ReLU function: σ<sub>ℓ</sub>(x) = max(α x, x), with variable parameter α ∈ (0, 1). It is a type of leaky ReLU, with α being a hyper-parameter, rather than a predetermined fixed value; see Figure 3 (middle).
- ReLU6 function:  $\sigma_{\ell}(x) = \min(\max(\alpha x, x), 6).$

This is ReLU restricted on the positive side by value 6. It suppresses very large activations and hence prevents the gradient from blowing up; see Figure 3 (bottom).



Figure 3: ReLU family of activation functions.

## 6 Loss functions

Quadratic loss. The quadratic loss function, given in (5),

$$C_m(\boldsymbol{\theta}) = \frac{1}{2} \sum_{j=1}^{n_{\text{out}}} (y_j^{(m)} - a_j^L(\boldsymbol{\theta}))^2,$$

is a very common loss function in regression problems.

**Cross-entropy loss.** The cross-entropy (or Kulback-Leibler) loss function is often used in classification problems. Given a set of labeled training data  $\{(\mathbf{x}^{(m)}, c^{(m)})\}_{m=1}^{n}$  with class labels  $c^{(m)} \in \{c_1, \ldots, c_{n_{\text{out}}}\}$ , the cross-entropy loss function reads

$$C_m(\boldsymbol{\theta}) = -\sum_{j=1}^{n_{\text{out}}} \delta_{m,j} \log(a_j^L(\boldsymbol{\theta})), \qquad a_j^L = p_j = \operatorname{Prob}(c_j | \mathbf{x}^{(m)}), \tag{10}$$

where

$$\delta_{m,j} = \begin{cases} 1 & \text{if } c^{(m)} = c_j, \\ 0 & \text{otherwise,} \end{cases}$$

is a 0-1 binary variable that indicates the "true" distribution of class membership, and  $a_j^L$  is the *j*-th output of the softmax activation function applied to the last layer of the network that indicates the "predicted" distribution. The cross-entropy loss is indeed the Kulbacl-Leibler divergence between the true and predicted distributions. It is easy to see that

$$C_m(\boldsymbol{\theta}) = -\log(\operatorname{Prob}(c^{(m)}|\mathbf{x}^{(m)})),$$

that is, for each m, we find the output index j for which  $c_j = c^{(m)}$ .

We note that adapting the backpropagation algorithm of Section 4.2 to either regressin problems, e.g. with the quadratic loss and identity activation at the last layer, or classification problems, e.g. with the cross-entropy loss and softmax activation at the last layer, will be straightforward, and hence we leave it as an exercise.

## 7 Overfitting and regularization

The training strategy discussed so far may suffer from *overfitting* (or overtraining), that is, the trained network  $\mathbf{f}_{\theta}(\mathbf{x})$  may not perform well in approximating  $\mathbf{y}$  outside the set of training data  $\{(\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}_{m=1}^{n}$ .

The first step in avoiding overfitting is to detect overfitting. For this purpose, we usually split the available data into two categories: training data and test data. While we train the network using the training data and monitor how the loss on training data changes as the network trains, we also keep track of the loss on the test data. This tells us if and how the trained network generalizes to the test data. Figure 4 shows a case where we observe a decrease in the loss on the training data, meaning that the network fits (or learns) the training data well, while the loss on the test data starts increasing after an initial drop. This is a sign of overfitting and lack of generalization.



Figure 4: The loss on training data versus test data as the number of epochs increases.

Obviously, increasing the amount of training data is one way of reducing overfitting. Another trivial approach is to reduce the complexity of our network, i.e. to reduce the number of network parameters. Unfortunately, both approaches are not very practical. Often, the number of training data cannot be increased because either they are not available to us or they are very expensive or difficult to obtain. Moreover, networks with larger number of parameters (such as very deep networks) have the potential to be more powerful, and hence reducing complexity of networks may not be a good option; we will discuss the power of "depth" in more details in Chapters 2-3. Fortunately, there are other techniques, known as *regularization* techniques, that can help reduce overfitting given a fixed number of data points and a fixed number of network parameters. Examples include the addition of a regularization (or penalty) term to the loss function, early stopping, and dropout [28].

## 7.1 Adding penalty terms

In this approach, we modify the loss function by adding a regularizing (or penalty) term. We consider two types of regularizers, namely  $L^2$  and  $L^1$  regularizers.

 $L^2$  regularizers. A common penalty term is the  $L^2$ -norm of all weights (all parameters excluding biases) scaled by a factor  $\lambda$  to the loss function. That is, we consider the  $L^2$ -regularized loss

$$R_n(\boldsymbol{\theta}) = \frac{1}{n} \sum_{m=1}^n C_m(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta}_w||_2^2,$$

where  $\boldsymbol{\theta}_w$  denotes the vector of all weights excluding biases. The parameter  $\lambda > 0$  is a hyper-parameter known as the regularization parameter.

The main effect of this regularization term is to learn smaller weights unless large weights substantially reduce the first part of the loss. It introduces a compromise between minimizing the original (non-regularized) loss and finding small wights. The regularization parameter  $\lambda$  determines the relative importance of each term: the large  $\lambda$ , the more emphasize is put on the penalty term.

Why does such regularization help reduce overfitting? In fact, smaller weights amount to smaller fluctuations in the output (more regularized outputs). When weights are small, the network output would not change much if we make small changes in the input. This is analogous to the behavior of simpler models which are more generalizable.

Beside reducing overfitting, smaller weights may help GD/SGD converge faster. When we minimize a non-regularized loss, the size of weights is likely to grow. This may cause the weights moving in pretty much the same direction and hence converging very slowly to the global minimum as GD/SGD makes only small changes to the direction. Another advantage of adding such a penalty term is that it makes the loss function more convex (and hence with less local minima) and the GD/SGD more robust with respect to the initial guess. A nonconvex loss with multiple local minima may cause the weights to get stuck in local minima. In this case, the learning process becomes sensitive to the initial guess. An initial guess that is close to a local minima would converge much slower than an initial guess that happen to be close to the global minimum.

It is to be noted that the main reason that biases are excluded in regularization is that large biases would not make the network output sensitive to the input.

 $L^1$  regularizers. Another penalty term is the  $L^1$ -norm (instead of  $L^2$  norm), which gives the  $L^1$ -regularized loss,

$$R_n(\boldsymbol{\theta}) = \frac{1}{n} \sum_{m=1}^n C_m(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta}_w||_1.$$

Similar to the case of  $L^2$  regularization, the  $L^1$ -regularized loss penalizes large weights, forcing the network to learn small weights. There is however a difference between the two approaches in the way the weights decrease at each iteration of GD/SGD. To see this, let us consider a single weight, say  $w := W_{j,k}^{\ell}$  for some fixed  $j, k, \ell$ . Then, a GD update of w reads

$$w^{(k+1)} = w^{(k)} - \eta \,\frac{\partial R_n}{\partial w}.$$

With  $L^1$  regularization, we get

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{n} \sum_{m=1}^{n} \frac{\partial C_m}{\partial w} - \eta \lambda \operatorname{sgn}(w^{(k)}),$$

and with  $L^2$  regularization, we get

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{n} \sum_{m=1}^{n} \frac{\partial C_m}{\partial w} - \eta \,\lambda \, w^{(k)}.$$

In both cases w decreases due to the penalty terms (i.e. the third terms in the right hand side of the above two formulas). However, with  $L^1$  regularization w decreases by a fixed amount, while with  $L^2$  regularization w decreases by an amount proportional to w. Consequently, when |w| is large,  $L^2$  regularization reduces the weight much more than  $L^1$  regularization, and when |w| is small,  $L^2$  regularization does not reduce it as much as  $L^1$  does. This amounts to a more balanced distribution of weights in the case of  $L^2$  regularization (pretty much like averaging), while  $L^1$  regularization tends to drive some weights toward zero keeping some others of high importance (pretty much like median). Moreover, since  $L^2$  does not change small weights much, and in particular it does not drive them to zero, it amounts to a dense weight vector (with many non-zeros). However,  $L^1$  brings **sparsity** to the weights by making small weights zero.

## 7.2 Early stopping

In this approach, we split the available data into three categories: training data, test data, and validation data. We will monitor the loss on the validation data (instead of the test loss) at the end of each epoch. Once we are confident that the validation loss does no longer decrease, we stop training; see Section 8 for more details on the validation set.

## 7.3 Dropout

In this approach, we modify the network and its training process, rather than modifying the loss function. The modification is as follows. Over a mini-batch of data points, we randomly select a portion, usually half, of hidden neurons (i.e. neurons in the hidden layers, excluding input and output layers). We then dropout the selected neurons, that is, we temporarily remove those neurons from the network, along with their incoming and outgoing connections. We perform all forward and backward passes in the mini-batch through the modified network and update the weights and biases corresponding to active neurons. For the next mini-batch, we restore the dropout neurons in the previous mini-batch and randomly select a new set of dropout neurons. We repeat this process: for each mini-batch we select a new set of dropout neurons and update the weights and biases for active neurons. In the end, we have updated the weights and biases of the full network, but always using half the hidden neurons. After training is over, dropout is no longer used when making a prediction with the full network. Since the full network has twice as many active hidden neurons, we halve the weights "outgoing" from hidden neurons. We can think of the network as one in which each hidden neuron is retained during training with a fixed probability p = 0.5 independent of other neurons. If a neuron is retained with probability p during training, the outgoing weights of that neuron are multiplied by p at the test time. We note that to effectively remove a neuron from a network, we can simply multiply the value of its output weight by zero. This is why only the output weights will be scaled by the selected rate, not biases.

Why would dropout help with regularization? Intuitively, we can think of dropout as a method of training a large number of neural networks with different architectures in parallel. This implies that we have trained multiple models and will be evaluating and averaging multiple models on each test example. Regularization is a direct consequence of averaging. In fact, different models may overfit in different ways, and averaging may help eliminate or reduce overfitting.

## 8 Validation and hyper-parameter tuning

Each optimization/regularization technique involves a few hyper-parameters, such as learning rates, number of epochs, mini-batch sizes, regularization parameters, dropout rates, and so forth. The hyper-parameters are often tuned using a *validation* set, i.e. a set of data points that are not directly used in the optimization process. A common practice is to select a large portion of the available set of n data points as training data and a smaller portion of the data as the validation data. Here, we will review a few heuristic strategies for selecting hyper-parameters. We refer to [2, 14] for more details on the subject.

**Learning rate**  $\eta$ . Recall that GD/SGD tries to bring us down to the valley of the loss function with steps proportional to  $\eta$ . If  $\eta$  is too large, the steps will also be large, and it is likely that we overshoot the minimum getting to the other side of valley. If  $\eta$  is too small, the steps will be very small, and it will slow down the convergence of GD/SGD. Usually, we start with a small value, say  $\eta = 0.1$ , and monitor the training-validation losses for a few epochs. Depending on the behavior of the loss we may need to increase or decrease the value of  $\eta$ . After we find a "good" value for  $\eta$ , we can keep it fixed during the training process. However, it is usually better to let  $\eta$  vary as we train, following the same strategy as early stopping. That is, we hold the learning rate constant until the validation loss stops decreasing, and then we decrease the learning rate by a factor, say 2-10, and continue this process.

Number of epochs  $n_e$ . We usually use early stopping to determine the number of epochs. At the end of each epoch, we monitor the validation loss. If it stops decreasing after a few epochs, we stop and this automatically determines the number of epochs. Hence, early stopping both prevents overfitting and provides a strategy to automatically take care of the number of epochs and to terminate training.

**Regularization parameter**  $\lambda$ . Similar to  $\eta$ , we first determine a reasonable value for  $\lambda$  by training over a few epochs. Then we monitor the validation loss and change and fine-tune  $\lambda$  whenever needed.

**Mini-batch size**  $n_b$ . In general, smaller batch sizes would give a faster GD/SGD convergence, as we update the parameters more often, but at the same time may result in a slower

learning, due to a poorer approximation of the sample gradient. There should be a compromise between the speed of learning and convergence. The mini-batch size is relatively independent of other hyper-parameters and mainly affect the speed of learning. Usually, we start with a reasonable set (not necessarily optimal) of other hyper-parameters. Then we use validation data to select a pretty optimal mini-batch size that gives the fastest improvement in performance (i.e. CPU-time). For this purpose we can plot the validation error versus CPU-time (not versus the number of epochs) for several different batch sizes and then select the mini-batch size that gives us the fastest decay in the validation loss. With the mini-batch size fixed, we can proceed to fine-tune other hyper-parameters.

## 9 Procedure summary

We finally summarize the general procedure:

- Data preparation: We first decide (based on our available budget) about the quantity and quality of data and collect the data. We then split the data into three non-overlapping sets: 1) a training set; 2) a validation set; and 3) a test set. All data sets should be representative of the problem in hand, and the usual size rule is to take the size of training set larger than the size of validation set which is in turn larger in size than the test set. As an example, we may consider a 70% 20% 10% split.
- Network architecture: We select the number of layers, neurons, and activation functions for the problem in hand.
- Network training: We use the training set (for computing cost gradients) and the validation set (for hyper-parameter tuning) to learn the network parameters.
- Network evaluation: We use the test set to evaluate (or test) the trained network. If the network passes the test, it is ready to be used for predication.
- Network prediction: The trained network that has passed evaluation can now be used to make predictions.

## Chapter 2: Deep neural networks and linear approximation

The goal of this chapter is to review a combination of classical ideas and more recent developments on the approximation theory of deep neural networks within the framework of linear approximation. We will start with an introduction to the concept of density in polynomial approximation and in particular will study the Stone-Weierstrass theorem for real-valued continuous functions. We will continue with a few classical results on the density and convergence rate of two-layer feedforward networks (or perceptrons). We finally discuss more recent developments on the complexity of deep networks in approximating Sobolev functions and by comparing networks with linear approximation methods.

## 10 Introduction

Let  $X \subset \mathbb{R}^d$  be a compact, convex domain in  $\mathbb{R}^d$ . Let  $f: X \to \mathbb{R}$  be a real-valued function, referred to as the target function, in some known function space  $\mathcal{F}(X)$ . Two particular examples of  $\mathcal{F}$  that we will consider include the set of continuous functions, denoted by C(X), and the set of Sobolev functions, denoted by  $W^{k,p}(X)$ ; see the definition of the Sobolev space  $W^{k,p}(X)$  in Section 13.2. We say f is a complicated function if its evaluation at any point  $\mathbf{x} \in X$  is computationally expensive, and our computational budget and resources allow only a finite (and often small) number of evaluations of f. We therefore wish to approximate the complicated target function f by a simpler approximant being a neural network (NN)  $f_{\theta}: \mathbb{R}^d \to \mathbb{R}$  that has  $n_{\theta}$  parameters.

In this context, where the approximant is a neural network, the main task of approximation theory is to study the approximability of the target function by neural networks. Specifically, by such studies we try to address three major questions:

- **Density**: if  $n_{\theta} \to \infty$ , is there  $f_{\theta}$  that can approximate f arbitrarily well?
- Convergence rate: if  $n_{\theta} < \infty$  fixed, how close can  $f_{\theta}$  be to f?
- Complexity: if we want  $||f f_{\theta}|| < \varepsilon$ , how large should  $n_{\theta}$  be?

The first question, i.e. the question of density, is a very important one concerning the possibility of approximation: when does  $f_{\theta}$  have the theoretical ability to approximate f arbitrarily well? In other words, can we approximate any traget function in the function space as accurately as we wish by a function from the family of neural networks? Or equivalently, is the family of neural networks "dense" in the target function space? For this reason, and in order to better grasp the concept of density, we will start with density in a simpler case where the approximants are algebraic polynomials, rather than neural networks (see Section 11). We then discuss a few classical density results and convergence rates for two-layer feedforward networks (see Sections 12-13). Finally, we study more recent developments on the complexity of deep NNs in Sobolev spaces (see Section 14).

**Remark 3.** (An open problem) One important problem that will not be addressed here concerns the selection and preparation of training data  $\{(\mathbf{x}^{(i)}, f(\mathbf{x}^{(i)}))\}_{i=1}^{n}$ . Precisely, to achieve

a desired accuracy constraint  $||f - f_{\theta}|| < \varepsilon$ , we would wish to know how to optimally select the number n of data points, the location (or distribution) of support points  $\{\mathbf{x}^{(i)}\}_{i=1}^{n}$  in X, and the quality of the output data points, i.e. how accurately to compute  $\{f(\mathbf{x}^{(i)})\}_{i=1}^{n}$ . This is still an open problem, and we simply assume that we have as many high-quality data points as needed.

**Remark 4.** (Approximation theory and numerical analysis) Both approximation theory and numerical analysis are branches of analysis. They share similar goals. In general, more accurate approximations of a target function can be achieved only by increasing the complexity of the approximants. The understanding of such a trade-off between accuracy and complexity is the main goal of "constructive approximation": how the approximation can be constructed. In this sense, the goals of approximation theory (and in particular constructive approximation) and numerical analysis are similar. Approximation theory is however less concerned with computational issues than numerical analysis. Also, in numerical computation, the target functions are often implicitly available, for instance through differential equations, integral equations, or integro-differential equations. Those interested in the field of approximation theory (not in the context of neural networks) may refer to [30, 25, 9, 8].

## 11 Density in polynomial approximation

In 1885 Karl Weierstrass, a German mathematician and the "father of modern analysis", proved that algebraic polynomials are dense in the set of real-valued continuous functions on closed intervals. Later, in 1937 Marshall Stone, an American mathematician, proved the Stone–Weierstrass theorem that generalizes Weierstrass's theorem on the uniform approximation of continuous functions by polynomials to higher dimensions. Here, we will review the key concepts in such density results.

## 11.1 Uniform convergence

For studying density (e.g. in the space of continuous functions) we will need to introduce a notion of "distance" that measures the "closeness" of the target function to the approximant. Here, we will consider uniform norms and hence review the notion of uniform convergence. A sequence  $\{f_m\}_{m=1}^{\infty}$  of functions is said to converge uniformly to a limiting function f on a set X if given any  $\varepsilon > 0$ , there exists a natural number N such that  $|f(\mathbf{x}) - f_m(\mathbf{x})| < \varepsilon$  for all  $m \geq N$  and for all  $\mathbf{x} \in X$ , and we write

$$f_m \to f$$
 uniformly. (11)

An equivalent formulation for uniform convergence can be given in terms of the supremum norm (also called infinity norm or uniform norm). Let the supremum norm of f be

$$||f||_{\infty} := \sup_{\mathbf{x} \in X} |f(\mathbf{x})|.$$

Then the uniform convergence (11) is equivalent to

$$||f - f_m||_{\infty} \xrightarrow{m \to \infty} 0. \tag{12}$$

It is to be noted that uniform convergence is stronger than pointwise convergence (defined by  $\lim_{m\to\infty} f_m(\mathbf{x}) = f(\mathbf{x}), \ \forall \mathbf{x} \in X$ ) in the sense that in uniform convergence  $N = N(\varepsilon)$  is independent of the point  $\mathbf{x} \in X$ . That is, the rate of convergence of  $f_m(\mathbf{x})$  to  $f(\mathbf{x})$  is uniform throughout the domain X, independent of where  $\mathbf{x} \in X$  is. On the contrary, pointwise convergence implies that for every point  $\mathbf{x} \in X$  given in advance, there exists  $N = N(\varepsilon, \mathbf{x})$ such that  $|f(\mathbf{x}) - f_m(\mathbf{x})| < \varepsilon$  for all  $m \geq N$  and for that particular  $\mathbf{x}$ , and we write

$$f_m \xrightarrow{m \to \infty} f, \quad \forall \mathbf{x} \in X.$$

In other words, uniform convergence implies pointwise convergence, but the converse is not necessarily true. A counter example is  $f_m(x) = x^m$  on X = [0,1]; show that while the sequence converges pointwise to f(x) = 0 when  $x \in [0,1)$  and f(x) = 1 when x = 1, it does not converge uniformly to f(x) at e.g.  $x = (3/4)^{1/m} < 1$  when e.g.  $\varepsilon = 1/2$ , because if it converges uniformly, it would converge to the limit (i.e. f(x) = 0 for x < 0) and we would get  $3/4 = |x^m - 0| < \varepsilon = 1/2$  which is a contradiction. Note that in this example the limiting function f(x) is discontinuous, and recall the uniform limit theorem that states if a sequence of a sequence of continuous functions converges uniformly to a limiting function, the limiting function must be continuous as well.

## 11.2 Weierstrass approximation theorem

Let C([a, b]) denote the space of real-valued continuous functions on the closed interval [a, b]on the real line, where  $a < b < \infty$ . Note that here we have  $X = [a, b] \subset \mathbb{R}$ . The Weierstrass approximation theorem states that the set of real-valued algebraic polynomials on [a, b] is dense in C([a, b]) with respect to the supremum norm. That is, given any function  $f \in C([a, b])$ , there exists a sequence of algebraic polynomials  $\{p_m\}_{m=1}^{\infty}$  such that  $p_m \to f$ uniformly. We say that  $f \in C([a, b])$  can be uniformly approximated as accurately as desired by an algebraic polynomial. The precise statement of the theorem follows.

Weierstrass Theorem. Given a function  $f \in C[a, b]$  and an arbitrary  $\varepsilon > 0$ , there exists an algebraic polynomial p such that  $|f(x) - p(x)| < \varepsilon$  for all  $x \in [a, b]$ , or equivalently  $||f - p||_{\infty} < \varepsilon$ .

**Proof.** A constructive proof of the Weierstrass theorem can be given using Bernstein polynomials; see the next two sub-sections.  $\Box$ 

We also note that a similar result holds for  $2\pi$ -periodic continuous functions and trigonometric polynomials, also due to Karl Weierstrass. That is, trigonometric polynomials are dense in the class of  $2\pi$ -periodic continuous functions. Recall that the space of algebraic polynomials of degree at most m is  $P_m = \text{span}\{1, x, \dots, x^m\}$ , while the space of trigonometric polynomials of degree at most m is  $T_m = \text{span}\{1, \sin x, \cos x \dots, \sin mx, \cos mx\}$ .

#### **11.3** Bernstein polynomials

A Bernstein polynomial of degree m is a linear combination of Bernstein basis polynomials:

$$B_m(x) = \sum_{k=0}^m \alpha_k \, b_{k,m}(x),$$

where  $b_{0,m}, \ldots, b_{m,m}$  are m+1 Bernstein basis polynomials of degree m,

$$b_{k,m}(x) = \binom{m}{k} x^k (1-x)^{m-k}, \qquad x \in [0,1], \qquad k = 0, 1, \dots, m.$$

with the binomial coefficients defined as

$$\binom{m}{k} = \frac{m!}{k!(m-k)!}$$

Figure 5 displays Bernstein basis polynomials of degree m = 2 (left) and m = 3 (right).



Figure 5: Bernstein basis polynomials of degree m = 2 (left) and m = 3 (right).

Bernstein basis polynomials have several interesting properties.

1. Bernstein basis polynomials of degree m form a basis for the vector space  $P_m$  of polynomials of degree at most m with real coefficients,

$$P_m = \operatorname{span}\{1, x, \dots, x^m\} = \operatorname{span}\{b_{0,m}, b_{1,m}, \dots, b_{m,m}\}, \qquad \forall m \in \mathbb{N}_0 := \{0, 1, 2, \dots\}.$$

2. Bernstein basis polynomials of degree m form a partition of unity,

$$\sum_{k=0}^{m} b_{k,m}(x) = 1.$$
(13)

This easily follows from the binomial theorem

$$(x+y)^m = \sum_{k=0}^m \binom{m}{k} x^k y^{m-k},$$

and setting y = 1 - x. However, since the support of all basis functions is the whole interval [0, 1], we obtain a partition of unity with a set of functions that are not locally supported on [0, 1]. That is, at each point  $x \in [0, 1]$  we cannot find a neighborhood of x where all but a finite number of basis functions in the set are 0.

- 3. There is an interesting connection between Bernstein basis polynomials and binomial random variables. To illustrate this, fix a point  $x \in [0, 1]$  and a degree m. Consider a random experiment where we carry out a sequence of m independent (Bernoulli) experiments, each with a Boolean-valued outcome: success with probability x and failure with probability 1 x. Then the probability of observing k successes in m independent Bernoulli trials, denoted by Pr(k), is given precisely by the Bernstein basis polynomial  $b_{k,m}(x) = {m \choose k} x^k (1-x)^{m-k}$ . In other words, a Bernstein basis polynomial  $b_{k,m}(x)$  is the probability mass function of a discrete random variable k that follows a binomial distribution, taking values  $0, 1, \ldots, m$ .
- 4. One can show that the expectation and variance of the discrete random variable  $\frac{k}{m}$  are given by:

$$\mathbb{E}\left[\frac{k}{m}\right] = \sum_{k=0}^{m} \frac{k}{m} \Pr(k) = \frac{1}{m} \sum_{k=0}^{m} k\binom{m}{k} x^{k} (1-x)^{m-k} = x,$$
$$\mathbb{V}\left[\frac{k}{m}\right] = \mathbb{E}\left[\left(\frac{k}{m} - x\right)^{2}\right] = \sum_{k=0}^{m} \left(\frac{k}{m} - x\right)^{2} \binom{m}{k} x^{k} (1-x)^{m-k} = \frac{x(1-x)}{m}.$$
(14)

This can be shown either by the binomial theorem or simply by the rules of probability. For instance, let  $k \sim B(m, x)$  be a binomially distributed random variable. Noting that  $k = k_1 + \ldots + k_m$  is the sum of m independent Bernoulli random variables each with expected value  $\mathbb{E}[k_i] = x, i = 1, \ldots, m$ , we can write

$$\mathbb{E}[k] = \mathbb{E}[k_1 + \ldots + k_m] = \mathbb{E}[k_1] + \ldots + \mathbb{E}[k_m] = m x.$$

The formula for variance follows similarly, noting that the variance of the sum of m independent variables is the sum of m variances, each equal to  $\mathbb{V}[k_i] = \mathbb{E}[k_i^2] - \mathbb{E}[k_i]^2 = x - x^2 = x(1-x), i = 1, ..., m.$ 

#### 11.4 A constructive proof of Weierstrass theorem

Consider a target function  $f \in C([0, 1])$ . We construct a Bernstein polynomial of degree at most m using a linear combination of Bernstein basis polynomials with coefficients f(k/m):

$$p_m(x) = \sum_{k=0}^m f(\frac{k}{m}) b_{k,m}(x).$$

Then by (13), we can write

$$p_m(x) - f(x) = \sum_{k=0}^m \left( f(\frac{k}{m}) - f(x) \right) b_{k,m}(x).$$

By triangle inequality, we then obtain

$$|p_m(x) - f(x)| \le \sum_{k=0}^m \left| f(\frac{k}{m}) - f(x) \right| b_{k,m}(x).$$
(15)

Continuity of f implies that there exists  $\delta > 0$  such that

$$|f(x_1) - f(x_2)| < \varepsilon \quad \text{whenever} \quad |x_1 - x_2| < \delta.$$
(16)

Continuity of f over a bounded set (here the interval [0, 1]) also implies that the function is bounded, that is,

$$M := \sup |f| < \infty. \tag{17}$$

We now split the sum in (15) into two parts and write

$$|p_m(x) - f(x)| \le \sum_{k: |x-k/m| < \delta} \left| f(\frac{k}{m}) - f(x) \right| b_{k,m}(x) + \sum_{k: |x-k/m| \ge \delta} \left| f(\frac{k}{m}) - f(x) \right| b_{k,m}(x).$$
(18)

The first sum in the right hand side of (18) is bounded by  $\varepsilon$ , thanks to (13) and (16) (with  $x_1 = x$  and  $x_2 = k/m$ ). In order to bound the second sum, we note that

$$\left|f(\frac{k}{m}) - f(x)\right| \le 2\sup|f| = 2M.$$

We then write

$$\sum_{k: |x-k/m| \ge \delta} \left| f\left(\frac{k}{m}\right) - f(x) \right| b_{k,m}(x) \le 2M \sum_{k: |x-k/m| \ge \delta} b_{k,m}(x)$$
$$\le 2M \sum_{k} \frac{(x-k/m)^2}{\delta^2} b_{k,m}(x)$$
$$= \frac{2M}{\delta^2} \frac{x(1-x)}{m}$$
$$\le \frac{2M}{\delta^2 m},$$

where the equality in the third step above follows from (14), and the inequality in the fourth step follows from  $0 \le x(1-x) \le 1$ . We finally obtain

$$|p_m(x) - f(x)| \le \varepsilon + \frac{2M}{\delta^2 m}.$$

Hence, since by (17)  $M < \infty$ , and  $\varepsilon > 0$  can be selected arbitrarily small, we get  $p_m \to f$  uniformly. The proof will be complete by simply extending Bernstein polynomials from [0, 1] to [a, b].

## 11.5 Stone-Weierstrass theorem

Stone generalized and proved the Weierstrass approximation theorem by replacing the closed interval [a, b] by an arbitrary compact Hausdorff space X and considering the space C(X) of real-valued continuous functions on X. We note that a Hausdorff space X is a topological space where for every two distinct points  $x, y \in X$  there exists a neighborhood U of x and a neighborhood V of y such that  $U \cap V = \emptyset$ . That is, any two points in X are separated by their neighborhoods that are disjoint. For example, all metric spaces (such as the Euclidean space  $\mathbb{R}^d$ ) are Hausdorff.

A particular case of Stone-Weierstrass theorem is when  $X = [a_1, b_1] \times \ldots \times [a_d, b_d] \subset \mathbb{R}^d$ . This case can be easily proved by generalizing 1D Bernstein basis polynomials to higher dimensions

$$b_{\mathbf{k},\mathbf{m}}(\mathbf{x}) = \prod_{i=1}^{d} b_{k_i,m_i}(x_i) = b_{k_1,m_1}(x_1) \dots b_{k_d,m_d}(x_d), \qquad \mathbf{x} = (x_1,\dots,x_d) \in X,$$

where  $\mathbf{k} = (k_1, \ldots, k_d)$  and  $\mathbf{m} = (m_1, \ldots, m_d)$  are two *d*-dimensional multi-indices.

## 12 Density of two-layer networks

In this section we will discuss the main density result for two-layer (i.e. shallow) feedforward networks.

#### 12.1 Two-layer feedforward networks

We will consider the family of two-layer feedforward networks with d input neurons, one hidden layer consisting of r neurons, where all r neurons use the same activation function  $\sigma: \mathbb{R} \to \mathbb{R}$ , and one output neuron without any activation function and without any bias. A schematic representation of the network with d = 2 inputs and r = 3 neurons in the hidden layer is displayed in Figure 6.



Figure 6: A representation of a feed-forward network with one hidden layer.

Let  $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$  be the input variable. Then, the output  $y \in \mathbb{R}$  of the network can be written as

$$y = \sum_{i=1}^{r} W_{1,i}^2 \, \sigma(\sum_{j=1}^{d} W_{i,j}^1 x_j + b_i),$$

where  $W^1 \in \mathbb{R}^{r \times d}$  and  $W^2 \in \mathbb{R}^{1 \times r}$  are the weight matrices of the two layers, and  $\mathbf{b} = (b_1, \ldots, b_r) \in \mathbb{R}^r$  is the bias in the hidden layer; see the notation in Chapter 1. Note that

there is no output bias here. Setting  $\mathbf{w}^{(i)} := (W_{i,1}^1, \ldots, W_{i,d}^1) \in \mathbb{R}^d$ , i.e. the *i*-th row of the weight matrix  $W^1$ , and  $c_i := W_{1,i}^2$ , i.e. the *i*-th column of the weight matrix  $W^2$ , we can write the network's output more succinctly as

$$y = \sum_{i=1}^{r} c_i \,\sigma(\mathbf{w}^{(i)} \cdot \mathbf{x} + b_i),$$

where  $\mathbf{a} \cdot \mathbf{b} := \sum_{j=1}^{d} a_j b_j$  is the inner product of  $\mathbf{a} \in \mathbb{R}^d$  and  $\mathbf{b} \in \mathbb{R}^d$ .

## 12.2 Pinkus theorem

We will consider the density question with such networks. Precisely, we consider the network space (corresponding to the case  $r \to \infty$ ),

$$\mathcal{M}(\sigma) := \operatorname{span}\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^d, \ b \in \mathbb{R}\},\$$

and ask for which class of activation functions the network space  $\mathcal{M}(\sigma)$  is dense in the space  $C(\mathbb{R}^d)$  of continuous functions with respect to the supremum norm (i.e. in the topology of uniform convergence on compact sets). Equivalently, we would like to know for what  $\sigma$ , given a target function  $f \in C(\mathbb{R}^d)$  and a compact subset  $X \subset \mathbb{R}^d$  and an arbitrary  $\varepsilon > 0$ , there exists  $g \in \mathcal{M}(\sigma)$  such that

$$\sup_{\mathbf{x}\in X} |f(\mathbf{x}) - g(\mathbf{x})| < \varepsilon.$$

Formally, we state the main density result as follows; see Theorem 3.1 in [23].

**Pinkus Theorem.** Let  $\sigma \in C(\mathbb{R})$ . Then  $\mathcal{M}(\sigma)$  is dense in  $C(\mathbb{R}^n)$  with respect to the supremum norm on compact sets, if and only if  $\sigma$  is not a polynomial.

An intuitive example. Let d = 1 and assume that X = [a, b] is a closed interval on  $\mathbb{R}$ . Consider a cosine activation function  $\sigma(x) = \cos x$ , which satisfies the conditions of the above theorem, i.e. it is continuous and not a polynomial. Then  $g(x) = \sum_{i\geq 1} c_i \cos(w_i x + b_i)$ , which reminds us of Fourier series. In fact this is the so called amplitude-phase form of Fourier series from which we can recover the more familiar sine-cosine form by the identity  $\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$ . Recall that if we have a continuous function, we can expand it in Fourier series, and hence it follows that g is dense in the space of continuous functions.

## 12.3 Proof sketch of Pinkus theorem

First, we need to show that if  $\mathcal{M}(\sigma)$  is dense, then  $\sigma$  is not a polynomial. Suppose  $\sigma \in P_m(\mathbb{R})$  is a polynomial of degree m, then for every choice of  $\mathbf{w} \in \mathbb{R}^d$  and  $b \in \mathbb{R}$ ,  $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$  is a multivariate polynomial of total degree at most m, and thus  $\mathcal{M}(\sigma)$  is the space of all polynomials of total degree at most m, that is  $\mathcal{M}(\sigma) = P_m(\mathbb{R}^d)$ , which does not span  $C(\mathbb{R}^d)$ , contradicting the density. We next show the converse result: if  $\sigma$  is not a polynomial, then  $\mathcal{M}(\sigma)$  is dense. This is done in four steps:

• Step 1. Consider the 1D case (i.e. d = 1) and the 1D counterpart of  $\mathcal{M}(\sigma)$ :

$$\mathcal{N}(\sigma) = \operatorname{span}\{\sigma(w\,x+b), \, w, b \in \mathbb{R}\}.$$

- Step 2. Show that for  $\sigma \in C^{\infty}(\mathbb{R})$  and not a polynomial,  $\mathcal{N}(\sigma)$  is dense in  $C(\mathbb{R})$ .
- Step 3. Weaken the smoothness demand on  $\sigma$ , using convolution by mollifiers, and show that for  $\sigma \in C(\mathbb{R})$  and not a polynomial,  $\mathcal{N}(\sigma)$  is dense in  $C(\mathbb{R})$ .
- Step 4. Extend the result to multiple dimensions: show that if  $\mathcal{N}(\sigma)$  is dense in  $C(\mathbb{R})$  then  $\mathcal{M}(\sigma)$  is dense in  $C(\mathbb{R}^d)$ .

Proof of step 2. We will use a very interesting theorem asserting that if a smooth function (i.e.  $C^{\infty}$ ) on an interval is such that its Taylor expansion about every point of the interval has at least one coefficient equal to zero, then the function is a polynomial.

**Lemma 1.** Let  $\sigma \in C^{\infty}((\alpha, \beta))$ , where  $(\alpha, \beta) \subset \mathbb{R}$  is an open interval on the real line. If for every point  $x \in (\alpha, \beta)$  on the interval there exists an integer k = k(x) such that the k-th derivative of  $\sigma$  vanishes at x, i.e.  $\sigma^{(k)}(x) = 0$ , then  $\sigma$  is a polynomial.

*Proof.* For a proof of this lemma see page 53 in [10].

Lemma 1 implies that since  $\sigma$  is smooth and not a polynomial, there exists a point  $b_0 \in \mathbb{R}$  at which  $\sigma^{(k)} \neq 0, k = 0, 1, 2, \ldots$  Now we note that

$$\frac{\sigma((w+h)x+b_0)-\sigma(wx+b_0)}{h} \in \mathcal{N}(\sigma), \qquad \forall h \neq 0,$$

where  $w \in \mathbb{R}$ . Taking the limit (as  $h \to 0$ ), it follows that the derivative of  $\sigma$  with respect to w is in  $\overline{\mathcal{N}(\sigma)}$ , which is the closure of  $\mathcal{N}(\sigma)$ . In particular, for w = 0 we get

$$\left. \frac{d}{dw} \sigma(w \, x + b_0) \right|_{w=0} = x \, \sigma'(b_0) \in \overline{\mathcal{N}(\sigma)},$$

Similarly (by considering k+1 terms of  $\sigma$  and taking the limit  $h \to 0$ ) we can show that

$$\left. \frac{d^k}{dw^k} \sigma(w \, x + b_0) \right|_{w=0} = x^k \, \sigma^{(k)}(b_0) \in \overline{\mathcal{N}(\sigma)}.$$

Since  $\sigma^{(k)} \neq 0$ ,  $k = 0, 1, 2, \ldots$ , the set  $\overline{\mathcal{N}(\sigma)}$  contains all monomials. By the Weierstrass Theorem this implies that  $\overline{\mathcal{N}(\sigma)}$  and hence  $\mathcal{N}(\sigma)$  is dense in  $C(\mathbb{R})$ , because if the closure of a function space is dense, the function space will be dense too: we can approximate any function in the closure space by functions in the space as accurately as we wish.

*Proof of step 3.* The proof utilizes the classical technique of convolution by mollifiers to weaken the smoothness requirement of  $\sigma$ . In this technique we consider a mollified activation

function  $\sigma_{\phi}(x)$  obtained by convolving  $\sigma \in C(\mathbb{R})$  with a smooth and compactly supported mollifier  $\phi \in C_0^{\infty}(\mathbb{R})$ ,

$$\sigma_{\phi}(x) = \int_{\mathbb{R}} \sigma(x-y) \,\phi(y) \, dy.$$

Since both  $\sigma$  and  $\phi$  are continuous and  $\phi$  has compact support, the above integral exists for all  $x \in \mathbb{R}$ . We also have  $\sigma_{\phi} \in C^{\infty}(\mathbb{R})$ . Moreover, taking the limit of Riemann sums, one can easily show that  $\sigma_{\phi}$  and  $\overline{\mathcal{N}(\sigma_{\phi})}$  are contained in  $\overline{\mathcal{N}(\sigma)}$ . Now since  $\sigma_{\phi} \in C^{\infty}(\mathbb{R})$ , provided we choose  $\phi$  such that  $\sigma_{\phi}$  is not a polynomial (also note that  $\sigma$  is not a polynomial), then by the method of proof of Step 2, all monomials are contained in  $\overline{\mathcal{N}(\sigma_{\phi})}$ . Hence  $\overline{\mathcal{N}(\sigma_{\phi})}$  and therefore  $\overline{\mathcal{N}(\sigma)}$  and  $\mathcal{N}(\sigma)$  are dense in  $C(\mathbb{R})$ .

Proof of step 4. One interesting technique to "reduce dimension" (here we want to reduce d to 1) is to utilize **ridge functions**, also known as plane waveforms in the context of hyperbolic PDEs. A ridge function  $g : \mathbb{R} \to \mathbb{R}$  is a multivariate function  $f : \mathbb{R}^d \to \mathbb{R}$  of the form

$$f(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}), \quad \mathbf{w} \in \mathbb{R}^d \setminus \{\mathbf{0}\}.$$

Figure 7 displays an example of a ridge function  $g(\mathbf{w} \cdot \mathbf{x}) = \sin \mathbf{w} \cdot \mathbf{x}$ , where  $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ , with three choices  $\mathbf{w} = (0, 1)$  (left),  $\mathbf{w} = (1, 0)$  (middle), and  $\mathbf{w} = (1/\sqrt{2}, 1/\sqrt{2})$  (right).



Figure 7: An example of a ridge function  $g(\mathbf{w} \cdot \mathbf{x}) = \sin \mathbf{w} \cdot \mathbf{x}$  in two dimensions, where  $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ , and for three different direction vectors  $\mathbf{w} = (0, 1)$  (left),  $\mathbf{w} = (1, 0)$  (middle), and  $\mathbf{w} = (1/\sqrt{2}, 1/\sqrt{2})$  (right).

As we observe, the vector  $\mathbf{w} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$  determines the direction of the plane wave. Importantly, for each  $\mathbf{w} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$  and  $b \in \mathbb{R}$ , the function  $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$  that is the building block of  $\mathcal{M}(\sigma)$  is also a ridge functions. Now let

$$\mathcal{R} := \operatorname{span}\{g(\mathbf{w} \cdot \mathbf{x}) : \mathbf{w} \in \mathbb{R}^d, ||\mathbf{w}||_2 = 1, g \in C(\mathbb{R})\}.$$

We first note that  $\mathcal{R}$  is dense in  $C(\mathbb{R}^d)$ , because it contains all functions of the form  $\cos(\mathbf{w} \cdot \mathbf{x})$ and  $\sin(\mathbf{w} \cdot \mathbf{x})$  which are dense on any compact subset of  $C(\mathbb{R}^d)$ . Of course, this does not directly imply that  $\mathcal{M}(\sigma)$  will also be dense, because g in  $\mathcal{R}$  can be any continuous function of  $\mathbf{w} \cdot \mathbf{x}$ , while  $\sigma$  is a fixed, given continuous function. However, if ridge functions in  $\mathcal{R}$  were not dense in  $C(\mathbb{R}^d)$ , then it would not be possible for  $\mathcal{M}(\sigma)$  to be dense in  $C(\mathbb{R}^d)$ . Now let  $f \in C(X)$  be a given target function on some compact set  $X \subset \mathbb{R}^d$ . Since  $\mathcal{R}$  is dense in C(X), then given  $\varepsilon > 0$  there exist  $\{g_i\}_{i=1}^r \in C(\mathbb{R})$  and  $\mathbf{w}^{(i)} \in \mathbb{R}^d$  with  $||\mathbf{w}^{(i)}||_2 = 1$ ,  $i = 1, \ldots, r$  (for some r) such that

$$|f(\mathbf{x}) - \sum_{i=1}^{r} g_i(\mathbf{w}^{(i)} \cdot \mathbf{x})| < \frac{\varepsilon}{2}, \quad \forall \mathbf{x} \in X.$$

Since X is compact, then for each i = 1, ..., r, we have  $\{\mathbf{w}^{(i)} \cdot \mathbf{x} : \mathbf{x} \in X\} \subseteq [a_i, b_i]$  for some bounded interval  $[a_i, b_i]$ . We next utilize the fact that  $\mathcal{N}(\sigma)$  is dense in  $C([a_i, b_i])$ , for all i = 1, ..., r, which implies that there exist  $c_{i,j}, w_{i,j}, b_{i,j} \in \mathbb{R}$ , with  $j = 1, ..., m_i$  for some  $m_i$ , such that

$$|g_i(t) - \sum_{j=1}^{m_i} c_{i,j} \,\sigma(w_{i,j} \,t + b_{i,j})| < \frac{\varepsilon}{2 \,r}, \qquad \forall t \in [a_i, b_i], \quad i = 1, \dots, r.$$

Hence, combining the above two inequalities, we get

$$\begin{aligned} |f(\mathbf{x}) - \sum_{i=1}^{r} \sum_{j=1}^{m_{i}} c_{i,j} \,\sigma(w_{i,j} \,\mathbf{w}^{(i)} \cdot \mathbf{x} + b_{i,j})| &= \\ |f(\mathbf{x}) - \sum_{i=1}^{r} g_{i}(\mathbf{w}^{(i)} \cdot \mathbf{x}) + \sum_{i=1}^{r} g_{i}(\mathbf{w}^{(i)} \cdot \mathbf{x}) - \sum_{i=1}^{r} \sum_{j=1}^{m_{i}} c_{i,j} \,\sigma(w_{i,j} \,\mathbf{w}^{(i)} \cdot \mathbf{x} + b_{i,j})| &\leq \\ |f(\mathbf{x}) - \sum_{i=1}^{r} g_{i}(\mathbf{w}^{(i)} \cdot \mathbf{x})| + \sum_{i=1}^{r} |g_{i}(\mathbf{w}^{(i)} \cdot \mathbf{x}) - \sum_{j=1}^{m_{i}} c_{i,j} \,\sigma(w_{i,j} \,\mathbf{w}^{(i)} \cdot \mathbf{x} + b_{i,j})| &< \\ \frac{\varepsilon}{2} + r \, \frac{\varepsilon}{2r} = \varepsilon, \qquad \forall \, \mathbf{x} \in X. \end{aligned}$$

Clearly, this implies that there exist  $c_{\ell}, b_{\ell} \in \mathbb{R}$  and  $\mathbf{w}^{(\ell)} \in \mathbb{R}^d$ , with  $\ell = 1, \ldots, m$  for some m, such that

$$|f(\mathbf{x}) - \sum_{\ell=1}^{m} c_{\ell} \,\sigma(\mathbf{w}^{(\ell)} \cdot \mathbf{x} + b_{\ell})| < \varepsilon, \qquad \forall \, \mathbf{x} \in X.$$

This in turn means that there exist  $f_{\theta} \in \mathcal{M}(\sigma)$ , given by  $f_{\theta}(\mathbf{x}) = \sum_{\ell=1}^{m} c_{\ell} \sigma(\mathbf{w}^{(\ell)} \cdot \mathbf{x} + b_{\ell})$  for some m, such that

 $|f(\mathbf{x}) - f_{\boldsymbol{\theta}}(\mathbf{x})| < \varepsilon, \qquad \forall \, \mathbf{x} \in X.$ 

Hence,  $\mathcal{M}(\sigma)$  is dense in C(X).

## 13 Convergence rate of approximation by two-layer networks

So far we have discussed the concept of density. We have seen that density addresses only the ability of approximation. It does not tell us anything about the rate of approximation. In particular, density does not imply that one can approximate well by an approximant from the set

$$\mathcal{M}_r(\sigma) := \{\sum_{i=1}^r c_i \, \sigma(\mathbf{w}^{(i)} \cdot \mathbf{x} + b_i), : \, \mathbf{w}^{(i)} \in \mathbb{R}^d, \, c_i, b_i \in \mathbb{R}\},\$$

with a fixed  $r < \infty$ . This is similar to the case of algebraic polynomials: the space of polynomials is dense in C(X), but polynomials of any fixed degree are rather sparse (not

dense). In other words, density assumes that the number of parameters in the approximant (such as the number r of neurons and the degree of polynomials) is infinite, and not fixed and finite. When r is fixed and finite, the rate question asks: is there  $\gamma > 0$  such that

$$\inf_{g \in \mathcal{M}_r(\sigma)} \sup_{\mathbf{x} \in X} |f(\mathbf{x}) - g(\mathbf{x})| < C r^{-\gamma},$$

for some constant C > 0? Such estimates would tell us the rate (how fast/slow) at which the error converges to zero. The infimum is taken because we want to know the "best" possible convergence rate that can be achieved by the functions in  $\mathcal{M}_r(\sigma)$  (the best case scenario). This question will be discussed in this section for the same family of single hidden-layer feedforward networks considered in Section 12.1.

Of course, similar to density, the rate of approximation has its own limitations. For instance, rate of approximation does not tell us how to find that "best" approximant. It also does not tell us if there are (efficient) methods/algorithms for finding "good" approximants. Nevertheless, it gives us more information than density.

We also note that when  $r \to \infty$ , then  $\mathcal{M}_r(\sigma)$  spans the whole  $\mathcal{M}(\sigma)$ . In fact, the density question in terms of the set  $\mathcal{M}_r(\sigma)$  reads: given  $f \in C(X)$ , is there  $g_r \in \mathcal{M}_r(\sigma)$  for which

$$\lim_{r \to \infty} \sup_{\mathbf{x} \in X} |f(\mathbf{x}) - g_r(\mathbf{x})| = 0.$$

Clearly, this question does not address the convergence rate of approximation.

Finally, we note that the set  $\mathcal{M}_r$  has the property

$$\mathcal{M}_{r_1} + \mathcal{M}_{r_2} = \mathcal{M}_{r_1 + r_2},$$

where the sumset is defined as  $\mathcal{M}_{r_1} + \mathcal{M}_{r_2} := \{g_1 + g_2 : g_1 \in \mathcal{M}_{r_1}, g_2 \in \mathcal{M}_{r_2}\}$ . This property will be used later to prove the main convergence result. The reason for this property is that  $\mathcal{M}_{r_1}$  and  $\mathcal{M}_{r_2}$  do not necessarily share basis functions  $\sigma(\mathbf{w}^{(i)} \cdot \mathbf{x} + b_i)$  with the very same  $(\mathbf{w}^{(i)}, b_i)$  parameters. In general we have  $\mathcal{M}_{r_1} = \operatorname{span}\{\sigma(\mathbf{w}^{(i)} \cdot \mathbf{x} + b_i)\}_{i=1}^{r_1}$  and  $\mathcal{M}_{r_2} = \operatorname{span}\{\sigma(\mathbf{w}^{(i)} \cdot \mathbf{x} + b_i)\}_{i=r_1+1}^{r_1+r_2}$ . In other words, the space  $\mathcal{M}_r$  is a nonlinear space. Note that this will make the neural networks in our setup nonlinear methods of approximation.

#### **13.1** Target functions of interest

Throughout Section 13 we let the domain X be the unit ball

$$X := \{ \mathbf{x} : ||\mathbf{x}||_2 \le 1 \} \subset \mathbb{R}^d, \qquad d \ge 2.$$

Moreover, instead of continuous target functions that we considered earlier, we will consider Sobolev target functions (see the definition of Sobolev function spaces in Section 13.2),

$$f \in W^{k,p}(X), \quad k \ge 1, \quad 1 \le p \le \infty, \quad ||f||_{W^{k,p}(X)} \le 1.$$

A major importance of Sobolev functions is that they often appear in the study of PDEs. We want to study the rate of approximation of  $f \in W^{k,p}(X)$  by  $g \in \mathcal{M}_r(\sigma)$  with fixed  $r < \infty$ .

## 13.2 Sobolev spaces

Here we will briefly review Sobolev spaces. For a more detailed discussion see Sec. 5 in [13]. **Strong derivatives.** Suppose  $f \in C^k(X)$ , for some non-negative integer k. Denote by  $\alpha = (\alpha_1, \ldots, \alpha_d)$  a multi-index of d non-negative integers of order  $|\alpha| = \alpha_1 + \ldots + \alpha_d \leq k$ . The (strong)  $\alpha$ -th partial derivative of f is defined as

$$D^{\alpha}f(\mathbf{x}) := \frac{\partial^{|\alpha|}f(\mathbf{x})}{\partial x_1^{\alpha_1} \dots \partial x_n^{\alpha_n}} = \partial_{x_1}^{\alpha_1} \dots \partial_{x_n}^{\alpha_n}f(\mathbf{x}), \qquad \mathbf{x} \in X.$$

Moreover, for every test function  $\phi \in C_0^{\infty}(X)$ , the following integration by parts formula holds

$$\int_X f D^{\alpha} \phi \, d\mathbf{x} = (-1)^{|\alpha|} \int_X D^{\alpha} f \, \phi \, d\mathbf{x}.$$

Note that there is no boundary terms involved in the above formula because tests functions have compact support on X, and hence all boundary terms vanish.

Weak derivatives. Sometimes (actually often), for example in the context of PDEs, we need to deal with functions f that have low regularity/smoothness. For instance, they may not belong to  $C^k(X)$ , i.e. they are not k-times continuously differentiable and hence  $D^{\alpha}f(\mathbf{x})$ would not exist in the above sense. For such functions we use a "weaker" notion of derivatives utilizing the integration by parts formula. Precisely, suppose  $f \in L^1(X)$  (this is why we assume  $p \ge 1$ ). We say that u is the weak  $\alpha$ -th partial derivative of f provided for all test functions  $\phi \in C_0^{\infty}(X)$  the following holds

$$\int_X f D^{\alpha} \phi \, d\mathbf{x} = (-1)^{|\alpha|} \int_X u \, \phi \, d\mathbf{x}.$$

In this case, we often use the same notation as in the case of strong derivatives and write

$$D^{\alpha}f = u$$

and say  $D^{\alpha}f$  is the  $\alpha$ -th partial derivative of f in the weak sense.

As an example, it is easy to see that the weak derivative of the ReLU activation function  $\sigma(x) = \max\{0, x\}$  on X = [-1, 1] is given by

$$u(x) = \begin{cases} 0 & -1 \le x < 0, \\ 1 & 0 \le x \le 1. \end{cases}$$

**Definition of Sobolev spaces.** Fix  $1 \le p \le \infty$ , and let k be a nonnegative integer. The Sobolev space  $W^{k,p}(X)$  consists of all functions  $f \in L^1(X)$  for which the derivatives  $D^{\alpha}f$  for all  $\alpha$  with  $|\alpha| \le k$  exist in the weak sense and belong to  $L^p(X)$ . We write

$$W^{k,p}(X) := \{ f : X \to \mathbb{R}; \ D^{\alpha} f \in L^p(X), \ |\alpha| \le k \}$$

where we understand  $D^{\alpha}f$  in the weak sense. In the particular case when p = 2, we usually use the letter H and omit p = 2, and write

$$H^k(X) := W^{k,2}(X),$$

which is a Hilbert space. Obviously,  $H^0(X) = W^{0,2}(X) = L^2(X)$ , which is the space of square-integrable functions on X.

**Remark 5.** Recall that the  $L^p$  space (sometimes called Lebesgue space) is defined as

$$L^{p}(X) := \{ f : X \to \mathbb{R}; \, ||f||_{L^{p}(X)} < \infty \},\$$

where

$$||f||_{L^p(X)} := \left(\int_X |f|^p \, d\mathbf{x}\right)^{1/p}, \qquad 1 \le p < \infty,$$

and

$$||f||_{L^{\infty}(X)} := ess \, sup_X |f| := \inf\{v \in \mathbb{R} | \ \mu(\{x \in X : |f| > v\}) = 0\}.$$

Here, for any subset  $A \subset X$  in  $\mathbb{R}^d$ ,  $\mu(A)$  denotes the d-dimensional Lebesgue measure of A, which determines the size or d-dimensional volume of the subset A. Informally,  $||f||_{L^{\infty}(X)}$  is the "almost everywhere" supremum of |f| over X, that is, the supremum of |f| everywhere on X except on a set of measure zero.

**Definition of Sobolev norm.** If  $f \in W^{k,p}(X)$ , we define its norm as

$$||f||_{W^{k,p}(X)} := \left(\sum_{|\alpha| \le k} ||D^{\alpha}f||_{L^{p}(X)}^{p}\right)^{1/p}, \qquad 1 \le p < \infty,$$

and

$$||f||_{W^{k,\infty}(X)} := \sum_{|\alpha| \le k} ||D^{\alpha}f||_{L^{\infty}(X)}.$$

## 13.3 A typical convergence rate result

**Error of interest.** To study the convergence rate, we will first need to define an error that we call the error of interest. We will consider the following error

$$\sup_{f \in W^{k,p}(X)} \inf_{g \in \mathcal{M}_r(\sigma)} ||f - g||_{L^p(X)}.$$

Clearly  $W^{k,p}$  is a subspace of  $L^p$ . Moreover, since X is compact and C(X) is dense in  $L^p(X)$ , then by the density result in Section 12,  $\mathcal{M}_r(\sigma)$  will also be dense in  $L^p(X)$  for each  $\sigma \in C(\mathbb{R})$ that is not a polynomial. Hence, the  $L^p$  space contains both  $W^{k,p}$  and  $\mathcal{M}_r(\sigma)$ , and therefore  $||f-g||_{L^p(X)}$  in the above error is well defined. The infimum over  $g \in \mathcal{M}_r(\sigma)$  is taken because fixing f we want to know how close the space  $\mathcal{M}_r(\sigma)$  can be to f (best case scenario). The supremum over  $f \in W^{k,p}(X)$  is taken because we want to consider the worst function f to be approximated (worst case scenario).

A Convergence Theorem. (Theorem 6.8 in [23]; also see Theorem 2.1 in [21]) Let  $\sigma \in C^{\infty}(I)$  be a smooth function on some open interval  $I \subseteq \mathbb{R}$ , and suppose that it is not a polynomial on I. Then, for each  $p \in [1, \infty]$ ,  $k \ge 1$ , and  $d \ge 2$ , there is a constant C > 0 independent of r such that

$$\sup_{f \in W^{k,p}(X)} \inf_{g \in \mathcal{M}_r(\sigma)} ||f - g||_{L^p(X)} \le C r^{-k/d}.$$

**Remark 6.** Note that, as stated above, we are also assuming that  $||f||_{W^{k,p}(X)} \leq 1$ . In the general case when the norm of f is bounded by another finite number, e.g.  $||f||_{W^{k,p}(X)} \leq a < \infty$ , the constant C may also depend on a (i.e. the size of f). See Section 14 for this more general case.

The proof of convergence theorem utilizes homogeneous polynomials, and hence here we quickly review them.

**Homogeneous polynomials.** A homogeneous polynomial of degree m in  $d \ge 2$  variables in  $\mathbb{R}^d$  is a polynomial whose non-zero terms all have the same degree m. For example,  $x_1^5 - 2x_1^3x_2^2 + x_1x_2^4$  is a homogeneous polynomial of degree m = 5 in two variables with d = 2. Let us denote by  $Q_m$  the linear space of such polynomials, i.e. the set of all terms  $\mathbf{x}^{\alpha} := \prod_{i=1}^d x_i^{\alpha_i}$  such that  $|\alpha| = \alpha_1 + \ldots + \alpha_d = m$ . A few interesting properties of homogeneous polynomials relevant to our convergence problem follows.

1. The dimension of  $Q_m$  is the maximal number of non-zero terms  $\mathbf{x}^{\alpha}$  with  $|\alpha| = m$ , or equivalently the cardinality of the set  $\{\alpha = (\alpha_1, \ldots, \alpha_d) : |\alpha| = m\}$ , which is

$$s := \dim Q_m = \binom{m+d-1}{m} \le C m^{d-1},$$

where C = C(d) is a constant that may depend only on d. The binomial formula can easily be shown by the method of stars and bars, which is a well-known technique in combinatorics. We skip its interesting proof here. The inequality is a simple consequence of  $\binom{m+d-1}{m} \leq \left(\frac{e(m+d-1)}{d-1}\right)^{d-1}$  and  $(a+b)^{d-1} \leq 2^{d-2} (a^{d-1}+b^{d-1})$ , for  $d \geq 2$ .

2. For a  $\mathbf{w} \in \mathbb{R}^d$  with unit norm  $||\mathbf{w}||_2 = 1$ , we have  $(\mathbf{w} \cdot \mathbf{x})^m \in Q_m$ . This can easily be seen by the multinomial theorem (which is a generalization of the binomial theorem),

$$(x_1 + \ldots + x_d)^m = \sum_{|\alpha|=m} \binom{m}{\alpha} \mathbf{x}^{\alpha}, \qquad \binom{m}{\alpha} := \frac{m!}{\prod_{i=1}^d \alpha_i!}, \qquad \mathbf{x}^{\alpha} := \prod_{i=1}^d x_i^{\alpha_i}.$$

3. Consequently, there exist  $s = \dim Q_m$  points  $\mathbf{w}^{(i)} \in \mathbb{R}^d$  with  $||\mathbf{w}^{(i)}||_2 = 1, i = 1, \ldots, s$ , such that  $\{(\mathbf{w}^{(i)} \cdot \mathbf{x})^m\}_{i=1}^s$  spans  $Q_m$ . This in particular implies that

$$Q_{\ell} = \operatorname{span}\{(\mathbf{w}^{(i)} \cdot \mathbf{x})^{\ell} : i = 1, \dots, s\}, \qquad \ell = 0, 1, \dots, m.$$

Hence, for the linear space  $P_m$  of multivariate polynomials of degree at most m,

$$P_m = \bigcup_{\ell=0}^m Q_\ell,$$

we have

$$P_m = \text{span}\{(\mathbf{w}^{(i)} \cdot \mathbf{x})^{\ell} : i = 1, \dots, s, \ell = 0, 1, \dots, m\}.$$

This last formula can also be written as

$$P_m = \{\sum_{i=1}^{s} p_i(\mathbf{w}^{(i)} \cdot \mathbf{x}) : p_i \in \pi_m, i = 1, \dots, s\}$$

where  $\pi_m$  denotes the linear space of univariate polynomials of degree at most m.
**Proof Sketch of Convergence Theorem.** Let  $Q_{\ell}$  denote the linear space of homogeneous polynomials of degree  $\ell$  in  $\mathbb{R}^d$ , and let  $P_m := \bigcup_{\ell=0}^m Q_{\ell}$  be the linear space of multivariate polynomials of degree at most m. By the third property of Homogeneous polynomials discussed above, there exists  $s = \dim Q_m$  points  $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(s)} \in \mathbb{R}^d$  with  $||\mathbf{w}^{(i)}||_2 = 1, i = 1, \ldots, s$ , such that

$$P_m = \{\sum_{i=1}^{s} p_i(\mathbf{w}^{(i)} \cdot \mathbf{x}) : p_i \in \pi_m, i = 1, \dots, s\},\$$

where  $\pi_m$  denotes the linear space of univariate algebraic polynomials of degree at most m. Now, if we consider the 1D counterpart of  $\mathcal{M}_r(\sigma)$ :

$$\mathcal{N}_r(\sigma) = \{\sum_{i=1}^r c_i \, \sigma(w_i \, x - b_i), \, c_i, w_i, b_i \in \mathbb{R}\},\$$

by the method of proof of step 2 in Section 12.3, under the conditions stated in the above theorem,  $\overline{\mathcal{N}_{m+1}(\sigma)}$ , which is the closure of  $\mathcal{N}_{m+1}(\sigma)$ , contains the space  $\pi_m$  of univariate algebraic polynomials of degree at most m. Note that in this method to show a monomial of degree up to m is in the network space, we will need up to the m-th derivative of  $\sigma$ . Using divided differences, this would need a linear combination of m+1 terms (e.g. we need 2 terms for first derivative, three terms for second derivative, etc.), and this amounts to a network space  $\mathcal{N}_{m+1}(\sigma)$  with m+1 basis functions. Now from  $p_i \in \pi_m$ , it follows that  $p_i \in \overline{\mathcal{N}_{m+1}(\sigma)}$ . Hence, using the property  $\mathcal{M}_{r_1} + \mathcal{M}_{r_2} = \mathcal{M}_{r_1+r_2}$ , we obtain

$$P_m \subseteq \overline{\mathcal{M}_{s(m+1)}(\sigma)}.$$

By setting r = s (m + 1), where  $s = \dim Q_m$ , we get  $P_m \subseteq \overline{\mathcal{M}_r(\sigma)}$ , which implies

$$\sup_{f \in W^{k,p}(X)} \inf_{g \in \mathcal{M}_{r}(\sigma)} ||f - g||_{L^{p}(X)} = \sup_{f \in W^{k,p}(X)} \inf_{g \in \mathcal{M}_{r}(\sigma)} ||f - g||_{L^{p}(X)}$$
$$\leq \sup_{f \in W^{k,p}(X)} \inf_{g \in P_{m}} ||f - g||_{L^{p}(X)}.$$

It is also well known (see e.g. Section 5 in [6]) that for every  $f \in W^{k,p}(X)$ , there exists a polynomial  $g \in P_m$  such that

$$||f - g||_{L^p(X)} \le C m^{-k} ||f||_{W^{k,p}(X)}.$$

Noting  $||f||_{W^{k,p}(X)} \leq 1$ , this implies that

$$\sup_{f \in W^{k,p}(X)} \inf_{g \in P_m} ||f - g||_{L^p(X)} \le C \, m^{-k}.$$

The desired estimate then follows noting that  $s = \dim Q_m \leq C(d) m^{d-1}$ .

#### 13.4 A short discussion on curse of dimension

Suppose that we want to construct an approximation  $g \in \mathcal{M}_r(\sigma)$  of a target function  $f \in W^{k,p}$ with an error at most  $\varepsilon > 0$ . Suppose further that the approximation error is proportional to

 $r^{-k/d}$ , as the above theorem suggests. This means that we need  $r = \mathcal{O}(\varepsilon^{-d/k})$  neurons, which demonstrates the **curse of dimension**: the computational cost (encoded in r) increases exponentially with the dimension d.

One of the main reasons for current interest in deep neural networks is their outstanding performance in high-dimensional problems, i.e. when d is large. However, the convergence rate  $\mathcal{O}(r^{-k/d})$  in the theorem in Section 13.3 depends strongly on the dimension d of the input space. Indeed, the theorem suggests that the rate of approximation from  $\mathcal{M}_r(\sigma)$  with  $r \sim m^d$ (note:  $r = s(m + 1) \sim m^d$ ) may not be better than the rate of polynomial approximation from the polynomial space  $P_m$  (which is  $\sim m^{-k}$ ). This makes one wonder whether it is really worthwhile using neural networks (at least in the case of a single hidden layer). One may think that the upper bound in the above theorem may not be sharp. However, there are lower bounds that confirm this is not the case; see e.g. [21, 23, 18]. For instance, with  $\sigma(x) = 1/(1 + \exp(-x))$  being the logistic sigmoid (which in particular satisfies the conditions of the theorem in Section 13.3), we get [18]

$$\sup_{f \in W^{k,p}(X)} \inf_{g \in \mathcal{M}_r(\sigma)} ||f - g||_{L^p(X)} \ge C \left( r \log r \right)^{-k/d}.$$

Therefore, we cannot expect to circumvent the curse of dimension, at least in the setting considered here.

An interesting approach to obtaining the rate of approximation from the set  $\mathcal{M}_r(\sigma)$  was initiated by Barron [1] and generalized by Makovoz [19]. Barron (and later Makovoz) started with a given  $\mathcal{M}_r(\sigma)$ , with a bounded sigmoidal  $\sigma$ , and tried to find classes of functions which are well approximated by  $\mathcal{M}_r(\sigma)$  with minimal dependence on the dimension d. Without going into details, here, we summarize their results.

**Barron-Makovoz Theorem.** Let  $\mathcal{B}(X)$  be the set of all functions  $f : \mathbb{R}^d \to \mathbb{R}$  defined on the unit ball  $X \subset \mathbb{R}^d$  that can be extended to all of  $\mathbb{R}^d$  such that some shift of f by a constant has a Fourier transform  $\hat{f}$  satisfying  $\int_{\mathbb{R}^d} ||\boldsymbol{\xi}||_2 |\hat{f}(\boldsymbol{\xi})| d\boldsymbol{\xi} \leq \gamma$  for some  $0 < \gamma < \infty$ , where  $||\boldsymbol{\xi}||_2 = (\boldsymbol{\xi} \cdot \boldsymbol{\xi})^{1/2}$ . Then for any bounded sigmoidal function  $\sigma$ ,

$$\sup_{f \in \mathcal{B}(X)} \inf_{g \in \mathcal{M}_r(\sigma)} ||f - g||_{L^2(X)} \le C r^{-(d+1)/2d},$$

where the constant C > 0 is independent of r.

It is to be noted that this result cannot be quite compared to the results of Section 13.3, because, unlike in the case of the Sobolev space  $W^{k,p}$ , the condition defining the function set  $\mathcal{B}$ , known as Barron space, cannot be restated in terms of the number of derivatives. Nevertheless, Barron has shown that on the unit ball X the following embeddings hold:

$$W^{\lfloor d/2 \rfloor + 2,\infty}(X) \subseteq \operatorname{span} \mathcal{B}(X) \subseteq W^{1,\infty}(X).$$

In particular, it tells us that functions f with sufficiently high orders  $k = \lfloor d/2 \rfloor + 2$  of derivatives are in the set  $\mathcal{B}$ . In other words, and roughly speaking, in order to achieve a rate  $\mathcal{O}(r^{-(d+1)/2d})$  that is almost independent of the dimension d when d is large, the function f would need to have many  $(k = \lfloor d/2 \rfloor + 2)$  derivatives. Putting this in the main result

of Section 13.3 we would obtain a comprable rate  $\mathcal{O}(r^{-k/d}) = \mathcal{O}(r^{-(d+4)/2d})$ . This in turn implies that in such smooth function spaces with a large number of derivatives available, then both ReLU networks with one hidden layer and polynomials achieve a rate that minimally depends on the dimension (i.e. -(d+4)/2d which approaches -1/2 in the limit).

**Remark 7.** The original estimate of Barron [1] was (mistakenly) of the form  $C r^{-1/2}$ . This initiated an unfortunate discussion (and misconception) that Barron's results have broken the curse of dimension.

We close this discussion by pointing out that other network architectures, such as deep convolutional neural network-type architectures that are invariant by construction and serve as the state-of-the-art architectures for image/text/graph-related tasks, may perform better in alleviating the curse of dimension when target functions have certain and a priori known hierarchical/geometric properties. Examples include target functions that have hierarchically local compositional structures, target functions that are invariant to transformations (i.e. translations, permutations, rotations), and target functions that possess geometric stability (or near invariance, smoothness) to small deformations. We refer to [20, 24, 3] and the references therein for further details on the subject.

# 14 Complexity of deep ReLU networks

So far we have been discussing the density and convergence rate questions regarding the approximation of continuous and Sobolev target functions by feedforward neural networks with one single hidden layer and continuous or smooth non-polynomial activation functions. In the remainder of this chapter we will consider the more practical case of "deep" feedforward networks with "ReLU" activation functions and obtain an estimate of the complexity of the network (i.e. number of layers, neurons, and non-zero parameters) needed for approximating Sobolev target functions within a desired small tolerance  $\varepsilon \in (0, 1/2)$ . We closely follow [31] and [15] in the sense that we utilize the basic element used in those two references: the "sawthooth" construction developed in [29]. Nevertheless, we will present some new results (that are more general than the results in [31] and [15]) with a few changes in the proofs presented in those two references. Other relevant references will be cited in the text whenever an idea/formula/concept is used.

#### 14.1 Target functions of interest

On the unit hypercube domain

$$X = [0, 1]^d \subset \mathbb{R}^d$$

we consider Sobolev target functions (see Section 13.2)

$$f \in W^{k,p}(X), \qquad k \ge 1, \qquad 1 \le p \le \infty.$$

Unlike in [31, 15], we do not assume  $||f||_{W^{k,p}(X)} \leq 1$  and will derive more general estimates, i.e. estimates that also depend on  $||f||_{W^{k,p}(X)} < \infty$ .

#### 14.2 Main complexity result

**Complexity Theorem.** Let  $\varepsilon \in (0, 1/2)$ . Consider a Sobolev target function  $f \in W^{k,p}(X)$ on  $X := [0, 1]^d$  with  $k \ge 1$  and  $p \in [1, \infty]$ . Then, there exists a standard ReLU neural network  $\Phi_{\varepsilon}$  (see definition in Section 14.8) with d inputs and one output and complexity

$$L(\Phi_{\varepsilon}) \leq C(d,k,p) \log_{2}(\varepsilon^{-1} ||f||_{W^{k-1,p}(X)}),$$
  

$$N(\Phi_{\varepsilon}), P(\Phi_{\varepsilon}) \leq C(d,k,p) \varepsilon^{-d/k} ||f||_{W^{k,p}(X)}^{d/k} \log_{2}^{2}(\varepsilon^{-1} ||f||_{W^{k-1,p}(X)}),$$

such that

$$||f - \Phi_{\varepsilon}||_{L^p(X)} \le \varepsilon.$$

Here, L, N, P denote the number of layers, neurons, and non-zero parameters of the network, respectively, and the constant C = C(d, k, p) > 0 is independent of  $\varepsilon$ .

### 14.3 Proof strategy

The general strategy is to first build a polynomial approximation of f and then approximate the polynomial by a ReLU network, according to the following steps:

- 1. Part 1: build a polynomial approximation of f:
  - Discretize  $X = [0, 1]^d$  into a uniform grid of  $(n + 1)^d$  grid points.
  - Construct a partition of unity as the collection of  $(n + 1)^d$  functions, where each function is supported around a grid point and is the product of d piecewise linear univariate functions (that can be represented by ReLU networks).
  - Around each grid point build a localized polynomial approximation of f by averaged Taylor polynomials (see Section 14.4 for a short introduction to averaged Taylor polynomials).
  - Build a global polynomial approximation of f using the local polynomials and partition of unity constructed in the previous two steps.
- 2. Part 2: Approximate the global polynomial by a ReLU network:
  - Using the sawtooth function approximate  $x^2$  by a ReLU network.
  - Utilizing  $xy = \frac{1}{2}((x+y)^2 x^2 y^2)$ , approximate xy by a ReLU network.
  - Construct a ReLU network with *d* outputs such that the product of its outputs is a function in the partition of unity.
  - Construct a ReLU network such that the product of its outputs expresses each term in the global polynomial.
  - Approximate the products of the outputs of a ReLU network by a ReLU network with one output.
  - Construct the final ReLU network by a linear combination of the one-output networks constructed in the previous step.

- Estimate the error in approximating the global polynomial by the constructed ReLU network.
- 3. Part 3: Finally, select the number n of grid points such that the total error (i.e. the sum of errors made in parts 1 and 2) remains below  $\varepsilon$  and compute the network complexity.

The remainder of this section will be devoted to the details of the proof. Part 1 will be covered in Sections 14.4-14.6. Part 2 will be discussed through Sections 14.7-14.12. Finally, Part 3 will be done in Section 14.13. Some very recent directions of research on the topic with further reading suggestions will be provided in Section 14.14.

### 14.4 Averaged Taylor polynomial approximation

We will review an averaged version of the Taylor polynomial of calculus. Averaged Taylor polynomials are particularly suitable for approximating Sobolev functions. A good reference on the topic is Chapter 4 of [5].

Consider a Sobolev function  $f \in W^{k,p}(X)$ , where  $X \subset \mathbb{R}^d$  is a bounded convex domain. Let

$$B := B(\mathbf{x}_0, r) = \{ \mathbf{x} \in X : ||\mathbf{x} - \mathbf{x}_0||_2 < r \},\$$

be the open ball centered at  $\mathbf{x}_0 \in X$  with radius r > 0 such that  $B \subset X$ . Consider a cut-off function  $\phi \in C_0^{\infty}(\mathbb{R}^d)$  with the properties

$$\phi \ge 0, \qquad \operatorname{supp} \phi = \overline{B}, \qquad \int_{\mathbb{R}^d} \phi(\mathbf{x}) \, d\mathbf{x} = 1$$

The Taylor polynomial of order k of f averaged over B is defined as

$$Q^k f(\mathbf{x}) := \int_B T^k_{\mathbf{y}} f(\mathbf{x}) \,\phi(\mathbf{y}) \, d\mathbf{y}, \qquad \mathbf{x} \in X,$$

where  $T_{\mathbf{y}}^{k} f(\mathbf{x})$  is the familiar Taylor polynomial of order k-1 about  $\mathbf{y}$ :

$$T_{\mathbf{y}}^{k}f(\mathbf{x}) := \sum_{|\boldsymbol{\alpha}| \le k-1} \frac{1}{\boldsymbol{\alpha}!} D^{\boldsymbol{\alpha}} f(\mathbf{y}) \, (\mathbf{x} - \mathbf{y})^{\boldsymbol{\alpha}}.$$

Here, we have used the standard multi-index notation for the multi-index  $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$ . Note that since  $f \in W^{k,p}(X)$  with  $k \ge 1$ , then  $f \in W^{k-1,p}(X)$  and  $D^{\boldsymbol{\alpha}} f$  with  $|\boldsymbol{\alpha}| \le k-1$  is bounded.

**Remark 8.** (Why do we use averaged Taylor polynomials for Sobolev functions?) In general, for a Sobolev function f, the derivatives  $D^{\alpha}f$  that appear in Taylor's formula may not exist in the usual pointwise (or strong) sense. One possibility to define a Taylor polynomial for such a function is by taking a smoothed average of  $T^k_{\mathbf{y}}f(\mathbf{x})$  over  $\mathbf{y} \in B$ .

A few properties of averaged Taylor polynomials (without proofs) follow.

**Linearity property.** From the linearity of weak derivatives, we can conclude that averaged Taylor polynomials are linear in f.

**Polynomial property.** Let  $f \in W^{k,p}(X)$ , where  $X \subset \mathbb{R}^d$  is a bounded convex domain. The Taylor polynomial of order k of f averaged over a ball  $B \subset X$  is a polynomial of degree less than k in **x** that can be written as

$$Q^k f(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \le k-1} c_{\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}, \qquad \mathbf{x} \in X,$$

with

$$|c_{\alpha}| \le c(k,d) \, r^{-d/p} \, ||f||_{W^{k-1,p}(X)}. \tag{19}$$

**Bramble-Hilbert Lemma.** (See Lemma 4.3.8 in [5]) Let  $f \in W^{k,p}(X)$ , where  $X \subset \mathbb{R}^d$  is a bounded convex domain. Let  $Q^k f(\mathbf{x})$  be the Taylor polynomial of order k of f averaged over a ball  $B \subset X$ . Then

$$|f - Q^k f|_{W^{s,p}(X)} \le c(k,d) h^{k-s} |f|_{W^{k,p}(X)}, \qquad h = \operatorname{diam}(X), \qquad s = 0, 1, \dots, k.$$

where  $|f|_{W^{k,p}(X)} := (\sum_{|\alpha|=k} ||D^{\alpha}f||_{L^{p}(X)}^{p})^{1/p}$  is the Sobolev semi-norm.

**Remark 9.** We note that for the Taylor expansion of f on X about any point  $\mathbf{y} \in X$  to give an approximation of f at some points  $\mathbf{x} = \mathbf{y} + \mathbf{h}$ , the whole path  $\mathbf{y} + t\mathbf{h}$ ,  $0 \le t \le 1$ from  $\mathbf{y}$  to  $\mathbf{x}$  must be within X. In case of the averaged Taylor polynomial the expansion point  $\mathbf{y}$  is replaced by a ball B, and hence we need the path (or the line segment) between each  $\mathbf{y} \in B$  and each  $\mathbf{x} \in X$  to be within X. This requires a geometrical condition on X: we say X is "star-shaped" (or star-convex) with respect to B. The original Bramble-Hilbert Lemma (see Lemma 4.3.8 in [5]) assumes X is star-shaped with respect to B, and introduces a parameter known as the "chunkiness" of X that appears in the constant of the estimate, i.e.  $c = c(k, d, \gamma)$ , with  $\gamma = \text{chunkiness}(X)$ . We avoid all these details by simply assuming X is convex, which implies that X is start-shaped and connected. Then the chunkiness will be a bounded constant depending on diam(X), which in turn depend on d for bounded X. For instance, in the particular case  $X = [0, 1]^d$  we have  $\gamma = 2\sqrt{d}$  which is the ratio between diam(X) =  $\sqrt{d}$  and the radius r = 1/2 of the ball inscribed in X.

#### 14.5 Partition of unity

There are different ways to construct a partition of unity. Here, we will construct a partition of unity as the product of piecewise linear functions. The main reason for a piecewise linear construction is that such construction can be realized by a ReLU network.

We first define a function  $\psi : \mathbb{R} \to \mathbb{R}$  as

$$\psi(z) = \begin{cases} 1 & |z| < 1, \\ 0 & |z| > 2, \\ 2 - |z| & 1 \le |z| \le 2. \end{cases}$$

Discretize  $X = [0,1]^d$  into a uniform grid of  $(n+1)^d$  grid points, where  $n \ge 1$ , and let  $\mathbf{m} = (m_1, \ldots, m_d) \in \{0, 1, \ldots, n\}^d$  be a *d*-tuple or multi-index corresponding to each grid point. For instance  $\mathbf{m} = (0, \ldots, 0)$  and  $\mathbf{m} = (n, \ldots, n)$  correspond to the very first and the

very last grid points, respectively. Clearly, there are  $(n+1)^d$  multi-indices **m** corresponding to  $(n+1)^d$  grid points. To each multi-index **m** we assign a function

$$\phi_{\mathbf{m}}(\mathbf{x}) = \prod_{\ell=1}^{d} \psi(3 n \left(x_{\ell} - \frac{m_{\ell}}{n}\right)), \qquad \mathbf{x} \in X, \qquad \mathbf{m} \in \{0, 1, \dots, n\}^{d},$$

which is obtained by the product of d piecewise linear univariate functions of the form  $\psi$ . The collection of all such functions for all  $(n + 1)^d$  multi-indices forms a partition of unity, denoted by

$$U := \{ \phi_{\mathbf{m}} : \, \mathbf{m} \in \{0, 1, \dots, n\}^d \}.$$

Figure 8 (left) shows the function  $\psi$ , and Figure 8 (right) shows the partition of unity in the case d = 1 and n = 5, i.e. a collection of n + 1 = 6 functions  $\phi_0, \phi_1, \ldots, \phi_5$ .



Figure 8: Left: the function  $\psi$ . Right: the partition of unity (a collection of n + 1 = 6 functions) for the case d = 1 and n = 5.

The partition of unity U has the following properties:

1.  $0 \le \phi_{\mathbf{m}}(\mathbf{x}) \le 1, \quad \forall \phi_{\mathbf{m}} \in U, \quad \forall \mathbf{x} \in \mathbb{R}^d.$ 

2. 
$$\sum_{\phi_{\mathbf{m}} \in U} \phi_{\mathbf{m}}(\mathbf{x}) = 1, \quad \forall \mathbf{x} \in X.$$

3.  $\operatorname{supp}(\phi_{\mathbf{m}}) \subset \{\mathbf{x} : |x_{\ell} - \frac{m_{\ell}}{n}| < \frac{1}{n}, \ell = 1, \dots, d\} =: B_{\mathbf{m}}^{\infty}, \quad \forall \phi_{\mathbf{m}} \in U.$ 

4. 
$$||\phi_{\mathbf{m}}||_{W^{1,\infty}(\mathbb{R}^d)} \le 4 n.$$

The first three properties follow easily from the definition of U. To show the fourth property, we note that  $||\phi_{\mathbf{m}}||_{W^{1,\infty}(\mathbb{R}^d)} = ||\phi_{\mathbf{m}}||_{L^{\infty}(\mathbb{R}^d)} + ||D\phi_{\mathbf{m}}||_{L^{\infty}(\mathbb{R}^d)}$ . The first term is bounded by one (thanks to property no. 1). It is also straightforward to show that  $\partial \phi_{\mathbf{m}}/\partial x_{\ell} \leq 3n$ , by directly computing the derivative of  $\phi_{\mathbf{m}}$  with respect to each coordinate  $x_{\ell}$ . Hence, the second term is bounded by 3n. We therefore obtain  $||\phi_{\mathbf{m}}||_{W^{1,\infty}(\mathbb{R}^d)} \leq 1+3n \leq 4n$ . **Remark 10.** (On the boundary points) For the points on the boundary of X and to construct averaged Taylor polynomials near the boundary of X one possibility is to use an extension operator and extend the function f slightly outside the domain X. This would introduce tedious technicalities in the exposition of formulas and proofs, and hence we avoid it here. Instead, we assume that  $\phi_{\mathbf{m}}$  for  $\mathbf{m}$  on the boundary vanishes outside X; see e.g.  $\phi_0$  and  $\phi_5$  in Figure 8. Similarly, we assume that the definition of  $B_{\mathbf{m}}^{\infty}$  is modified by  $B_{\mathbf{m}}^{\infty} \cap X$  for any  $\mathbf{m}$  on the boundary. With such modifications, we will have  $\cup_{\mathbf{m}} B_{\mathbf{m}}^{\infty} = X$ . The same modification is assumed to be applied to all balls near the boundary over which we compute averaged Taylor polynomials.

#### 14.6 Construction of a global polynomial approximant

Now, we will use averaged Taylor polynomials to build a local approximation (and obtain local estimates) and then combine them using the partition of unity to build a global approximation (and obtain a global estimate).

**Local approximations.** For each multi-index  $\mathbf{m} \in \{0, 1, ..., n\}^d$ , corresponding to a grid point, consider the ball

$$B_{\mathbf{m}} := B(\frac{\mathbf{m}}{n}, r = \frac{3}{4n}) \subset B_{\mathbf{m}}^{\infty},$$

where  $B_{\mathbf{m}}^{\infty}$  is defined in property no. 3 in Section 14.5 and contains the support of  $\phi_{\mathbf{m}}$ . We note that  $B_{\mathbf{m}}^{\infty}$  is bounded and convex (and in particular star-shaped with respect to the ball  $B_{\mathbf{m}}$ ). Hence, we can use the Bramble-Hilbert Lemma with  $h = \operatorname{diam}(B_{\mathbf{m}}^{\infty}) = 2\sqrt{d}/n$ . Specifically, we let  $p_{\mathbf{m}}$  be the Taylor polynomial of order k of f averaged over  $B_{\mathbf{m}}$  that (thanks to the polynomial property of averaged Taylor polynomials in Section 14.4) can be written as

$$p_{\mathbf{m}}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \le k-1} c_{\mathbf{m},\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}, \qquad \mathbf{x} \in B^{\infty}_{\mathbf{m}},$$

where by (19) and noting  $r = \frac{3}{4n}$ ,

$$|c_{\mathbf{m}, \alpha}| \le c(k, d) n^{d/p} ||f||_{W^{k-1, p}(B_{\mathbf{m}}^{\infty})}.$$

Note that this is a local approximation of f, as  $p_{\mathbf{m}}(\mathbf{x})$  is the approximating polynomial defined on  $B_{\mathbf{m}}^{\infty}$ , not on the whole domain X. By the Bramble-Hilbert Lemma with  $h = \text{diam}(B_{\mathbf{m}}^{\infty}) = 2\sqrt{d/n}$ , we get the local estimate (with s = 0),

$$||f - p_{\mathbf{m}}||_{L^{p}(B_{\mathbf{m}}^{\infty})} \le c(k,d) \left(\frac{1}{n}\right)^{k} |f|_{W^{k,p}(B_{\mathbf{m}}^{\infty})} \le c(k,d) \left(\frac{1}{n}\right)^{k} ||f||_{W^{k,p}(B_{\mathbf{m}}^{\infty})}$$

A global approximation. We next combine the partition of unity with all local polynomials to build a global polynomial approximation of f on X. Specifically, we define the global approximating polynomial

$$f_n(\mathbf{x}) := \sum_{\mathbf{m}} \phi_{\mathbf{m}}(\mathbf{x}) p_{\mathbf{m}}(\mathbf{x}), \qquad \mathbf{x} \in X,$$

where the sum is taken over all  $(n+1)^d$  multi-indices  $\mathbf{m} \in \{0, 1, \ldots, n\}^d$  and  $\{\phi_{\mathbf{m}}\}$  forms the partition of unity defined in Section 14.5. We can now write

$$||f - f_n||_{L^p(X)}^p = ||\sum_{\mathbf{m}} \phi_{\mathbf{m}} (f - p_{\mathbf{m}})||_{L^p(X)}^p \le \sum_{\tilde{\mathbf{m}}} ||\sum_{\mathbf{m}} \phi_{\mathbf{m}} (f - p_{\mathbf{m}})||_{L^p(B_{\tilde{\mathbf{m}}}^{\infty})}^p,$$
(20)

where the last inequality follows from the integral-definition of  $L^p$ -norm and that  $X = \bigcup_{\tilde{\mathbf{m}}} B^{\infty}_{\tilde{\mathbf{m}}}$ .

For each multi-index  $\tilde{\mathbf{m}}$  we have

$$\begin{split} ||\sum_{\mathbf{m}} \phi_{\mathbf{m}} \left(f - p_{\mathbf{m}}\right)||_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})} &\leq \sum_{\mathbf{m}: \, ||\mathbf{m} - \tilde{\mathbf{m}}||_{\infty} \leq 1} ||\phi_{\mathbf{m}} \left(f - p_{\mathbf{m}}\right)||_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})} \\ &\leq \sum_{\mathbf{m}: \, ||\mathbf{m} - \tilde{\mathbf{m}}||_{\infty} \leq 1} ||\phi_{\mathbf{m}} \left(f - p_{\mathbf{m}}\right)||_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})}, \end{split}$$

where the first inequality is a result of the triangle inequality and noting that  $\operatorname{supp}(\phi_{\mathbf{m}}) \subset B_{\mathbf{m}}^{\infty}$ , implying that we only need to consider multi-indices  $\mathbf{m}$  that are adjacent to  $\tilde{\mathbf{m}}$ , because only  $\phi_{\mathbf{m}}$  corresponding to such multi-indices take values on  $B_{\tilde{\mathbf{m}}}^{\infty}$  and hence contribute to the norm over  $B_{\tilde{\mathbf{m}}}^{\infty}$ . The second inequality is again a consequence of the support of  $\phi_{\mathbf{m}}$ , that is,

$$\operatorname{supp}(g) \subset B^{\infty}_{\mathbf{m}} \quad \Rightarrow \quad ||g||_{L^{p}(B^{\infty}_{\mathbf{m}})} = ||g||_{L^{p}(B^{\infty}_{\mathbf{m}} \cap B^{\infty}_{\mathbf{m}})} \leq ||g||_{L^{p}(B^{\infty}_{\mathbf{m}})}$$

Each term in the sum in the last inequality above can further be bounded by,

$$||\phi_{\mathbf{m}}(f - p_{\mathbf{m}})||_{L^{p}(B_{\mathbf{m}}^{\infty})} \leq ||\phi_{\mathbf{m}}||_{L^{\infty}(B_{\mathbf{m}}^{\infty})} ||(f - p_{\mathbf{m}})||_{L^{p}(B_{\mathbf{m}}^{\infty})} \leq c(k, d) \left(\frac{1}{n}\right)^{k} ||f||_{W^{k, p}(B_{\mathbf{m}}^{\infty})},$$

where we used the local estimates obtained above and the fact that  $\phi_{\mathbf{m}} \leq 1$  everywhere. Combining the last two inequalities we get for each multi-index  $\tilde{\mathbf{m}}$ ,

$$\|\sum_{\mathbf{m}} \phi_{\mathbf{m}} (f - p_{\mathbf{m}})\|_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})} \le c(k, d) \left(\frac{1}{n}\right)^{k} \sum_{\mathbf{m}: \|\mathbf{m} - \tilde{\mathbf{m}}\|_{\infty} \le 1} \|f\|_{W^{k, p}(B_{\mathbf{m}}^{\infty})}.$$
 (21)

Now, by (20) and (21), we get

$$||f - f_n||_{L^p(X)}^p \le c^p(k, d) \left(\frac{1}{n}\right)^{kp} \sum_{\tilde{\mathbf{m}}} \left(\sum_{\mathbf{m}: \, ||\mathbf{m} - \tilde{\mathbf{m}}||_{\infty} \le 1} ||f||_{W^{k, p}(B_{\mathbf{m}}^{\infty})}\right)^p.$$
(22)

Now, let us recall Hölder's inequality

$$\sum_{\mathbf{m}} |g_{\mathbf{m}} h_{\mathbf{m}}| \le \left(\sum_{\mathbf{m}} |g_{\mathbf{m}}|^p\right)^{1/p} \left(\sum_{\mathbf{m}} |h_{\mathbf{m}}|^q\right)^{1/q}, \qquad g_{\mathbf{m}}, h_{\mathbf{m}} \in \mathbb{R}, \qquad p \in (1, \infty), \qquad \frac{1}{p} + \frac{1}{q} = 1.$$

Setting  $g_{\mathbf{m}} := ||f||_{W^{k,p}(B_{\mathbf{m}}^{\infty})}$  and  $h_{\mathbf{m}} := 1$ , and raising both sides of Hölder's inequality to power p, we obtain

$$\left(\sum_{\mathbf{m}: ||\mathbf{m} - \tilde{\mathbf{m}}||_{\infty} \le 1} ||f||_{W^{k,p}(B_{\mathbf{m}}^{\infty})}\right)^{p} \le \sum_{\mathbf{m}: ||\mathbf{m} - \tilde{\mathbf{m}}||_{\infty} \le 1} ||f||_{W^{k,p}(B_{\mathbf{m}}^{\infty})}^{p} (3^{d})^{p/q},$$
(23)

where  $3^d = \sum_{\mathbf{m}: ||\mathbf{m} - \tilde{\mathbf{m}}||_{\infty} \leq 1} 1$  is the total number of indices  $\mathbf{m}$  that are either adjacent or identical to  $\tilde{\mathbf{m}}$ . By the definition of  $B_{\mathbf{m}}$ , it is also not difficult to see that (show this as an exercise),

$$\sum_{\tilde{\mathbf{m}}} \sum_{\mathbf{m}: ||\mathbf{m} - \tilde{\mathbf{m}}||_{\infty} \le 1} ||f||_{W^{k,p}(B_{\tilde{\mathbf{m}}}^{\infty})}^{p} \le 3^{d} \sum_{\tilde{\mathbf{m}}} ||f||_{W^{k,p}(B_{\tilde{\mathbf{m}}}^{\infty})}^{p}.$$
(24)

Moreover, by the definition of  $B_{\mathbf{m}}$  and the integral-definition of  $W^{k,p}$ -norm, one gets (show this as an exercise),

$$\sum_{\tilde{\mathbf{m}}} ||f||_{W^{k,p}(B_{\tilde{\mathbf{m}}}^{\infty})}^{p} \le 2^{d} ||f||_{W^{k,p}(\cup_{\tilde{\mathbf{m}}} B_{\tilde{\mathbf{m}}}^{\infty})}^{p} = 2^{d} ||f||_{W^{k,p}(X)}^{p}.$$
(25)

Combining (22)-(25), we finally obtain the global estimate

$$||f - f_n||_{L^p(X)} \le c(k, d, p) \left(\frac{1}{n}\right)^k ||f||_{W^{k, p}(X)}.$$

We summarize the results of Sections 14.4-14.6 in the following proposition.

**Proposition 1.** Let  $U = \{\phi_{\mathbf{m}} : \mathbf{m} \in \{0, 1, ..., n\}^d\}$ , with  $supp(\phi_{\mathbf{m}}) \subset B_{\mathbf{m}}^{\infty}$  and  $\cup_{\mathbf{m}} B_{\mathbf{m}}^{\infty} = X$ , be the partition of unity over  $X = [0, 1]^d$  built in Section 14.5. Then, for every  $f \in W^{k, p}(X)$ , with  $k \ge 1$  and  $p \in [1, \infty]$ , there exists a polynomial

$$f_n(\mathbf{x}) := \sum_{\mathbf{m}} \phi_{\mathbf{m}}(\mathbf{x}) p_{\mathbf{m}}(\mathbf{x}), \qquad \mathbf{x} \in X,$$

with

$$p_{\mathbf{m}}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \le k-1} c_{\mathbf{m},\boldsymbol{\alpha}} \, \mathbf{x}^{\boldsymbol{\alpha}}, \qquad \mathbf{x} \in B_{\mathbf{m}}^{\infty}, \qquad |c_{\mathbf{m},\boldsymbol{\alpha}}| \le c(k,d) \, n^{d/p} \, ||f||_{W^{k-1,p}(B_{\mathbf{m}}^{\infty})},$$

that satisfies

$$||f - f_n||_{L^p(X)} \le c(k, d, p) \left(\frac{1}{n}\right)^k ||f||_{W^{k, p}(X)}.$$
(26)

# 14.7 Expressive power of ReLU networks: an intuitive example

We consider the quadratic function on the interval [0, 1], displayed in Figure 9,

$$g(x) = x^2, \qquad x \in [0, 1]$$

Our goal is to construct a ReLU network approximant, say  $\tilde{g}$ , of g with the error

$$||g - \tilde{g}||_{L^{\infty}([0,1])} \le \varepsilon, \tag{27}$$

with minimal complexity, i.e. with the number of layers, neurons, and parameters as small as possible. Indeed, we will show that this can be achieved with complexity  $L, N, P = \mathcal{O}(\varepsilon^{-1})$ . Our construction will use the "sawtooth" function that (first) appeared in [29].



Figure 9: The quadratic function  $g(x) = x^2$  on the interval [0, 1] to be approximated by a ReLU network.



Figure 10: The hat function h(x) on the interval [0, 1].

Sawtooth function made by the composition of hat functions. Consider the hat function (also known as tent or triangle function), displayed in Figure 10,

$$h(x) = \begin{cases} 2x, & 0 \le x < 1/2, \\ 2(1-x), & 1/2 \le x \le 1, \\ 0, & \text{elsewhere.} \end{cases}$$

Now let

$$h_0(x) = x, \qquad h_1(x) = h(x),$$

and denote by  $h_s(x)$ , with  $s \ge 2$ , the s-fold composition of h with itself

$$h_s(x) = \underbrace{h \circ h \circ \ldots \circ h}_{s \text{ times}}(x), \qquad s \ge 2.$$

Telgarsky [29] has shown that  $h_s$  is a sawtooth function with  $2^{s-1}$  evenly distributed teeth (or tents), where each application of h doubles the number of teeth; see Figure 11.



Figure 11: The sawtooth function  $h_s(x)$  on the interval [0, 1] with s = 1, 2, 3.

We make two observations and will discuss each in turn:

- $g(x) = x^2$  can be approximated by linear combinations of  $h_0, h_1, h_2, \ldots$
- $h_0, h_1, h_2, \ldots$  and hence their linear combinations can be expressed by a ReLU network.

**Observation 1.** We start with approximating  $g(x) = x^2$  by a piecewise-linear interpolation  $g_m$  of g on a uniform grid of  $2^m + 1$  evenly distributed grid points  $\{\frac{i}{2^m}\}_{i=0}^{2^m}$ ; see Figure 12. Note that by definition of interpolation, we have

$$g_m(x) = x^2, \qquad x = \frac{i}{2^m}, \qquad i = 0, 1, \dots, 2^m.$$

It is not difficult to see that refining the interpolation from  $g_{m-1}$  to  $g_m$  amounts to adjusting it by a function proportional to a sawtooth function,

$$g_{m-1}(x) - g_m(x) = \frac{h_m(x)}{2^{2m}}.$$



Figure 12: Piecewise-linear approximation  $g_m(x)$  of  $g(x) = x^2$  at  $2^m + 1$  evenly distributed grid points on the interval [0, 1], with m = 0, 1, 2.

We hence obtain

$$g_m(x) = x - \sum_{s=1}^m \frac{h_s(x)}{2^{2s}}.$$
(28)

That is, the function  $g(x) = x^2$  can be approximated by linear combinations of  $h_0, h_1, h_2, \ldots$ 

It is to be noted that the construction of  $g_m$  only involves  $\mathcal{O}(m)$  linear operations and compositions of h. This shows the "super power" of composition! For instance, to build  $h_m$  (and hence  $g_m$ ) without composition, we would need  $\mathcal{O}(2^m)$  operations. However, using composition, we would only need  $\mathcal{O}(m)$  operations, a huge reduction from an exponential complexity down to a linear complexity. As we will see later, this is a major reason why deep learning (which involves multiple applications of composition) has been so successful in solving science and engineering problems.

We can also derive a clear formula for the approximation error  $||g(x) - g_m(x)||_{L^{\infty}([0,1])}$ . For this purpose, we will carry out a simple numerical experiment, and show how such numerical experiments could help derive theoretical results. To this end, we will compute the error  $|g(x) - g_m(x)|$  versus  $x \in [0, 1]$  for the first three approximations  $g_m$  with m = 0, 1, 2. The result is depicted in Figure 13. Obviously, the initial maximum error is

$$\max_{x \in [0,1]} |g(x) - g_0(x)| = \max_{x \in [0,1]} |x^2 - x| = 1/4.$$

This can be easily shown by equating the derivative of  $x^2 - x$  to zero (i.e. 2x - 1 = 0), which tells us that the maximum is at x = 1/2 and is equal to  $|(1/2)^2 - (1/2)| = 1/4$ . Figure 13 also tells us that with increasing m by one, the maximum error gets multiplied by 1/4. Overall,

the figure suggests that the maximum error is  $2^{-2(m+1)}$ , that is,

$$||g(x) - g_m(x)||_{L^{\infty}([0,1])} = 2^{-2(m+1)}, \qquad m = 0, 1, 2, \dots$$
 (29)

This is indeed an example of the power of numerical experiments in deriving theoretical



Figure 13: The error  $|g(x) - g_m(x)|$  in the approximation of  $g(x) = x^2$  by a piecewise linear interpolant  $g_m$  at  $2^m + 1$  evenly distributed grid points on the interval [0, 1], with m = 0, 1, 2. This suggests that with increasing m, the maximum error decays as  $2^{-2(m+1)}$ .

results. Interestingly, the figure also tells us how to theoretically prove (29). For every  $m = 0, 1, 2, \ldots$ , the error  $|g(x) - g_m(x)|$  is identical on each subinterval between two consecutive points of interpolation. In particular, we will have

$$\max_{x \in [0,1]} |g(x) - g_m(x)| = \max_{x \in [0,1/2^m]} |g(x) - g_m(x)|.$$

It is now straightforward to show (29) using the right hand side of the above formula.

**Observation 2.** We start the second observation with noting that the hat function h = h(x) can be realized (or expressed) by a ReLU network. To derive such a network, we first note that h can be written as a linear combination of three ReLU functions (see Figure 14):

$$h(x) = 2\sigma(x) - 4\sigma(x - 1/2) + 2\sigma(x - 1).$$

One can then easily see that h(x) is given by a ReLU network with L = 2 layers,

$$\Phi_h(x) := A_2 \circ \sigma \circ A_1(x),$$



Figure 14: The three ReLU functions (left) that combine to form the har function (right).

where

$$A_{1}(x) = \begin{pmatrix} 1\\ 1\\ 1 \end{pmatrix} x + \begin{pmatrix} 0\\ -1/2\\ -1 \end{pmatrix}, \quad A_{2}(\mathbf{z}) = (2 - 4 \ 2) \begin{pmatrix} z_{1}\\ z_{2}\\ z_{3} \end{pmatrix}$$

Figure 15 shows a graph representation of  $\Phi_h(x) = h(x)$ , with L = 2 layers, N = 5 neurons, and P = 10 parameters (i.e. weights and biases).



Figure 15: The two-layer ReLU network that realizes  $h(x) = 2\sigma(x) - 4\sigma(x-1/2) + 2\sigma(x-1)$ .

It will now be straightforward to construct a ReLU network with L = s + 1 layers that realizes  $h_s(x)$ , for any  $s \ge 2$ . One such network is given by

$$\Phi_{h_s}(x) := A_{s+1} \circ \sigma \circ A_s \circ \sigma \circ \ldots \circ A_2 \circ \sigma \circ A_1(x),$$

where

$$A_{1}(x) = \begin{pmatrix} 1\\ 1\\ 1 \end{pmatrix} x + \begin{pmatrix} 0\\ -1/2\\ -1 \end{pmatrix}, \quad A_{s+1}(\mathbf{z}) = (2 - 4 2) \begin{pmatrix} z_{1}\\ z_{2}\\ z_{3} \end{pmatrix},$$

and

$$A_{\ell}(x) = \begin{pmatrix} 2 & -4 & 2 \\ 2 & -4 & 2 \\ 2 & -4 & 2 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} + \begin{pmatrix} 0 \\ -1/2 \\ -1 \end{pmatrix}, \qquad \ell = 2, \dots, s$$

The complexity of such network is

$$L(\Phi_{h_s}) = s + 1,$$
  $N(\Phi_{h_s}) = 3s + 2,$   $P(\Phi_{h_s}) = 12s - 2.$ 

Figure 16 shows the graph representation of  $\Phi_{h_2}(x) = h_2(x)$ , with L = 3 layers, N = 8 neurons, and P = 22 parameters.



Figure 16: The three-layer ReLU network that realizes  $h_2(x)$ .

Finally, we can construct a ReLU network with L = m + 1 layers that realizes  $g_m(x)$ , for any  $m \ge 1$ . One such network is given by

$$\Phi_{g_m}(x) := A_{m+1} \circ \sigma \circ A_m \circ \sigma \circ \ldots \circ A_2 \circ \sigma \circ A_1(x),$$

where

$$A_1(x) = \begin{pmatrix} 1\\1\\1\\1 \end{pmatrix} x + \begin{pmatrix} 0\\-1/2\\-1\\0 \end{pmatrix}, \quad A_{m+1}(\mathbf{z}) = (-1/2 \ 1 \ -1/2 \ 1) \begin{pmatrix} z_1\\z_2\\z_3\\z_4 \end{pmatrix},$$

and

$$A_{\ell}(x) = \begin{pmatrix} 1/2 & -1 & 1/2 & 0\\ 1/2 & -1 & 1/2 & 0\\ 1/2 & -1 & 1/2 & 0\\ -1/2 & 1 & -1/2 & 1 \end{pmatrix} \begin{pmatrix} z_1\\ z_2\\ z_3\\ z_4 \end{pmatrix} + \begin{pmatrix} 0\\ -2^{-2\ell+1}\\ -2^{-2\ell+2}\\ 0 \end{pmatrix}, \qquad \ell = 2, \dots, m.$$

The complexity of such network is

$$L(\Phi_{g_m}) = m + 1,$$
  $N(\Phi_{g_m}) = 4m + 2,$   $P(\Phi_{g_m}) = 20m - 7.$ 

Figure 17 and Figure 18 show the graph representations of  $\Phi_{g_1}(x) = x - \frac{1}{4}h_1(x)$  and  $\Phi_{g_2}(x) = x - \frac{1}{4}h_1(x) - \frac{1}{16}h_2(x)$ , respectively.

We can now state the final result on the approximation of the quadratic function by ReLU networks.



Figure 17: The two-layer ReLU network that realizes  $g_1(x) = x - \frac{1}{4}h_1(x)$ .



Figure 18: The three-layer ReLU network that realizes  $g_2(x) = x - \frac{1}{4}h_1(x) - \frac{1}{16}h_2(x)$ .

**Lemma 2.** The function  $g(x) = x^2$  on [0,1] can be approximated within any small error  $\varepsilon \in (0,1/2)$  in the sense (27) by a ReLU network with complexity  $L, M, P = \mathcal{O}(\log_2 \varepsilon^{-1})$ .

*Proof.* The proof follows directly by choosing  $m = \lfloor \frac{1}{2} \log_2 \varepsilon^{-1} \rfloor$ .

**Remark 11.** We have discussed how the power of composition in neural networks makes them efficient approximants in the sense that their complexity increases logarithmically with the reciprocal of the approximation error. This is a very desirable approximation property of neural networks. Note that the input dimension is one here, and hence this result does not address the effect of composition on dimension.

#### 14.8 Connected and standard ReLU networks

We closely follow the setup used in [22, 15] and define two classes of feedforward ReLU networks:

- 1) connected networks, where connection is allowed between all layers;
- 2) standard networks, where connection is allowed only between neighboring layers.

The latter, that we have been considering so far, is a particular case of the former. We will also present an important result that states if a function can be realized by a connected network, it can also be realized by a standard network with comparable complexity. In this sense, any accuracy-complexity result obtained for connected networks will also hold for standard networks with minor modifications.

**Connected networks.** A connected network with  $N_0 = d$  inputs and L layeres, each with  $N_{\ell}$  neurons,  $\ell = 1, \ldots, L$ , is given by a sequence of matrix-vector tuples (or weight-bias tuples)

$$\Phi := \{ (W_1, b_1), \dots, (W_L, b_L) \}, \qquad W_\ell \in \mathbb{R}^{N_\ell \times \sum_{i=0}^{\ell-1} N_i}, \qquad b_\ell \in \mathbb{R}^{N_\ell}.$$

The weight matrices  $W_{\ell}$  may be written in block matrix form

$$W_{\ell} = \left(W_{\ell,0} \mid W_{\ell,1} \mid \ldots \mid W_{\ell,\ell-1}\right) \in \mathbb{R}^{N_{\ell} \times \sum_{i=0}^{\ell-1} N_i}, \qquad W_{\ell,i} \in \mathbb{R}^{N_{\ell} \times N_i}$$

With a slight abuse of notation, we also use  $\Phi$  to denote the function that the network realizes, and write

$$\Phi(\mathbf{x}) = \mathbf{x}_L, \qquad \Phi : \mathbb{R}^d \to \mathbb{R}^{N_L},$$

where  $\mathbf{x}_L$  results from the following scheme:

$$\mathbf{x}_0 = \mathbf{x},$$
  

$$\mathbf{x}_{\ell} = \sigma(W_{\ell,0}\mathbf{x}_0 + \ldots + W_{\ell,\ell-1}\mathbf{x}_{\ell-1} + b_{\ell}), \qquad \ell = 1, \ldots, L-1,$$
  

$$\mathbf{x}_L = W_{L,0}\mathbf{x}_0 + \ldots + W_{L,L-1}\mathbf{x}_{L-1} + b_L.$$

The distinction between  $\Phi$  being a function or a set of matrix-vector tuples should be easily made from the context.

As before, we consider ReLU activation function  $\sigma : \mathbb{R} \to \mathbb{R}$  that acts componentwise on all hidden layers. We do not apply any activation function on the output layer. The complexity of a connected network  $\Phi$ , represented by the number of layers  $L(\Phi)$ , number of neurons  $N(\Phi)$ , and number of non-zero parameters  $P(\Phi)$ , is given by

$$L(\Phi) = L, \qquad N(\Phi) = \sum_{\ell=0}^{L} N_{\ell}, \qquad P(\Phi) = \sum_{\ell=1}^{L} (||W_{\ell}||_{0} + ||b_{\ell}||_{0}), \qquad ||A||_{0} := \operatorname{nnz}(A).$$

**Standard networks**. A standard network with  $N_0 = d$  inputs and L layers, each with  $N_\ell$  neurons,  $\ell = 1, \ldots, L$ , is given by a sequence of matrix-vector tuples (or weight-bias tuples)

$$\Phi := \{ (W_1, b_1), \dots, (W_L, b_L) \}, \qquad W_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}, \qquad b_\ell \in \mathbb{R}^{N_\ell}$$

We also denote by  $\Phi$  the function that the network realizes, and write

$$\Phi(\mathbf{x}) = \mathbf{x}_L, \qquad \Phi : \mathbb{R}^d \to \mathbb{R}^{N_L},$$

where  $\mathbf{x}_L$  results from the following scheme:

$$\mathbf{x}_0 = \mathbf{x},$$
  

$$\mathbf{x}_\ell = \sigma(W_\ell \, \mathbf{x}_{\ell-1} + b_\ell), \qquad \ell = 1, \dots, L-1,$$
  

$$\mathbf{x}_L = W_L \, \mathbf{x}_{L-1} + b_L.$$

It is to be noted that a standard network is a particular type of connected network whose weight matrices are in the block form

$$W_{\ell} = \left( \mathbf{0} \mid \ldots \mid \mathbf{0} \mid W_{\ell,\ell-1} \right) \in \mathbb{R}^{N_{\ell} \times \sum_{i=0}^{\ell-1} N_i}, \qquad W_{\ell,\ell-1} \in \mathbb{R}^{N_{\ell} \times N_{\ell-1}}$$

**Identity ReLU network.** Let  $I_d \in \mathbb{R}^{d \times d}$  be the identity matrix, i.e.  $I_d \mathbf{x} = \mathbf{x}$ . Consider the two-layer standard ReLU network

$$\Phi_I := \{ (W_1, b_1), (W_2, b_2) \},\$$

with

$$W_1 = \begin{pmatrix} I_d \\ -I_d \end{pmatrix} \in \mathbb{R}^{2d \times d}, \qquad b_1 = \mathbf{0} \in \mathbb{R}^{2d}, \qquad W_2 = (I_d \mid -I_d) \in \mathbb{R}^{d \times 2d}, \qquad b_2 = \mathbf{0} \in \mathbb{R}^d.$$

Then, using the identity relation  $\sigma(x) - \sigma(-x) = x$ , it can easily be shown that

$$\Phi_I(\mathbf{x}) = I_d \, \mathbf{x} = \mathbf{x}.$$

We hence call  $\Phi_I$  the identity ReLU network.

Composition of networks. Consider two (connected) networks

$$\Phi_1 = \{ (W_1^1, b_1^1), \dots, (W_{L_1}^1, b_{L_1}^1) \}, \qquad \Phi_2 = \{ (W_1^2, b_1^2), \dots, (W_{L_2}^2, b_{L_2}^2) \},\$$

where the input dimension of  $\Phi_1$  is equal to the output dimension of  $\Phi_2$ , and  $\Phi_2$  takes  $\mathbf{x} \in \mathbb{R}^d$  as input. Then, an efficient way of constructing a (connected) network that gives the composition  $\Phi_1 \circ \Phi_2$  is by utilizing the identity network as  $\Phi_1 \circ \Phi_I \circ \Phi_2$ . Such a composition network is given by

$$\Phi_1 \odot \Phi_2 := \left\{ (W_1^2, b_1^2), \dots, (W_{L_2-1}^2, b_{L_2-1}^2), \left( \begin{pmatrix} W_{L_2}^2 \\ -W_{L_2}^2 \end{pmatrix}, \begin{pmatrix} b_{L_2}^2 \\ -b_{L_2}^2 \end{pmatrix} \right), (\hat{W}_1^1, b_1^1), \dots, (\hat{W}_{L_1}^1, b_{L_1}^1) \right\},$$

where

$$\hat{W}_{\ell}^{1} = \left(\mathbf{0} \mid W_{\ell,0}^{1} \mid -W_{\ell,0}^{1} \mid W_{\ell,1}^{1} \mid \dots \mid W_{\ell,\ell-1}^{1}\right), \qquad \ell = 1,\dots,L_{1}$$

We have

$$\Phi_1 \odot \Phi_2(\mathbf{x}) = \Phi_1 \circ \Phi_2(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^d.$$

Furthermore, we have

$$L(\Phi_1 \odot \Phi_2) = L_1 + L_2, \quad N(\Phi_1 \odot \Phi_2) \le 2 N(\Phi_1) + 2 N(\Phi_2), \quad P(\Phi_1 \odot \Phi_2) \le 2 P(\Phi_1) + 2 P(\Phi_2).$$

Combination (or concatenation) of networks. Consider two (connected) networks

$$\Phi_1 = \{ (W_1^1, b_1^1), \dots, (W_{L_1}^1, b_{L_1}^1) \}, \qquad \Phi_2 = \{ (W_1^2, b_1^2), \dots, (W_{L_2}^2, b_{L_2}^2) \},$$

which take the same input  $\mathbf{x} \in \mathbb{R}^d$ . We want to construct a network  $P(\Phi_1, \Phi_2)$  such that it takes the same input and gives the outputs of the two networks, that is,

$$\Phi_1 \parallel \Phi_2(\mathbf{x}) = (\Phi_1(\mathbf{x}), \Phi_2(\mathbf{x})), \qquad \mathbf{x} \in \mathbb{R}^d.$$

Assuming  $L_1 \leq L_2$ , such a combination network is given by

$$\Phi_1 \parallel \Phi_2 := \{ (\tilde{W}_1, \tilde{b}_1), \dots, (\tilde{W}_{L_2}, \tilde{b}_{L_2}) \},\$$

where

$$\begin{split} \tilde{W}_{\ell} &= \left( \begin{pmatrix} W_{\ell,0}^{1} \\ W_{\ell,0}^{2} \end{pmatrix} \middle| \begin{pmatrix} W_{\ell,1}^{1} & \mathbf{0} \\ \mathbf{0} & W_{\ell,1}^{2} \end{pmatrix} \middle| \dots \middle| \begin{pmatrix} W_{\ell,\ell-1}^{1} & \mathbf{0} \\ \mathbf{0} & W_{\ell,\ell-1}^{2} \end{pmatrix} \right) \rangle, \quad \tilde{b}_{\ell} = \begin{pmatrix} b_{\ell}^{1} \\ b_{\ell}^{2} \end{pmatrix}, \quad 1 \leq \ell < L_{1}, \\ \tilde{W}_{\ell} &= \begin{pmatrix} W_{\ell,0}^{2} \mid (\mathbf{0} \ W_{\ell,1}^{2}) \mid \dots \mid (\mathbf{0} \ W_{\ell,L_{1}-1}^{2}) \mid W_{\ell,L_{1}}^{2} \mid \dots \mid W_{\ell,\ell-1}^{2} \end{pmatrix}, \quad \tilde{b}_{\ell} = b_{\ell}^{2}, \quad L_{1} \leq \ell < L_{2}, \\ \tilde{W}_{L_{2}} &= \left( \begin{pmatrix} W_{L_{1},0}^{1} \\ W_{L_{2},0}^{2} \end{pmatrix} \middle| \begin{pmatrix} W_{L_{1},1}^{1} & \mathbf{0} \\ \mathbf{0} & W_{L_{2},1}^{2} \end{pmatrix} \middle| \dots \middle| \begin{pmatrix} W_{L_{1},L_{1}-1}^{1} & \mathbf{0} \\ \mathbf{0} & W_{L_{2},L_{1}-1}^{2} \end{pmatrix} \middle| \begin{pmatrix} \mathbf{0} \\ W_{L_{2},L_{1}}^{2} \end{pmatrix} \middle| \dots \middle| \begin{pmatrix} \mathbf{0} \\ W_{L_{2},L_{2}-1}^{2} \end{pmatrix} \right) \\ &= \tilde{b}_{L_{2}} = \begin{pmatrix} b_{L_{1}}^{1} \\ b_{L_{2}}^{2} \end{pmatrix}. \end{split}$$

Furthermore, we have

$$L(\Phi_1 \| \Phi_2) = \max\{L_1, L_2\}, \quad N(\Phi_1 \| \Phi_2) = N(\Phi_1) + N(\Phi_2) - d, \quad P(\Phi_1 \| \Phi_2) = P(\Phi_1) + P(\Phi_2).$$

**Connection between connected and standard networks.** We will now present a result that states if a function can be realized by a connected network, it can also be realized by a standard network with comparable complexity.

**Lemma 3.** For any connected network  $\Phi_c$  there is a standard network  $\Phi_s$  with the same input dimension such that

$$\Phi_s(\mathbf{x}) = \Phi_c(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^d$$

and

$$L(\Phi_s) = L(\Phi_c) = L, \qquad N(\Phi_s) \le C L N(\Phi_c), \qquad P(\Phi_s) \le C (L N(\Phi_c) + P(\Phi_c))$$

where C > 0 is a constant.

*Proof.* For the proof refer to the proof of Lemma 2.11 in [15].

Lemma 3 enables us to switch between connected and standard networks as desired.

# 14.9 Approximating product of two numbers by ReLU networks

Now consider two real numbers  $x \in \mathbb{R}$  and  $y \in \mathbb{R}$ , where  $\max\{|x|, |y|\} \leq D$ , with  $D \geq 1$ . Note that the case D < 1 can always be replaced by D = 1 and hence falls into the case  $D \geq 1$  that we consider here. Utilizing the identity formula  $xy = \frac{1}{2}((x+y)^2 - x^2 - y^2)$  and the approximation of the quadratic function by ReLU networks discussed in Section 14.7 we show that the product xy can be well approximated by a ReLU network.

**Lemma 4.** The function g(x, y) = xy on the domain  $[-D, D]^2$ , where  $D \ge 1$ , can be approximated by a ReLU network  $\Phi_{\varepsilon}^{xy}$  within any small error  $\varepsilon \in (0, 1/2)$  in the sense

$$||\Phi_{\varepsilon}^{xy}(x,y) - xy||_{L^{\infty}([-D,D]^2)} \le \varepsilon,$$

where the network has two input dimensions (one for x and one for y) and has the complexity  $L, M, P = \mathcal{O}(\log_2 \varepsilon^{-1} + \log_2 D)$ . Moreover, if x = 0 or y = 0, then  $\Phi_{\varepsilon}^{xy}(x, y) = 0$ .

*Proof.* Let  $\Phi_{\delta}$  be the network from Lemma 2 that approximates the quadratic function such that

$$||\Phi_{\delta}(z) - z^2||_{L^{\infty}([0,1])} \le \delta, \qquad z \in [0,1],$$

with complexity

$$L(\Phi_{\delta}), N(\Phi_{\delta}), P(\Phi_{\delta}) = \mathcal{O}(\log_2 \delta^{-1}).$$

Noting that  $xy = \frac{1}{2}((x+y)^2 - x^2 - y^2)$ , we write

$$xy = 4D^2 \frac{x}{2D} \frac{y}{2D} = 2D^2 \left( \left(\frac{x+y}{2D}\right)^2 - \left(\frac{x}{2D}\right)^2 - \left(\frac{y}{2D}\right)^2 \right),$$

and set

$$\Phi_{\varepsilon}^{xy}(x,y) = 2D^2 \Big( \Phi_{\delta}(\frac{|x+y|}{2D}) - \Phi_{\delta}(\frac{|x|}{2D}) - \Phi_{\delta}(\frac{|y|}{2D}) \Big).$$

With this setup, it easily follows that  $\Phi_{\varepsilon}^{xy}(x,y) = 0$  if x = 0 or y = 0. Moreover, we will have

$$||\Phi_{\varepsilon}^{xy}(x,y) - xy||_{L^{\infty}([-D,D]^2)} \le 6D^2C_0||\Phi_{\delta}(z) - z^2||_{L^{\infty}([0,1])} \le \varepsilon_1$$

provided we choose  $\delta = \varepsilon/(6D^2C_0)$ . Here,  $C_0 > 0$  is the constant in the inequality

$$||f_1 \circ f_2||_{L^{\infty}(\Omega_2)} \le C_0 ||f_1||_{L^{\infty}(\Omega_1)} ||f_2||_{L^{\infty}(\Omega_2)}, \qquad f_1 : \Omega_1 \subset \mathbb{R} \to \mathbb{R}, \quad f_2 : \Omega_2 \subset \mathbb{R}^2 \to \mathbb{R},$$

where  $\Omega_1$  and  $\Omega_2$  are compact, convex sets. In our case, we have  $\Omega_1 = [0, 1]$  and  $\Omega_2 = [-D, D]^2$ . It remains to show that the complexity of  $\Phi_{\varepsilon}^{xy}$  is of the same order as the complexity of  $\Phi_{\delta}$ . We can do this in a constructive way by building the network. By the identity formula  $|x| = \sigma(x) + \sigma(-x)$ , the network  $\Phi_{\varepsilon}^{xy}$  has the complexity (show this as an exercise),

$$L(\Phi_{\varepsilon}^{xy}) = L(\Phi_{\delta}) + 2, \quad N(\Phi_{\varepsilon}^{xy}) = 3N(\Phi_{\delta}) + 9, \quad P(\Phi_{\varepsilon}^{xy}) = 3P(\Phi_{\delta}) + 23.$$

Then we have

$$L(\Phi_{\varepsilon}^{xy}), N(\Phi_{\varepsilon}^{xy}), P(\Phi_{\varepsilon}^{xy}) = \mathcal{O}(\log_2 \delta^{-1}) = \mathcal{O}(\log_2 \varepsilon^{-1} + \log_2 D),$$

which completes the proof of the lemma.

# 14.10 ReLU networks and the product of their output components

Utilizing the result obtained in Section 14.9, we now discuss an interesting result on the approximation of the product of the output components of a network by another network with one output.

**Lemma 5.** Let  $\varepsilon \in (0, 1/2)$ . Let  $\Phi$  be a network with d inputs, m outputs, and complexity  $L, N, P \leq M$ , and suppose its output components  $\Phi_1, \ldots, \Phi_m$  are all bounded

$$||\Phi_i||_{L^{\infty}([0,1]^d)} \le 1, \qquad i = 1, \dots, m.$$

Then, there exists a network  $\Psi_{\varepsilon}$  with d inputs, one output, and complexity  $L, N, P \leq M C(m) \log_2 \varepsilon^{-1}$ such that

$$||\Psi_{\varepsilon}(\mathbf{x}) - \prod_{i=1}^{m} \Phi_i(\mathbf{x})||_{L^{\infty}([0,1]^d)} \le m \varepsilon.$$

Moreover,  $\Psi_{\varepsilon}(\mathbf{x}) = 0$  if  $\prod_{i=1}^{m} \Phi_i(\mathbf{x}) = 0$ , for  $\mathbf{x} \in [0, 1]^d$ .

*Proof.* The proof is by induction. For m = 1, we clearly have  $\Psi_{\varepsilon} = \Phi$ . Suppose the theorem holds for  $m \ge 1$ . We show that it also holds for m + 1. Let

$$\Phi = \{ (W_1, b_1), \dots, (W_L, b_L) \},\$$

be a network with d inputs,  $m + 1 \ge 2$  outputs, and complexity  $L, N, P \le M$ . In particular, it will be useful to note that we have

$$W_{L} = \begin{pmatrix} W_{L}^{(1:m)} \\ W_{L}^{(m+1)} \end{pmatrix} \in \mathbb{R}^{(m+1) \times \sum_{\ell=0}^{L-1} N_{\ell}}, \qquad W_{L}^{(1:m)} \in \mathbb{R}^{m \times \sum_{\ell=0}^{L-1} N_{\ell}}, \qquad W_{L}^{(m+1)} \in \mathbb{R}^{1 \times \sum_{\ell=0}^{L-1} N_{\ell}},$$

and

$$b_L = \begin{pmatrix} b_L^{(1:m)} \\ b_L^{(m+1)} \end{pmatrix} \in \mathbb{R}^{m+1}, \qquad b_L^{(1:m)} \in \mathbb{R}^m, \qquad b_L^{(m+1)} \in \mathbb{R}.$$

Now set

$$\hat{\Phi} = \{ (W_1, b_1), \dots, (W_{L-1}, b_{L-1}), (W_L^{(1:m)}, b_L^{(1:m)}) \},$$

be the network with d inputs and m outputs, formed by removing the last output neuron and its corresponding weights of  $\Phi$ . Note that the outputs of  $\hat{\Phi}$  are precisely the first m outputs of  $\Psi$ . By induction hypothesis, there exists a network with one output

$$\hat{\Psi}_{\varepsilon} = \{ (\hat{W}_1, \hat{b}_1), \dots, (\hat{W}_{\hat{L}}, \hat{b}_{\hat{L}}) \}, \qquad \hat{L} \ge L_{\varepsilon}$$

with the same first L-1 layers as in  $\hat{\Phi}$  and hence the same first L-1 layers as in  $\Phi$  such that

$$||\hat{\Psi}_{\varepsilon} - \prod_{i=1}^{m} \Phi_{i}||_{L^{\infty}([0,1]^{d})} = ||\hat{\Psi}_{\varepsilon} - \prod_{i=1}^{m} \hat{\Phi}_{i}||_{L^{\infty}([0,1]^{d})} \le m \varepsilon.$$

We modify  $\hat{\Psi}_{\varepsilon}$  by adding the removed neuron and its corresponding weights to the last layer of  $\hat{\Psi}_{\varepsilon}$ , and define a new network with two outputs

$$\tilde{\Psi}_{\varepsilon} = \{ (\hat{W}_1, \hat{b}_1), \dots, (\hat{W}_{\hat{L}-1}, \hat{b}_{\hat{L}-1}), (\tilde{W}_{\hat{L}}, \tilde{b}_{\hat{L}}) \}, \quad \tilde{W}_{\hat{L}} = \begin{pmatrix} \hat{W}_{\hat{L}} \\ W_L^{(m+1)} & \mathbf{0} \end{pmatrix}, \quad \tilde{b}_{\hat{L}} = \begin{pmatrix} \hat{b}_{\hat{L}} \\ b_L^{(m+1)} \end{pmatrix}$$

Note that the first output component of  $\tilde{\Psi}_{\varepsilon}$  is the same as the only output component of  $\hat{\Psi}_{\varepsilon}$ . We have

$$||(\tilde{\Psi}_{\varepsilon})_{1}||_{L^{\infty}([0,1]^{d})} = ||\hat{\Psi}_{\varepsilon}||_{L^{\infty}([0,1]^{d})} \le ||\hat{\Psi}_{\varepsilon} - \prod_{i=1}^{m} \Phi_{i}||_{L^{\infty}([0,1]^{d})} + ||\prod_{i=1}^{m} \Phi_{i}||_{L^{\infty}([0,1]^{d})} \le m\varepsilon + 1,$$

and

$$||(\tilde{\Psi}_{\varepsilon})_{2}||_{L^{\infty}([0,1]^{d})} = ||\Phi_{m+1}||_{L^{\infty}([0,1]^{d})} \le 1.$$

We can hence use Lemma 4 with D = m + 1 and construct the desired network as

$$\Psi_{\varepsilon}: \Phi_{\varepsilon}^{xy} \odot \tilde{\Psi}_{\varepsilon}.$$

It is easy to see that the complexity of network is as desired (verify this). We now work out the approximation error and write

$$\begin{split} ||\Psi_{\varepsilon} - \prod_{i=1}^{m+1} \Phi_i||_{L^{\infty}([0,1]^d)} &= ||\Phi_{\varepsilon}^{xy} \circ \tilde{\Psi}_{\varepsilon} - \Phi_{m+1} \prod_{i=1}^m \Phi_i||_{L^{\infty}([0,1]^d)} \\ &\leq ||\Phi_{\varepsilon}^{xy} \circ \tilde{\Psi}_{\varepsilon} - \Phi_{m+1} \hat{\Psi}_{\varepsilon}||_{L^{\infty}([0,1]^d)} + ||\Phi_{m+1} (\hat{\Psi}_{\varepsilon} - \prod_{i=1}^m \Phi_i)||_{L^{\infty}([0,1]^d)}. \end{split}$$

The first term in the right hand side of the above inequality is bounded by  $\varepsilon$  thanks to Lemma 4 with D = m + 1. The second term can also be bounded as

$$||\Phi_{m+1}(\hat{\Psi}_{\varepsilon} - \prod_{i=1}^{m} \Phi_{i})||_{L^{\infty}([0,1]^{d})} \le ||\Phi_{m+1}||_{L^{\infty}([0,1]^{d})} ||\hat{\Psi}_{\varepsilon} - \prod_{i=1}^{m} \Phi_{i}||_{L^{\infty}([0,1]^{d})} \le m\varepsilon,$$

thanks to the induction hypothesis. It hence follows that

$$||\Psi_{\varepsilon} - \prod_{i=1}^{m+1} \Phi_i||_{L^{\infty}([0,1]^d)} \le (m+1)\varepsilon$$

And this completes the proof.

# 14.11 ReLU networks and the partition of unity

We now return to the partition of unity and construct a two-layer network  $\Phi_{\mathbf{m}}$  with d inputs, d outputs, and complexity  $N, P = \mathcal{O}(d)$  such that the product of its output component realizes a function  $\phi_{\mathbf{m}} \in U$  in the partition of unity; see Section 14.5, that is,

$$\prod_{i=1}^{d} (\Phi_{\mathbf{m}})_{i}(\mathbf{x}) = \phi_{\mathbf{m}}(\mathbf{x}), \qquad \mathbf{x} \in [0, 1]^{d}.$$

Step 1. Construct a two-layer network  $\Phi_{\psi}$  that realizes the function  $\psi$  over  $\mathbb{R}$ , i.e.  $\Phi_{\psi}(z) = \psi(z)$ , for  $z \in \mathbb{R}$ ; see Figure 19. It is easy to verify that the output of the network is

$$\Phi_{\psi}(z) = \sigma(z+2) - \sigma(z+1) - \sigma(z-1) + \sigma(z-2) = \psi(z), \qquad z \in \mathbb{R}.$$

The network  $\Phi_{\psi}$  has the complexity L = 2, N = 6, and P = 12.



Figure 19: The two-layer ReLU network that realizes  $\psi(z)$  for  $z \in \mathbb{R}$ .

Step 2. By modifying the network in step 1, construct a two-layer network  $\Phi_{\psi}^{(i)}$  that realizes  $\psi(3n(x_i - \frac{m_i}{n}))$ ; see Figure 20. It is easy to verify that

$$\Phi_{\psi}^{(i)}(x_i) = \psi(3 n (x_i - \frac{m_i}{n})), \qquad x_i \in [0, 1].$$

The network  $\Phi_{\psi}^{(i)}$  has the complexity L = 2, N = 6, and P = 12.

Step 3. Simply stack up the *d* networks  $\Phi_{\psi}^{(1)}, \ldots, \Phi_{\psi}^{(d)}$  constructed in Step 2 in a vertical position (one below another) to build the desired two-layer network  $\Phi_{\mathbf{m}}$ , with complexity  $N(\Phi_{\mathbf{m}}) = 6 d$  and  $P(\Phi_{\mathbf{m}}) = 12 d$ .

#### 14.12 Approximating global polynomials by ReLU networks

We are now ready to complete Part 2 of the proof strategy in Section 14.3.

Specifically, we will construct a ReLU network that approximates the global polynomial  $f_n$  from Proposition 1, constructed in Section 14.6, and obtain an estimation for the error.

**Proposition 2.** Let  $f_n(\mathbf{x}) = \sum_{\mathbf{m}} \sum_{|\alpha| \le k-1} c_{\mathbf{m},\alpha} \phi_{\mathbf{m}}(\mathbf{x}) \mathbf{x}^{\alpha}$  be the global polynomial from Proposition 1 that approximates  $f \in W^{k,p}(X)$ , with  $k \ge 1$  and  $p \in [1, \infty]$ , where  $\mathbf{x} \in X = [0, 1]^d$ .



Figure 20: The two-layer ReLU network that realizes  $\psi(3n(x_i - \frac{m_i}{n}))$  for  $x_i \in [0, 1]$ .

Let  $\varepsilon \in (0, 1/2)$ . Then, there exists a ReLU network  $\Phi_{\varepsilon}$  with d inputs, one output, and complexity

$$L(\Phi_{\varepsilon}) \le C_1(k,d) \log_2 \varepsilon^{-1}, \qquad N(\Phi_{\varepsilon}), P(\Phi_{\varepsilon}) \le C_2(k,d) (n+1)^d \log_2 \varepsilon^{-1},$$

such that

$$||f_n - \Phi_{\varepsilon}||_{L^p(X)} \le C(k, d, p) ||f||_{W^{k-1, p}(X)} \varepsilon.$$

*Proof.* We first approximate the localized terms  $\phi_{\mathbf{m}}(\mathbf{x}) \mathbf{x}^{\alpha}$  for a fixed  $(\mathbf{m}, \alpha)$ , where  $|\alpha| \leq k - 1$ , by a ReLU network  $\Psi_{\varepsilon, \mathbf{m}, \alpha}$  as follows:

• There is a one-layer ReLU network  $\Phi_{\alpha}$  with d input,  $|\alpha|$  outputs, and at most d+k-1 neurons and at most k-1 weights such that

$$\mathbf{x}^{\boldsymbol{\alpha}} = \prod_{i=1}^{|\boldsymbol{\alpha}|} (\Phi_{\boldsymbol{\alpha}})_i(\mathbf{x}), \qquad ||(\Phi_{\boldsymbol{\alpha}})_i||_{L^{\infty}(X)} \le 1, \qquad \mathbf{x} \in X = [0,1]^d.$$

• From Section 14.11, we know that there is a two-layer network  $\Phi_{\mathbf{m}}$  with d inputs, d outputs, 6d neurons, and 12d parameters such that

$$\phi_{\mathbf{m}}(\mathbf{x}) = \prod_{i=1}^{d} (\Phi_{\mathbf{m}})_{i}(\mathbf{x}), \qquad ||(\Phi_{\mathbf{m}})_{i}||_{L^{\infty}(X)} \le 1, \qquad \mathbf{x} \in X = [0, 1]^{d}.$$

• We concatenate the above two networks and build a network  $\Phi_{\mathbf{m},\alpha} := \Phi_{\alpha} \parallel \Phi_{\mathbf{m}}$  that will have two layers, d inputs,  $d + |\alpha|$  outputs, at most 6d + k - 1 neurons, and at most 12d + k - 1 parameters, such that

$$\phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} = \prod_{i=1}^{|\boldsymbol{\alpha}|+d} (\Phi_{\mathbf{m},\boldsymbol{\alpha}})_i(\mathbf{x}), \qquad ||(\Phi_{\mathbf{m},\boldsymbol{\alpha}})_i||_{L^{\infty}(X)} \le 1, \qquad \mathbf{x} \in X = [0,1]^d.$$

• By Lemma 5, there is a ReLU network  $\Psi_{\varepsilon,\mathbf{m},\alpha}$  with d inputs, one output, and complexity

$$L(\Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}), N(\Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}), P(\Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}) \leq \hat{C}(k,d) \log_2 \varepsilon^{-1}$$

such that

$$||\Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}} - \prod_{i=1}^{|\boldsymbol{\alpha}|+d} (\Phi_{\mathbf{m},\boldsymbol{\alpha}})_i||_{L^{\infty}(X)} \le C'(k,d)\varepsilon,$$
(30)

and  $\Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x}) = 0$  if  $\phi_{\mathbf{m}}(\mathbf{x}) \mathbf{x}^{\boldsymbol{\alpha}} = 0$ , for  $\mathbf{x} \in X$ .

We next approximate the global polynomial, i.e. the sums of localized polynomials, by a ReLU network  $\Phi_{\varepsilon}$  as follows:

- Let  $\Lambda := \{(\mathbf{m}, \boldsymbol{\alpha}) : \mathbf{m} \in \{0, 1, \dots, n\}^d, |\boldsymbol{\alpha}| \le k-1\}$ , noting that  $|\Lambda| = \tilde{C}(k, d) (n+1)^d$ .
- Let  $\Phi_{\text{sum}}$  be the one-layer ReLU network that takes  $|\Lambda|$  inputs and gives their weighted sum with the weights  $c_{\mathbf{m},\alpha}$ . That is,  $\Phi_{\text{sum}} := \{(W, \mathbf{0})\}$ , where  $W \in \mathbb{R}^{1 \times |\Lambda|}$  is the matrix with components  $c_{\mathbf{m},\alpha} \in \Lambda$ .
- Let  $\Phi_{\varepsilon} := \Phi_{\text{sum}} \odot \left( \|_{(\mathbf{m}, \alpha) \in \Lambda} \Psi_{\varepsilon, \mathbf{m}, \alpha} \right)$ , where  $\|_{(\mathbf{m}, \alpha) \in \Lambda}$  denotes network concatenation over all elements of the set  $\Lambda$ . Clearly,  $\Phi_{\varepsilon}$  is a ReLU network with d inputs, one output, number of layers

$$L(\Phi_{\varepsilon}) \le 1 + \hat{C}(k, d) \log_2 \varepsilon^{-1} \le C_1(k, d) \log_2 \varepsilon^{-1},$$

and number of neurons and parameters

$$N(\Phi_{\varepsilon}), M(\Phi_{\varepsilon}) \le 2|\Lambda|(1+\hat{c}(k,d)\log_2\varepsilon^{-1}) \le C_2(k,d) (n+1)^d \log_2\varepsilon^{-1},$$

satisfying

$$\Phi_{\varepsilon}(\mathbf{x}) = \sum_{\mathbf{m}} \sum_{|\boldsymbol{\alpha}| \le k-1} c_{\mathbf{m},\boldsymbol{\alpha}} \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x}).$$

It remains to derive the approximation error. We write

$$\begin{split} ||f_n - \Phi_{\varepsilon}||_{L^p(X)}^p &= ||\sum_{\mathbf{m}} \sum_{|\alpha| \le k-1} c_{\mathbf{m},\alpha} \left( \phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\alpha} - \Psi_{\varepsilon,\mathbf{m},\alpha}(\mathbf{x}) \right)||_{L^p(X)}^p \\ &\le \sum_{\tilde{\mathbf{m}}} ||\sum_{\mathbf{m}} \sum_{|\alpha| \le k-1} c_{\mathbf{m},\alpha} \left( \phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\alpha} - \Psi_{\varepsilon,\mathbf{m},\alpha}(\mathbf{x}) \right)||_{L^p(B_{\tilde{\mathbf{m}}}^{\infty})}^p, \end{split}$$

where, similar to (20), the inequality follows from the integral-definition of  $L^p$ -norm and that  $X = \bigcup_{\tilde{\mathbf{m}}} B^{\infty}_{\tilde{\mathbf{m}}}$ . Now, for each multi-index  $\tilde{\mathbf{m}}$  we have

$$\begin{split} ||\sum_{\mathbf{m}}\sum_{|\boldsymbol{\alpha}|\leq k-1} c_{\mathbf{m},\boldsymbol{\alpha}} \left( \phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} - \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x}) \right)||_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})} \\ &\leq \sum_{\mathbf{m}: \, ||\mathbf{m}-\tilde{\mathbf{m}}||_{\infty}\leq 1} \sum_{|\boldsymbol{\alpha}|\leq k-1} |c_{\mathbf{m},\boldsymbol{\alpha}}| \, ||\phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} - \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x})||_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})} \\ &\leq c(k,d) \, n^{d/p} \sum_{\mathbf{m}: \, ||\mathbf{m}-\tilde{\mathbf{m}}||_{\infty}\leq 1} \sum_{|\boldsymbol{\alpha}|\leq k-1} ||f||_{W^{k-1,p}(B_{\tilde{\mathbf{m}}}^{\infty})} \, ||\phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} - \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x})||_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})} \end{split}$$

where the first inequality is a result of the triangle inequality and noting that  $\operatorname{supp}(\phi_{\mathbf{m}}) \subset B_{\mathbf{m}}^{\infty}$ and  $\Psi_{\varepsilon,\mathbf{m},\alpha}(\mathbf{x}) = 0$  if  $\phi_{\mathbf{m}}(\mathbf{x}) \mathbf{x}^{\alpha} = 0$ , which implies that we only need to consider multi-indices  $\mathbf{m}$  that are adjacent to  $\tilde{\mathbf{m}}$ , because only  $\phi_{\mathbf{m}}$  corresponding to such multi-indices take values on  $B_{\mathbf{m}}^{\infty}$  and hence contribute to the norm over  $B_{\mathbf{m}}^{\infty}$ . The second inequality is due to the bound on  $|c_{\mathbf{m},\alpha}|$  in Proposition 1.

We next state and use a useful theorem on the inclusion of  $L^p$  spaces. Let Y be a finite measure domain, such as  $B^{\infty}_{\tilde{\mathbf{m}}} \subset \mathbb{R}^d$ . Then for every  $1 \leq p < q \leq \infty$ , we have  $L^q(Y) \subset L^p(Y)$ . Moreover, by Hölder's inequality, we have

$$||f||_{L^p(Y)} \le \max(Y)^{1/p-1/q} ||f||_{L^q(Y)}.$$

In the particular case  $q = \infty$ , the inequality reads

$$||f||_{L^p(Y)} \le \max(Y)^{1/p} ||f||_{L^\infty(Y)}$$

Here, meas(Y) is the finite Lebesgue measure of the finite measure domain Y. We will use this last inequality for  $Y = B^{\infty}_{\hat{\mathbf{m}}}$ , where meas $(B^{\infty}_{\hat{\mathbf{m}}}) = c(d) (1/n)^d$ , and write

$$\begin{aligned} ||\phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} - \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x})||_{L^{p}(B_{\tilde{\mathbf{m}}}^{\infty})} &\leq c(d,p) \, n^{-d/p} \, ||\phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} - \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x})||_{L^{\infty}(B_{\tilde{\mathbf{m}}}^{\infty})} \\ &\leq c(k,d,p) \, n^{-d/p} \, \varepsilon, \end{aligned}$$

where the last inequality follows from (30). Combining the last two inequalities, we obtain

$$||\sum_{\mathbf{m}}\sum_{|\boldsymbol{\alpha}|\leq k-1} c_{\mathbf{m},\boldsymbol{\alpha}} \left( \phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} - \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x}) \right)||_{L^{p}(B^{\infty}_{\tilde{\mathbf{m}}})} \leq c(k,d,p) \, \varepsilon \, \sum_{\mathbf{m}: \, ||\mathbf{m}-\tilde{\mathbf{m}}||_{\infty} \leq 1} \, \sum_{|\boldsymbol{\alpha}|\leq k-1} ||f||_{W^{k-1,p}(B^{\infty}_{\tilde{\mathbf{m}}})}$$

Also noting that  $|\{\boldsymbol{\alpha}: |\boldsymbol{\alpha}| \leq k-1\}| \leq (k-1)^d$ , we have

$$||\sum_{\mathbf{m}}\sum_{|\boldsymbol{\alpha}|\leq k-1} c_{\mathbf{m},\boldsymbol{\alpha}} \left( \phi_{\mathbf{m}}(\mathbf{x}) \, \mathbf{x}^{\boldsymbol{\alpha}} - \Psi_{\varepsilon,\mathbf{m},\boldsymbol{\alpha}}(\mathbf{x}) \right) ||_{L^{p}(B^{\infty}_{\tilde{\mathbf{m}}})} \leq c(k,d,p) \, \varepsilon \sum_{\mathbf{m}: \, ||\mathbf{m}-\tilde{\mathbf{m}}||_{\infty} \leq 1} ||f||_{W^{k-1,p}(B^{\infty}_{\tilde{\mathbf{m}}})}.$$
(31)

Now, with the very same arguments as those in (23)-(25), we get

$$||f_n - \Phi_{\varepsilon}||_{L^p(X)} \le c(k, d, p) \varepsilon ||f||_{W^{k-1, p}(X)}.$$

The proof is complete.

### 14.13 Proof of Complexity Theorem

We are finally ready to complete the proof of the complexity theorem. From Proposition 1 we have

$$||f - f_n||_{L^p(X)} \le c(k, d, p) n^{-k} ||f||_{W^{k,p}(X)}.$$

From Proposition 2 with  $\frac{\varepsilon}{2C(k,d,p)||f||_{W^{k-1,p}(X)}}$  instead of  $\varepsilon$  in the proposition, we have

$$||f_n - \Phi_{\varepsilon}||_{L^p(X)} \le \frac{\varepsilon}{2},$$

with

$$L(\Phi_{\varepsilon}) \leq C_{1}(k, d, p) \log_{2}(\varepsilon^{-1} ||f||_{W^{k-1, p}(X)}),$$
  

$$N(\Phi_{\varepsilon}), P(\Phi_{\varepsilon}) \leq C_{2}(k, d, p) (n+1)^{d} \log_{2}(\varepsilon^{-1} ||f||_{W^{k-1, p}(X)}).$$

We now set

$$n = \left\lceil \left( \frac{\varepsilon}{2 c ||f||_{W^{k,p}(X)}} \right)^{-1/k} \right\rceil,$$

where c = c(k, d, p) is the constant from Proposition 1. It follows that

$$||f - \Phi_{\varepsilon}||_{L^{p}(X)} \le ||f - f_{n}||_{L^{p}(X)} + ||f_{n} - \Phi_{\varepsilon}||_{L^{p}(X)} \le c \, n^{-k} \, ||f||_{W^{k,p}(X)} + \frac{\varepsilon}{2} \le \varepsilon.$$

With this choice of n it also follows that  $(n+1)^d \leq C_0(k, p, d) \varepsilon^{-d/k} ||f||_{W^{k,p}(X)}^{d/k}$ , which in turn implies

$$N(\Phi_{\varepsilon}), P(\Phi_{\varepsilon}) \le C_2(k, d, p) \varepsilon^{-d/k} ||f||_{W^{k, p}(X)}^{d/k} \log_2(\varepsilon^{-1} ||f||_{W^{k-1, p}(X)})$$

To complete the proof, it is enough to note that by Lemma 3 there exists a standard ReLU network with the same input dimension as the connected network  $\Phi_{\varepsilon}$  with the same number of layers and with number of neurons and parameters

$$N, P \leq \hat{C}(k, d, p) \, \varepsilon^{-d/k} \, ||f||_{W^{k, p}(X)}^{d/k} \, \log_2^2(\varepsilon^{-1} \, ||f||_{W^{k-1, p}(X)}).$$

#### 14.14 Further reading and nonlinear approximation

The results and approaches commonly found in the literature (including the results discussed in this chapter) basically show that ReLU networks can approximate target functions with the same complexity-accuracy as classical linear methods of approximation such as approximation by polynomials and piecewise polynomials on prescribed partitions. Approximation by neural networks is however a form of nonlinear approximation and as such should be compared with other nonlinear methods. Two good references on the subject that interested readers are encouraged to refer to include [7] and [12]. The first paper [7] studies the approximation of univariate functions by deep ReLU networks and compares their approximation power with that of other non-linear approximation methods such as free knot linear splines. Some knowledge of non-linear approximation (see e.g. [8]) may be required. The second paper [12] provides a comprehensive overview of the recently developed Kolmogorov-Donoho nonlinear approximation, which relates the complexity of a given function class to the complexity of a deep network within prescribed accuracy. Other useful references on the subject may be found in [11]. We will summarize and discuss some of these results in Chapter 3.

# Chapter 3: Deep neural networks and nonlinear approximation

The goal of this chapter is to review a few recent developments on the approximation theory of deep neural networks within the framework of nonlinear approximation. We follow two specific goals. First we show that standard ReLU networks are at least as expressive as other nonlinear approximation methods, such as free knot linear splines. Next, we will mathematically justify the power of depth, giving a convincing reason for the success of deep networks. Specifically, we show that there exists a large class of functions, spanning from smooth/analytic functions to functions that are not smooth in any classical sense, that possess self-similarity and can be well approximated by deep ReLU networks with exponential accuracy.

# 15 Introduction

Let X = [0, 1] and consider a target function  $f : X \to \mathbb{R}$  in some known function space, e.g. the space of continuous functions:  $f \in C(X)$ . Suppose that we wish to approximate f by a standard ReLU network with width W and depth L; see the precise definition in Section 16. We will closely follow [7], focusing on two specific goals:

- Our first goal is to show that standard ReLU networks are at least as expressive as other nonlinear approximation methods, such as free knot linear splines.
- The second goal is to mathematically justify the power of depth, giving a convincing reason for the success of deep networks. Specifically, we show that there exists a large class of functions, spanning from smooth functions to functions that are not smooth in any classical sense, that possess self-similarity and can be approximated by deep ReLU networks with exponential accuracy.

# 16 Standard ReLU networks and free knot linear splines

In this section we consider and introduce the approximation space of connected feed-forward ReLU networks and their closest classical nonlinear approximation family of free know linear splines.

# 16.1 Space of standard ReLU networks

We consider a connected feed-forward ReLU network with one input neuron, one output neuron, and a fixed width W and a depth L. The depth here refers to the number of hidden layers (excluding the input and output layers), and each hidden layer has W neurons with ReLU activation function. No activation function will be applied on the output neuron. Such a network will have L + 1 layers and can uniquely be represented by a collection of L + 1weight matrices and bias vectors,

$$\Phi := \{ (W^{(0)}, b^{(0)}), (W^{(1)}, b^{(1)}), \dots, (W^{(L)}, b^{(L)}) \},\$$

where

$$W^{(0)} \in \mathbb{R}^{W \times 1}, \qquad W^{(1)}, \dots, W^{(L-1)} \in \mathbb{R}^{W \times W}, \qquad W^{(L)} \in \mathbb{R}^{1 \times W},$$

and

$$b^{(0)}, \ldots, b^{(L-1)} \in \mathbb{R}^{W \times 1}, \qquad b^{(L)} \in \mathbb{R}.$$

As before, we also denote by  $\Phi$  the univariate real-valued function that the network produces,

$$\Phi(x) = A^{(L)} \circ \sigma \circ A^{(L-1)} \circ \ldots \circ \sigma \circ A^{(0)}(x),$$

where

$$A^{(\ell)}(z) = W^{(\ell)}z + b^{(\ell)}, \qquad \ell = 0, 1, \dots, L,$$

and

$$\sigma(z) = (\sigma(z_1), \dots, \sigma(z_m)) = (\max\{0, z_1\}, \dots, \max\{0, z_m\}), \qquad z = (z_1, \dots, z_m) \in \mathbb{R}^m.$$

The distinction between  $\Phi$  being a network (a set of matrix-vector tuples) or a function that the network realizes should be easily made from the context.

We denote by  $\mathcal{M}_{W,L}$  the set of all functions generated by standard ReLU networks,

 $\mathcal{M}_{W,L} = \{ \Phi : [0,1] \to \mathbb{R} \text{ produced by all possible choices of weights and biases} \}.$ 

Any function in  $\mathcal{M}_{W,L}$  is determined by

$$n(W,L) = W(W+1)L - (W-1)^2 + 2 \sim W^2L,$$

parameters (i.e. total number of network parameters). Here, the symbol ~ stands for "comparable to", that is, there exist constants  $c_1, c_2$  such that  $c_1W^2L \leq n(W,L) \leq c_2W^2L$ . For instance, when  $W, L \geq 2$  these inequalities hold with  $c_1 = 1/2$  and  $c_2 = 2$ .

Throughout this chapter we keep W fixed and let L vary. This would in particular enable us to study the effect of depth on the approximation power of standard ReLU networks.

#### 16.2 Space of free knot linear splines

Every function in  $\mathcal{M}_{W,L}$  is a continuous piecewise linear function on X = [0, 1]. Hence, the closest classical nonlinear approximation family to  $\mathcal{M}_{W,L}$  may be the set of free knot linear splines,

 $S_n = \{S : [0,1] \to \mathbb{R} \text{ continuous piecewise linear with at most } n \text{ distinct breakpoints in } (0,1)\}.$ 

Any function in  $S_n$  is determined by at most 2n + 2 parameters: at most 2n parameters for the location of n breakpoints in (0, 1) and the values of the function at those breakpoints, and 2 parameters for the values of the function at the two boundary points 0 and 1. Note that when  $n(W, L) \sim n$ , then  $\mathcal{M}_{W,L}$  and  $S_n$  have comparable complexity in terms of the number of parameters.

# 16.3 A comparison between one-layer ReLU networks and free knot linear splines

It is easy to see that the space  $S_W$  is essentially the same as  $\mathcal{M}_{W,1}$ . Precisely, the following inclusion holds on [0, 1],

$$\mathcal{S}_{W-1} \subset \mathcal{M}_{W,1} \subset \mathcal{S}_W.$$

That is, for large W, the space  $\mathcal{M}_{W,1}$  of one-layer networks is essentially the same as the space  $\mathcal{S}_W$  of free knot linear splines. This is perhaps why practitioners at early stages of neural network approximation focused more on shallow networks, and in particular on one-layer networks.

We first show the second inclusion, noting that a network with one hidden layer (L = 1)and width W generates a continuous piecewise linear function with at most W breakpoints, where the number and location of breakpoints are determined by the network parameters. This can easily be seen by,

$$A^{(1)} \circ \sigma \circ A^{(0)}(x) = b^{(1)} + \begin{bmatrix} w_1^{(1)} & w_2^{(1)} & \cdots & w_W^{(1)} \end{bmatrix} \begin{bmatrix} \sigma(w_1^{(0)}x + b_1^{(0)}) \\ \vdots \\ \sigma(w_W^{(0)}x + b_W^{(0)}) \end{bmatrix} = b^{(1)} + \sum_{j=1}^W w_j^{(1)} \sigma(w_j^{(0)}x + b_j^{(0)}).$$

This means that if  $g \in \mathcal{M}_{W,1}$  then  $g \in \mathcal{S}_W$ , and hence  $\mathcal{M}_{W,1} \subset \mathcal{S}_W$ . As an example, Figure 21 displays four different outputs of a network with W = 3 and L = 1 corresponding to four different sets of weight-bias parameters, generating four piecewise linear functions with 0, 1, 2, and  $W^L = 3$  break points.



Figure 21: Different outputs of four ReLU networks with a fixed architecture (W, L) = (3, 1)and four different parameter sets. Network outputs are piecewise linear functions with up to W = 3 breakpoints.

To see the first inclusion, we note that every continuous piecewise linear function on [0, 1]with W - 1 interior breakpoints on (0, 1) is the restriction of a function from  $\mathcal{M}_{W,1}$  to [0, 1]. Indeed, every  $S \in \mathcal{S}_{W-1}$  on [0, 1] with interior breakpoints  $x_1, \ldots, x_{W-1}$  can be represented

$$S(x) = a x + b + \sum_{j=1}^{W-1} m_j \sigma(x - x_j) = \begin{bmatrix} a & m_1 & \cdots & m_{W-1} \end{bmatrix} \sigma \left( \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} x + \begin{bmatrix} 0 \\ -x_1 \\ \vdots \\ -x_{W-1} \end{bmatrix} \right) + b,$$

which means  $S \in \mathcal{M}_{W,1}$ . Hence  $\mathcal{S}_{W-1} \subset \mathcal{M}_{W,1}$ . As an example, the hat function  $H : [0, 1] \rightarrow [0, 1]$  that has one interior breakpoint belongs to  $\mathcal{M}_{2,1}$ ,

$$H(x) = 2\sigma(x) - 4\sigma(x - 0.5).$$

Note that as we have sen before, H belongs to  $\mathcal{M}_{3,1}$  on  $\mathbb{R}$ , and only its restriction to [0,1] belongs to  $\mathcal{M}_{2,1}$ .

#### 16.4 A comparison between deep ReLU networks and free knot linear splines

The composition of two continuous piecewise linear functions will be another continuous piecewise linear function with the maximum number of breakpoints equal to the product of the number of breakpoints of individual functions. That is, if f has n breakpoints and g has m breakpoints, then  $f \circ g$  will have at most nm breakpoints. This allows networks with L hidden layers of width W to create roughly  $W^L$  breakpoints by  $n(W,L) \sim W^2L$  special choices of parameters (weights and biases). This means that a "deep" network with n(W, L) parameters can produce functions with many more breakpoints than the number of parameters of a free knot spline is roughly double the number of breakpoints  $(2n + 2 \sim 2n)$ . As we will see in the remainder of this chapter, this gives deep networks approximation superiority over free knot linear splines provided the target function has a special (compositional/self-similarity) structure.

#### 17 ReLU networks are at least as expressive as free knot linear splines

Throughout this section, we fix  $W \ge 8$  and let  $L \ge 2$  and consider  $\mathcal{M}_{W,L}$ . The goal is to show that  $\mathcal{S}_n \subset \mathcal{M}_{W,L}$  where  $n(W,L) \le Cn$  with an absolute constant C. Precisely, we will show that for every fixed  $W \ge 8$ , any function in  $\mathcal{S}_n$  is the output of a (special) network with the number of parameters comparable to n. We extensively use the following quantity

$$q := \left\lfloor \frac{W-2}{6} \right\rfloor, \qquad W \ge 8.$$
(32)

**Theorem 1.** Fix  $W \ge 8$  and let  $n \ge 1$ . The set  $S_n$  of free knot linear splines with at most n breakpoints is contained in the set  $\mathcal{M}_{W,L}$ , where L = 2 when n < q(W - 2) and  $L = 2\lceil n/q(W - 2)\rceil$  when  $n \ge q(W - 2)$ . The complexity of such network space  $\mathcal{M}_{W,L}$  is  $n(W,L) = W^2 + 4W + 1$  when n < q(W - 2) and  $n(W,L) \le 61n$  when  $n \ge q(W - 2)$ .

The rest of this section will be devoted to the proof of Theorem 1. The proof is constructive and utilizes a special type of ReLU networks.

by

# 17.1 Special ReLU networks

For  $W \ge 4$  and  $L \ge 2$  we consider a special ReLU network as a "special" subset of a ReLU network with the same width and depth, where special roles are reserved for the top and bottom neurons of hidden layers. Specifically, we define three types of channels for special ReLU networks:

- A source channel formed by top neurons that are assumed to be ReLU-free with unit weights and without any bias. The neurons in a source channel do not take any input from neurons in other channels and do not do any computation. This channel simply carries forward the input x.
- A collation channel formed by bottom neurons that are also assumed to be ReLU-free. This channel is used to collect all intermediate computations (i.e. outputs of hidden layers). The neurons in a collation channel do not feed into subsequent calculations. They only take outputs of neurons in previous layer and carry them over with unit weight to subsequent bottom neurons.
- The rest of channels are regular computational channels.

Figure 22 displays the graph representation of a special ReLU network with W = 4 and L = 3.



Figure 22: Graph representation of a special ReLU network in  $\hat{\mathcal{M}}_{4,3}$ .

We denote by  $\hat{\mathcal{M}}_{W,L}$  the set of all functions generated by special ReLU networks,

 $\hat{\mathcal{M}}_{W,L} = \{ \Phi : [0,1] \to \mathbb{R} \text{ produced by a special network of width } W \ge 4 \text{ and depth } L \ge 2 \}.$ 

It is to be noted that since top and bottom neurons are ReLU free, special networks do not form a direct subset of ReLU networks. However, we make the following observations:

- Since the input is non-negative,  $x \in [0, 1]$ , then  $x = \sigma(x)$ , and hence the assumption that top neurons are ReLU free is not restrictive.
- The first bottom neuron in the first hidden layer  $(\ell = 0)$  is simply taking zero and since  $0 = \sigma(0)$ , the ReLU free assumption is not restrictive.

• Any other bottom neuron in the remaining L-1 hidden layers  $(\ell = 1, \ldots, L-1)$  takes an input function, say  $g_{\ell}(x)$  that depends continuously on x. Hence there is a constant, say  $C_{\ell}$ , such that  $g_{\ell}(x) + C_{\ell} \ge 0$  for every  $x \in [0, 1]$ . This implies that  $g_{\ell}(x) = \sigma(g_{\ell}(x) + C_{\ell}) - C_{\ell}$ .

Consequently, given any function  $\Phi \in \hat{\mathcal{M}}_{W,L}$ , determined by the parameter set  $\{\hat{W}^{(\ell)}, \hat{b}^{(\ell)}\}_{\ell=0}^{L}$ , one can construct a ReLU network with the same complexity that produces the same function  $\Phi$ . The parameters  $\{W^{(\ell)}, b^{(\ell)}\}_{\ell=0}^{L}$  of such ReLU network are given in terms of the parameters of the special network by

$$W^{(\ell)} = \hat{W}^{(\ell)}, \qquad \ell = 0, \dots, L,$$
  

$$b_j^{(\ell)} = \hat{b}_j^{(\ell)}, \qquad j = 1, \dots, W - 1, \qquad b_W^{(\ell)} = \hat{b}_W^{(\ell)} + C_\ell, \qquad \ell = 1, \dots, L - 1,$$
  

$$b^L = \hat{b}^L - \sum_{\ell=1}^{L-1} C_\ell.$$

This implies that given any  $\Phi \in \hat{\mathcal{M}}_{W,L}$ , there exists a standard ReLU network in  $\mathcal{M}_{W,L}$  with the same complexity that produces  $\Phi$ . Hence, although special networks are not direct subsets of ReLU networks, in terms of sets of functions produced by them the following inclusion holds,

$$\mathcal{M}_{W,L} \subset \mathcal{M}_{W,L}.$$

**Proposition 3.** Special ReLU networks have the following properties:

- (i) For every  $W, L_1, L_2$ :  $\hat{\mathcal{M}}_{W,L_1} + \hat{\mathcal{M}}_{W,L_2} \subset \hat{\mathcal{M}}_{W,L_1+L_2}$ .
- (ii) For L < P:  $\hat{\mathcal{M}}_{W,L} \subset \hat{\mathcal{M}}_{W,P}$ .

*Proof.* We first prove (i). Fix two networks/functions  $\Phi_1 \in \hat{\mathcal{M}}_{W,L_1}$  and  $\Phi_2 \in \hat{\mathcal{M}}_{W,L_2}$ , and construct a concatenated network as follows.

- 1. Start by placing the input and all  $L_1$  hidden layes of  $\Phi_1$ ;
- 2. Continue with placing the first hidden layer of  $\Phi_2$  with one change: in its collation neuron place the output of  $\Phi_1$  rather than zero;
- 3. Finally, place the remaining hidden layers and output layer of  $\Phi_2$ .

The output of such concatenated network, which has  $L_1 + L_2$  hidden layers, will be  $\Phi_1 + \Phi_2$  as desired, thanks to the concatenated collation channel that carries over  $\Phi_1$  and adds it to  $\Phi_2$ ; see Figure 23.

Proof of (ii) follows from (i) with  $\Phi_2 \equiv 0$  and  $L_1 = L$  and  $L_2 = P - L$ .



Figure 23: Graph representation of the concatenated network that outputs the sum of  $\Phi_1$  (in blue) and  $\Phi_2$  (in red).

#### 17.2 Special ReLU networks that realize continuous piecewise linear functions

Consider a positive integer M = q(W-2), with q in (32). Note that since q is of order of W, the integer M will be of order of  $W^2$  and may hence be a small number. We first show how to construct a special ReLU network with only 2 hidden layers that generates continuous piecewise linear functions with M (that may be small) breakpoints.

Let  $x_1 < \ldots < x_M \in (0, 1)$  be any M given breakpoints in the interior of [0, 1]. Let  $x_0 = 0$ and  $x_{M+1} = 1$ . Let further  $\mathcal{S}_M^0 = \mathcal{S}_M^0(x_0, x_1, \ldots, x_{M+1})$  denote the set of all continuous piecewise linear functions that vanish outside (0, 1) and with breakpoints only at  $x_0, x_1, \ldots, x_{M+1}$ . The space  $\mathcal{S}_M^0$  is a linear space of dimension M (and not M + 2 because its elements vanish at  $x_0$  and  $x_{M+1}$ ).

**Lemma 6.** For every M breakpoints  $x_1 < \ldots < x_M \in (0,1)$ , where M = q(W-2), with q in (32), we have

$$\mathcal{S}_M^0 \subset \hat{\mathcal{M}}_{W,2}.$$

*Proof.* Step 1. We first create a particular basis for  $S_M^0$  as follows. We introduce a set  $\{\xi_j = x_{jq}\}_{j=1}^{W-2}$  of W-2 "principal breakpoints", where each  $\xi_j$  is associated with q hat basis functions  $H_{i,j}$ ,  $i = 1, \ldots, q$ , that take value 1 at  $\xi_j$  and are supported on  $[x_{jq-i}, x_{jq+1}]$ ; see Figure 24. We notice that the case q = 1 (when  $8 \leq W \leq 13$ ) simply coincides with the



Figure 24: Schematic representation of q basis hat functions  $H_{1,j}, \ldots, H_{q,j}$  associated to the principle breakpoint  $\xi_j$ .

familiar finite element basis with Lagrange elements. There will be a total number of M basis

function, and we name them  $\{\phi_k\}_{k=1}^M$ , ordered in such a way that  $\phi_k$  has leftmost breakpoint  $x_{k-1}$ . We say  $\phi_k$  is associated with  $\xi_j$  if  $\phi_k(\xi_j) = 1$ . Clearly,  $\{\phi_k\}_{k=1}^M$  forms a basis for  $\mathcal{S}_M^0$ , as  $\phi_k$ 's are linearly independent. Indeed, if  $\sum_{k=1}^M c_k \phi_k = 0$ , then  $c_1 = 0$  because  $\phi_1$  is the only nonzero function on  $[x_0, x_1]$ . With a similar argument, considering  $\phi_2$ , we get  $c_2 = 0$ , etc.

Step 2. Let  $S \in \mathcal{S}_M^0$ . By Step 1, we can write  $S(x) = \sum_{k=1}^M c_k \phi_k(x)$ . Note that with principal breakpoints  $\xi_1, \ldots, \xi_{W-2}$ , we have  $S(\xi_j) = \sum_{k:\phi_k(\xi_j)=1} c_k$ , and there are q indices in the summation associated to each principal breakpoint. We take the index set  $\Lambda = \{1, \ldots, M\}$  and divide it into two sets  $\Lambda = \Lambda^+ \cup \Lambda^-$ , where

$$\Lambda^+ = \{k \in \Lambda : c_k \ge 0\}, \qquad \Lambda^- = \{k \in \Lambda : c_k < 0\}.$$

We then divide each of  $\Lambda^+$  and  $\Lambda^-$  into at most 3q subsets. For instance, we split  $\Lambda^+$  into

$$\Lambda_{i,m}^{+} = \{ s \in \Lambda^{+} : \phi_{s} = H_{i,j} \text{ such that } j \text{ mod } 3 = m \}, \qquad i = 1, \dots, q, \quad m = 0, 1, 2.$$

More explicitly, we get three sets

$$\Lambda_{i,0}^{+} = \{ s \in \Lambda^{+} : \phi_{s} = H_{i,3} \text{ or } \phi_{s} = H_{i,6} \text{ or } \phi_{s} = H_{i,9} \dots \}$$
  
$$\Lambda_{i,1}^{+} = \{ s \in \Lambda^{+} : \phi_{s} = H_{i,1} \text{ or } \phi_{s} = H_{i,4} \text{ or } \phi_{s} = H_{i,7} \dots \}$$
  
$$\Lambda_{i,2}^{+} = \{ s \in \Lambda^{+} : \phi_{s} = H_{i,2} \text{ or } \phi_{s} = H_{i,5} \text{ or } \phi_{s} = H_{i,8} \dots \}.$$

Similarly, we split  $\Lambda^-$  into at most 3q subsets,

$$\Lambda_{i,m}^{-} = \{ s \in \Lambda^{-} : \phi_s = H_{i,j} \text{ such that } j \mod 3 = m \}, \qquad i = 1, \dots, q, \quad m = 0, 1, 2.$$

We can hence write  $\Lambda = \{1, \ldots, M\}$  as a disjoint union of  $K \leq 6q \leq W - 2$  sets  $\Lambda_p$ :

$$\Lambda = \cup_{p=1}^{K} \Lambda_p,$$

where each  $\Lambda_p$  has two properties: 1) all coefficients  $c_k$  of S with  $k \in \Lambda_p$  have the same sign, and 2) for every  $k, k' \in \Lambda_p$ , the associated principle breakpoints  $\xi_j$  and  $\xi_{j'}$  (associated to  $\phi_k$  and  $\phi_{k'}$ ) satisfy the separation/non-overlapping property  $|j - j'| \geq 3$ . We note that some of  $\Lambda_p$ 's may be empty, and hence we may get K < W - 2. In this case we set  $\Lambda_{K+1} = \ldots = \Lambda_{W-2} = \emptyset$ . We then write

$$S(x) = \sum_{p=1}^{W-2} S_p(x), \qquad S_p = \begin{cases} \sum_{i \in \Lambda_p} c_i \phi_i(x), & \Lambda_p \neq \emptyset, \\ 0, & \Lambda_p = \emptyset. \end{cases}$$
(33)

Because of the separation property, all  $\phi_i$ 's with  $i \in \Lambda_p$  have disjoint supports, and hence  $c_i = S_p(\xi_j)$ , where  $\xi_j$  is the principal breakpoint associated to  $\phi_i$ .

Step 3. We next show that each  $S_p$  corresponding to a nonempty  $\Lambda_p$  is of the form  $\pm \sigma(g_p(x))$ , where  $g_p$  is a continuous piecewise linear function whose breakpoints are the principal breakpoints  $\xi_1, \ldots, \xi_{W-2}$ . We fix p and first consider the case where all  $c_i$ 's in  $\Lambda_p$  are non-negative. We let  $g_p$  be the continuous piecewise linear function that takes the value  $c_i$  at each principal breakpoint  $\xi_j$  associated to  $i \in \Lambda_p$ . Note that each  $i \in \Lambda_p$  has a different associated  $\xi_j$  thanks


Figure 25: A simple example of constructing  $g_p$  (thin red line) such that  $\sigma(g_p(x)) = S_p(x)$ . The function  $S_p$  is displayed with thick blue line. Clearly,  $g_p$  is a continuous piecewise linear function whose breakpoints are the principal breakpoints  $\xi_1, \ldots, \xi_{W-2}$  (red circles). Note that  $g_p$  vanishes at the leftmost and rightmost breakpoints (black circles) of all  $\phi_i$ 's with  $i \in \Lambda_p$ .

to the separation property. At the remaining principal breakpoints, we assign negative values for  $g_p$  such that  $g_p$  vanishes at the leftmost and rightmost breakpoints of all  $\phi_i$ 's with  $i \in \Lambda_p$ . Again, this is possible thanks to the separation property. A simple example is depicted in Figure 25. It follows that  $\sigma(g_p(x)) = S_p(x)$ . Similarly, when all  $c_i$ 's in  $\Lambda_p$  are negative, we will do a similar procedure noting that  $-\sigma(-g_p(x)) = S_p(x)$ .

Step 4. Finally, we construct a special ReLU network with two hidden layers that generates  $S(x) = \sum_{p=1}^{W-2} S_p(x)$ ; see Figure 26. The W-2 compute neurons labeled  $j = 1, \ldots, W-2$ 



Figure 26: A special ReLU network with two hidden layers that outputs  $S(x) = \sum_{p=1}^{W-2} S_p(x)$ . in the first hidden layer are used to generate  $\sigma(x - \xi_j)$ , with  $j = 1, \ldots, W - 2$ . This can be

easily done by setting input weights to 1 and biases to  $\xi_1, \ldots, \xi_{W-2}$ . In the second hidden layer, each compute unit, labeled  $p = 1, \ldots, W-2$ , is used to take x and all  $\sigma(x - \xi_j)$ 's with  $j = 1, \ldots, W-2$  to output  $S_p$ . To find the corresponding weights and biases, we first note that the index  $p = 1, \ldots, W-2$  corresponds to a different labeling of the index set  $\{(i, m, \pm), i =$  $1, \ldots, q, m = 0, 1, 2\}$ . Each p corresponds to a triplet (i, m, +) or (i, m, -). We take a fixed p, say corresponding to (i, m, +), and this gives us the set  $\Lambda_{i,m}^+$  which corresponds to the set  $\Lambda_p$  which in turn contains indices  $i \in \Lambda_p$  in  $S_p(x) = \sum_{i \in \Lambda_p} c_i \phi_i(x) = \sigma(g_p(x))$ . All we will need to do is to find the weights  $\{m_{p,1}, \ldots, m_{p,W-1}$  and the bias  $b_p$  in the second hidden layer such that

$$g_p(x) = m_{p,1}x + \sum_{j=2}^{W-1} m_{p,j} \sigma(x - \xi_{j-1}) + b_p.$$

This can be easily done recalling that  $g_p$  is a continuous piecewise linear function whose breakpoints are the principal breakpoints  $\xi_1, \ldots, \xi_{W-2}$ , and hence it can be represented by a linear combination of  $\{\sigma(x - \xi_j)\}_{j=1}^{W-2}$  and x terms. To do this we start with noting that  $b_p = g_p(0)$ . Then we take the first principal breakpoints and write  $g_p(\xi_1) = m_{p,1}\xi_1 + b_p$ , which gives us  $m_{p,1}$ . Then we proceed with the second principal breakpoints and write  $g_p(\xi_2) = m_{p,1}\xi_2 + m_{p,2}(\xi_2 - \xi_1) + b_p$ , which gives us  $m_{p,2}$ , and so forth. We note that in this particular setup, the source and collation channels are not used. Nevertheless we keep them in case we would need to perform concatenation or other operations on the input-output. This completes the proof.

We will now focus on the construction of deep special ReLU networks with many hidden layers that generate continuous piecewise linear functions with a large number of breakpoints.

**Lemma 7.** For every N breakpoints  $x_1 < \ldots < x_N \in (0, 1)$ , where N = q(W - 2)L, with q in (32), we have

$$\mathcal{S}_N \subset \mathcal{M}_{W,2L}.$$

Proof. We need to show that every  $S \in S_N$  is the output of a special ReLU network with width W and depth 2L. Let  $x_1 < \ldots < x_N$  be the breakpoints of S in (0, 1), and set  $x_0 = 0$  and  $x_{N+1} = 1$ . Let g(x) = ax + b be the linear function that interpolates S at the two endpoints 0 and 1. Then  $T = S - g \in S_N^0$  vanishes at the endpoints, and it will be enough to show that  $T \in \hat{\mathcal{M}}_{W,2L}$ . We first write  $T = T_0 + T_1 + \ldots + T_{L-1}$ , where each  $T_j \in S_N^0$  is a continuous piecewise linear function that agrees with T at the points  $\{x_{jq(W-2)+1}, \ldots, x_{(j+1)q(W-2)}\}$  and is zero at all other breakpoints of T; see an example in Figure 27. Since each  $T_j$  may be non-zero only at M = q(W - 2) breakpoints, we will have



Figure 27: Decomposition of  $T = T_0 + T_1 + \ldots + T_{L-1} \in \mathcal{S}_N^0$  into L functions  $T_j \in \mathcal{S}_M^0$ , with  $j = 0, 1, \ldots, L-1$ , where N = ML.

 $T_j \in \mathcal{S}_M^0$  for  $j = 0, 1, \ldots, L-1$ . By Lemma 6 we have  $T_j \in \hat{\mathcal{M}}_{W,2}$ , with  $j = 0, 1, \ldots, L-1$ . We can now simply concatenate these L networks as in Proposition 3 and construct a network with 2L hidden layers that realizes T. Finally, in order to produce S = T + g, where g(x) = ax + b, we can simply use the source channel and assign weight a to the link the top neuron of the last hidden layer and the output neuron (which was not used when generating  $T_{L-1}$ ). We also assign bias b to the output neuron. This complete the proof.  $\Box$ 

#### 17.3 Proof of Theorem 1

We need to show that  $\mathcal{S}_n \subset \mathcal{M}_{W,L}$  where  $n(W,L) \sim n$ . By Lemma 7 and inclusion  $\hat{\mathcal{M}}_{W,2L} \subset \mathcal{M}_{W,2L}$ , we have

$$\mathcal{S}_N \subset \mathcal{M}_{W,2L}, \qquad N = ML, \qquad M = q (W - 2),$$
(34)

with q given in (32). We consider two cases.

If n < M, then we have  $S_n \subset S_M \subset \mathcal{M}_{W,2}$ , where the second inclusion follows from (34) with L = 1. Clearly we have  $n(W, 2) = 2W(W+1) - (W-1)^2 + 2 = W^2 + 4W + 1$  as desired. If  $n \ge M$ , we choose

$$L = \left\lceil \frac{n}{q(W-2)} \right\rceil < \frac{n}{q(W-2)} + 1,$$

and hence

$$n(W, 2L) = 2LW(W+1) - (W-1)^2 + 2$$
  

$$< 2W(W+1)(1+n/q(W-2)) - (W-1)^2 + 2$$
  

$$= \frac{2W(W+1)}{q(W-2)}n + W^2 + 4W + 1$$
  

$$< 34n + W^2 + 4W + 1.$$

The last inequality is because 2W(W+1)/q(W-2) attains its maximum for W = 13 and q = 1, and hence  $2W(W+1)/q(W-2) \le 364/11 < 34$ . Moreover, it is easy to see that by (32) we have  $W^2 + 4W + 1 < 27q(W-2) \le 27n$ . Hence we get n(W, 2L) < 34n + 27n = 61n, as desired. This completes the proof of Theorem 1.

### 18 The power of depth

So far we have seen that ReLU networks are at last as expressiv as free knot linear splines with at most n breakpoints. Similar results also hold when comparing ReLU networks with other nonlinear approximation methods; see e.g. Section 6 of [7] that comapres ReLU networks with n-term approximation from a Fourier-like basis.

In this section we will discuss the power of depth. Precisely, we show that there exists a large class of functions that possess self-similarity and hence can be approximated by ReLU networks with approximation rates that far exceed the approximation rates of free knot linear splines or any other classical approximation family.

We will first present a few additive and compositional properties of standard and special ReLU networks (see Section 18.1) and then utilize them to discuss and exemplify the power of depth for self-similar functions (see Section 18.2).

### 18.1 Additive and compositional properties of ReLU networks

**Proposition 4.** Let  $W \ge 2$ . For any  $\Phi_j \in \mathcal{M}_{W,L_j}$  with  $j = 1, \ldots, J$ , the followings hold:

(i) 
$$\Phi_J \circ \ldots \circ \Phi_1 \in \mathcal{M}_{W,L_1+\ldots+L_J}$$

(*ii*) 
$$\Phi_1 + \ldots + \Phi_J \in \hat{\mathcal{M}}_{W+2,L_1+\ldots+L_J}$$

*Proof.* To prove (i), we first concatenate the networks  $\Phi_1$  and  $\Phi_2$  as follows to construct a network, say  $\Phi_{2\circ 1}$ , that outputs  $\Phi_2 \circ \Phi_1$ :

- 1. The input and the first  $L_1$  hidden layers of  $\Phi_{2\circ 1}$  will be the same as the input and  $L_1$  hidden layers of  $\Phi_1$ .
- 2. The  $(L_1 + 1)$ -st hidden layer of  $\Phi_{2\circ 1}$  the first hidden layer of  $\Phi_2$ .
- 3. The weights between hidden layers  $L_1$  and  $L_1 + 1$  (a  $W \times W$  matrix) will be the output weights of  $\Phi_1$  (a  $1 \times W$  vector) multiplied from left by the input weights of the first hidden layer of  $\Phi_1$  (a  $W \times 1$  vector).
- 4. The bias of hidden layer  $L_1 + 1$  will be the bias of the first hidden layer of  $\Phi_2$  plus the product of the output bias of  $\Phi_1$  and the input weights of the first hidden layer of  $\Phi_2$ .
- 5. The remaining hidden layers of  $\Phi_{2\circ 1}$  will be the same as the remaining hidden layers of  $\Phi_2$ .

The resulting network will have  $L_1 + L_2$  hidden layers. This procedure can be applied J - 1 times to generate  $\Phi_{J \circ \dots \circ 1}$  with  $L_1 + \dots + L_J$  hidden layers.

To prove (ii), we concatenate the standard ReLU networks  $\Phi_1$  and  $\Phi_2$  by adding a source and a collation channel, as shown in Figure 28, to construct a special ReLU network, say  $\Phi_{1+2}$ , with width W + 2 and depth  $L_1 + L_2$ , that outputs  $\Phi_1 + \Phi_2$ . Again this can be repeated J - 1 times to build a network with width W + 2 and depth  $L_1 + \ldots + L_J$  that outputs  $\Phi_1 + \ldots + \Phi_J$ . This completes the proof.

In what follows, we denote by  $g^{\circ k}$  the k-fold composition of a function g with itself

$$g^{\circ k}(x) = \underbrace{g \circ g \circ \ldots \circ g}_{k \text{ times}}(x), \qquad s \ge 1.$$

Note that  $g^{\circ 1}(x) = g(x)$ .

**Proposition 5.** Let  $W \geq 2$ . For any  $T \in \mathcal{M}_{W,L}$ , then

$$\Phi(x) = \sum_{i=1}^{m} a_i T^{\circ i}(x) \in \hat{\mathcal{M}}_{W+2,mL}.$$



Figure 28: By adding source and collation channels we can concatenate two standard ReLU networks  $\Phi_1$  (in blue) and  $\Phi_2$  (in red) and generate a special ReLU network that outputs  $\Phi_1 + \Phi_2$ .



Figure 29: Concatenation of m standard ReLU networks, each generating the same function T, and addition of a collation channel to output  $\Phi(x) = \sum_{i=1}^{m} a_i T^{\circ i}(x)$ .

*Proof.* We first generate  $T^{\circ m}(x)$  as discussed in the proof of Proposition 4 (i), displayed in blue and red in Figure 29. We add a collation channel and modify the input weights (displayed in green) of every *i*-th output collation neuron so that it produces  $a_i T^{\circ i}(x)$ . The source channel is not needed in this case. Nevertheless, we include it in case we need to add another function of x to  $\Phi$  later on.

**Proposition 6.** Let  $W \geq 2$ . For any  $T \in \mathcal{M}_{W_1,L}$  and  $g \in \mathcal{M}_{W_2,L}$ , then

$$\Phi_g(x) = \sum_{i=1}^m a_i \, g \circ T^{\circ i}(x) \in \hat{\mathcal{M}}_{W_1 + W_2 + 2, (m+1)L}.$$

*Proof.* The construction of  $\Phi_g$  is similar to the construction of  $\Phi$  in Proposition 5 with the extra step of adding m copies of g to  $T^{\circ m}$  as depicted in Figure 30.



Figure 30: Concatenation of m standard ReLU networks, each generating the same function  $g \circ T$ , and addition of a collation channel to output  $\Phi_g(x) = \sum_{i=1}^m a_i g \circ T^{\circ i}(x)$ .

# 18.2 A large class of functions with self-similarity

Consider target functions of the form

$$f(x) = \sum_{k \ge 1} a^{-k} g \circ T^{\circ k}(x), \qquad |a| > 1, \qquad g : [0,1] \to \mathbb{R}, \qquad T : [0,1] \to [0,1].$$

This is an example of a self-similar function. By Proposition 6, if

$$T \in \mathcal{M}_{W_1,L}, \qquad g \in \mathcal{M}_{W_2,L},$$

then

$$\Phi_m(x) = \sum_{k=1}^m a^{-k} g \circ T^{\circ k}(x) \in \hat{\mathcal{M}}_{W,(m+1)L}, \qquad W = W_1 + W_2 + 2.$$

We now define the (best) approximation error in approximating f by special ReLU networks with a fixed width W and depth (m + 1)L by

$$\varepsilon_{(m+1)L}(f)_{C[0,1]} := \inf_{\Phi \in \hat{\mathcal{M}}_{W,(m+1)L}} ||f - \Phi||_{C[0,1]}.$$

Assuming  $||g||_{C[0,1]} = 1$  (this can easily be relaxed), then we will have

$$\begin{aligned} \varepsilon_{(m+1)L}(f)_{C[0,1]} &\leq ||f - \Phi_m||_{C[0,1]} \\ &= ||\sum_{k>m} a^{-k} g \circ T^{\circ k}||_{C[0,1]} \\ &\leq \sum_{k>m} |a|^{-k} ||g \circ T^{\circ k}||_{C[0,1]} \\ &= |a|^{-(m+1)} \left(1 + \sum_{k>m+1} |a|^{-k}\right) \leq C |a|^{-(m+1)}, \end{aligned}$$

where  $C < \infty$  is a constant, noting that  $\sum_{k\geq 0} |a|^{-k} = 1/(1-|a|^{-1})$ . We note that the second inequality is a simple consequence of triangle inequality. The above estimate implies that the target function f can be approximated by a ReLU network with exponential accuracy (i.e. the approximation error decays exponentially). Moreover, the deeper the network, that is, the more terms m in the sum, the higher the exponential rate of decay.

We now consider a special class of such self-similar functions, known as Takagi functions, and present two examples of Takagi functions showing that they form a large class of functions that can be well approximated by deep ReLU networks.

Takagi functions. Consider continuous functions of the form

$$f(x) = \sum_{k \ge 1} a_k H^{\circ k}(x),$$
(35)

where  $\{a_k\}_{k\geq 1}$  is an absolutely summable sequence of real numbers, and  $H \in \mathcal{M}_{2,1}$  is the hat function. Note that this is a self-similar function with g(x) = x and T(x) = H(x). By Proposition 4(i), we will have  $H^{\circ k} \in \mathcal{M}_{2,k}$ . Following Proposition 5, a special ReLU network approximating any Takagi function of the form (35) can be easily constructed; see Figure 31.



Figure 31: A special ReLU network with width W = 4 and L hidden layers that outputs  $\Phi_L(x) = \sum_{k=1}^L a_k H^{\circ k}(x)$ , approximating a Takagi function  $\Phi(x) = \sum_{k\geq 1} a_k H^{\circ k}(x)$ . The deeper the network, the higher the rate of decay in approximation error.

The depth L of such special ReLU network, which is equal to the number of terms in the approximant  $\Phi_L(x) = \sum_{k=1}^L a_k H^{\circ k}(x)$ , will determine the exponential decay rate of approximation error. We will give two specific examples.

**Example 1.** Consider a Takagi function (35) with  $a_k = 2^{-k}$ ,

$$f_1(x) = \sum_{k \ge 1} 2^{-k} H^{\circ k}(x)$$

Let  $\Phi_L(x) = \sum_{k=1}^L 2^{-k} H^{\circ k}(x)$  be the ReLU network approximant of  $f_1$ , with depth  $L \ge 1$ . Then, by the error estimate obtained above, we will have

$$\varepsilon_L(f_1)_{C[0,1]} \le C \, 2^{-L}$$

This implies that theoretically  $f_1$  can be approximated with exponential accuracy by ReLU networks with roughly  $W^2L \sim L$  parameters (noting that W = 4 in the above construction). This reveals an amazing power of deep networks, as  $f_1$  is nowhere differentiable and hence has very little smoothness in classical sense. In fact, all traditional methods of approximation would fail to approximate  $f_1$  well. The key to the success of deep network is however the self-similarity feature of  $f_1$ ; it possesses a simple compositional/recursive pattern that can be exploited by deep networks.

**Example 2.** Consider a Takagi function (35) with  $a_k = 4^{-k}$ ,

$$f_2(x) = \sum_{k \ge 1} 4^{-k} H^{\circ k}(x) = x(1-x).$$

Unlike  $f_1$  which has very little smoothness,  $f_2$  is an analytic (very smooth) function. This representation can be used to show that the quadratic function  $x^2$  can be approximated with exponential accuracy by ReLU networks (as we did earlier). We can then show that all monomials  $x^3, x^4, \ldots$  can also be approximated with exponential accuracy by ReLU networks. Using the additive property of ReLU networks, discussed in Section 18.1, we can conclude that analytic functions (and Sobolev functions) can be approximated by ReLU networks with the same accuracy as their approximation by algebraic polynomials (a result that we obtained in previous chapter).

A concluding remark. The above two examples show that the approximation space of deep ReLU networks is quite large. It contains many functions, spanning from smooth/analytic functions to functions that are not smooth in any classical sense. This brings flexibility to deep ReLU networks: they well approximate functions with little classical smoothness and retain the good property of approximating smooth functions with comparable accuracy to other methods of approximation.

Acknowledgments. This survey is mainly based on two courses on "Mathematics of deep learning" that the author designed and gave at The University of New Mexico in Fall 2021 and at Uppsala University, Sweden, in Spring 2022. The author would like to thank all students and researchers who actively participated in class and colleagues and collaborators who contributed to the survey through interesting discussions. The author is also indebted to Gunilla Kreiss for the support she provided during a one-month visit to Uppsala University.

## References

- A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inform. Theory*, 39:930–945, 1993.
- [2] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In Müller KR. Montavon G., Orr G.B., editor, *Neural Networks: Tricks of the Trades*, pages 437–478. Springer, Berlin, 2012.
- [3] Alberto Bietti, Luca Venturi, and Joan Bruna. On the sample complexity of learning under geometric stability. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, 2021.
- [4] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. SIAM Rev., 60:223–311, 2018.
- [5] S. C. Brenner and L. R. Scott. The Mathematical Theory of Finite Element Methods. Springer, 2008.
- [6] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. Spectral Methods: Fundamentals in Single Domains. Springer, 2006.
- [7] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova. Nonlinear approximation and (deep) ReLU networks. (accepted/in press). *Constructive Approximation*, 2021.
- [8] R. A. DeVore. Nonlinear approximation. Acta Numer., 7:51–150, 1998.
- [9] R. A. DeVore and G. G. Lorentz. *Constructive Approximation*. Springer-Verlag, 1991.
- [10] W. F. Donoghue. Distributions and Fourier Transforms. Academic Press, Inc., New York, 1969.
- [11] W. E, C. Ma, S. Wojtowytsch, and L. Wu. Towards a mathematical understanding of neural network-based machine learning: What we know and what we don't. CSIAM Trans. Appl. Math., 1:561–615, 2020.
- [12] D. Elbrächter, D. Perekrestenko, P. Grohs, and H. Bölcskei. Deep neural network approximation theory. arxiv:1901.02220, 2021.
- [13] L. C. Evans. Partial Differential Equations, volume 19 of Graduate Studies in Mathematics. American Mathematical Society, Providence, Rhode Island, 1998.
- [14] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.
- [15] I. Gühring, G. Kutyniok, and P. Petersen. Error bounds for approximations with deep ReLU neural networks in W<sup>s,p</sup> norms. Analysis and Applications, 18:803–859, 2020.

- [16] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. Ann. Math. Statist., 23:462–466, 1952.
- [17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980v9, 2017.
- [18] V. Maiorov and R. Meir. On the near optimality of the stochastic approximation of smooth functions by neural networks. Advances in Computational Mathematics, 13:79– 103, 2000.
- [19] Y. Makovoz. Random approximants and neural networks. Journal of Approximation Theory, 85:98–109, 1996.
- [20] H. Mhaskar and T. Poggio. Deep vs. shallow networks: an approximation theory perspective. Analysis and Applications, 14:829–848, 2016.
- [21] H. N. Mhaskar. Neural networks for optimal approximation of smooth and analytic functions. *Neural Computation*, 8:164–177, 1996.
- [22] P. Petersen and F. Voigtländer. Optimal approximation of piecewise smooth functions using deep ReLU neural networks. *Neural Networks*, 108:296–330, 2018.
- [23] A. Pinkus. Approximation theory of the MLP model in neural networks. Acta Numer., 8:143–195, 1999.
- [24] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Why and when can deepbut not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14:503–519, 2017.
- [25] M. J. D. Powell. Approximation Theory and Methods. Cambridge University Press, 1996.
- [26] H. Robbins and S. Monro. A stochastic approximation method. Ann. Math. Statist., 22:400–407, 1951.
- [27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by backpropagating errors. *Nature*, 323:533–536, 1986.
- [28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [29] M. Telgarsky. Representation benefits of deep feedforward networks. arxiv:1509.08101, 2015.
- [30] L. N. Trefethen. Approximation Theory and Approximation Practice, Extended Edition. SIAM, 2019.
- [31] D. Yarotsky. Error bounds for approximations with deep ReLU networks. Neural Networks, 94:103-114, 2017.