CS 839: Theoretical Foundations of Deep Learning		Spring 2023
	Lecture 3 Approximation I	
Instructor: Yingyu Liang	*Date:*	*Scriber: Zhenmei Shi*

1 Overview

In the previous lecture, we decomposed the risk into three parts: approximation, estimation/generalization and optimization. In this lecture we will focus on approximation power of neural networks.

2 Problem Setup

2.1 Two-Layer Neural Network

Let $\mathcal{X}$ denote the input space, and $\mathcal{Y}$ the label space. $\mathcal{X} \subseteq \mathbb{R}^d$, and $\mathcal{Y} = \{-1, +1\}$ for binary classification. The hypothesis/model class $\mathcal{H}$ is 2-layer neural networks which can also be thought of as a 1-hidden-layer neural network with functions $h : \mathcal{X} \mapsto \mathcal{Y}$,

$$h(\mathbf{x}) = \sum_{i=1}^{m} a_i \sigma(\langle \mathbf{w}_i, \mathbf{x} \rangle + b_i) = \mathbf{a}\sigma(\mathbf{W}\mathbf{x} + \boldsymbol{b}), \tag{1}$$

where $\mathbf{w}_i \in \mathbb{R}^d, a_i, b_i \in \mathbb{R}, \mathbf{W} = [\mathbf{w}_1^\top, \ldots, \mathbf{w}_m^\top]^\top, \boldsymbol{b} \in \mathbb{R}^m, \mathbf{a} \in \mathbb{R}^{1\times m}$, and σ is an activation function. Below is a picture to better understand a neural network's set of parameters applied to the input.

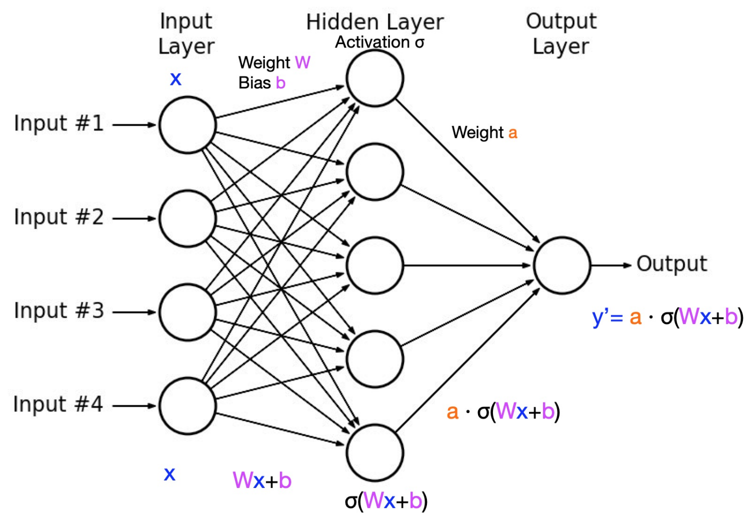

Figure 1: Example of a 2-layer neural network given parameters

Example 1 (Activation Functions)**.** Some common activation functions are:

- ReLU: $\sigma(z) = \max(0, z)$.
- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$.

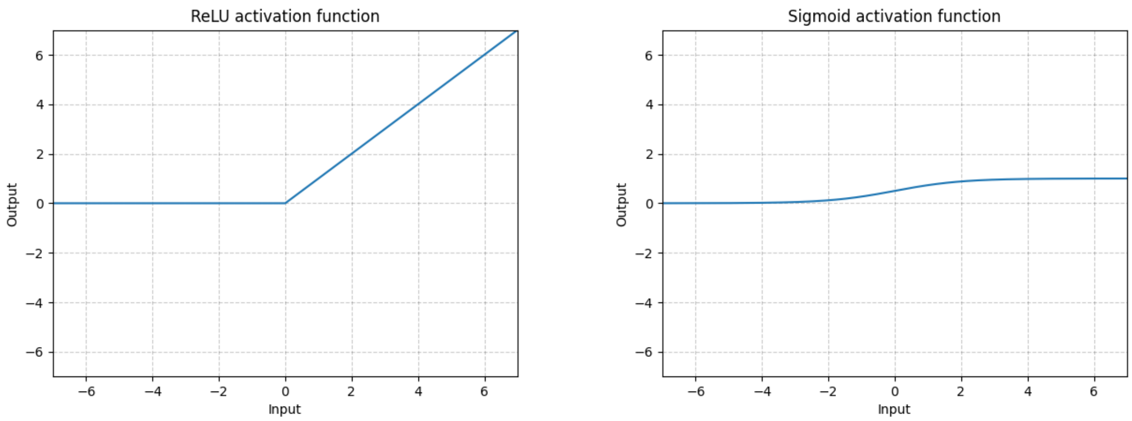

Figure 2: ReLU and Sigmoid

2.2 L-Layer Neural Network

Now to define a Deep Neural Network (DNN) or multiple-layer neural network which has L layers:

$$
\begin{aligned}
h_0(\mathbf{x}) &= \mathbf{x} \\
h_j(\mathbf{x}) &= \sigma_j(\mathbf{W}_j h_{j-1}(\mathbf{x}) + \boldsymbol{b}_j) \\
h(\mathbf{x}) &= h_L(\mathbf{x}),
\end{aligned}
$$

where $\mathbf{x} \in \mathbb{R}^d, \mathbf{W}_1 \in \mathbb{R}^{m_1 \times d}, \mathbf{W}_i \in \mathbb{R}^{m_i \times m_{i-1}}$ for $i = 2, \ldots, L-1, \mathbf{W}_L \in \mathbb{R}^{k \times m_{L-1}}$ and $\boldsymbol{b}_i \in \mathbb{R}^{m_i}$. The function h is a mapping of $h : \mathbb{R}^d \mapsto \mathbb{R}^k$. Normally, the same activation function is used in all layers except the last one.

For multi-class classification with $y \in \{1, \ldots, k\}$, k is the number of classes in the problem, the last layer uses the softmax function to generate a probabilistic vector, and the output $h(\mathbf{x})$ is regarded as the probabilities over the k classes.

For binary classification with $y \in \{-1, +1\}$, typically $k = 1$, the last layer uses the sigmoid function to generate a value in $(0, 1)$, and the output $h(\mathbf{x})$ is regarded as the probability of the label $+1$.

For regression with $y \in \mathbb{R}$, typically $k = 1$, the last layer uses no activation function (or equivalently, uses the identity function as the activation), and the output $h(\mathbf{x})$ is regarded as an estimation of y.

Example 2 (Loss Functions)**.** Some typical examples of loss functions are:

- Cross-entropy loss for multi-class classification: $l(\widehat{y}, y) = -\log(\widehat{y}_y) = -\sum_{i=1}^{k} \bar{y}_i \log(\widehat{y}_i)$, where $y \in \{1, \ldots, k\}$, $\bar{y}$ is the one-hot encoding of y, $\widehat{y} \in \mathbb{R}^k$ is a probabilistic vector (i.e., the softmax output of the network).
- Logistic loss for binary classification: $l(\widehat{y}, y) = \log(1 + \exp(-\widehat{y}y))$, where $y \in \{-1, +1\}$, and $\widehat{y} \in \mathbb{R}$ is the output of the network (usually the last layer uses no activation function).
- Square loss for regression: $l(\widehat{y}, y) = (\widehat{y} - y)^2$, where $y \in \mathbb{R}$, and $\widehat{y} \in \mathbb{R}$ is the output of the network (usually the last layer uses no activation function).

For $L = 2$,

$$h(\mathbf{x}) = \sigma_2(\mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2),$$

is equivalent to (1) when σ_2 is the identity function, $\mathbf{W}_2 = \mathbf{a}$, and $\boldsymbol{b}_2 = 0$.

3 Approximation

Recall that the risk is,

$$\mathcal{R}(h) = \mathbb{E}_{(\mathbf{x},y)}[l(h(\mathbf{x}), y)].$$

Suppose we have a hypothesis class $\mathcal{H}$ and ground truth function class $\mathcal{G}$ and $l(\cdot, y)$ is 1-Lipschitz, $|l(\widehat{y}, y) - l(y', y)| \leqslant |\widehat{y} - y'|$. Let $h \in \mathcal{H}, g \in \mathcal{G}$, we define the distance d between two functions and l_1 norm as,

$$\begin{aligned} d(h, g) &:= \mathbb{E}_{(x,y)}[l(h(\mathbf{x}), y)] - \mathbb{E}_{(x,y)}[l(g(\mathbf{x}), y)] \\ &\leqslant \mathbb{E}_{(x,y)}[|h(\mathbf{x}) - g(\mathbf{x})|] := \|h - g\|_1. \end{aligned}$$

Their difference is also called the approximation error of function class $\mathcal{H}$ on $\mathcal{G}$. We use l_1 norm because it is convenient for theoretical analysis; other definitions are possible.

Our goal is to bound the following term:

$$\sup_{g \in \mathcal{G}} \inf_{h \in \mathcal{H}} \|h - g\|_1$$

to measure the approximation error of $\mathcal{H}$ on $\mathcal{G}$.

4 Neural Networks Approximate Lipschitz Functions

4.1 1-dimension Case

Consider Lipschitz function family (non-parametric function family) $g : [0, 1) \mapsto \mathbb{R}$,

$$|g(x_1) - g(x_2)| \leqslant \rho|x_1 - x_2|, \forall x_1, x_2 \in [0, 1).$$

We have the following theorem.

Theorem 3. Suppose $g : [0, 1) \mapsto \mathbb{R}$ is ρ-Lipschitz. For any $\epsilon > 0$, there exists a 2-layer-neural-network $h(x)$ with $2m$ neurons where $m := \lceil \frac{\rho}{\epsilon} \rceil$, s.t. $\forall x \in [0, 1), |h(x) - g(x)| \leqslant \epsilon$.

Proof of Theorem 3. The proof idea is doing partition. The target function is almost constant in each small interval. Construct a function by assigning left hand side value of the target function in the interval. Let $b_i = \frac{(i-1)}{m}, i = 1, \ldots, m$. Consider

$$h(x) = \sum_{j=1}^{m} g(b_j)\mathbb{I}[x \in [b_j, b_{j+1})], \tag{2}$$

we need to replace the indicator function by neurons. Consider step activation function (threshold function),

$$\sigma(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geqslant 0 \end{cases}$$

we have,

$$\sigma(z - b_j) - \sigma(z - b_{j+1}) = \mathbb{I}[z \in [b_j, b_{j+1})].$$

Thus $h(x)$ in (2) is a 2-layer-neural-network with $2m$ neurons. The error is bounded by,

$$|h(x) - g(x)| = |g(b_j) - g(x)| \leqslant \rho|b_j - x| \leqslant \rho \cdot \frac{\epsilon}{\rho} = \epsilon.$$

□

We make use of threshold functions that use several 1-dimensional flat regions to approximate arbitrary ρ-Lipschitz unitary functions.

4.2 High-dimension Case

In this section, we consider multivariate approximation, and similarly make use of higher dimensional bumps or flat regions to approximate continuous multivariate functions. In particular, we will use 3-layer-neural-network with ReLU activation function to approximate high-dimension Lipschitz function family.

Theorem 4. Suppose g is a continuous function and $\epsilon > 0$. Additionally, assume that $\forall \delta > 0$, $\forall \mathbf{x}_1, \mathbf{x}_2 \in [0, 1]^d$, if $\|\mathbf{x}_1 - \mathbf{x}_2\|_\infty \leqslant \delta$, we have $|g(\mathbf{x}_1) - g(\mathbf{x}_2)| \leqslant \epsilon$. Then, there exists a 3-layer-neural-network $h(\mathbf{x})$ with $\Omega(\frac{1}{\epsilon^d})$ ReLU neurons s.t. $\|h(\mathbf{x}) - g(\mathbf{x})\|_1 = \int_{[0,1]^d} |h(\mathbf{x}) - g(\mathbf{x})|dx \leqslant 2\epsilon$.

We can use similarly ideas in Theorem 3 to construct an intermediate function (piece-wise constant function).

Lemma 5. Let g, δ, ϵ be defined as in Theorem (4). For any partition P of $[0, 1]^d$ into hyper rectangles, $P = \{R_i\}_{i=1}^N$ with side length $\leqslant \delta$, there exists a piece-wise constant function $h(\mathbf{x}) = \sum_{i=1}^N \alpha_i \mathbb{I}[\mathbf{x} \in R_i]$ s.t. $\forall \mathbf{x} \in [0, 1]^d, |h(\mathbf{x}) - g(\mathbf{x})| \leqslant \epsilon$.

Proof of Lemma 5. Let α_i be the value of g on any point in the region R_i. □

With this lemma, we are equipped to prove Theorem 1. Our strategy will be to apply Lemma 5 to claim that a piece-wise constant function approximates a continuous function. Then, we will prove that a 2-layer ReLU network can approximate an indicator function, representing a bump in high dimensional space. Finally, we show by construction a 3-layer neural network that uses the first 2 layers to approximate selector functions, and the final layer to apply corresponding constants α_i to the correct indicator approximation.

Proof of Theorem 4. Divide the domain $[0,1]^d$ into sufficiently small hyper rectangles of the form $R_i :=:= \bigtimes_{j=1}^{d}[a_{ij}, b_{ij})$. If we can approximate $\mathbb{I}[x_j \in [a_{ij}, b_{ij})]$ by 1 layer of the neural network with ReLU activation function and approximate $\mathbb{I}[\mathbf{x} \in R_i]$ by 2 layers of the neural network with ReLU activation function then we have three layers of the neural network as

$$f(\mathbf{x}) = \sum_i \alpha_i \widetilde{\mathbb{I}}[\mathbf{x} \in R_i],$$

where $\widetilde{\mathbb{I}}$ is approximate by neurons with ReLU activation function. Then we have,

$$\mathbb{E}_{\mathbf{x}}|f(\mathbf{x}) - g(\mathbf{x})| \leqslant \mathbb{E}_{\mathbf{x}}|f(\mathbf{x}) - h(\mathbf{x})| + \mathbb{E}_{\mathbf{x}}|h(\mathbf{x}) - g(\mathbf{x})| \leqslant \epsilon + \epsilon = 2\epsilon. \tag{3}$$

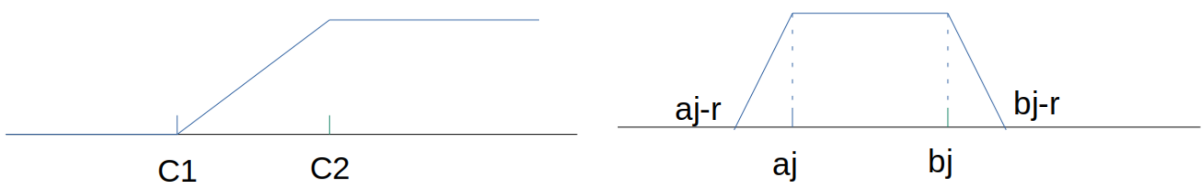

Figure 3: Linear combination of ReLU activations

Now, we will construct our two-layer networks to satisfy the above property. First, we are going to approximate $\mathbb{I}[x \in [a_{ij}, b_{ij})]$ by neurons with ReLU activation function. Consider

$$\frac{\sigma(z - c_1) - \sigma(z - c_2)}{c_2 - c_1}, \tag{4}$$

for $c_2 > c_1$; see left figure in Figure 3. Two equation (4) can form an approximation function of indicator function,

$$\begin{aligned} f_{i,j,\gamma}(\mathbf{x}) =& \frac{1}{\gamma}[\sigma(x_j - a_{ij}) - \sigma(x_j - (a_{ij} - \gamma))] - \frac{1}{\gamma}[\sigma(x_j - (b_{ij} + \gamma)) - \sigma(x_j - b_{ij})] & (5) \\ :=& \widetilde{\mathbb{I}}[x_j \in [a_{ij}, b_{ij})], & (6) \end{aligned}$$

see right figure in Figure 3. Note that the γ can be chosen to be sufficiently small so that we have as small approximation error of $\widetilde{\mathbb{I}}$ on $\mathbb{I}$ as desired. $f_{i,j,\gamma}(\mathbf{x})$ can only approximate one dimension. To approximate d dimension, we can compose these 1-layer single-coordinate selector functions to form a 2-layer selector function that selects rectangles in all coordinates:

$$\widetilde{\mathbb{I}}[\mathbf{x} \in R_i] := f_{i,\gamma}(\mathbf{x}) = \sigma(\sum_{j=1}^{d} f_{i,j,\gamma}(\mathbf{x}) - (d-1)).$$

Then it satisfies:

$$f_{i,\gamma}(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in R_i \\ [0, 1), & \text{o.w.} \\ 0, & \mathbf{x} \notin \times_{j=1}^{d} [a_{ij} - \gamma, b_{ij} + \gamma]. \end{cases}$$

Finally, we pick a sufficiently small γ. By equation (3), we conclude that 3-layer-neural-networks with ReLU activation function can approximate high-dimension Lipschitz function family with small approximation error. □

Remark 6 (Curse of dimension)**.** Note that this theorem requires a number of ReLU units exponential in the dimensionality. A neural network satisfying this requirement is likely to be impractical in high-dimensional domains.