Compare Shallow and Deep Neural Network for Function Approximation

Zhan Shu, Yuan Dou, Yingqi Ma, Qingyang Gu

Reference Paper

Liang, Shiyu and R. Srikant. “Why Deep Neural Networks for Function Approximation?” International Conference on Learning Representations (2017) [1].

Shallow networks need exponentially more neurons than deep networks to approximate a function.

Goal

Focus on bounds on the size of deep network for function approximation to guarantee an ε-approximation.

Given a function f, our goal is to find out whether a deep neural network ˜f of upper bounds on the depth L(ε) and size N(ε) exists such that it solves $\min_{\tilde{f} \in \mathcal{F}(N,L)} \|f - \tilde{f}\| \leq \varepsilon.$

Approximation of univariate functions.

Then extend these results to certain classes of multivariate functions.

Theorem 1 of Univariate Function

Theorem 1. *For function* $f(x) = x^2, x \in [0,1]$*, there exists a multilayer neural network* $\tilde{f}(x)$ *with* $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$ *layers,* $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$ *binary step units and* $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$ *rectifier linear units such that* $|f(x) - \tilde{f}(x)| \leq \varepsilon, \quad \forall x \in [0,1]$.

Theorem of approximating a quadratic function.

Proof

For any x ∈ [0, 1], we first use the multilayer neural network to approximate x by its finite binary expansion $\sum_{i=0}^{n} \frac{x_i}{2^i}$

Next, implement the function $\tilde{f}(x) = f\left(\sum_{i=0}^{n} \frac{x_i}{2^i}\right)$ by a two-layer neural network

Since f(x) = x^2, we can further get: $\tilde{f}(x) = \left(\sum_{i=0}^{n} \frac{x_i}{2^i}\right)^2 = \sum_{i=0}^{n}\left[x_i \cdot \left(\frac{1}{2^i}\sum_{j=0}^{n}\frac{x_j}{2^j}\right)\right] = \sum_{i=0}^{n} \max\left(0, 2(x_i - 1) + \frac{1}{2^i}\sum_{j=0}^{n}\frac{x_j}{2^j}\right).$

The approximate error function can then be considered as:

$$|f(x) - \tilde{f}(x)| = \left| x^2 - \left(\sum_{i=0}^{n} \frac{x_i}{2^i}\right)^2 \right| \leq 2\left| x - \sum_{i=0}^{n} \frac{x_i}{2^i} \right| = 2\left| \sum_{i=n+1}^{\infty} \frac{x_i}{2^i} \right| \leq \frac{1}{2^{n-1}}.$$

In order to achieve ε-approximation error, $n = \left\lceil \log_2 \frac{1}{\varepsilon} \right\rceil + 1$

Since we used O(n + p) layers with O(n) binary step units and O(pn) rectifier linear units in total, the deep neural network has O(log 1/ε) layers, O(log 1/ε) binary step units and O(log 1/ε) rectifier linear units.

Theorem 2 of Univariate Function

Theorem 2. *For polynomials* $f(x) = \sum_{i=0}^{p} a_i x^i$, $x \in [0,1]$ *and* $\sum_{i=1}^{p} |a_i| \leq 1$, *there exists a multilayer neural network* $\tilde{f}(x)$ *with* $\mathcal{O}\left(p + \log \frac{p}{\varepsilon}\right)$ *layers,* $\mathcal{O}\left(\log \frac{p}{\varepsilon}\right)$ *binary step units and* $\mathcal{O}\left(p \log \frac{p}{\varepsilon}\right)$ *rectifier linear units such that* $|f(x) - \tilde{f}(x)| \leq \varepsilon, \forall x \in [0,1]$.

Theorem of the network for approximating general polynomials.

Proof

First use the deep structure to find the n-bit binary expansion $\sum_{i=0}^{n} a_i x^i$ of x.

Then construct a multilayer network to approximate polynomials gi(x) = x^i , i=1,...,p, rewrite it as $g_{m+1}(\sum_{i=0}^{n} \frac{x_i}{2^n})$

$$g_{m+1}\left(\sum_{i=0}^{n}\frac{x_i}{2^i}\right)=\sum_{j=0}^{n}\left[x_j\cdot\frac{1}{2^j}g_m\left(\sum_{i=0}^{n}\frac{x_i}{2^i}\right)\right]=\sum_{j=0}^{n}\max\left[2(x_j-1)+\frac{1}{2^j}g_m\left(\sum_{i=0}^{n}\frac{x_i}{2^i}\right),0\right]$$

The expansion defines iterations between the outputs of neighbor layers.

Proof

Define the output of the multilayer neural network as: $\tilde{f}(x) = \sum_{i=0}^{p} a_i g_i \left(\sum_{j=0}^{n} \frac{x_j}{2^j} \right)$

For this deep network, the approximation error is:

$$|f(x) - \tilde{f}(x)| = \left| \sum_{i=0}^{p} a_i g_i \left(\sum_{j=0}^{n} \frac{x_j}{2^j} \right) - \sum_{i=0}^{p} a_i x^i \right| \leq \sum_{i=0}^{p} \left[|a_i| \cdot \left| g_i \left(\sum_{j=0}^{n} \frac{x_j}{2^j} \right) - x^i \right| \right] \leq \frac{p}{2^{n-1}}$$

Choose n = log (p/ε) + 1.

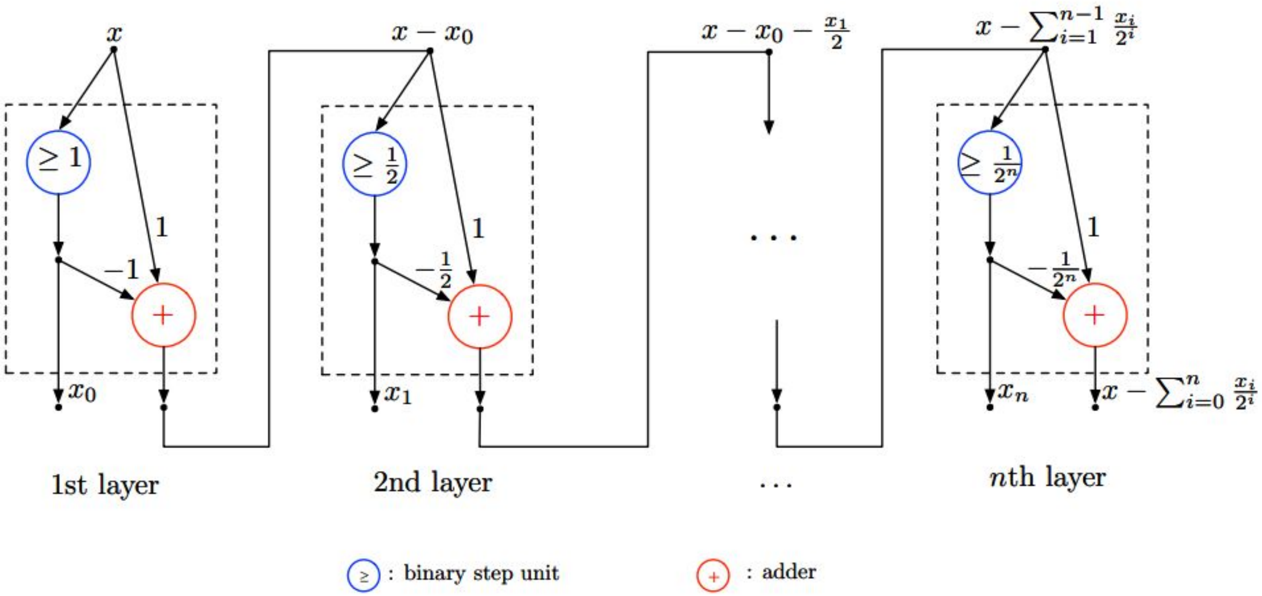
This deep neural network has O (p + log p/ε) layers, O (log p/ε) binary step units and O (p*log p/ε) rectifier linear units.

Theorem 8 of Multivariate Function

Theorem 8. *Let* $W = \{\boldsymbol{w} \in \mathbb{R}^d : \|\boldsymbol{w}\|_1 = 1\}$. *For* $f(\boldsymbol{x}) = \prod_{i=1}^{p} (\boldsymbol{w}_i^T \boldsymbol{x})$, $\boldsymbol{x} \in [0,1]^d$ *and* $\boldsymbol{w}_i \in W$, $i = 1, ..., p$, *there exists a deep neural network* $\tilde{f}(\boldsymbol{x})$ *with* $\mathcal{O}\left(p + \log \frac{pd}{\varepsilon}\right)$ *layers and* $\mathcal{O}\left(\log \frac{pd}{\varepsilon}\right)$ *binary step units and* $\mathcal{O}\left(pd \log \frac{pd}{\varepsilon}\right)$ *rectifier linear units such that* $|f(\boldsymbol{x}) - \tilde{f}(\boldsymbol{x})| \leq \varepsilon$, $\forall \boldsymbol{x} \in [0,1]^d$.

Theorem 8 is for a product of multivariate linear functions.

N bit binary expansion



Universal Approximation Theorem

- According to the Universal Approximation Theorem, Neural Networks has a kind of universality i.e. there is a network that can approximate f(x) no matter what kind of function f(x) is. [3]
- This Universal Approximation Theorem holds even if the neural network has just a single layer and an input and the output layer.
- **Universal approximation theorem** *(L1 distance, ReLU activation, arbitrary depth, minimal width).* For any Bochner–Lebesgue p-integrable function $f : \mathbb{R}^n \to \mathbb{R}^m$ and any $\epsilon > 0$, there exists a fully-connected ReLU network F of width exactly $d_m = \max\{n+1, m\}$, satisfying

$$\int_{\mathbb{R}^n} \|f(x) - F(x)\|^p \mathrm{d}x < \epsilon.$$

Moreover, there exists a function $f \in L^p(\mathbb{R}^n, \mathbb{R}^m)$ and some $\epsilon > 0$, for which there is no fully-connected ReLU network of width less than $d_m = \max\{n+1, m\}$ satisfying the above approximation bound.

Quadratic function

Data:

- 1000 data points, x∈[0,1]
- Binary expansion to $x = \sum_{i=0}^{\infty} \frac{x_i}{2^i}$, where $x_i \in \{0, 1\}$

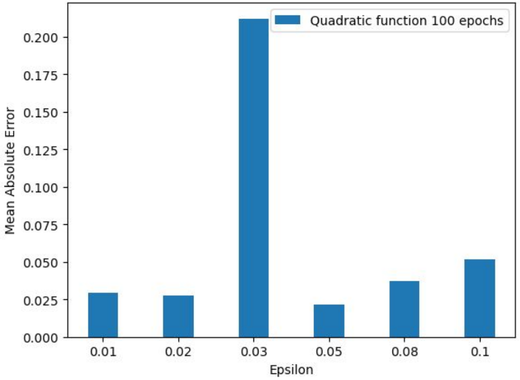
Neural network:

1 input layer of binary step unit, 1 hidden layer of ReLU, 1 output layer

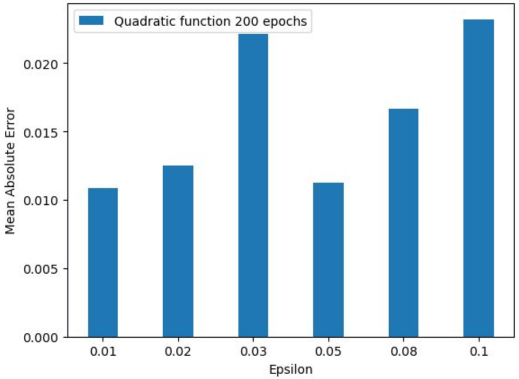
Choose $n = \left\lceil \log_2 \frac{1}{\varepsilon} \right\rceil + 1$

Therefore, the model has $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$ layers, $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$ binary step units and $\mathcal{O}\left(\log \frac{1}{\varepsilon}\right)$ ReLU

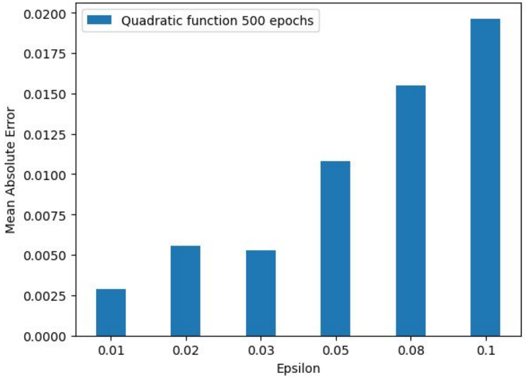
Result for different epsilon and epoches



100 epoches



200 epochs



500 epoches

Polynomial Function Approximation

Data:

- $f(x) = \sum_{i=0}^{p} a_i x^i,\ x \in [0,1]\ and\ \sum_{i=1}^{p} |a_i| \leq 1$
- 1000 data points, x∈[0,1]
- p polynomial coefficients, randomly generated and normalized
- Binary expansion to $x = \sum_{i=0}^{\infty} \frac{x_i}{2^i}$, where $x_i \in \{0, 1\}$

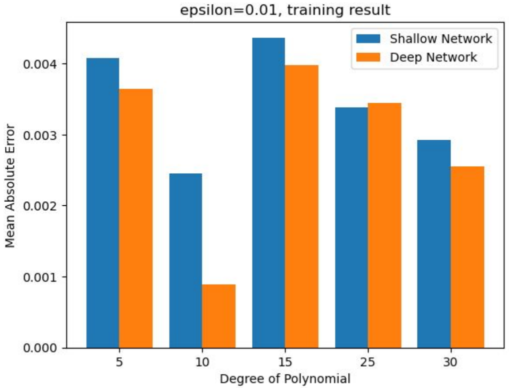
Total number of rectifier linear units = $p \log \frac{p}{\varepsilon}$

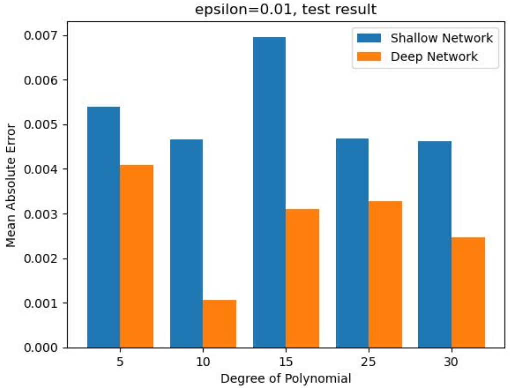
Number of rectifier linear units in each layer = $\log \frac{p}{\varepsilon}$

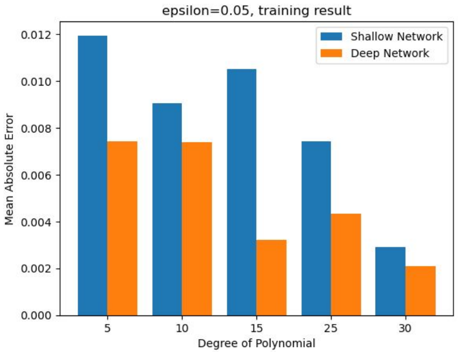
Shallow Neural Network: 1 input layer, 1 hidden layer, 1 output layer

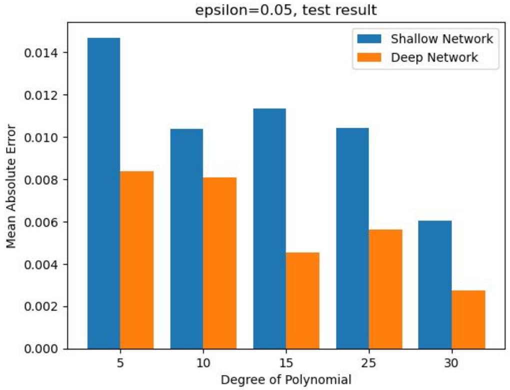
Deep Neural Network: 1 input layer, p hidden layers, 1 output layer

Approximation of Polynomial Functions









Approximation of Multivariate Functions

Data:

- $f(\boldsymbol{x}) = \prod_{i=1}^{p} (\boldsymbol{w}_i^T \boldsymbol{x}),\ \boldsymbol{x} \in [0,1]^d$ *and* $\boldsymbol{w}_i \in W,\ i = 1, \ldots, p,$
- 1000 data points for each pair of (p, d)

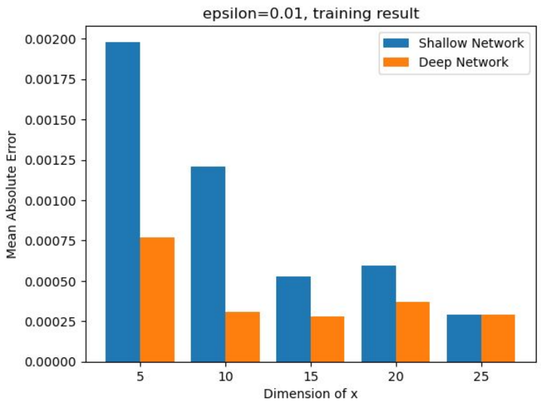
Total Number of rectifier linear units = $pd \log \frac{pd}{\varepsilon}$

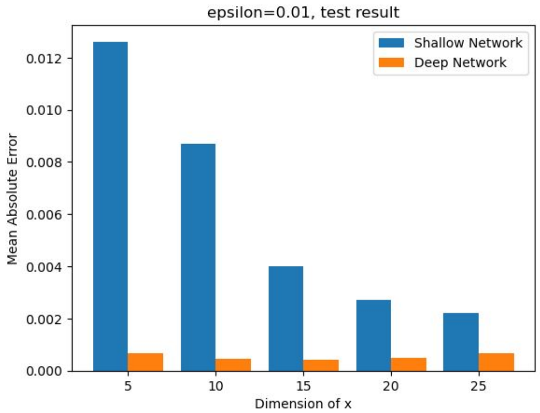
Number of rectifier linear unit in each layer = $d \log \frac{pd}{\varepsilon}$

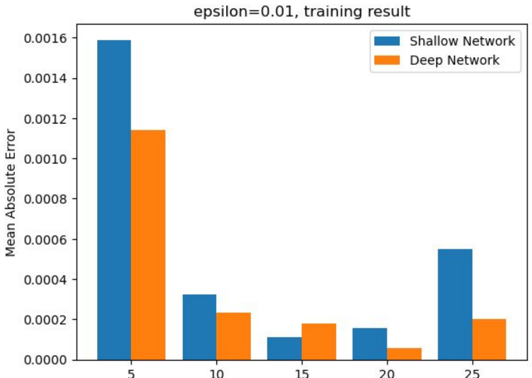
Shallow Neural Network: 1 input layer, 1 hidden layer, 1 output layer

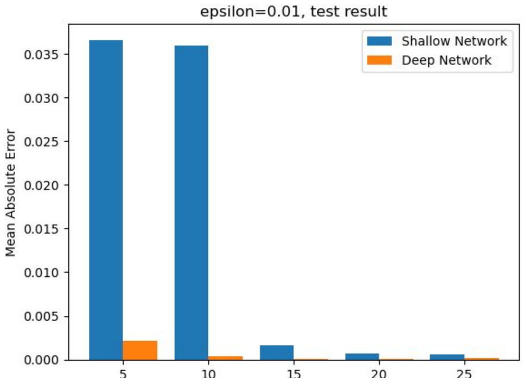
Deep Neural Network: 1 input layer, p hidden layers, 1 output layer

Approximation of Multivariate Functions









Reference

[1] WHY DEEP NEURAL NETWORKS FOR FUNCTION APPROXIMATION?, Liang and Srikant, 2017

[2] https://en.wikipedia.org/wiki/Universal_approximation_theorem

[3] http://neuralnetworksanddeeplearning.com/chap4.html