Approximation Theory

Maria K. Cameron

September 19, 2025

Contents

1 Introduction

Currently, we observe a revolution in scientific computing associated with a qualitatively new level of artificial intelligence, machine learning, and data-driven methods. Function approximation is at the very ground level of these techniques.

This chapter will discuss classical results on function approximation with polynomials, including the Bernstein polynomials and Chebyshev interpolation, and the approximation

of functions by neural networks.

1.1 The Weierstrass Approximation Theorem

The possibility of approximating a continuous function on a closed interval in a uniform norm was established by Karl Weierstrass in 1885:

Theorem 1. *Suppose f is a continuous real-valued function defined on the real interval $[a, b]$. For every $\epsilon > 0$, there exists a polynomial p such that*

$$\max_{x \in [a,b]} \|f(x) - p(x)\| < \epsilon. \tag{1}$$

1.2 The Bernstein polynomials

References for this subsection:

- [Farouki(2012)]: R. Faruoki, The Bernstein polynomial basis: A centennial retrospective, Computer Aided Geometric Design, 29/6, August 2012, pp. 379–419
- Slides by Rida T. Farouki (UC Davis)
- Slides by Richard V. Kadison (U Penn)

A simple and constructive proof of this interval was given by Sergei Bernstein in 1912. He introduced what we call now the *Bernstein polynomials*.

Consider the identity

$$1 = (x + (1 - x))^n = \sum_{k=0}^{n} \binom{n}{k} x^k (1 - x)^{n-k}. \tag{2}$$

The polynomials in the right-hand side of (2),

$$p_{n,k}(x) := \binom{n}{k} x^k (1 - x)^{n-k}, \tag{3}$$

are called the *Bernstein basis functions*. Equation (2) shows that they form a partition of unity, i.e, the sum of all Bernstein basis functions with any fixed $n \in \mathbb{N}$ is identically equal

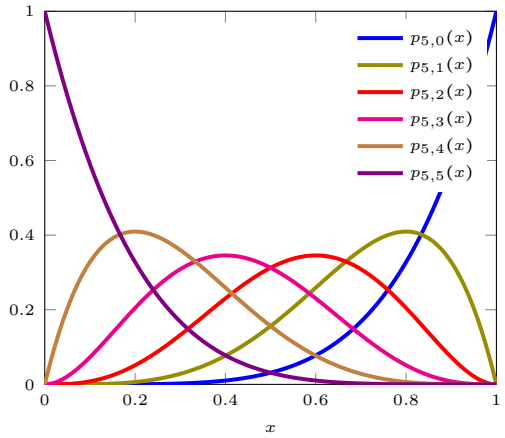

Figure 1: Bernstein basis functions $p_{n,k}$ for $n = 5$, $k = 0, 1, 2, 3, 4, 5$.

to one. It is also clear that these polynomials are nonnegative on $[0, 1]$. The Bernstein basis functions for $n = 5$ are plotted in Fig. 1 Several useful identities involving Bernstein basis functions are obtained using the differentiation trick. Consider the identity

$$x\frac{d}{dy}(y + (1-x))^n\Big|_{y=x} = nx. \tag{4}$$

On the other hand, using the left-hand side of (4) we get

$$nx = x\frac{d}{dy}(y + (1-x))^n\Big|_{y=x} = x\sum_{k=1}^{n}\binom{n}{k}kx^{k-1}(1-x)^{n-k}. \tag{5}$$

Hence, we get the following identity:

$$\sum_{k=0}^{n}\binom{n}{k}kx^k(1-x)^{n-k} = nx. \tag{6}$$

By a similar trick, we can derive that

$$\sum_{k=0}^{n}\binom{n}{k}k^2x^k(1-x)^{n-k} = n(n-1)x^2 + nx. \tag{7}$$

Equations (2), (4) and (6) imply the following identity that we will need to prove the Bernstein approximation theorem below:

$$\sum_{k=0}^{n}\binom{n}{k}\left(\frac{k}{n} - x\right)^2 x^k(1-x)^{n-k} = \frac{x(1-x)}{n}. \tag{8}$$

Definition 1. *With f a real-valued function defined and bounded on the interval [0,1], let $B_n(f)$ be the polynomial on $[0,1]$ that assigns to x the value*

$$B_n(x) := \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k} f\left(\frac{k}{n}\right), \tag{9}$$

$B_n(f)$ is called the nth **Bernstein polynomial** *for f.*

Using identities (2), (6), and (7) one can check that

$$B_n(1) = 1, \tag{10}$$

$$B_n(x) = x, \tag{11}$$

$$B_n(x^2) = \left(1 - \frac{1}{n}\right) x^2 + \frac{x}{n}. \tag{12}$$

Theorem 2. (Bernstein, 1912) *Let f be a real-valued function defined and bounded by M on the interval $[0,1]$.*

1. *For each point x of continuity of f, $B_n(f)(x) \to f(x)$ as $n \to \infty$.*
2. *If f is continuous on $[0,1]$, the Bernstein polynomial $B_n(f)$ tends uniformly to f as $n \to \infty$.*
3. *With x a point of differentiability of f, $B_n'(f)(x) \to f'(x)$ as $n \to \infty$.*
4. *If f is continuously differentiable on $[0,1]$, then $B_n'(f)$ tends to f' uniformly as $n \to \infty$.*

See R. Kadison's slides for the proof.

Thus, the Bernstein polynomial is a constructive tool for approximating any continuous function on a compact interval in $\mathbb{R}$ in the uniform norm. However, the error of this approximation decays as $O(n^{-1})$ even for smooth functions, e.g. $B_n(x^2) = x^2(1-n^{-1}) + xn^{-1}$. In the next section, we review the Chebyshev interpolation and its application to solving boundary-value problems. You will see that the error decays faster than any power of n, the number of interpolation (or collocation) points, on smooth functions.

2 Chebyshev interpolation and Chebyshev spectral methods

2.1 Lagrangian interpolation

Suppose a continuous function $f(x)$ defined on the interval $[a, b]$ is given at a finite number of points x_j, $j = 0, 1, 2, \ldots, n$, a total of $n + 1$ points. We will denote $f(x_j)$ by f_j. The task of interpolation is to find a polynomial of degree at most n passing through all these points. Lagrange's approach to this task is to construct $n+1$ polynomials $L_j(x)$ of degree n such that each $L_j(x)$ is 1 at x_j and zero at all other x_k's. Then the linear combination

$$P(x) = f_0 L_0(x) + f_1 L_1(x) + \ldots + f_n L_n(x)$$

is a polynomial of degree at most n, and $P(x_j) = f_j$ for all $j = 0, 1, \ldots, n$. The polynomial $P(x)$ written in the form above is called the Lagrange interpolation polynomial. The polynomials $L_j(x)$ are easily constructed. The error of interpolation can also be found. These results are summarized in the following theorem.

Theorem 3. *Given a function f that is defined at $n + 1$ points $x_0 < x_1 < \ldots < x_n \in [a, b]$ there exists a unique polynomial of degree $\leq n$ such that*

$$P_n(x_j) = f(x_j), \quad j = 0, 1, 2, \ldots, n.$$

This polynomial is given by

$$P_n(x) = \sum_{j=0}^{n} f(x_j) L_j(x),$$

where $L_j(x)$ is defined by

$$L_j(x) = \frac{\pi_{n+1}(x)}{(x - x_j)\pi'_{n+1}(x_j)} = \frac{\prod_{k=0, k\neq j}^{n}(x - x_k)}{\prod_{k=0, k\neq j}^{n}(x_j - x_k)},$$

$\pi_{n+1}(x)$ *being the nodal polynomial*

$$\pi_{n+1}(x) = \prod_{k=0}^{n}(x - x_k).$$

Additionally, if f is $n + 1$ times continuously differentiable in (a, b), then for any $x \in [a, b]$ there exists a value $\zeta_x \in (a, b)$ depending on x, such that

$$E_n(x) = f(x) - P_n(x) = \frac{f^{(n+1)}(\zeta_x)}{(n+1)!}\pi_{n+1}(x). \tag{13}$$

Proof. 1. *Existence* By construction, P_n is a polynomial of degree n passing through x_j, $j = 0, 1, 2, \ldots, n$.

2. *Uniqueness* Suppose that there are two such polynomials, $P_n(x)$ and $Q_n(x)$. Then the polynomial $d(x) := P_n(x) - Q_n(x)$ is at most of degree n. On the other hand, it has at least $n+1$ roots x_j, $j = 0, 1, 2, \ldots, n$. Therefore, it must be identically zero.

3. *Error formula* Consider the function

$$F(z) = f(z) - P_n(z) - [f(x) - P_n(x)]\frac{\pi_{n+1}(z)}{\pi_{n+1}(x)}.$$

This function has $n+2$ zeros at $z = x_j$, $j = 0, 1, 2, \ldots, n$, and $z = x$. Recall Rolle's theorem that says that if a function $f(z)$ is continuously differentiable on $[a, b]$ and $f(a) = f(b)$ then there is $c \in (a, b)$ such that $f'(c) = 0$. Therefore, if we apply Rolle's theorem $n+1$ times we get that the function

$$F^{(n+1)}(z) = f^{(n+1)}(z) - P_n^{(n+1)}(z) - [f(x) - P_n(x)]\frac{(n+1)!}{\pi_{n+1}(x)}$$

has at least one zero in (x_0, x_n). We denote this zero by ζ_x. Therefore, taking into account that $P_n^{(n+1)}(z) \equiv 0$, we get

$$0 = f^{(n+1)}(\zeta_x) - [f(x) - P_n(x)]\frac{(n+1)!}{\pi_{n+1}(x)},$$

and Eq. (13) follows.

□

Example Let us find the Lagrange interpolant $p(x)$ passing through the points $(0, f_0)$, $(1, f_1)$, and $(2, f_2)$. The polynomial $p(x)$ is of the form

$$\begin{aligned} p(x) &= f_0\frac{(x-1)(x-2)}{(0-1)(0-2)} + f_1\frac{(x-0)(x-2)}{(1-0)(1-2)} + f_2\frac{(x-0)(x-1)}{(2-0)(2-1)} \\ &= \tfrac{1}{2}f_0(x^2 - 3x + 2) - f_1(x^2 - 2x) + \tfrac{1}{2}f_2(x^2 - x) \\ &= f_0 + \tfrac{1}{2}(-3f_0 + 4f_1 - f_2)x + \tfrac{1}{2}(f_0 - 2f_1 + f_2)x^2. \end{aligned} \tag{14}$$

2.2 The Newton interpolation polynomial

Lagrangian interpolation is convenient because it gives an explicit formula for the interpolant. However, it does not provide a convenient way to modify the polynomial to

accommodate additional interpolation points. An alternative form of the interpolation polynomial, the Newton form, gives such a way. The Newton interpolation formula is used, for example, for deriving linear multistep methods with varying time step for solving ODE's. The Newton interpolation formula is defined via the divided differences. We set

$$\begin{aligned} f[x_0] &= f(x_0), \\ f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0}, \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}, \\ &\dots \\ f[x_0, x_1, \dots, x_k] &= \frac{f[x_1, \dots, x_k] - f[x_0, x_1, \dots, x_{k-1}]}{x_k - x_0}, \\ &\dots . \end{aligned}$$

Theorem 4. *The polynomial interpolating $f(x)$ at x_j, $j = 0, 1, 2, \dots, n$ is given by*

$$\begin{aligned} P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \dots \\ + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \dots (x - x_{n-1}). \end{aligned} \tag{15}$$

Example Let us find the Newton interpolant $p(x)$ passing through the points $(0, f_0)$, $(1, f_1)$, and $(2, f_2)$. The polynomial $p(x)$ is of the form

$$p(x) = f[0] + f[0, 1](x - 0) + f[0, 1, 2](x - 0)(x - 1), \tag{16}$$

where

$$f[0] = f_0, \quad f[0, 1] = \frac{f_1 - f_0}{1 - 0} = f_1 - f_0, \quad f[1, 2] = \frac{f_2 - f_1}{2 - 1} = f_2 - f_1,$$

$$f[0, 1, 2] = \frac{f[1, 2] - f[0, 1]}{2 - 0} = \frac{1}{2}(f_2 - f_1 - (f_1 + f_0)) = \frac{1}{2}(f_0 - 2f_1 + f_2).$$

Plugging these coefficients into Eq. (16) we get

$$\begin{aligned} p(x) &= f_0 + (f_1 - f_0)x + \tfrac{1}{2}(f_0 - 2f_1 + f_2)x(x - 1) \\ &= f_0 + \tfrac{1}{2}\left(-3f_0 + 4f_1 - f_2\right)x + \tfrac{1}{2}(f_0 - 2f_1 + f_2)x^2. \end{aligned} \tag{17}$$

The polynomial $p(x)$ in Eq. (17) coincides with the one in Eq. (14) as it should.

Proof. We will proceed by induction. For $n = 1$ Eq. (15) holds. Suppose the polynomials $P[x_0, \dots, x_{n-1}](x)$ and $P[x_1, \dots, x_n](x)$ of the form of Eq. (15) interpolate f at the points

$x_0, \ldots, x_{n-1}$ and $x_1, \ldots, x_n$ respectively. Note that both of them are of degree $n-1$. Hence they differ by a polynomial of degree at most $n-1$, and this polynomial has zeros at x_1, ..., x_{n-1}. Therefore,

$$P[x_1, \ldots, x_n](x) - P[x_0, \ldots, x_{n-1}](x) = a(x - x_1) \ldots (x - x_{n-1})$$

where a is a number. Obviously, a is the difference of the leading coefficients of the polynomials $P[x_0, \ldots, x_{n-1}](x)$ and $P[x_1, \ldots, x_n](x)$, i.e.,

$$a = f[x_1, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}].$$

At the same time,

$$a = \frac{P[x_1, \ldots, x_n](x) - P[x_0, \ldots, x_{n-1}](x)}{(x - x_1) \ldots (x - x_{n-1})}.$$

Now consider the polynomial

$$P[x_0, x_1, \ldots, x_n](x) = P[x_0, \ldots, x_{n-1}](x) + \frac{a}{x_n - x_0}(x - x_0)(x - x_1) \ldots (x - x_{n-1}). \quad (18)$$

The last term of this polynomial is chosen so that it is zero at $x_0, \ldots, x_{n-1}$, and the coefficient $a/(x_n - x_0)$ is our guess that we will verify below. By construction, $P[x_0, x_1, \ldots, x_n](x)$ interpolates f at $x_0, \ldots, x_{n-1}$. Let us verify that it also does so at $x = x_n$. We evaluate $P[x_0, x_1, \ldots, x_n](x)$ at x_n and obtain:

$$\begin{aligned}
&P[x_0, x_1, \ldots, x_n](x_n) \\
=&P[x_0, \ldots, x_{n-1}](x_n) + \frac{a}{x_n - x_0}(x_n - x_0)(x_n - x_1) \ldots (x_n - x_{n-1}) \\
=&P[x_0, \ldots, x_{n-1}](x_n) \\
+&\frac{P[x_1, \ldots, x_n](x_n) - P[x_0, \ldots, x_{n-1}](x_n)}{(x_n - x_0)(x_n - x_1) \ldots (x_n - x_{n-1})}(x_n - x_0)(x_n - x_1) \ldots (x_n - x_{n-1}) \\
=&P[x_1, \ldots, x_n](x_n) = f(x_n).
\end{aligned}$$

Therefore, the polynomial given by Eq. (18) interpolates f at x_j, $j = 0, 1, 2, \ldots, n$. □

Remark The function $f[x_0, \ldots, x_n]$ is a symmetric function of its arguments i.e., it does not change if we permute x_0, ..., x_n. This is because the interpolation polynomial is independent of the order of nodes.

2.3 Runge phenomenon

Read Section 3.2.1 in Chapter 3 in Gil et al. [Gil et al.(2007)Gil, Segura, and Temme] and Chapter 5 in Trefethen [Trefethen(2000)].

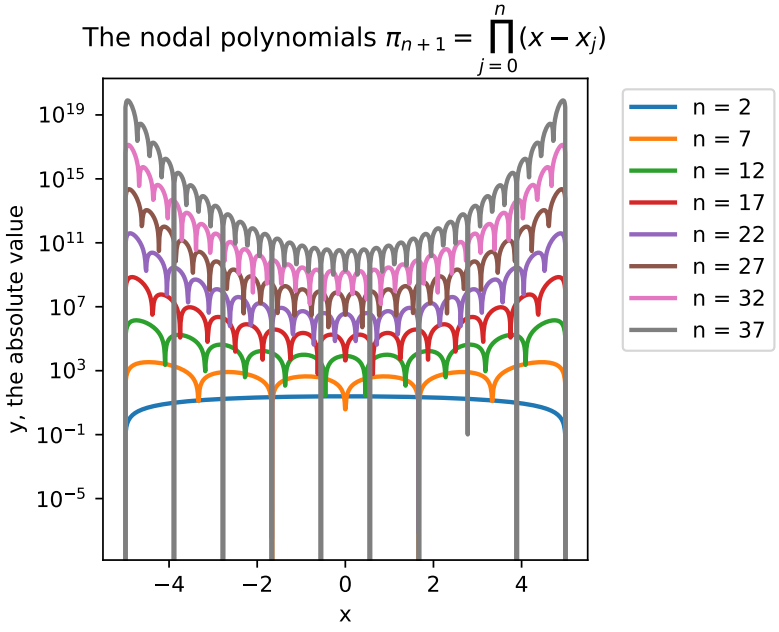

Figure 2: The nodal polynomials $\pi_{n+1}(x)$ with uniform nodes grow exponentially in the max norm.

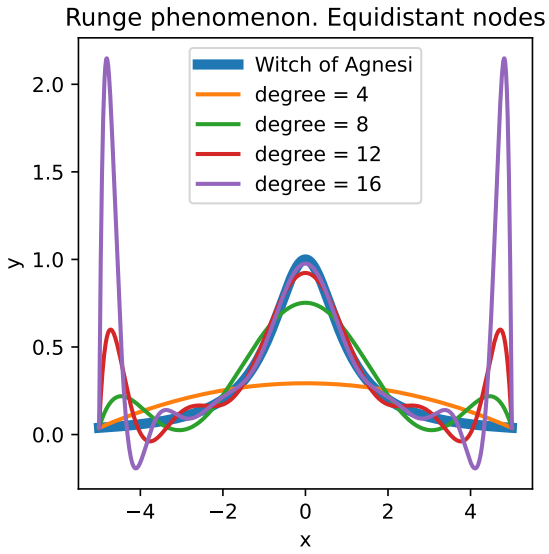

Figure 3: A demonstration of the Runge phenomenon on the example of a smooth function "Witch of Agnesi" $f(x) = (1 + x^2)^{-1}$ on the interval $[-5, 5]$ that has poles at $\pm i$ in the complex plane.

2.4 Chebyshev polynomials

Read Section 3.3 in Chapter 3 in Gil et al. [Gil et al.(2007)Gil, Segura, and Temme].

We will study the Chebyshev polynomials in more detail as they lead to several remarkable numerical tools:

- Chebyshev interpolation, where the Runge phenomenon is eliminated [Gil et al.(2007)Gil, Segura, and Temme];
- Chebyshev least squares approximation, which is close to the minimax one;
- Chebyshev spectral methods for solving PDEs with non-periodic boundary conditions – L. N. Trefethen's book [Trefethen(2000)] Spectral Methods in Matlab.

Chebyshev polynomials are defined as follows:

$$T_n(x) = \cos\left[n \arccos(x)\right], \quad x \in [-1, 1], \quad n = 0, 1, 2, \ldots. \tag{19}$$

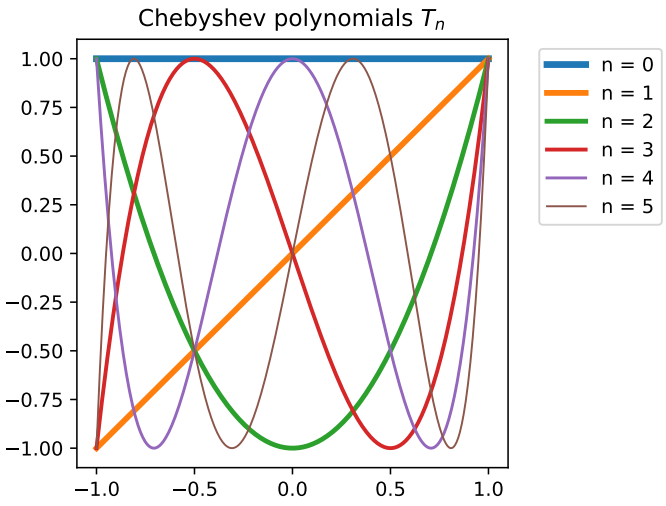

Figure 4: Graphs of the Chebyshev polynomials $T_0(x)$, $T_1(x)$, ..., $T_6(x)$.

It follows from the definition that

$$T_n(\cos\theta) = \cos(n\theta), \quad \theta \in [0, \pi], \quad n = 0, 1, 2, \ldots. \tag{20}$$

The first few Chebyshev polynomials are

$$\begin{aligned}
T_0(x) &= 1, \\
T_1(x) &= x, \\
T_2(x) &= 2x^2 - 1, \\
T_3(x) &= 4x^3 - 3x, \\
T_4(x) &= 8x^4 - 8x^2 + 1, \\
T_5(x) &= 16x^5 - 20x^3 + 5x, \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1.
\end{aligned}$$

2.5 Properties of the Chebyshev polynomials

1. The Chebyshev polynomials satisfy the following three-term recurrence relationships (TTRR):

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad T_0(x) = 1, \quad T_1(x) = x, \quad n = 0, 1, 2, \ldots. \tag{21}$$

This TTRR immediately follows from Eq. (20) and the trigonometric formula

$$2\cos(a)\cos(b) = \cos(a+b) + \cos(a-b). \tag{22}$$

Indeed, let $x = \cos\theta$. Then setting $a = n\theta$ and $b = \theta$ we get (21).

2. The leading coefficient of $T_n(x)$ is 2^{n-1}. This follows from Eq. (21).

3.
$$T_n(-x) = (-1)^n T_n(x).$$

This means that if n is even, then $T_n(x) = T_n(-x)$, i.e., $T_n(x)$ is an even function, and if n is odd then $T_n(x) = -T_n(-x)$, i.e., $T_n(x)$ is an odd function. This follows from the formulas

$$\arccos(-x) = \pi - \arccos(x) \quad \text{and} \quad \cos(\pi n - x) = (-1)^n \cos x.$$

4. Zeros of $T_n(x)$ are

$$x_k = \cos\left(\frac{k+\frac{1}{2}}{n}\pi\right), \quad k = 0, 1, \ldots, n-1. \tag{23}$$

Indeed, $T_n(\cos\theta) = \cos n\theta = 0$. Hence $n\theta = \frac{\pi}{2} + \pi k$, then $\theta = \frac{\pi}{n}(k + \frac{1}{2})$, therefore, $x = \cos\theta$ is given by (23). The values of $k = 0, 1, \ldots, n-1$ define fistinct roots.

5. Extrema of $T_n(x)$ are

$$x'_k = \cos\left(\frac{\pi k}{n}\right), \quad k = 0, 1, \ldots, n. \tag{24}$$

Note that $T_n(x'_k) = (-1)^k$. Also note that x'_k follow the decreasing order and $x'_0 = 1$ and $x'_n = -1$.

6. The deviation of $2^{-n}T_{n+1}(x)$ from zero on $[-1, 1]$ is minimal possible among all polynomials with leading coefficient 1 of degree $n+1$. The theorem establishing this fact is below.

Theorem 5. *Let*

$$x_k = \cos\left(\frac{k+\frac{1}{2}}{n+1}\pi\right), \quad k = 0, 1, \ldots, n.$$

Then the monic polynomial (i.e., its leading coefficient is 1)

$$\hat{T}_{n+1} = \prod_{k=0}^{n}(x - x_k)$$

of degree $n+1$ has the smallest possible uniform (maximum) norm 2^{-n} in $[-1,1]$ among all polynomials of degree $n+1$. I.e.,

$$2^{-n} = \max_{x\in[-1,1]} |\hat{T}_{n+1}(x)| = \min_{\substack{p\in\mathcal{P}_{n+1}\\ p=x^{n+1}+\dots}} \max_{x\in[-1,1]} |p(x)|.$$

Proof. Suppose there is a monic polynomial $p(x)$ of degree $n+1$ such that $|p(x)| < 2^{-n}$ for all $x \in [-1,1]$. Let x'_k, $k = 0, 1, \dots, n+1$ be the abscissas of the extreme values of Chebyshev polynomials of degree $n+1$. Hence we have

$$\begin{aligned} &p(x'_0) < 2^{-n} T_{n+1}(x'_0),\\ &p(x'_1) > 2^{-n} T_{n+1}(x'_1),\\ &p(x'_2) < 2^{-n} T_{n+1}(x'_2),\\ &\quad \dots. \end{aligned}$$

Therefore the polynomial

$$Q(x) = p(x) - 2^{-n} T_{n+1}(x)$$

changes sign between each two consecutive extrema of T_{n+1}. $T_{n+1}(x)$ has $n+2$ extrema on $[-1,1]$. Hence $Q(x)$ has $n+1$ zeros. But $Q(x)$ is of degree $\le n$. Thus we have arrived to a contradiction. Hence there is no such monic polynomial p of degree $n+1$ such that $|p(x)| < 2^{-n}$ for $x \in [-1,1]$. □

7. Relations with derivatives.

$$T_0(x) = T'_1(x), \tag{25}$$

$$T_1(x) = \frac{1}{4} T'_2(x), \tag{26}$$

$$T_n(x) = \frac{1}{2}\left(\frac{T'_{n+1}(x)}{n+1} - \frac{T'_{n-1}(x)}{n-1}\right), \quad n \ge 2. \tag{27}$$

The first two equalities are easy-to-check. The last one can be proven from the following observation:

$$T'_n(x) = \frac{n\sin(n\theta)}{\sin\theta}, \quad \text{where} \quad x = \cos\theta.$$

Exercise Prove Eq. (27).

8. Multiplication relationship:

$$2T_r(x)T_q(x) = T_{r+q}(x) + T_{|r-q|}(x). \tag{28}$$

Exercise Prove Eq. (28).

9. Orthogonality relationship:

$$\int_{-1}^{1} \frac{T_r(x)T_s(x)}{\sqrt{1-x^2}}dx = \begin{cases} 0, & r \neq s \\ \pi, & r = s = 0, \\ \frac{\pi}{2}, & r = s \neq 0. \end{cases} \tag{29}$$

Let us prove it. We make a variable change $x = \cos\theta$. Then $dx = -\sin\theta d\theta$, $T_r(x) = \cos(r\theta)$, $T_s(x) = \cos(s\theta)$, and the integration limits are changed to: $-1 \mapsto \pi$, $1 \mapsto 0$. Therefore, using the formula

$$\cos a \cos b = {}^1\!/_2 \cos(a+b) + {}^1\!/_2 \cos(a-b)$$

we get:

$$\begin{aligned} \int_{-1}^{1} \frac{T_r(x)T_s(x)}{\sqrt{1-x^2}}dx &= \int_0^{\pi} \cos(r\theta)\cos(s\theta)d\theta = \frac{1}{2}\int_0^{\pi} \left[\cos(r+s)\theta + \cos(r-s)\theta\right] d\theta \\ &= \frac{1}{2}\begin{cases} \left[\frac{\sin(r+s)\theta}{r+s} + \frac{\sin(r-s)\theta}{r-s}\right]_0^{\pi}, & r \neq s \\ 2\pi, & r = s = 0, \\ \pi + \left[\frac{\sin(2r\theta)}{2r}\right]_0^{\pi}, & r = s \neq 0. \end{cases} \\ &= \begin{cases} 0, & r \neq s \\ \pi, & r = s = 0, \\ \frac{\pi}{2}, & r = s \neq 0. \end{cases} \end{aligned}$$

10. Discrete orthogonality relationship: Take the points

$$x_j = \cos\left(\frac{\pi(j+\frac{1}{2})}{n+1}\right), \quad j = 0, 1, \ldots, n$$

that are the zeros of $T_{n+1}(x)$. Then for all $0 \leq r, s \leq n$ we have

$$\sum_{j=0}^{n} T_r(x_j)T_s(x_j) = \begin{cases} 0, & r \neq s \\ n+1, & r = s = 0, \quad j = 0, 1, \ldots, n. \\ \frac{n+1}{2}, & r = s \neq 0, \end{cases} \tag{30}$$

Exercise Prove Eq. (30). *Hint: first use the trigonometric formula*

$$\cos(r\theta)\cos(s\theta) = \tfrac{1}{2}\cos(r+s)\theta + \tfrac{1}{2}\cos(r-s)\theta,$$

then use the formula $\cos y = \frac{1}{2}(e^{iy} + e^{-iy})$. *You will obtain four geometric series. Compute their sums.*

2.6 Chebyshev interpolation

Read Section 3.4 in Gil et al. [Gil et al.(2007)Gil, Segura, and Temme]. Properties of the Chebyshev polynomials offer a nice way for computing the Chebyshev interpolant of degree n. Fix some integer n and consider the zeros of $T_{n+1}(x)$. They are

$$x_j = \cos\left(\frac{\pi(j+\frac{1}{2})}{n+1}\right), \quad j = 0, 1, \ldots, n.$$

For a given function $f(x)$ on the interval $[-1, 1]$, the polynomial p_n of degree n interpolating $f(x)$ at the Chebyshev points x_j, $j = 0, 1, \ldots, n$, is given by

$$p_n(x) = {\sum_{k=0}^{n}}' c_k T_k(x) \equiv \frac{c_0}{2} + \sum_{k=1}^{n} c_k T_k(x).$$

The symbol $'$ indicates that the first term in the sum should be divided by two. The coefficients c_k are found from the requirement that $p_n(x_j) = f(x_j)$, $j = 1, 2, \ldots, n$, i.e.,

$$f(x_j) = {\sum_{k=0}^{n}}' c_k T_k(x_j).$$

Then we have

$$\sum_{j=0}^{n} f(x_j) T_m(x_j) = {\sum_{k=0}^{n}}' c_k \sum_{j=0}^{n} T_k(x_j) T_m(x_j) = {\sum_{k=0}^{n}}' c_k \frac{n+1}{2} \delta_{mk} = \tfrac{1}{2}(n+1) c_k.$$

Hence the coefficients are given by

$$c_k = \frac{2}{n+1} \sum_{j=0}^{n} f(x_j) T_k(x_j), \quad x_j = \cos\left(\frac{\pi(j+\frac{1}{2})}{n+1}\right). \tag{31}$$

To summarize, the Chebyshev interplant of $f(x)$ is given by

$$p_n(x) = \frac{c_0}{2} + \sum_{k=1}^{n} c_k T_k(x), \tag{32}$$

where the coefficients c_k are given by Eq. (31).

2.7 Chebyshev polynomials shifted to the interval $[a, b]$

Suppose we need to find the Chebyshev interpolant for $f(y)$ on the interval $y \in [a, b]$. We proceed as follows.

1. First we set up a linear map of $[-1,1]$ onto $[a,b]$ and map the Chebyshev points, the roots of T_{n+1}, lying in the interval $[-1,1]$, to the interval $[a,b]$:

$$l(x) = \frac{b-a}{2}x + \frac{a+b}{2}; \quad x_k = \cos\left(\frac{k+1/2}{n+1}\pi\right), \quad y_k = l(x_k), \quad k = 0,1,\ldots,n. \tag{33}$$

2. Next, we compute the Chebyshev coefficients:

$$c_j = \frac{2}{n+1}\sum_{k=0}^{n} T_j(x_k) f(y_k). \tag{34}$$

3. Finally, we write out the Chebyshev interpolant

$$p_n(y) = \frac{c_0}{2} + \sum_{j=1}^{n} c_j T_j\left(\frac{2y-a-b}{b-a}\right). \tag{35}$$

Exercise Show that the shifted Chebyshev polynomials satisfy the following orthogonality relationships

$$\int_a^b \frac{T_r^{[a,b]}(y) T_s^{[a,b]}(y)}{\sqrt{1-\left(\frac{2y-a-b}{b-a}\right)^2}} dy = \begin{cases} 0, & r \neq s \\ \frac{b-a}{2}\pi, & r = s = 0, \\ \frac{(b-a)\pi}{4}, & r = s \neq 0. \end{cases} \tag{36}$$

2.8 Clenshaw's method for evaluating Chebyshev sums

Chebyshev interpolant can be evaluated at the point $y \in [a,b]$ directly from Eq. (32)

$$p_n(y) = \frac{c_0}{2} + \sum_{k=1}^{n} c_k \cos(k \arccos(y)), \quad \text{where} \quad x = \frac{2y-a-b}{b-a}. \tag{37}$$

This formula has a shortcoming in that it requires evaluation of cos and arccos which are typically built-in functions, but their evaluation is time-consuming in comparison with basic floating-point operations.

An elegant way to evaluate Chebyshev interpolant that avoids calculations of cos and arccos was proposed by Clenshaw (1955). A detailed description of Clenshaw's algorithm is given in Chapter 3 "Chebyshev Expansions" from "Numerical Methods for Special Functions" by Amparo Gil, Javier Segura, and Nico Temme (see pages 75-76). Here, we show how one can implement it in MATLAB.

Suppose we need to evaluate the sum

$$p_n(y) = \frac{c_0}{2} + \sum_{k=1}^{n} c_k T_k(x), \quad \text{where} \quad x = \frac{2y - a - b}{b - a}.$$

We rewrite this sum in the vector form:

$$p_n(y) = \mathbf{c}^\top \mathbf{t} - \frac{c_0}{2},$$

where $\mathbf{c} \coloneqq [c_0, c_1, \ldots, c_n]^\top$, $\mathbf{t} \coloneqq [T_0(x), T_1(x), \ldots, T_n(x)]^\top$. Recall that the Chebyshev polynomials satisfy TTRR (21) that can be written in the matrix form as

$$\begin{bmatrix} 1 & & & & & \\ -2x & 1 & & & & \\ 1 & -2x & 1 & & & \\ & 1 & -2x & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2x & 1 \end{bmatrix} \begin{bmatrix} T_0(x) \\ T_1(x) \\ T_2(x) \\ \vdots \\ T_n(x) \end{bmatrix} = \begin{bmatrix} 1 \\ -x \\ 0 \\ \vdots \\ 0 \end{bmatrix} \equiv A\mathbf{t} = \mathbf{d}.$$

Let $\mathbf{b}$ be a vector such that

$$\mathbf{b}^\top A = \mathbf{c}^\top, \quad \text{or} \quad A^\top \mathbf{b} = \mathbf{c}. \tag{38}$$

Note that $\mathbf{b}$ is readily found starting from b_n as $A^\top$ is upper-triangular. Then

$$\begin{aligned} p_n(y) &= \mathbf{c}^\top \mathbf{t} - \frac{c_0}{2} = \mathbf{b}^\top A \mathbf{t} - \frac{c_0}{2} \\ &= \mathbf{b}^\top \mathbf{d} - \frac{c_0}{2} = b_0 - b_1 x - \frac{c_0}{2}. \end{aligned}$$

From (38) we find

$$c_0 = b_0 - 2x b_1 + b_2.$$

Therefore,

$$p_n(y) = b_0 - b_1 x - \frac{c_0}{2} = b_0 - b_1 x - \frac{1}{2}(b_0 - 2x b_1 + b_2) = \frac{1}{2}(b_0 - b_2). \tag{39}$$

3 Approximation of functions with neural networks

References for this section:

1. [Pinkus(1999)]: A very good review of the results on the density of NN spaces of the 1980s and 1990s.

   Allan Pinkus, Approximation theory of the MLP model in neural networks, Acta Numerica (1999), pp. 143—195

2. [Yarotsky(2017)]: An important work that addresses the power of the depth of NNs.

   Dmitry Yarotsky, Error bounds for approximations with deep ReLU networks, Neural Networks, 94, October 2017, pp. 103—114

3. [He et al.(2022)He, Li, and Xu]: An educational work connecting NN spaces with finite element spaces.

   Juncai He, Lin Li, Jinchao Xu, ReLU Neural Networks from Hierarchical Basis Perspective, Computer and Mathematics with Applications, 120(15), August 2022, pp. 105—114

4. [Lu et al.(2021)Lu, Shen, Yang, and Zhang]: An important work establishing the scaling of the required width and depth of NNs to represent a smooth function with a given accuracy. It shows the curse of dimensionality in neural network-based approximations.

   Jianfeng Lu, Zuowei Shen, Haizhao Yang, Shijun Zhang, Deep Network Approximation for Smooth Functions, SIAM Journal for Mathematical Analysis 53(5), pp. 5465-5506 (2021)

5. [Shen et al.(2021)Shen, Yang, and Zhang]: An interesting work showing how to eliminate the curse of dimensionality in terms of the number of parameters (but not in terms of the memory required).

   Zuowei Shen, Haizhao Yang, Shijun Zhang, Neural network approximation: Three hidden layers are enough, Neural Networks 141, pp. 160—173

3.1 What is a neural network?

A *feed-forward fully connected neural network* with l *hidden layers* is a composition of functions of the form

$$\mathcal{N}(x;\theta) = \mathcal{L}_{l+1} \circ \sigma_l \circ \mathcal{L}_l \circ \sigma_{l-1} \circ \ldots \circ \sigma_1 \circ \mathcal{L}_1. \tag{40}$$

The symbol $\mathcal{L}_k$ denotes the k's affine operator of the form $\mathcal{L}_k(x) = A_k x + b_k$, while σ_k denotes a nonlinear function called an *activation function*. The activation functions act entry-wise on vector arguments.

The user chooses the activation functions, the dimensions of matrices A_k, and the number of hidden layers l and called the *hyperparameters* of the neural network.

The matrices A_k and shift vectors (or bias vectors) b_k are encoded into the argument θ: $\theta = \{A_k, b_k\}_{k=1}^{l+1}$. They are called the *parameters* of the neural network. The term *training neural network* means optimizing $\{A_k, b_k\}_{k=1}^{l+1}$ with respect to a certain objective function called the *loss function*. The loss function is set up by the user so that it is minimized if the neural network $\mathcal{N}(x;\theta)$ satisfies certain conditions. For example, one might want the neural network to assume certain values f_j at certain points x_j, $j = 1, \ldots, N$. These points x are called the *training data*. In this case, a common choice of the loss function is the least squares error:

$$\mathsf{Loss}(x;\theta) = \frac{1}{n}\sum_{j=1}^{n} \|N(x_j;\theta) - f_j\|^2. \tag{41}$$

The activation functions σ_k can be arbitrary **non-polynomial** functions. Popular choices are (Fig. 3.1)

- sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}};$$

- hyperbolic tangent:

$$\tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}}; \tag{42}$$

- rectified linear unit:

$$\mathsf{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0. \end{cases}$$

There are many other possible choices for the activation functions.

3.2 Approximation of continuous functions in $\mathbb{R}^n$ by neural networks with one hidden layer

Ref.: [Pinkus(1999)]

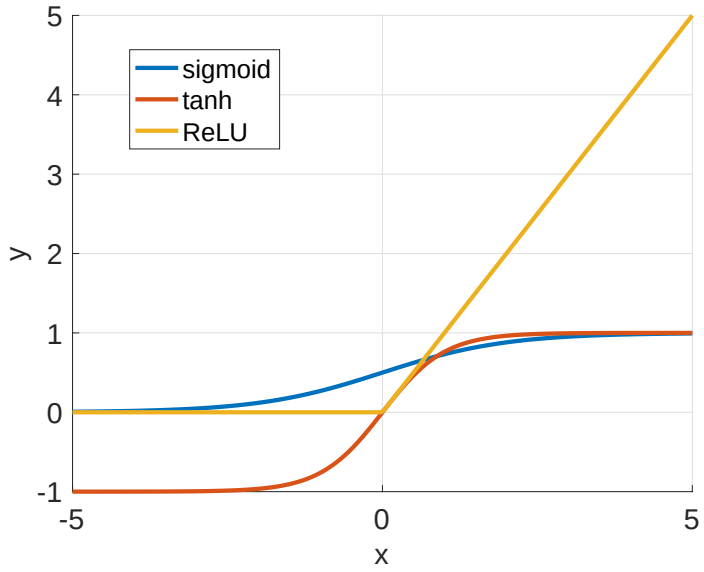

Figure 5: Common activation functions for neural networks.

Definition 2. *We will say that a set of functions $\mathcal{M}$ is* **dense** *in the set of continuous functions $C(\mathbb{R}^n)$ in the* **topology of uniform convergence on compacta** *if for any $\epsilon > 0$, any compact set $K \subset \mathbb{R}^n$, and any continuous function $f(x) : \mathbb{R}^n \to \mathbb{R}$, there exists a function $g(x) \in \mathcal{M}$ such that*

$$\max_{x \in K} |f(x) - g(x)| < \epsilon. \tag{43}$$

We will consider the sets $\mathcal{M}$ of the form

$$\mathcal{M}(\sigma) := \operatorname{span} \{\sigma(w \cdot x - \theta) \mid w \in \mathbb{R}^n,\ \theta \in \mathbb{R}\}. \tag{44}$$

The question whether $\mathcal{M}(\sigma)$ is dense in $C(\mathbb{R})$ in the topology of uniform convergence on compacta addresses the necessary part of the function approximation. If the answer is yes, then, in principle, one can approximate a function $f \in C(\mathbb{R}^n)$ with a function $g \in \mathcal{M}(\sigma)$ with any given accuracy. However, it does not mean that this approximation will be efficient regarding the number of parameters involved, and it does not give us a good recipe to construct such an approximation. We will address these questions in later sections.

The earliest works addressing the density of $\mathcal{M}(\sigma)$ are Hecht-Nielsen (1987), Gallant and White (1988), and Irie-Miyake (1988).

The first proofs of density of $\mathcal{M}(\sigma)$ in $C(\mathbb{R}^n)$ in the uniform topology on compacta belong to Cybenko (1989) [Cybenko(1989)] and Funahashi (1989). Cybenko and Funahashi independently proved very similar results using different methods. Cybenko proved it for any continuous sigmoidal function, which is not necessarily monotone, while Funahashi proved it for any continuous monotone sigmoidal function. Hornik, Stinchcombe, and White (1989) proved almost the same results for σ bounded and monotone, but not necessarily continuous. Their method of proof is also different.

The main theorem in [Pinkus(1999)] is

Theorem 6. *Let $\sigma \in C(\mathbb{R})$. Then $\mathcal{M}(\sigma)$ is dense in $C(\mathbb{R}^n)$, in the topology of uniform convergence on compacta, if and only if σ is not a polynomial.*

If σ is a polynomial of degree r, then $\mathcal{M}$ is a set of polynomials of degree r and hence not dense in $C(\mathbb{R}^n)$.

If σ is not a polynomial, the proof of the density property of $\mathcal{M}$ is quite long and consists of several stages. The following proposition reveals the two components that we need to prove to establish the density of (σ) in $C(\mathbb{R}^n)$, in the topology of uniform convergence on compacta.

Proposition 1. *Assumptions:*

1. *$\Lambda, \Theta \subset \mathbb{R}$ are such that the set*

$$\mathcal{N}(\sigma; \Lambda, \Theta) := \text{span}\{\sigma(\lambda t - \theta) \mid \lambda \in \Lambda, \theta \in \Theta\}, \tag{45}$$

   is dense in $C(\mathbb{R})$ in the topology of uniform convergence on compacta.

2. *Let $A \subset \mathbb{S}^{n-1}$ be a set of unit vectors in $\mathbb{R}^n$, where $\mathbb{S}^{n-1}$ is the unit sphere in $\mathbb{R}^n$, such that the set of* **ridge functions**

$$\mathcal{R}(A) = \text{span}\,\{g(a \cdot x) \ : \ g \in C(\mathbb{R}),\ a \in A\} \tag{46}$$

   is dense in $C(\mathbb{R})$ in topology of uniform convergence on compacta.

Then

$$\mathcal{M}(\sigma; \Lambda \times A, \Theta) := \text{span}\{\sigma(w \cdot x - \theta) \mid w \in \Lambda \times A, \theta \in \Theta\} \tag{47}$$

is dense in $C(\mathbb{R})$ in topology of uniform convergence on compacta.

Proof. Let $f \in C(K)$ for some compact set in $\mathbb{R}^n$. Since $\mathcal{R}(A)$ is dense in $C(K)$, given $\epsilon > 0$ there exist $g_i \in C(\mathbb{R}$ and $a^i \in A$, $i = 1, \ldots, r$ such that

$$\left| f(x) - \sum_{i=1}^{r} g_i(a^i \cdot x) \right| < \frac{\epsilon}{2} \quad \forall x \in K. \tag{48}$$

Since K is compact,

$$\{a^i \cdot x \mid x \in K\} \subseteq [\alpha_i, \beta_i],$$

for some finite interval $[\alpha_i, \beta_i]$, $i = 1, \ldots, r$. Because $\mathcal{N}(\sigma; \Lambda, \Theta)$ is dense in $C([\alpha_i, \beta_i])$, $i = 1, \ldots, r$, there exist constants $c_{ij} \in \mathbb{R}$, $\lambda_{ij} \in \Lambda$, and $\theta_{ij} \in \Theta$, $j = 1, \ldots, m_i$, $i = 1, \ldots, r$, for which

$$\left| g_i(t) - \sum_{j=1}^{m_i} c_{ij} \sigma(\lambda_{ij} t - \theta_{ij}) \right| < \frac{\epsilon}{2r} \quad \forall t \in [\alpha_i, \beta_i], \quad i = 1, \ldots, r. \tag{49}$$

Therefore,

$$\left| f(x) - \sum_{i=1}^{r} \sum_{j=1}^{m_i} c_{ij} \sigma(\lambda_{ij} a^i \cdot x - \theta_{ij}) \right| < \epsilon \quad \forall x \in K. \tag{50}$$

□

The question of whether the set $\mathcal{R}(A)$ of ridge functions is dense in $C(\mathbb{R}^n)$ was answered in 1961 by Vostrecov and Kreines:

Theorem 7. *$\mathcal{R}(A)$ is dense in $C(\mathbb{R}^n)$ in topology of uniform convergence on compacta, if and only if there is no nontrivial (not identically zero) homogeneous (all terms have the same total degree) polynomial that vanishes on A.*

Note that a ridge function $g(a \cdot x)$ is constant on all hyperplanes normal to the direction of a.

The proof of the density of $\mathcal{N}(\cdot)$ in $C(\mathbb{R})$ is conducted in a series of propositions that gradually extend the set of permissible σ and shrink the sets Λ and Θ.

Proposition 2. *Let $\sigma \in C^\infty(R)$ and is not a polynomial. Then $\mathcal{N}(\sigma; \mathbb{R}, \mathbb{R})$ is dense in $\mathbb{R}$.*

We will need the following lemma by Corominas and Sunyer Balaguer (1954):

Lemma 1. *Let $\sigma \in C^\infty(R)$ and is not a polynomial. Let I be any open interval in $\mathbb{R}$. Then there exists a point $t \in I$ such that $\sigma^{(k)}(t) \neq 0$ for all $k \in \{0\} \cup \mathbb{N}$.*

Proof. (Proposition 2.) Set $-\theta_0 = t$ where t is the point from Lemma 1. We have that

$$\frac{1}{h}\left[\sigma((\lambda + h)t - \theta_0) - \sigma(\lambda t - \theta_0)\right] \in \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}).$$

Therefore, letting $h \to 0$ and choosing $\lambda = 0$, we obtain

$$\frac{d}{d\lambda}\sigma(\lambda t - \theta_0)\Big|_{\lambda=0} = t\sigma'(-\theta_0) \in \overline{\mathcal{N}(\sigma; \mathbb{R}, \mathbb{R})}.$$

Inductively applying the same argument, we establish that

$$\frac{d^k}{d\lambda^k}\sigma(\lambda t - \theta_0)\Big|_{\lambda=0} = t^k\sigma^{(k)}(-\theta_0) \in \overline{\mathcal{N}(\sigma; \mathbb{R}, \mathbb{R})} \quad \forall k \in \{0\} \cup \mathbb{N}.$$

Since $\sigma^{(k)}(-\theta_0) \neq 0$ for any k, the set $\overline{\mathcal{N}(\sigma; \mathbb{R}, \mathbb{R})}$ contains all monomials t^k, $k = 0, 1, \ldots$, hence all polynomials. Hence, it is dense in $C(K)$ for any compact set K by the Weierstrass theorem. □

Note that the proof of Proposition 2 only requires that Λ contains a sequence tending to zero. In the further series of propositions, Pinkus [Pinkus(1999)] relaxes the requirements on σ. I am referring interested students to his paper.

3.3 Approximation of differentiable functions in $\mathbb{R}^n$ by neural networks with one hidden layer

Notation:

- $$D^{\mathbf{m}} := \frac{\partial^{|\mathbf{m}|}}{\partial^{m_1} \ldots \partial^{m_n}}, \quad |m| = m_1 + \ldots + m_n, \quad m_1, \ldots, m_n \in \mathbb{Z}_+^n \equiv \{0\} \cup \mathbb{R}^n;$$
- $$C^{\mathbf{m}^1, \ldots, \mathbf{m}^s}(\mathbb{R}^n) = \bigcap_{j=1}^{s} C^{\mathbf{m}^j}(\mathbb{R}^n).$$

Definition 3. *We say that $\mathcal{M}(\sigma)$ is dense in $C^{\mathbf{m}^1,\ldots,\mathbf{m}^s}$ if, for any $f \in C^{\mathbf{m}^1,\ldots,\mathbf{m}^s}(\mathbb{R}^n)$, any compact set $K \subset \mathbb{R}^n$, any $\epsilon > 0$, there exists $g \in \mathcal{M}(\sigma)$ such that*

$$\max_{x\in K} \left| D^{\mathbf{k}} f(x) - D^{\mathbf{k}} g(x) \right| < \epsilon \quad \forall \mathbf{k} \in \mathbb{Z}_+^n \text{ s.t. } \mathbf{k} \le \mathbf{m}^i \text{ for some } i. \tag{51}$$

Theorem 8. *Let $\mathbf{m}^i \in \mathbb{Z}_+^n$, $1 \le i \le s$, and set $m = \max_{1\le i\le s} |\mathbf{m}^i|$. Assume $\sigma \in C^m(\mathbb{R})$ and sigma is not a polynomial. Then $\mathcal{M}(\sigma)$ is dense in $C^{\mathbf{m}^1,\ldots,\mathbf{m}^s}(\mathbb{R}^n)$.*

This result was established as a results of a series of works by various authors in the 1990s: Hornik, Stinchcombe and White (1990), Hornik (1991), Ito (1993), Li (1996). Pinkus's review paper contains an outline of its proof.

3.4 How many neurons do we need?

Pinkus [Pinkus(1999)] reviews the results on the number of neurons in one-hidden-layer neural networks required to approximate a given function $f \in \mathcal{B}_p^s(\mathbb{R}^n)$, where

$$\mathcal{B}_p^s(\mathbb{R}^n) := \left\{ f \in \mathcal{W}_p^s \mid \|f\|_{s,p} \le 1 \right\}. \tag{52}$$

Here $\mathcal{W}_p^m$ is the Sobolev space consisting of all functions with weak derivatives up to order m such that their L_p-norm over the unit ball in $\mathbb{R}^n$ is finite. Note that $\mathcal{W}_p^m$ includes all functions with continuous derivatives up to order m that have finite L_p-norm over the unit ball in $\mathbb{R}^n$. It also includes all functions with continuous derivatives up to order $m-1$ that have finite L_p-norm over the unit ball in $\mathbb{R}^n$, and the $(m-1)$st derivative is Lipschitz-continuous.

Hence, we define the set

$$\mathcal{M}_r(\sigma) := \left\{ \sum_{i=1}^{r} c_i \sigma(w_i \cdot x - \theta_i) \mid c_i, \theta_i \in \mathbb{R},\ w_i \in \mathbb{R}^n \right\}. \tag{53}$$

The infimum of the approximation error of f in $\mathcal{M}_r(\sigma)$ in a normed space X is denoted by

$$E(f, \mathcal{M}_r(\sigma), X) := \inf_{g\in\mathcal{M}_r(\sigma)} \|f - g\|_X. \tag{54}$$

The following two theorems reveal how the lower and upper bounds on $E(\cdot)$ scale with s and n.

Theorem 9. Maiorov and Meir, 1999. Lower bound. *Let $p \in [1, \infty]$, $s \geq 1$, $n \geq 2$, and*

$$\sigma(t) = \frac{1}{1 + e^{-t}},$$

or σ be a polynomial spline of a fixed degree with a finite number of knots. Then

$$E(\mathcal{B}_p^s(\mathbb{R}^n), \mathcal{M}_r(\sigma), L_p) \geq C(r \log r)^{-s/n} \tag{55}$$

for some constant C independent of r.

Theorem 10. Mhaskar, 1996. Upper bound. *Assume that $\sigma : \mathbb{R} \to \mathbb{R}$ is $C^\infty(\mathbb{R})$ and not a polynomial. For any $p \in [1, \infty]$, $s \geq 1$, $n \geq 2$,*

$$E(\mathcal{B}_p^s(\mathbb{R}^n), \mathcal{M}_r(\sigma), L_p) \leq Cr^{-s/n} \tag{56}$$

for some constant C independent of r.

This theorem can be derived using the fact that the error of approximating a function by a polynomial scales with its degree as k as k^{-s}, as follows from the Chebyshev interpolation and its connection with the Fourier theory. The number of neurons r such that the closure of $\mathcal{M}_r$ contains a polynomial of degree k in 1D scales as k, as can be seen from the proof of Proposition 2. The number of monomials of degree k in $\mathbb{R}^n$ is

$$\left(\begin{array}{c} k+n-1 \\ k \end{array} \right) \equiv \left(\begin{array}{c} k+n-1 \\ n-1 \end{array} \right) = \frac{(k+n-1)(k+n-2)\ldots(k+1)}{(n-1)!}.$$

If $k \gg n$, the number of monomials scales at k^n.

The upper bound in Theorem 10 was generalized on other classes of activation functions.

3.5 Benefits of depth of neural networks

The first work that addresses the benefits of depth is the PhD dissertation by Johan Hastad (MIT,1986) [Hastad(1987)] who has shown that Boolean circuits with only `or` and `and` operations require exponential size in order to represent a parity function well. The parity function takes a boolean vector as input and outputs one if the input vector has odd number of ones and zero otherwise.

3.5.1 Sawtooth functions

Matus Telgarsky [Telgarsky(2015)] considered the sawtooth function

$$g(x) \equiv g_1(x) := \begin{cases} 2x, & x \in [0, 1/2), \\ 2(1-x), & x \in [1/2, 1], \\ 0, & \text{otherwise.} \end{cases} \tag{57}$$

This function can be iterated as

$$g_m(x) = \underbrace{g \circ \ldots \circ g}_{m \text{ times}}(x). \tag{58}$$

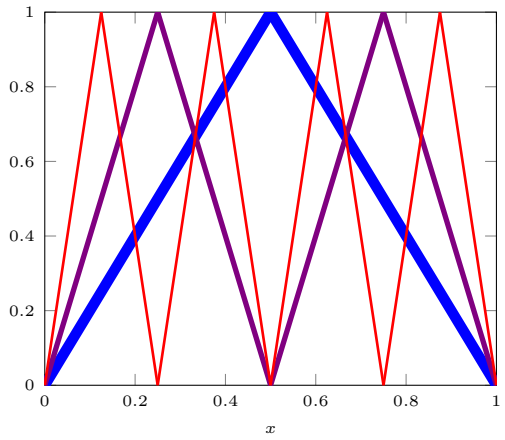

Figure 6: Sawtooth functions g_m for $m = 1, 2, 3$.

For example, the functions $g_2(x)$ and $g_3(x)$ are

$$g_2(x) = g(g(x)), \quad g_3(x) = g(g(g(x))). \tag{59}$$

The graphs of g, g_2, and g_3 are shown in Fig. 6. The function $g_m(x)$ has 2^{m-1} teeth.

It is easy to check that $g(x)$ can be expressed as a linear combination of three ReLU functions:

$$g(x) = 2\mathsf{ReLU}(x) - 4\mathsf{ReLU}(x - 1/2) + 2\mathsf{ReLU}(x - 1). \tag{60}$$

Indeed,

$$2\mathsf{ReLU}(x) - 4\mathsf{ReLU}(x - 1/2) + 2\mathsf{ReLU}(x - 1) = \begin{cases} 0, & x \le 0, \\ 2x, & x \in [0, 1/2), \\ 2x - 4x + 2 = 2(1 - x), & x \in [1/2, 1], \\ 2x - 4x + 2 + 2x - 2 = 0, & x > 1 \end{cases} = g(x).$$

Therefore, the function g_2 can be represented as a two-layer ReLU neural network with three neurons in each layer. In fact, g_2 can be represented as a linear combination of five ReLU functions, i.e., as a one-layer ReLU neural network with 5 neurons:

$$g_2(x) = 4\mathsf{ReLU}(x) - 8\mathsf{ReLU}(x - 1/4) + 8\mathsf{ReLU}(x - 1/2) - 8\mathsf{ReLU}(x - 3/4) + 4\mathsf{ReLU}(x - 1). \quad (61)$$

The number of ReLU functions required to represent g_m, if composition is not allowed, is $1 + 2^m$. Indeed, we need 2 ReLUs to represent each tooth and one more ReLU to flatten the function for $x > 1$. However, using function composition, $g_m(x)$ is represented by m layers with three ReLUs in each layer.

Telgarsky explores the approximation of oscillating functions with function-dependent and constant depth neural networks.

Consider a classification problem. Let $\{(x_i, y_i)\}_{i=1}^N$ be a collection of 2^k point-label pairs where $x_i \in [0, 1]$ and $y \in \{0, 1\}$ $\forall i$. The classification problem aims at defining a function f such that $f(x_i) = y_i$. More generally, x_i may belong to $\mathbb{R}^n$, but it suffices to consider $x_i \in [0, 1]$ for our purpose. The classifier function $\tilde{f}$ for a given function f is defined as

$$\tilde{f}(x) = \begin{cases} 1, & x \ge \frac{1}{2} \\ 0, & x < \frac{1}{2} \end{cases}. \quad (62)$$

The classification error is measured as the fraction of misclassified points:

$$\mathcal{R}(f) := \frac{1}{N} \sum_{i=1}^{N} |\tilde{f}(x_i) - y_i|. \quad (63)$$

Theorem 11 ([Telgarsky(2015)])**.** *Let positive integer k, number of layers l, and number of nodes per layer m be given with $m \le 2^{(k-3)/l-1}$. Then there exists a collection of 2^k point-label pairs $\{(x_i, y_i)\}_{i=1}^{2^k}$ with $x_i \in [0, 1]$ and $y \in \{0, 1\}$ $\forall i$ such that*

$$\min_{f \in \mathcal{N}(\mathrm{ReLU}; 2, 2k)} \mathcal{R}(f) = 0 \quad \text{and} \quad \min_{f \in \mathcal{N}(\mathrm{ReLU}; m, l)} \mathcal{R}(f) \ge \frac{1}{6}, \quad (64)$$

where $\mathcal{N}(\sigma; m, l)$ denotes the set of all neural networks with l layers and at most m nodes (neurons) per layer.

The proof of this theorem is based on choosing f being an alternating sign function on $\{x_i\}_{i=1}^n$ and an analysis of the effects of the composition and sum operations on the sawtooth functions. Note that, if one is not concerned with the behavior of the hat function g (Eq. (80)) outside the interval $[0,1]$ then suffices to use two ReLUs to express it.

The three main points made by Telgarsky are [Telgarsky(2016)]:

1. Functions with few oscillations poorly approximate functions with many oscillations.
2. Functions computed by networks with few layers must have few oscillations.
3. Functions computed by networks with many layers can have many oscillations.

To reconcile these conclusions with Theorems 9 and 10, we note that the oscillatory functions of nondecreasing amplitude with the growth of the number of oscillations k tend to have L_∞ norms growing exponentially with the order of the derivative m. For example, consider $f(x) = 1 + 0.5\sin(k\pi x)$ with $k \geq 1$ on $x \in [0,1]$. Its mth derivative is

$$\frac{d^m}{dx^m}\left(1 + \frac{1}{2}\sin(k\pi x)\right) = \frac{1}{2}(k\pi)^m \begin{cases} (-1)^{m/2}\sin(k\pi x), & m \text{ is even} \\ (-1)^{(m-1)/2}\cos(k\pi x), & m \text{ is odd} \end{cases}.$$

Hence, to adjust this function so that it belongs to the unit ball $\mathcal{B}_\infty^s(\mathbb{R})$ we must scale it by the factor $2(k\pi)^{-s}$ which is exponentially small.

3.6 Computation units

An important step in the benefit of depth of neural networks was accomplished by Dmitry Yarotsky [Yarotsky(2017)]. To simplify and generalize the analysis of the approximation power of deep neural networks,

Yarotsky defined the *computation unit* as

$$y = \sigma(w \cdot x + b), \quad w \in \mathbb{R}^N \quad b \in \mathbb{R}, \tag{65}$$

where the weights w and the bias b are adjustable parameters, x is the input, and sigma is an activation function. The computational units are groupped in layers where the inputs of layer j may be the output of any layer $i < j$. The output layer of the neural network may have no activation function applied to it which is equivalent to making the activation

function of the output layer of the neural network identity. Note that this construct admits architectures of networks with connections in non-neighboring layers.

The estimates of the complexity of neural network may be given in the number of weights and the number of computational units.

3.6.1 The approximation of the square and product functions

Any function can be approximated by a polynomial on a compact set (e.g. a unit hypercube) in $\mathbb{R}^n$ can be approximated by a polynomial according to the *Stone-Weierstrass theorem*. Any polynomial over $\mathbb{R}^n$ is a linear combination of monomials. Any monomial over $\mathbb{R}^n$ can be written using the product operations: $\times(x, y) = xy$. In turn, the product function can be written as a linear combination of squares:

$$xy = \frac{1}{2}\left((x+y)^2 - x^2 - y^2\right). \tag{66}$$

Thus, we review Yarotsky's approximation of square and product functions by deep ReLU neural networks.

Consider the function $f(x) = x^2$ on $[0,1]$. We construct its approximation using the sawtooth functions $g_m(x)$ as shown in Fig. 7. We start with $f_0 = x$. The maximum of the

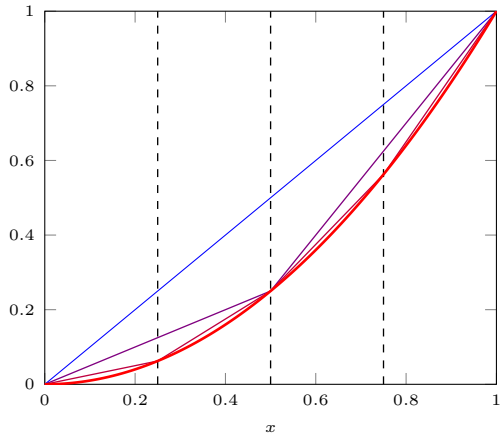

Figure 7: The approximation of $f(x) = x^2$ on $[0,1]$ using sawtooth functions g_m, $m = 1, 2, \ldots$.

error

$$e_0(x) = f_0 - x = x^2 - x \tag{67}$$

is achieved at $x = \frac{1}{2}$ and is equal to $\frac{1}{4}$. Then we update the approximation to

$$f_1(x) = f_0 - \frac{1}{4} g_1(x). \tag{68}$$

The maximum of the error

$$e_1(x) = f_1 - f(x) \tag{69}$$

is achieved at $x = \frac{1}{4}$ and $x = \frac{3}{4}$ and is equal to $\frac{1}{16}$. We update the approximation as Then we update the approximation to

$$f_2(x) = f_1 - \frac{1}{16} g_2(x). \tag{70}$$

And so on. Therefore, the mth approximation is

$$f_m(x) = f_{m-1} - \frac{1}{2^{2m}} g_m(x), \quad m = 1, 2, \ldots. \tag{71}$$

and the maximum of the error of this approximation,

$$e_m = f_m(x) - x^2, \quad m = 1, 2, \ldots, \tag{72}$$

is equal to 2^{-2m-2}

Exercise 1. Calculate the maximal error of the approximation x^2 by its linear interpolant on the interval $[a, b]$ and find the point where this maximal error is achieved.

2. Use this result to show that equation (72) is correct, and that $\max_{x \in [0,1]} e_m(x) = 2^{-2m-2}$.

From (72) we find that

$$f_m(x) = x - \sum_{k=1}^{m} \frac{g_k}{2^{2k}} \tag{73}$$

Noting that

$$\max_{x \in [0,1]} |e_m(x)| \equiv \|e_m\|_\infty = 2^{-2m-2} \to 0 \quad \text{as} \quad m \to \infty, \tag{74}$$

we conclude that f_m converges to $f(x) = x^2$ uniformly on $[0, 1]$. Now the following proposition can be readily proven.

Proposition 3 ([Yarotsky(2017)]). *For any $\epsilon > 0$, the function $f(x) = x^2$ of the segment $[0, 1]$ can be approximated by a* ReLU *deep neural network so that the maximum of the approximation error does exceed ϵ and the depth, the number of weights, and the number of computation units are $O(\log(\epsilon^{-1}))$.*

Proof. WE pick m such that $e^{-2m-2} < \epsilon$ and approximate $f(x) = x^2$ with f_m given by (73). The number of functions g_k is m. Each g_k is a composition by k functions g each of which, in turn, is a linear combination of three ReLUs – see equation (82). This ReLU network is shown in Fig. 8. Therefore, we can rewrite (73) a deep ReLU network of

- depth m,
- with the number of weights $3m + 1 + 9m$ (the number of edges in Fig. 8),
- and the number of computation units $1+3m$ (the number of nodes in Fig. 8 excluding the input node).

Noting that $m = \text{ceil}[\frac{1}{2}\log_2(\epsilon^{-1}) - 1]$ and that the numbers of weights and computation units scale linearly with m we obtain the desired scaling $O(\log(\epsilon^{-1}))$. □

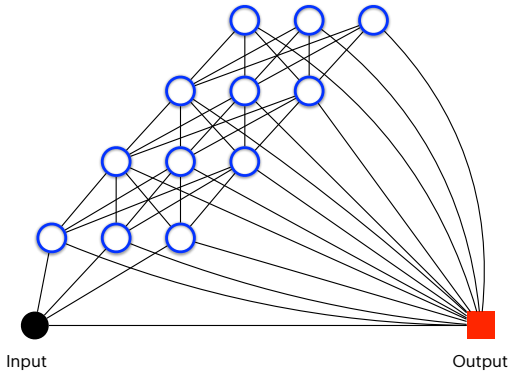

Figure 8: The architecture of the ReLU neural network approximating x^2 by equation (73) with $m = 4$.

The error bound for the approximation of the product function on the square $[-M, M] \times [-M, M]$ is given the following proposition.

Proposition 4 ([Yarotsky(2017)]). *Given $M > 0$ and $\epsilon > 0$, there is a* ReLU *neural network $\mathcal{N}$ with two input units that implements a function $\tilde{\times} : \mathbb{R}^2 \to \mathbb{R}$ so that*

(a) for any inputs x and y, if $|x| \le M$ and $|y| \le M$, then $|\tilde{\times}(x,y) - xy| \le \epsilon$;

(b) if $x = 0$ or $y = 0$ then $\tilde{\times}(x,y) = 0$;

(c) the depth and the number of weights and computation units in $\mathcal{N}$ are bounded by $c_1 \log(\epsilon^{-1}) + c_2$ where c_1 is an absolute constant and c_2 depends on M but not on ϵ.

Proof. Motivated by (66), we define the function $\times_m(x,y)$ as

$$\times_m(x,y) = 2M^2 \left[f_m\left(\frac{x+y}{2M}\right) - f_m\left(\frac{x}{2M}\right) - f_m\left(\frac{y}{2M}\right)\right], \tag{75}$$

where f_m is given by (73). Then

$$\begin{aligned}
|\times_m(x,y) - xy| &= \left| 2M^2 \left[f_m\left(\frac{x+y}{2M}\right) - f_m\left(\frac{x}{2M}\right) - f_m\left(\frac{y}{2M}\right)\right] - 4M^2 \frac{x}{2M}\frac{y}{2M} \right| \\
&= 2M^2 \left| \left[f_m\left(\frac{x+y}{2M}\right) - f_m\left(\frac{x}{2M}\right) - f_m\left(\frac{y}{2M}\right)\right] - \left[\frac{(x+y)^2}{4M^2} - \frac{x^2}{4M^2} - \frac{y^2}{4M^2}\right] \right| \\
&\le 2M^2 \left[\left| f_m\left(\frac{x+y}{2M}\right) - \frac{(x+y)^2}{4M^2} \right| + \left| f_m\left(\frac{x}{2M}\right) - \frac{x^2}{4M^2} \right| + \left| f_m\left(\frac{y}{2M}\right) - \frac{y^2}{4M^2} \right| \right] \\
&\le 2M^2 \left[2^{-2(m+1)} + 2^{-2(m+1)} + 2^{-2(m+1)} \right] = 6M^2 2^{-2(m+1)}.
\end{aligned}$$

In this calculation, we used the fact that $|x/(2M)| \in [0,1]$ and (74). To make the last expression less than ϵ, we choose

$$m = \operatorname{ceil}\left(\frac{1}{2}\log_2\left(\frac{3M^2}{2}\right) + \frac{1}{2}\log_2(\epsilon^{-1}) \right). \tag{76}$$

This proves item (a). Item (b) readily follows from (75) and (73). Item (c) follows from equations (76) and (75) and the fact that the depth and the number of weights and computation units linearly depend on m – see the proof of Proposition 3. □

3.7 Convergence rates for the approximation of smooth functions with ReLU networks

As [Pinkus(1999)], Yarotsky [Yarotsky(2017)] considered the unit ball

$$\mathcal{B}^s_\infty(\mathbb{R}^n) := \left\{ f \in \mathcal{W}^s_p \mid \|f\|_{s,p} \le 1 \right\} \tag{77}$$

in the functional space with the uniform norm. This ball consists of all functions with $s-1$ continuous derivatives and sth derivative being defined in a weak sense (i.e., it may

be piecewise continuous), and all these derivatives do not exceed one in absolute value. Contrary to the theorems presented in [Pinkus(1999)], the activation functions are ReLU, i.e., not continuously differentiable.

Theorem 12 ([Yarotsky(2017)]). *For any dimension n, number of weak derivatives s, there is a* ReLU *network architecture that*

(a) is capable of approximating $f \in \mathcal{B}^s_\infty(\mathbb{R}^n)$ with error ϵ, i.e., there is a weight assignment to the achitecture such that the resulting network function $\mathcal{N}$ satisfies $\|f - \mathcal{N}\|_\infty < \epsilon$;

(b) has the depth of at most $c(\log(\epsilon^{-1}) + 1)$ and at most $c\epsilon^{-n/s}(\log(\epsilon^{-1}) + 1)$ weights and computation units, with some c dependent on n and s but not on f.

The proof of this theorem is conducted via constructing the desired approximation using a partition of unity by piecewise-linear functions and Taylor polynomials. These Taylor polynomials, in turn, are constructed out of the product functions that are represented by ReLU networks as in Proposition 4. The proof takes several pages. I am sending interested students to [Yarotsky(2017)].

How does the scaling of the number of computation units in Yarotsky's Theorem 12 compare to the ones in Theorem 9 by Maiorov and Meir Theorem 10 by Mhaskar? Recall that the width of a one-hidden layer neural network r is exactly the number of computation units in Yarotsky's terminology. Expressing r via ϵ from Mhaskar's upper bound $Cr^{-s/n} = \epsilon$, we get $r = O(\epsilon^{-n/s})$. Therefore, the scaling of the numbers of computational units in Yarotsky's work and Maiorov's and Meir's and Mhaskar's works are similar, about $\epsilon^{-n/s}$. The scaling prefactors may be drastically different. An important difference is that Yarotsky abandoned the smoothess requirement for the activation function.

[Yarotsky(2017)] also proved that if the architecture of a neural network allowed to depend on the function to be approximated, its approximation may be more efficient.

Theorem 13 ([Yarotsky(2017)]). *For any $f \in \mathcal{B}^1_\infty(\mathbb{R})$ and any $\epsilon \in (0, \frac{1}{2})$, there exists a depth-6* ReLU *network $\mathcal{N}$ with architecture depending on f such that $\|t - \mathcal{N}\|_\infty < \epsilon$ and $\mathcal{N}$ has not more than $c\epsilon^{-1}[\log(\epsilon^{-1})]^{-1}$ weights and computation units, where c is an absolute constant.*

This theorem shows that adaptive architecture of a ReLU network only affects the complexity of a neural network by a log factor.

Yarotsky conducted further investigation of benefits of depth in [Yarotsky(2017)]. The bottom line is that the depth of ReLU neural network as well as adaptive architecture

affects scaling constants and log factors, but the curse of dimensionality manifesting itself via the factor $\epsilon^{-n/s}$ is still there.

3.8 Scaling of the width and depth in ReLU networks

Jianfeng Lu (Duke), Zuowei Shen (National University of Singapore), Haizhao Yang (UMD), Shijun Zhang (National University of Singapore) proved an important theorem showing how the approximation error of a smooth function scales with the width and depth of a feed-forward fully-connected neural network with ReLU activation functions.

Theorem 14 ([Lu et al.(2021)Lu, Shen, Yang, and Zhang]). *Given a smooth function $f \in C^s([0,1]^n)$ with $s \in \mathbb{N}$, for any $N, L \in \mathbb{N}$, there exists a function ϕ implemented by a* ReLU *feedforward neural network with width $C_1(N+1)\log_2(8N)$ and depth $C_2(L+2)\log_2(4L)+2n$ such that*

$$\|\phi - f\|_\infty \le C_3 \|f\|_{C^s([0,1]^n)} N^{-2s/n} L^{-2s/n}, \tag{78}$$

where $C_1 = 17s^{n+1}3^n n$, $C_2 = 18s^2$, and $C_3 = 85(s+1)^n 8^s$.

The norm of f in the right-hand side of (78) is the maximal uniform norm of all derivatives of f up to order s.

In simpler terms, Theorem 14 says that a feedforward ReLU network with width $O(N \log N)$ and depth $O(L \log L)$ can approximate a function with continuous derivatives up to order s with error $O(N^{-2s/n} L^{-2s/n})$. The dimension n in the denominator of the power shows the curse of dimensionality. The factor of 2 improves the previous estimates with $r^{-s/n}$. The scaling is given for the type of neural networks widely used by researchers – ReLU feedforward networks with prescribed width and depth. The factor $\|f\|_{C^s([0,1]^n)}$ in the right-hand side of (78) shows the proportionality of the derivative growth and the approximation error.

3.9 Approximation with special activation functions

As we have seen, ReLU neural network approximations of smooth high-dimensional functions suffer from the curse of dimensionality. So is true for one-hidden-layer neural networks with smooth activation functions. [Shen et al.(2021)Shen, Yang, and Zhang] showed that one can use specially crafted activation functions to overcome the curse of dimensionality in terms of the number of weights. These activation functions are Floor, Exponential, and

Step. The neural networks with three hidden layers with these activation functions are called, respectively, FLES networks. The floor function of $x \in \mathbb{R}$ returns the largest integer less than or equal to x. The exponential is 2^x. The step function is equal to zero on $[i, i+\frac{1}{2})$, and is equal to one on $[i+\frac{1}{2}, i+1)$, $i \in \mathbb{Z}$. Interested students can find more details in the referenced paper.

3.10 Connection to finite elements in 2D

Reference: Juncai He, Lin Li, Jinchao Xu, *ReLU Deep Neural Networks from the Hierarchical Basis Perspective, Computer & Mathematics with Applications, Vol. 120, 15 August 2022, pp. 105–114, arXiv:2105.04156.*

Let us fix a triangulation with the set of nodes (x_j, y_j), $1 \le j \le N$, and fineness h in a rectangular domain $\Omega \subset \mathbb{R}^2$ constructed out of a regular rectangular mesh as shown in Fig. 9. We define the set of finite element basis functions

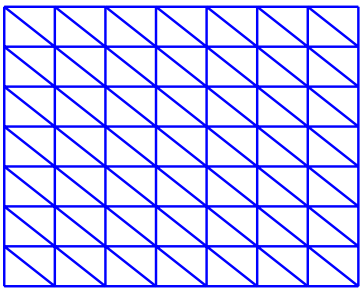

Figure 9: The hat-function g and its composition with itself g_2.

$$\eta_j(x,y) = \begin{cases} 1, & (x,y) = (x_j, y_j), \\ 0, & (x,y) = (x_i, y_i),\ i \neq j, \\ \text{linear}, & \text{on each triangle} \\ \text{continuous}, & (x,y) \in \Omega. \end{cases} \tag{79}$$

It follows from the classical approximation theory that any twice continuously differentiable function $f : \Omega \to \mathbb{R}$ can be approximated by its linear interpolant $I_h f$ with the set of collocation points (x_j, y_j), $1 \le j \le N$, with accuracy $O(h^2)$. Now we will follow Juncai He, Lin Li, Jinchao Xu (2021) to show that f can be approximated by a 2-layer ReLU neural network with $15N$ neurons per layer.

Recall the hat function $g : \mathbb{R} \to \mathbb{R}$ is defined by

$$g(x) = \begin{cases} 2x, & x \in [0, 1/2), \\ 2(1-x), & x \in [1/2, 1], \\ 0, & \text{otherwise.} \end{cases} \tag{80}$$

We encountered it before in Telgarsky's and Yarotsky's works. The function $g_2 : \mathbb{R} \to \mathbb{R}$ is defined as

$$g_2(x) = g(g(x)). \tag{81}$$

The graphs of g and g_2 are shown in Fig. 6.

It is easy to check that $g(x)$ can be expressed as a linear combination of three ReLU functions:

$$g(x) = 2\mathsf{ReLU}(x) - 4\mathsf{ReLU}(x - 1/2) + 2\mathsf{ReLU}(x - 1). \tag{82}$$

Indeed,

$$2\mathsf{ReLU}(x) - 4\mathsf{ReLU}(x - 1/2) + 2\mathsf{ReLU}(x - 1) = \begin{cases} 0, & x \le 0, \\ 2x, & x \in [0, 1/2), \\ 2x - 4x + 2 = 2(1-x), & x \in [1/2, 1], \\ 2x - 4x + 2 + 2x - 2 = 0, & x > 1 \end{cases} = g(x).$$

Therefore, the function g_2 can be represented as a two-layer ReLU neural network with three neurons in each layer. In fact, g_2 can be represented as a linear combination of five ReLU functions, i.e., as a one-layer ReLU neural network with 5 neurons:

$$g_2(x) = 4\mathsf{ReLU}(x) - 8\mathsf{ReLU}(x - 1/4) + 8\mathsf{ReLU}(x - 1/2) - 8\mathsf{ReLU}(x - 3/4) + 4\mathsf{ReLU}(x - 1). \tag{83}$$

Now we consider the standard FEM basis function $\phi(x, y)$ on the unit square $[0, 1]^2$ shown in Fig. 10. The basis functions $\eta_j(x, y)$ (79) are obtained from $\phi(x, y)$ via appropriate affine transformations:

$$\eta_j(x, y) = \phi\left(\frac{x - x_j}{h}, \frac{y - y_j}{h}\right). \tag{84}$$

One can verify that $\phi(x, y)$ can be expressed as a linear combination of three g_2 functions:

$$\phi(x, y) = \frac{1}{2}\left[g_2\left(\frac{x}{2}\right) + g_2\left(\frac{y}{2}\right) - g_2\left(\frac{x + y}{2}\right)\right]. \tag{85}$$

The problem with representation (85) is that it is valid on $[0,1]^2$ but not globally. The right-hand side in (85) is equal to $1/2$ at $(3/2, 3/2)$ rather than 0. The global representation of $\phi(x,y)$ can be constructed by applying the function

$$\mathsf{ReLU1}(x) = \mathsf{ReLU}(x) - \mathsf{ReLU}(x-1) = \begin{cases} 0, & x < 0, \\ x, & x \in [0,1], \\ 1, & x > 1, \end{cases} \tag{86}$$

to the arguments in the right-hand side of (85):

$$\phi(x,y) = \frac{1}{2}\left[g_2\left(\frac{\mathsf{ReLU1}(x)}{2}\right) + g_2\left(\frac{\mathsf{ReLU1}(y)}{2}\right) - g_2\left(\frac{\mathsf{ReLU1}(x) + \mathsf{ReLU1}(y)}{2}\right)\right]. \tag{87}$$

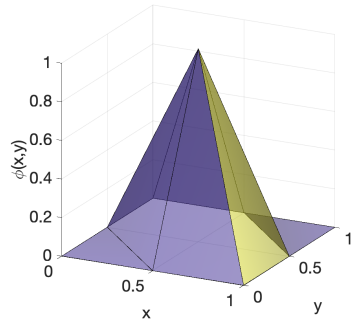

Figure 10: The standard basis finite element hat-function in 2D.

Eqs. (83) and (87) imply that the linear interpolant $I_h f(x,y)$ of a function f

$$I_h f(x,y) = \sum_{j=1}^{N} f(x_j, y_j)\eta_j(x,y) = \sum_{j=1}^{N} f(x_j, y_j)\phi\left(\frac{x-x_j}{h}, \frac{y-y_j}{h}\right). \tag{88}$$

can be represented as a two-layer neural network with at most $15N$ neurons per layer.

References

[Farouki(2012)] Rida T. Farouki. The bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29(6):379–419, 2012. ISSN 0167-8396. doi: https://doi.org/10.1016/j.cagd.2012.03.001. URL `https://www.sciencedirect.com/science/article/pii/S0167839612000192`.

[Gil et al.(2007)Gil, Segura, and Temme] J. Gil, P. Segura, and N. Temme. *Numerical Methods for Special Functions*. SIAM, 2007. ISBN 978-0-89871-634-4. doi: 10.1137/1.9780898717822.

[Trefethen(2000)] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2000. URL https://epubs.siam.org/doi/book/10.1137/1.9780898719598.

[Pinkus(1999)] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

[Yarotsky(2017)] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2017.07.002. URL https://www.sciencedirect.com/science/article/pii/S0893608017301545.

[He et al.(2022)He, Li, and Xu] Juncai He, Lin Li, and Jinchao Xu. Relu deep neural networks from the hierarchical basis perspectiveimage 1. *Computers Mathematics with Applications*, 120:105–114, 2022. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2022.06.006. URL https://www.sciencedirect.com/science/article/pii/S0898122122002516.

[Lu et al.(2021)Lu, Shen, Yang, and Zhang] Jianfeng Lu, Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation for smooth functions. *SIAM Journal on Mathematical Analysis*, 53(5):5465–5506, 2021. doi: 10.1137/20M134695X. URL https://doi.org/10.1137/20M134695X.

[Shen et al.(2021)Shen, Yang, and Zhang] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Neural network approximation: Three hidden layers are enough. *Neural Networks*, 141:160–173, 2021. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2021.04.011. URL https://www.sciencedirect.com/science/article/pii/S0893608021001465.

[Cybenko(1989)] George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989. URL https://link.springer.com/article/10.1007/BF02551274.

[Hastad(1987)] Johan Hastad. *Computational Limitations of Small-Depth Circuits*. MIT Press (MA), 1987.

[Telgarsky(2015)] Matus Telgarsky. Representation benefits of deep feedforward networks. *ArXiv*, abs/1509.08101, 2015. URL https://api.semanticscholar.org/CorpusID:1219694.

[Telgarsky(2016)] Matus Telgarsky. Benefits of depth in neural networks. In *Annual Conference Computational Learning Theory*, 2016. URL `https://api.semanticscholar.org/CorpusID:8145558`.