Verification of Control Systems with Neural Networks

pippia.eleonora

1. Introduction

The problem of verify closed-loop systems with neural net controllers has recently attracted a lot attention due to the increasing use of AI components in cyber-physical systems. [...]

2. Problem Statement

We consider a continuous-time system S described by

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{1}$$
$$\mathbf{u}(t_k) = N_C(\mathbf{x}(t_k)) \tag{2}$$

where $\mathbf{x}$ is the state variables, $\mathbf{u}$ is the control input that is updated at discrete time points t_k by a neural network controller N_C with $t_k = t_{k-1} + \delta_{k-1}$, $k \in \{1, 2, \dots\}$ and $\delta_{k-1} > 0$. Between these time points the control input remains constant.

The controller N_C is a feedforward neural network. Each neural network can be described layer by layer. Given a neural network N with l layers we have

$$u = N(x) = N_1 \circ \cdots \circ N_l(x) \tag{3}$$

with $N_i(x) = \sigma(W^i x + b^i)$ and σ is the activation function. $W^i \in \mathbb{R}^{n \times m}$ with n_i number of neurons in layer i and m number of input in each neuron.

We usually have three types of activation function:

- ReLU: $\sigma(x) = \max(0, x)$
- sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- tanh: $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$

We assume that the neural network controller N_C is an approximation of a controller C which was designed so that the closed-loop system $S - C$ satisfies a property $\mathcal{P}$. Our goal is to check whether the closed-system $S - N_C$ guarantees $\mathcal{P}$. We can sketch an abstract algorithm (Algorithm 1) to perform reachability on this system. Step five of Algorithm 1 can be done in principle by the existing reachability algorithms. In this scheme what is missing is the computation of the image $N_C(X_i)$.

Different authors are focusing on the problem of over-approximate the reachable region of a neural networks ([IWA+19, DCJ+19, HFL+19]). In particular in [HFL+19] they are trying to approximate a neural network with the Bernstein polynomial operator. This approach does not take advantage of the structure of the neural network and on the particular type of activation functions that we can consider. It can be proved that a ReLU and a sigmoid function can be approximated by a rational function $p(\mathbf{x})$ (ratio of two polynomials) as accurately as desired ([Tel17]). Following this idea, we want to approximate each layer with a rational function and compute the image using the Bernstein expansion and assuming that the input lies inside a polytope.

Algorithm 1 Closed-loop reachable set computation

```
function REACH(X_0, k_max)
    X_0 ⊂ R^n set of initial states, k_max ∈ N
    for i = 0, ..., k_max do
        U_i ← N_C(X_i)
        X_{i+1} ←Reach_f(X_i, U_i, δ_i)
    end for
end function
```

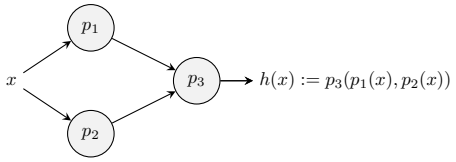
First of all we recall some definition for the Bernstein expansion over a hyperrectangle and then we address the problem of over-approximate a polynomial expression over a polytope. Be able to generalize the Bernstein properties over a polytope seems mandatory to increase the accuracy of the over-approximation. In Example 2.1 we design a scenario that we will avoid using the polytopes generalization.

Example 2.1. *Let consider 3 different maps:*

$p_1 : \mathbb{R} \to \mathbb{R}$ *such as* $p_1(x) = x$.

$p_2 : \mathbb{R} \to \mathbb{R}$ *such as* $p_2(x) = -x$.

$p_3 : \mathbb{R}^2 \to \mathbb{R}$ *such as* $p_3(x, y) = x + y$.



Now let image that we want to compose this three maps such as $h(x) = p_3(p_1(x), p_2(x))$. *In this toy example it is clear that* $h(x) \equiv 0$. *What it is important to notice is that if we consider* $x \in [0, 1]$ *and we approximate the*

composition of these function with hyperboxes we obtain that $p_1(x) \in [0,1]$ *and* $p_2(x) \in [-1,0]$ *so the hyperbox* $[0,1]\times[-1,0]$ *and finally* $p_3(x,y) \in [-1,1]$. *The problem is that we are loosing the relation between the input* x *and the output of* p_3.

3. Bernstein polynomial

The following is developed for function $f : \mathbb{R}^n \to \mathbb{R}$ but can be easily expanded to $f : \mathbb{R}^n \to \mathbb{R}^m$ just repeating each step for each f_j, $j = 1, \dots, m$.

Every polynomial $p : \mathbb{R}^n \to \mathbb{R}$ can be represented using the power basis as follows:

$$p(\mathbf{x}) = \sum_{\mathbf{i}\in I^n} a_{\mathbf{i}}\mathbf{x}^{\mathbf{i}} \tag{4}$$

where the index $\mathbf{i} = (i_1, \dots, i_n)$ is a multi-index of size $n \in \mathbb{N}$ and $\mathbf{x}^{\mathbf{i}}$ denotes the monomial $x_1^{i_1}x_2^{i_2}\dots x_n^{i_n}$.

We want to represent $p(\mathbf{x})$ using Bernstein basis. We need some additional definitions to continue. Given two multi-index $\mathbf{i} = (i_1, \dots, i_n)$ and $\mathbf{j} = (j_1, \dots, j_n)$ we write $\mathbf{j} \le \mathbf{i}$ if $j_k \le i_k$ for $k = 1, \dots, n$. We write $\mathbf{i}/\mathbf{j}$ for the multi-index $(i_1/j_1, \dots, i_n/j_n)$ and $\binom{\mathbf{i}}{\mathbf{j}}$ for the product $\binom{i_1}{j_1}\dots\binom{i_n}{j_n}$. Finally we define the degree $\mathbf{deg}$ of p as the smallest multi-index such as $\mathbf{i} \le \mathbf{deg}$ for all $\mathbf{i} \in I^n$. In the following we will refer to $I^{\mathbf{deg}} = \{\mathbf{i} \in \mathbb{N}^n \mid \mathbf{i} \le \mathbf{deg}\}$

A polynomial $p(\mathbf{x})$ can be represented using Bernstein basis as

$$p(\mathbf{x}) = \sum_{\mathbf{i}\in I^{\mathbf{deg}}} b_{\mathbf{i}}(p) B_{(\mathbf{deg},\mathbf{i})}(\mathbf{x}) \tag{5}$$

where $b_{\mathbf{i}}(p)$ are called *Bernstein coefficient* for a polynomial p and they are defined as

$$b_{\mathbf{i}}(p) = \sum_{\mathbf{j}\le\mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{deg}}{\mathbf{j}}} a_{\mathbf{j}} \tag{6}$$

and the $\mathbf{i}$-th *Bernstein polynomial* $B_{(\mathbf{deg},\mathbf{i})}$ of degree $\mathbf{deg}$ is

$$B_{(\mathbf{deg},\mathbf{i})}(\mathbf{x}) = \beta_{(d_1,i_1)}(x_1)\dots\beta_{(d_n,i_n)}(x_n) \tag{7}$$

with $\mathbf{deg} = (d_1, \dots, d_n)$ and

$$\beta_{(d_j,i_j)}(x) = \binom{d_j}{i_j} x^{i_j}(1-x)^{d_j-i_j} \tag{8}$$

Bernstein coefficients present an interesting property that can be used to compute them.

Property 3.1. (Sharpness) *For all* $\mathbf{i} \in \mathcal{V}_{\mathbf{deg}}$

$$b_{\mathbf{i}}(p) = p(\mathbf{i}/\mathbf{deg}) \tag{9}$$

with $\mathcal{V}_{\mathbf{deg}}$ *the set of vertices of the hyperrectangle* $[0, d_1] \times \dots [0, d_n]$ *with* $\mathbf{deg} = (d_1, \dots, d_n)$.

If we consider the minimum and the maximum coefficient we obtain two bounds on the polynomial.

Property 3.2. (Range Enclosing)

$$\min_{\mathbf{i}\in I^{\mathbf{deg}}} b_{\mathbf{i}}(p) \le p(\mathbf{x}) \le \max_{\mathbf{i}\in I^{\mathbf{deg}}} b_{\mathbf{i}}(p) \tag{10}$$

for all $\mathbf{x} \in [0,1]^n$

Property 3.2 is true on the unitary hyperbox, but can be easily extended to a hyperrectangle [Dre17]. This property is fundamental to compute the hyperrectangle but does not maintain the relation between the input and the output. In order to obtain a better approximation, we are more interesting on the property that maintain a convex structure between the input/output points and their over-approximation.

Definition 3.1. Let $V = \{v_1, \dots, v_l\}$ be a sequence of points in $\mathbb{R}^n$. The *convex hull* of V is the set

$$\text{Convex}_{\text{hull}}(V) = \{y \in \mathbb{R}^n \mid y = \sum_{i=1}^{l} \alpha_i v_i, \sum_{i=1}^{l} \alpha_i = 1,\ \alpha_i \ge 0\}$$

Property 3.3. (Convex hull) *Let* $p \in [0,1]^n \to \mathbb{R}^m$ *be a polynomial of degree* $\mathbf{deg}$, *we have*

$$\begin{pmatrix}\mathbf{x}\\ p(\mathbf{x})\end{pmatrix} \subseteq \mathit{Convex}_{\mathit{hull}}\Big(\Big\{\begin{pmatrix}v_{\mathbf{i}}\\ b_{\mathbf{i}}(p)\end{pmatrix} \mid \mathbf{i} \in I^{\mathbf{deg}}\Big\}\Big) \tag{11}$$

were $v_{\mathbf{i}} := \frac{\mathbf{i}}{\mathbf{deg}}$ *and* $(v_{\mathbf{i}}, b_{\mathbf{i}}(p))$ *is called* control point *associated with the* $\mathbf{i}$*th Bernstein coefficient.*

In order to be able to iterate the procedure throw the layers of a neural networks, without loosing information, we need to define the Bernstein expansion over a polytope.

4. Generalized Bernstein polynomial

NOTE: starting from here I will not use the bold notation for vectors.

Definition 4.1. Let $V = \{v_1, \dots, v_m\}$ be a sequence of points in $\mathbb{R}^n$. V is *affinely independent* if

$$\forall x \in \mathbb{R}^n\ \exists!\xi(x) \in \mathbb{R}^m : x = \sum_{i=1}^{m} \xi_i(x) v_i, \quad \sum_{i=1}^{m} \xi_i(x) = 1$$

The functions ξ_i so defined are called *barycentric coordinates*. The functions $\xi : \mathbb{R}^n \to \mathbb{R}$ are affine functions, which are nonnegative on the convex hull of the points.

Let $v_0, \dots, v_n$ be $n+1$ points in $\mathbb{R}^n$. The ordered list $V = [v_0, \dots, v_n]$ is called *simplex* of the vertices $v_0, \dots, v_n$. The *realization* of a simplex V is the set of $\mathbb{R}^n$ defined as the convex hull of the points $v_0, \dots, v_n$. If the $n+1$ points are affinely independent, the simplex is non-degenerete. We denote by $e_1, \dots, e_n$ the canonical basis of $\mathbb{R}^n$ and by e_0 the zero vector in $\mathbb{R}^n$. We denote with $\Delta := [e_0, e_1, \dots, e_n]$ the *standard simplex*. If $x = (x_1, \dots, x_n) \in \Delta$ then the barycentric coordinates with respect to Δ are $\xi(x) = (\xi_0(x), \dots, \xi_n(x)) = (1 - |x|, x_1, \dots, x_n)$ with $|x| = x_1 + \dots + x_n$.

For a finite sequence of points, which are possibly not affinely independent, we can define a generalisation of the barycentric coordinates that are not unique anymore.

Given an $x \in \mathbb{R}$ and $k \in \mathbb{N}$ we have

$$1 = 1^k = (x + (1-x))^k = \sum_{i=1}^{k} \binom{k}{i} x^i (1-x)^{k-i} = \sum_{i=1}^{k} \beta_{(k,i)}(x)$$

Given a vector $\xi = \{\xi_1, \dots, \xi_m\}$ with $\sum_i \xi_i = 1$ we introduce the following notation for multinomial expansion

$$(\xi_1 + \cdots + \xi_m)^k = \sum_{\alpha_1 + \cdots + \alpha_m = k} \binom{k}{\alpha_1, \dots, \alpha_m} \xi_1^{\alpha_1} \dots \xi_m^{\alpha_m} = \sum_{|\alpha|=k} \binom{|\alpha|}{\alpha} \xi^\alpha$$

with $\alpha \in \mathbb{N}^m$ and

$$\binom{k}{\alpha} = \binom{k}{\alpha_1, \dots, \alpha_m} = \frac{k!}{\alpha_1! \dots \alpha_m!}$$

Definition 4.2. Given $V = \{v_1, \dots, v_m\}$ and a polynomial $p : Convex_{hull}(\{v_1, \dots, v_m\}) \to \mathbb{R}$, the *generalised Bernstein polynomial* of degree k is

$$B_V^{(k)} p = \sum_{|\mathbf{i}|=k} b_{\mathbf{i}}^{(k)}(p, V) B_{\mathbf{i},V}^{(k)}$$

with $\xi = (\xi_1, \dots, \xi_m)$ the generalised barycentric coordinate with respect to V, and

$$B_{\mathbf{i},V}^{(k)}(\mathbf{x}) = \binom{|\mathbf{i}|}{\mathbf{i}} \xi^{\mathbf{i}}(\mathbf{x})$$

We can find different definition of generalized barycentric coordinates. It is not enough to define a general system of coordinates but we are searching for non-negative coordinates because we want to maintain Property 3.3. More over the coordinates should be easy enough to be generalized in n-dimension and we should also be able to compute the new Bernstein coefficients. In [Wal11, Do11] they define an easy extension for general polytope. Unfortunately this coordinates are not positive in the total convex hull. An alternative definition in given in [Lan08]. Here the coordinates are positive on the convex hull but the the construction is not easy to be generalized in n-dimensions. In both cases the estimation of the Bernstein coefficients is not explained. The majority of the generalized barycentric coordinates are developed just for 3D and used for graphical purposes [Rus07, LKCOL07, JMD+07, Flo03, WSHD07].

The easiest generalization that keeps the desired property is the Simplicial Bernstein generalization, i.e. the Bernstein base over a simplex [Far86].

4.1. Generalized Bernstein polynomial over a simplex

Given a general simplex $S = \{v_0, \dots, v_n\}$ with $v_i \in \mathbb{R}^n$ we can map affinely each point of its realization into the standard simplex Δ and vice-versa. In particular given $x \in \Delta$ we obtain $y \in S$ as

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = T(x) := V \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + v_0 \tag{12}$$

with

$$V = \begin{pmatrix} v_1(1) - v_0(1) & \dots & v_n(1) - v_0(1) \\ \vdots & & \vdots \\ v_1(n) - v_0(n) & \dots & v_n(n) - v_0(n) \end{pmatrix}$$

So, we start with the case of the standard simplex Δ. We introduce additional notations. Given $\alpha = (\alpha_0, \dots, \alpha_n) \in \mathbb{N}^{n+1}$ we indicate with $\hat{\alpha} = (\alpha_1, \dots, \alpha_n)$. For $\hat{\alpha}, \hat{\beta} \in \mathbb{N}^n$ with $\hat{\alpha} \le \hat{\beta}$, we use

$$\binom{\hat{\alpha}}{\hat{\beta}} := \prod_{i=1}^{n} \binom{\alpha_i}{\beta_i} \tag{13}$$

and if $|\hat{\alpha}| < k$ then

$$\binom{k}{\hat{\alpha}} := \binom{k}{\alpha_1, \dots, \alpha_n} = \frac{k!}{\alpha_1! \dots \alpha_n! (k - |\hat{\alpha}|)!} \tag{14}$$

Property 4.1. ([Ham18]) *Given a (multivariate) polynomial of degree l*

$$p(x) = \sum_{|\hat{\beta}| \le l} a_{\hat{\beta}} x^{\hat{\beta}}$$

the simplicial Bernstein form of p of degree $l \le k$ on Δ is

$$p(x) = \sum_{|\alpha|=k} b_\alpha^{(k)}(p, \Delta) B_\alpha^{(k)} \tag{15}$$

where

$$B_\alpha^{(k)} = \binom{k}{\alpha} \xi^\alpha \tag{16}$$

and

$$b_\alpha^{(k)}(p, \Delta) = \sum_{\hat{\beta} \le \hat{\alpha}} \frac{\binom{\hat{\alpha}}{\hat{\beta}}}{\binom{k}{\hat{\beta}}} a_{\hat{\beta}}, \quad |\hat{\alpha}| = k \tag{17}$$

Definition 4.3. Given the Bernstein coefficients of degree $k \ge l$ of a (multivariate) polynomial p of degree l over a general simplex S, the *control point* associated with the αth Bernstein coefficient $b_\alpha^{(k)}(p, S)$ is the point $(v_\alpha^{(k)}(p, S), b_\alpha^{(k)}(p, S))$ with

$$v_\alpha^{(k)}(p, S) := \frac{\alpha_0 v_0 + \cdots + \alpha_n v_n}{k} \in \mathbb{R}^n, \quad |\alpha| = k \tag{18}$$

Property 4.2. (Sharpness [LKCOL07]) *Keeping the same notations, we have*

$$b_{ke_i}^{(k)}(p, \Delta) = p(v_{ke_i}) \quad \forall i \in \{0, \dots, n\} \tag{19}$$

Property 4.3. (Convex Hull [Ham18]) *The polynomial p over S is contained within the convex hull of the control points*

$$\begin{pmatrix} x \\ p(x) \end{pmatrix} \subseteq Convex_{hull}\Big(\Big\{ \begin{pmatrix} v_\alpha^{(k)}(p,S) \\ b_\alpha^{(k)}(p,S) \end{pmatrix} \mid |\alpha| = k \Big\}\Big) \quad (20)$$

4.2. Matrix method to compute Bernstein coefficients

It is well known that we can compute the Bernstein coefficients over a hyperrectangle using a matrix product. This method was introduced by Garloff in [Gar85]. In [TG17], Garloff and Titi expand the matrix calculation for simplicial Bernstein expressions.

Let $q : \mathbb{R}^n \to \mathbb{R}$ be a polynomial defined over a simplex $S = \{v_0, \ldots, v_n\}$. Using the transformation 12 we define the polynomial $p(x) = q(T(x))$ over the standard simplex Δ.

$$p(x) = \sum_{|\hat{\alpha}| \le l} a_{\hat{\alpha}} x^{\hat{\alpha}} \quad (21)$$

The Bernstein polynomial has a degree k such as $l \le k$. We arranged the coefficients of p in an $(k+1) \times k^*$ matrix A with $k^* := (k+1)^{n-1}$

$$A := \begin{pmatrix} a_{0,0,\ldots,0} & a_{0,1,\ldots,0} & \cdots & a_{0,k,\ldots,0} & \cdots & a_{0,k\ldots,k} \\ a_{1,0,\ldots,0} & a_{1,1,\ldots,0} & \cdots & a_{1,k,\ldots,0} & \cdots & a_{1,k\ldots,k} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{k,0,\ldots,0} & a_{k,1,\ldots,0} & \cdots & a_{k,k,\ldots,0} & \cdots & a_{k,k\ldots,k} \end{pmatrix}$$

with $a_{\hat{\alpha}} = 0$ if $|\hat{\alpha}| > l$. We also define the lower triangular Pascal matrix P, i.e.

$$(P_k)_{i,j} = \begin{cases} \binom{i-1}{j-i} & j \le i \\ 0 & \text{otherwise} \end{cases} \quad (22)$$

We can compute $P_k = \prod_{h=1}^{k} K_h$ from the matrices K_h, $h = 1, \ldots, k$ defined as

$$(K_h)_{i,j} = \begin{cases} 1 & j \le i \\ 1 & i = j+1,\ j \ge k-h+1 \\ 0 & \text{otherwise} \end{cases} \quad (23)$$

In introduce the *cyclic ordering* of the entries of a matrix denoted with the superscript c, i.e. the order of the indices of the entries of the matrix under a cyclic permutation. This means that the index in the first position is replaced by the index in the second one, the index in i-th position by the one in the $i+1$-th position and the index in the n.th position by the one in the first position. In two dimension the cyclic ordering is the transposition. We put $C_0(\Delta) := A$ and define for $t = 1, \ldots, n$

$$C_t := (P_k C_{t-1})^c \quad (24)$$

The Bernstein coefficients of the polinomial p over Δ can be obtained for the entries of the matrix C_n considering just $(C_n)_{\hat{\alpha}}$ with $|\hat{\alpha}| \le k$.

$$b_\alpha^{(k)} = b_{(k-|\hat{\alpha}|,\hat{\alpha})}^{(k)} = (C_n)_{\hat{\alpha}} \quad (25)$$

Finally we compute v_α as

$$v_\alpha^{(k)} = T\left(\frac{\alpha_0 e_0 + \cdots + \alpha_n e_n}{k}\right) = T\left(\frac{\hat{\alpha}}{k}\right) \quad (26)$$

5. EXTRA NOTES

The idea now is to triangulate a polytope and compute for each simplex the control points. A sketch of a possible algorithm can be Algorithm 2.

Algorithm 2 Over-approximation for a polynomial composition

```
Input: P = {p_1, ..., p_n} polynomial function
       V_0 = {v_1, ..., v_m} vertices of the polytope that is the domain for p_1.
for i = 1, ..., n do
    split V_{i-1} into k simplicis S_1, ..., S_k.
    for j = 1, ..., k do
        compute b_α(S_j, p_i)
    end for
    V_i := {b_α(S_j, p_i)}
end for
```

Questions to be addressed:

1. To triangulate a polytope an easy way is use the delaunay triangulation (already implemented in Matlab). This function does not work when the polytope is degenerate or when there are at least $n+1$ points in n dimension over a sphere.
2. Given a set $U = \{u_1, \ldots, u_m\}$ of points in $\mathbb{R}^n$, do we need to compute the set $V \subseteq U$ of vertices of the convex hull of U? Which is the trade off?
3. For each simplex generate by triangulation we obtain a convex hull by Proposition 4.3. Do we need to keep each convex hull separately or we will compute the total convex hull? NOTE: if we keep them separately we do not have a convex domain anymore.
4. We should consider for each $i = 1, \ldots, n$ a polynomial function in $p_i : \mathbb{R}^{n_i} \to \mathbb{R}^{m_i}$. So we should repeat the step (7) for m_i times. If the j-th and the k-th components of p_i have a different degree, how we compute the convex hull? Should we take the maximum degree?
5. How to generalize to fractions (possibility: [Ham18])

Answers 1 for the degenerate case: In [Mat20] we can find two matlab function to convert a set of points in a set of linear constraints (equalities and inequalities) and vice versa. Given a set of point $V = \{v_1, \ldots, v_m\}$ with $v_i \in \mathbb{R}^n$, we can computer $[A, b, Aeq, beq] = \text{vert2lcon}(V)$ such as $A * x \le b$ and $Aeq * x = beq$ for $x \in Convex_{hull}(V)$. If Aeq is not empty than the convex hull is degenerate. *General Case:* $Aeq \in \mathbb{R}^{k \times n}$ with $k \le n$. If $k = n$ that the convex hull is a single point and the next computations are trivial. So we suppose $k < n$. We can split the matrix in

two submatrix $Aeq_1 \in \mathbb{R}^{k\times k}$ and $Aeq_2 \in \mathbb{R}^{k\times n-k}$. We suppose $det(Aeq_1) \neq 0$, the case $= 0$ is the *Special case*. We indicate with $x' = (x_1, \dots, x_k)$ and $x'' = (x_{k+1}, \dots, x_n)$

$$(Aeq_1 \mid Aeq_2) \cdot \begin{pmatrix} x' \\ x'' \end{pmatrix} = beq$$

then we can compute x'

$$x' = Aeq_1^{-1}(beq - Aeq_2 \cdot x'')$$

and we can substitute it into the set of inequalities, splitting again the matrix $A \in \mathbb{R}^{h\times n}$ in $A_1 \in \mathbb{R}^{h\times k}$ and $A_2 \in \mathbb{R}^{h\times n-k}$

$$(A_2 - A_1 Aeq_1^{-1} Aeq_2) \cdot x'' \leq b - A_1 Aeq_1^{-1} beq$$

and we call $A_{new} := A_2 - A_1 Aeq_1^{-1} Aeq_2$ and $b_{new} := b - A_1 Aeq_1^{-1} beq$. We can finally use the function $V' = \text{lcon2vert}(A_{new}, b_{new})$ to obtain the vertex of the degenerate polytope in $n-k$ dimensions.

Special Case: (when some variable are fixed to a certain value, so $det(Aeq_1) = 0$) TBD

6. Approximate a Neural Network with a rational function

Lets consider $N : \mathbb{R}^n \to \mathbb{R}^m$ a feed-forward (FF) neural network with $L > 2$ layers in total ($L-2$ hidden layers plus the input layer and the output layer). We indicate with n_l the number of neurons in the l-th layer, with $l = 1, \dots, L$, and we have $n_1 = n$ and $n_L = m$.

Given $x \in \mathbb{R}^{n_{l-1}}$ we indicate with h_l the function

$$h_l(x) = \sigma(W_l x + a_l) \quad l = 2, \dots, L \tag{27}$$

where σ is the activation function, $W_l \in \mathbb{R}^{n_{l-1}\times n_l}$ and $a_l \in \mathbb{R}^{n_l}$. The select the hyperbolic tangent (tanh) as activation function for each hidden layer ($l = 2, \dots, L-1$) and the last activation function for the output layer is the identity. The output of the neural network is the composition of these function

$$N(x) = (h_L \circ h_{L-1} \cdots \circ h_2)(x) \quad x \in \mathbb{R}^n \tag{28}$$

The hyperbolic tangent is defined as

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{29}$$

It is known that tanh can be defined also with the Lambert's continued fraction defined as

$$\tanh(x) = \cfrac{x}{1 + \cfrac{x^2}{3 + \cfrac{x^2}{5 + \cfrac{x^2}{7 + \dots}}}} \tag{30}$$

We can approximate the hyperbolic tangent function taking Equation (30) up to a finite number of fraction and we will call $r_d(x)$ the rational function with d fractions. In Figure 1 we plot the approximation in the domain $[-10, 10]$.

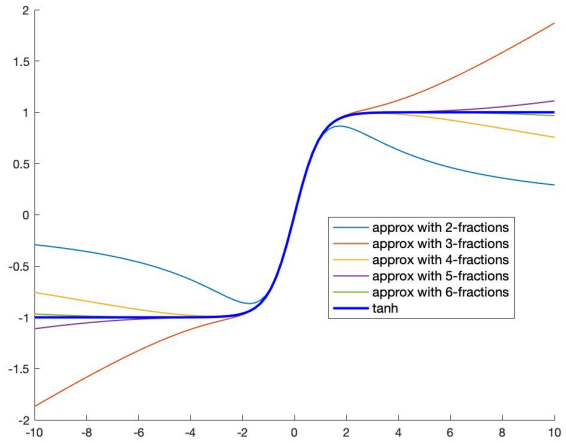


Figure 1: Approximation of tanh with Lambert's finite fraction.

6.1. Bernstein for a rational function

We consider $f = \frac{p}{q}$ rational function with p, q polynomial functions. Lets compute the Bernstein coefficients $b_\alpha^{(k)}(p)$ and $b_\alpha^{(k)}(q)$, over a box X or the standard simplex Δ. Assume that all Bernstein coefficients $b_\alpha^{(k)}(q)$ have the same sign and are non-zero (this implies that $q(x) \neq 0$, for all $x \in X$). Then in [NGSM12] they prove that holds Property 3.2 so

$$\min_{|\alpha|=k} \frac{b_\alpha^{(k)}(p)}{b_\alpha^{(k)}(q)} \leq f(x) \leq \max_{|\alpha|=k} \frac{b_\alpha^{(k)}(p)}{b_\alpha^{(k)}(q)} \tag{31}$$

with $k \geq$ the degree of p and q.

Unfortunately in [NGSM12] they also show that the convex hull property does holds for rational functions and they create also a counterexample.

To apply the range enclosing property we need to guarantee that the Bernstein coefficients of the denominator have the same sign. We consider the Lambert's finite fraction with d fraction. It's easy to observe that the denominator is a positive even polynomial function with degree $\lfloor \frac{d}{2} \rfloor$. It has only even monomials and the coefficients are all positive. It is also immediate to prove that this polynomial has a minimum in $x = 0$ with value $\prod_{i=0}^{d-1} 2i + 1$. All this properties are not enough to guarantee that the Bernstein coefficients are all positive in every interval. As we can see in Example 6.1, we can have negative Bernstein coefficients even with a positive function.

Example 6.1. *Consider the polynomial function* $q(x) = x^2 + 1$ *over* $[-2, 1]$ *the control points are*

$$(v_0(q), b_0(q)) = (-2, 5);$$
$$(v_1(q), b_1(q)) = (-0.5, -1);$$
$$(v_2(q), b_2(q)) = (1, 2).$$

The second Bernstein coefficient is negative and the others are positive. This happens because the point $x = 0$ *is not*

included in the discretization of the interval. See Figure 2

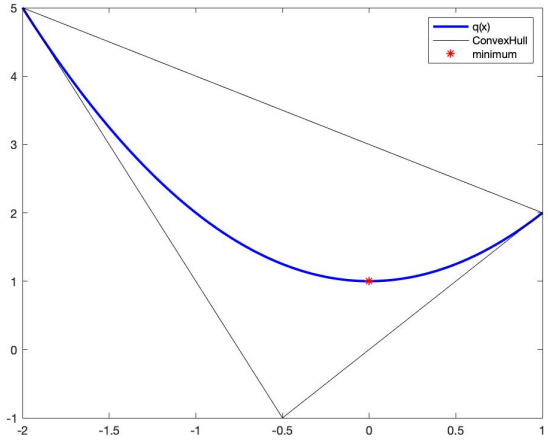


Figure 2: Convex hull of a positive function with a negative Bernstein coefficient.

We need to address the problem of deciding whether a given polynomial or rational function in n-variables has a *positive certificate*, i.e. all its Bernstein coefficients are positive. In [THG15] they prove Theorem 6.1 on a simplex, but the same holds also for hyper-rectangle.

Theorem 6.1. *Let p be a polynomial of degree $d \leq k$ over a simplex Δ. Then*

$$min_{x\in\Delta} p(x) = min_{|\alpha|=k} b_\alpha^{(k)}(p) \tag{32}$$

if and only if

$$min_{|\alpha|=k} b_\alpha^{(k)}(p) = b_{\alpha^*}^{(k)}(p) \tag{33}$$

for some $\alpha^* = ke_i$ with $i \in \{0, \dots, n\}$

This results suggest us that the Bernstein coefficients of the denominator function are positive only if we are able to select $x = 0$ in our control points. For this reason we can split our domain and compute the Bernstein coefficients when $x \geq 0$ and $x \leq 0$. In Example 6.2 we apply this technique for the polynomial expression in Example 6.1.

Example 6.2. *Consider again the polynomial function $q(x) = x^2 + 1$ over $[-2, 1]$. This function has the minimum in $x = 0$ so we slit the domain in $X_1 = [-2, 0]$ and $X_2 = [0, 1]$ and we compute the control points for the two domains*

$$\begin{aligned}
&(v_0(q, X_1), b_0(q, X_1)) = (-2, 5);\\
&(v_1(q, X_1), b_1(q, X_1)) = (-1, 1);\\
&(v_2(q, X_1), b_2(q, X_1)) = (0, 1);\\
&(v_0(q, X_2), b_0(q, X_2)) = (0, 1);\\
&(v_1(q, X_2), b_1(q, X_2)) = (0.5, 1);\\
&(v_2(q, X_2), b_2(q, X_2)) = (1, 2).
\end{aligned}$$

Now all the Bernstein coefficients are positive and we obtain two convex hull as in Figure 3

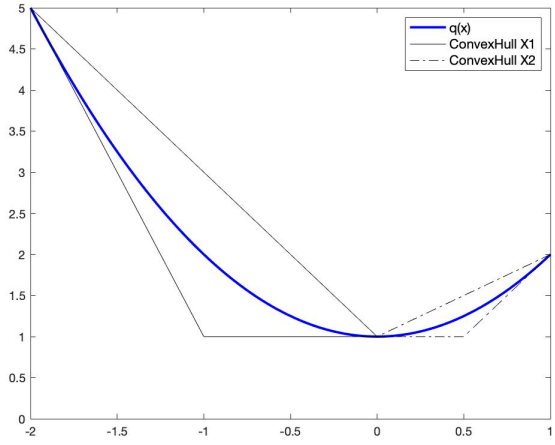


Figure 3: Convex hull of a positive function with only positive Bernstein coefficient.

6.2. Approximation of a single layer

We consider a single layer with n input m neurons. We consider $X = \prod_{i=1}^n [\underline{x}_i, \overline{x}_i]$ an hyper-rectangle and we consider the function $f : X \subset \mathbb{R}^n \to \mathbb{R}^m$

$$f(x) = \tanh(Wx + a) \tag{34}$$

with $W \in \mathbb{R}^{n\times m}$ and $a \in \mathbb{R}^m$. We can split this function in two steps

$$y = Wx + a \tag{35}$$
$$z = \tanh(y) \tag{36}$$

and finally we consider the associated rational function using Lambert's finite series

$$y = Wx + a \tag{37}$$
$$z_j = r_d(y_j) \quad j = 1, \dots m \tag{38}$$

Equation (37). Functions is equation (37) are m linear functions with n variable. We can compute quickly the Bernstein coefficients using sharpness property 3.1. First of all we need to translate the domain X in the unitary cube $[0, 1]^n$

$$y = W \begin{pmatrix} (\overline{x}_1 - \underline{x}_1)u + \underline{x}_1 \\ \vdots \\ (\overline{x}_n - \underline{x}_n)u + \underline{x}_n \end{pmatrix} + a \tag{39}$$

with $u \in [0, 1]^n$. We define $a^* := \underline{x} + a$ and

$$W^* := \begin{pmatrix} w_{1,1}(\overline{x}_1 - \underline{x}_1) & \dots & w_{1,n}(\overline{x}_n - \underline{x}_n) \\ \vdots & & \vdots \\ w_{n,1}(\overline{x}_1 - \underline{x}_1) & \dots & w_{n,n}(\overline{x}_n - \underline{x}_n) \end{pmatrix} \tag{40}$$

For the sharpness property we can compute the Bernstein coefficients as

$$b_\alpha(Wx+a, X) = W^*\alpha + a^* \quad \alpha \in \{0,1\}^n \tag{41}$$

If we compute just the minimum and the maximum we can simplify just looking at the positive and negative values of W^*. In particular if we want to compute the bound for y_j, $j = 1, \dots, m$, we consider the row W_j^* and we have

$$\sum_{i:W^*_{i,j}<0} W^*_{i,j} + a^*_j \le y_j \le \sum_{i:W^*_{i,j}>0} W^*_{i,j} + a^*_j \tag{42}$$

and we indicate this interval with $[\underline{y}_j, \overline{y}_j]$.

Equation (38). The activation function is a univariate rational function over the interval $[\underline{y}_j, \overline{y}_j]$ and we can compute easily the Bernstein coefficients. We just need to split the interval with respect to zero to ensure the positiveness of the coefficients for the denominator.

We call $Y_j := [\underline{y}_j, \overline{y}_j]$. If $Y_j \cap \{0\} = \emptyset$ or $\underline{y}_j = 0$ or $\overline{y}_j = 0$ we just compute

$$\underline{b} = \min_\alpha b_\alpha(r_d(y_j), Y_j) \le z_j \le \max_\alpha b_\alpha(r_d(y_j), Y_j) = \overline{b} \tag{43}$$

otherwise we split the interval in $Y_j^1 := [\underline{y}_j, 0]$ and $Y_j^2 := [0, \overline{y}_j]$, we compute

$$b^1_\alpha = b_\alpha(r_d(y_j), Y_j^1) \tag{44}$$

$$b^2_\beta = b_\beta(r_d(y_j), Y_j^2) \tag{45}$$

and we consider

$$\underline{b} = \min_{\alpha,\beta}\{b^1_\alpha, b^2_\beta\} \le z_j \le \max_{\alpha,\beta}\{b^1_\alpha, b^2_\beta\} = \overline{b} \tag{46}$$

In this step, in order to be more precise also when we are far from 0, we can reduce the interval $[\underline{b}, \overline{b}]$ to the interval $[-1, 1]$ because $\tanh y \in [-1, 1]$.

$$\max\{\min\{\underline{b}, 1\}, -1\} \le z_j \le \min\{\max\{\overline{b}, -1\}, 1\} \tag{47}$$

7. Approximate a Neural Network with a polynomial function

Lets consider the same feed-forward neural network N with $L > 2$ layers and tanh as activation function. We focus again on a single layer with n input m neurons. We consider a polytope P and we consider the function $f : P \subset \mathbb{R}^n \to \mathbb{R}^m$

$$f(x) = \tanh(Wx + a) \tag{48}$$

with $W \in \mathbb{R}^{n\times m}$ and $a \in \mathbb{R}^m$. Suppose that there exists a polynomial function π that approximate the hyperbolic tangent. Then we can consider the polynomial

$$p(x) = \pi(Wx + a) \tag{49}$$

The i-th component is

$$p_i(x) = \pi(W_i x + a_i) \tag{50}$$

with W_i the i-th row of the matrix W.

We now go through Algorithm 2 in detail for this single layer. Lets suppose to have the polytope P define by a set of inequalities. First of all we need to triangulate the polytope P into k simplicies $S_1, \dots, S_k$. If the polytope is not degenerate we can do it using the Delaunay triangulation by transforming the constraints into vertices and by adding eventually some extra points to deal with non-unique solutions. Otherwise we need to reduce the space domain.

Suppose that P is non degenerate, in this case each simplex $S \in \{S_1, \dots, S_k\}$ is defined by $n+1$ vertices $S = \{s_0, \dots, s_n\}$ in $\mathbb{R}^n$. To compute the Bernstein coefficients over S we have to transform S in the standard simplex Δ. As in Equation (12) we define the matrix

$$V = \begin{pmatrix} s_1(1) - s_0(1) & \dots & s_n(1) - s_0(1) \\ \vdots & & \vdots \\ s_1(n) - s_0(n) & \dots & s_n(n) - s_0(n) \end{pmatrix} \tag{51}$$

where $s_i(j)$ is the j-th component of the vector s_i. The transformation $T : \Delta \to S$ is

$$T(u) := Vu + s_0 \quad u \in \Delta \tag{52}$$

We consider the new polynomial $q : \Delta \to \mathbb{R}^m$

$$q(u) = \pi(W(Vu + s_0) + a) \tag{53}$$

Again we are interested in a single component

$$q_i(u) = \pi(W_i(Vu + s_0) + a_i) \tag{54}$$

and if π is a polynomial of degree d in one variable then q_i is a polynomial of degree $\le d$ in n variables.

We arranged the coefficients of q_i in an $(d+1) \times d^*$ matrix C with $d^* := (d+1)^{n-1}$

$$C := \begin{pmatrix} c_{0,0,\dots,0} & c_{0,1,\dots,0} & \dots & c_{0,d,\dots,0} & \dots & c_{0,d\dots,d} \\ c_{1,0,\dots,0} & c_{1,1,\dots,0} & \dots & c_{1,d,\dots,0} & \dots & c_{1,d\dots,d} \\ \vdots & \vdots & & \vdots & & \vdots \\ c_{d,0,\dots,0} & c_{d,1,\dots,0} & \dots & c_{d,d,\dots,0} & \dots & c_{d,d\dots,d} \end{pmatrix}$$

with $c_{\hat\alpha} = 0$ if $|\hat\alpha| > d$. We can finally compute the Bernstein coefficients using the matrix method in Equation (24). For Property 4.3 we have

$$\begin{pmatrix} u \\ q(u) \end{pmatrix} \subseteq \mathrm{Convex}_{\mathrm{hull}}\Big(\Big\{ \begin{pmatrix} v^{(d)}_\alpha(q_i, \Delta) \\ b^{(d)}_\alpha(q_i, \Delta) \end{pmatrix} \mid |\alpha| = d \Big\}\Big) \tag{55}$$

with $\alpha = (d - |\hat\alpha|, \hat\alpha)$. In particular $q_i(u) = p_i(x)$ for $i = 1, \dots, m$ with $u \in \Delta$, $x \in S$ and $x = T(u)$, so also $b^{(d)}_\alpha(q_i, \Delta) = b^{(d)}_\alpha(p_i, S)$. We can then write

$$\begin{pmatrix} x \\ p_i(x) \end{pmatrix} \subseteq \mathrm{Convex}_{\mathrm{hull}}\Big(\Big\{ \begin{pmatrix} T(v^{(d)}_\alpha(q_i, \Delta)) \\ b^{(d)}_\alpha(q_i, \Delta) \end{pmatrix} \mid |\alpha| = d \Big\}\Big) \tag{56}$$

For each function p_i over the polytope P we should consider the union of each convex hull obtained for each simplex. If we want to maintain we convexity we can collect the set points and create the total convex hull.

$$Q_i := \text{Convex}_{\text{hull}}\Big(\Big\{ \begin{pmatrix} T(v_\alpha^{(d)}(p_i, S_j)) \\ b_\alpha^{(d)}(p_i, S_j) \end{pmatrix} \mid |\alpha| = d, j = 1, \dots, k \Big\}\Big) \tag{57}$$

The output set can be obtained in two ways.

1. We transform the convex hull in a set of inequalities, we collect each set for every $p_i, i = 1, \dots, m$. The final set is a set of inequalities for $n + m$ variables we need to project away the first n variables x.
2. We can observe that each p_i has the same degree, and so q_i. We have that $T(v_\alpha^{(d)}(q_i, \Delta))$ are the same for each $i = 1, \dots, n$. We can collect each vector $(b_\alpha^{(d)}(q_i, \Delta)) \in \mathbb{R}^m$ assembly the components by matching the index α and we create the convex hull of these points. The codomain of p over each simplex S_i will be the set of inequalities that defines this convex hull.

$$\text{Convex}_{\text{hull}}\Big\{ \begin{pmatrix} b_\alpha^{(d)}(q_1, \Delta) \\ \vdots \\ b_\alpha^{(d)}(q_n, \Delta) \end{pmatrix} \mid |\alpha| = d \Big\} \tag{58}$$

7.1. Polynomial function for hyperbolic tangent

A common way to approximate a function with a polynomial expression is by taking the Taylor expansion up to a certain degree. Given a function $f : \mathbb{R} \to \mathbb{R}$ we can write the Taylor expansion in x_0 as

$$f(x_0) + f^{(1)}(x_0)(x - x_0) + \frac{f^{(2)}(x_0)}{2!}(x - x_0)^2 + \dots \tag{59}$$

with $f^{(i)}$ is the i-th derivative of f. For the hyperbolic tangent we can use the Taylor expansion in $x_0 = 0$ and we have

$$\tanh(x) \approx x - \frac{y}{3} + \frac{2y^5}{15} + \dots \tag{60}$$

In Figure 4 we plot different approximations with a saturation to $[-1, 1]$ if we exceed this interval.

In [Vlč12] they suggest to use the Chebyshev expansion to better approximate the sigmoid function, and so also the hyperbolic tangent. To clarify, it's trivial to prove that

$$\tanh(x) = 2 \cdot \text{sigmoind}(2x) - 1 \tag{61}$$

They define the Chebyshev expansion as the integration of the Bernstein expression

$$C_{p,q}(x) = \left(\frac{1 + x}{2}\right)^{q+1} \sum_{i=0}^{p} \binom{i + q}{i} \left(\frac{1 - x}{2}\right)^i \tag{62}$$

Vlček obtains approximation of sigmoid$(8x)$ with the expression $C_{11,11}(x)$. We plot in Figure 5 the modified polynomial $2 \cdot C_{11,11}(x/4) - 1$ in comparison with the hyperbolic

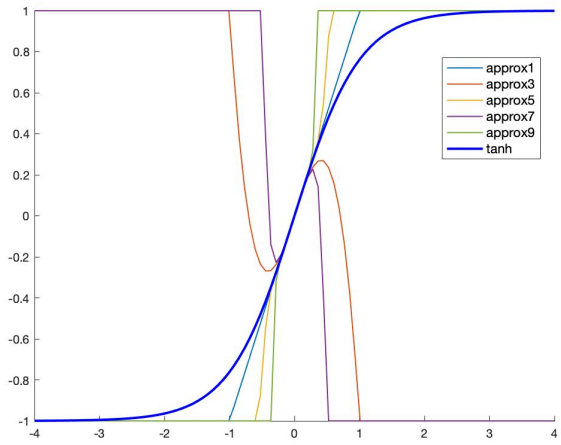


Figure 4: Approximation of hyperbolic tangent with Taylor expansion around $x_0 = 0$ from degree 1 up to degree 9.

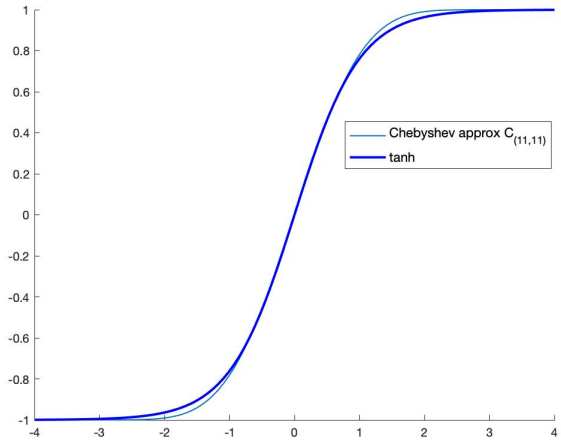


Figure 5: Chebyshev approximation of the hyperbolic tangent.

tangent. **Question TBA:** if Chebyshev is the integral form of Bernstein, can I transform easily the Chebyshev coefficients in Bernstein coefficients?

This is a polynomial function with degree 23. The approximation is acceptable in the interval [-4,4] and we need to bound the output outside this interval. In any case we have to define a interval of confidence $[x_1, x_2]$. So given an polynomial expression $p(x)$, Taylor expansion or Chebsyshev expansion, we can just define a piece-wise function

$$\tanh(x) \approx \pi(x) = \begin{cases} 1 & x > x_2 \\ p(x) & x \in [x_1, x_2] \\ -1 & x < x_1 \end{cases} \tag{63}$$

To apply Algorithm 2 in this case we need to split the polytope P for each $i = 1, \ldots, m$ in

- $P_i^+ = P \cap \{W_i x + a_i > x_1\}$
- $P_i^- = P \cap \{W_i x + a_i < x_2\}$
- $\bar{P}_i = P \setminus (P^+ \cup P^-)$

and apply the procedure just for $\bar{P}_i$. Consider a $\bar{P}_i$ with $i = 1, \ldots, m$, we compute $\bar{Q}_i$ as in Equation (57). If $P_i^+ \neq \emptyset$ we add also the set of points $(v_j, 1)$ where v_j is a vertex of the polytope P_i^+. If $P_i^- \neq \emptyset$ we add also the set of points $(w_j, -1)$ where w_j is a vertex of the polytope P_i^-. Finally we merge the vertices of $\bar{Q}_i$, $(v_j, -1)$ and $(w_j, -1)$ and we compute the convex hull. The output set can be obtained by transforming each convex hull in a set of inequalities and by projecting away the first n variables x.

8. Test

In this section we report the result obtained using the above algorithms implemented in Matlab.

We select a function $f : \mathbb{R} \to \mathbb{R}$ we sample the input/output values over an interval and we train a neural network with the toolbox already implemented in Matlab.

8.1. Test 1: approximation with rational function

We choose the rational approximation with 9 fractions and we set the interval of confidence $I := [-6, 6]$. The rational function approximates the hyperbolic tangent with a maximum error of 1.23×10^{-5}.

Exponential function.

For the first test we select the exponential function $y = e^x$ over the interval $[-1, 7]$, we sample randomly 100 values and we train a neural network with 2 hidden layers with 3 and 2 neurons respectively.

We use the box algorithm described in Section 6. It takes around $0.05s$ and we obtain the result in Figure 6.

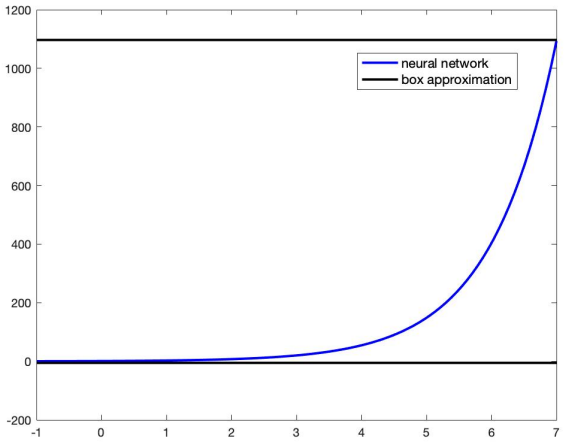


Figure 6: Neural network for the exponential function and bounds approximation with the box algorithm

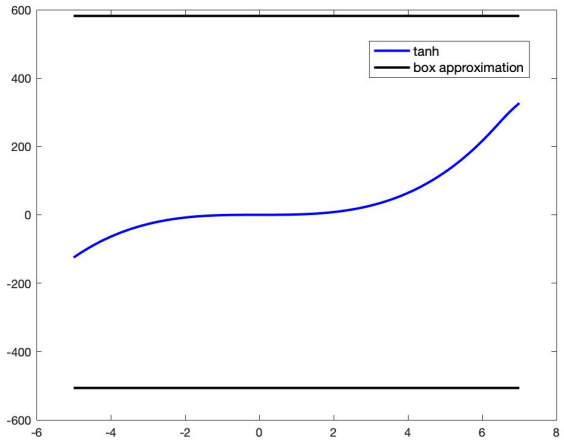


Figure 7: Neural network for the cubic function and bounds approximation with the box algorithm

Cubic function

For the second test we select the cubic function $y = x^3$ over the interval $[-5, 7]$, we sample randomly 100 values and we train a neural network with 2 hidden layers with 5 and 3 neurons respectively.

We use the box algorithm described in Section 6 an we obtain the result in Figure 7.

The error of this approximation is huge compared to the exponential function. What we can try to do is to split the domain. In particular we split uniformly the interval in 20 sub intervals and we compute the bounds for each interval. It takes around $0.8s$ and we obtain a better approximation. We can see in Figure 8 in black the bounds for each interval and in red the convex hull.

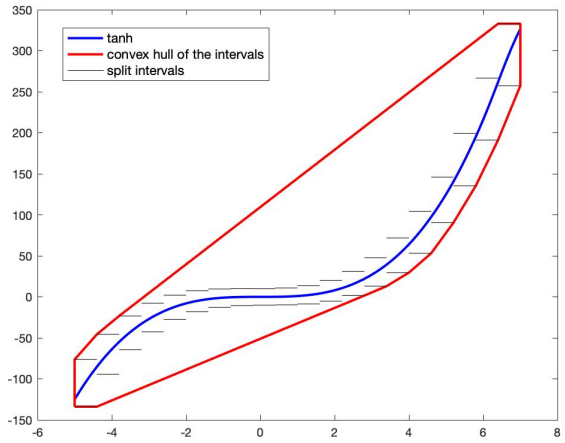


Figure 8: Neural network for the cubic function and bounds approximation with the box algorithm splitting the domain in 20 sub intervals

References

[DCJ+19] S. Dutta, X. Chen, S. Jha, S. Sankaranarayanan, and A. Tiwari. Sherlock - a tool for verification of neural network feedback systems: Demo abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 262–263, 2019.

[Do11] Tan Do. *Generalised Bernstein Operators on Polytopes*. PhD thesis, University of Auckland, 05 2011.

[Dre17] T. Dreossi. Sapo: Reachability computation and parameter synthesis of polynomial dynamical systems. *Hybrid Systems: Computation and Control (HSCC 2017)*, 2017.

[Far86] G. Farin. Triangular bernstein-bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.

[Flo03] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19 – 27, 2003.

[Gar85] Juergen Garloff. Convergent bounds for the range of multivariate polynomials. In *Interval Mathematics 1985*, pages 37–56, 01 1985.

[Ham18] T. Hamadneh. *Bounding Polynomials and Rational Functions in the Tensorial and Simplicial Bernstein Forms*. PhD thesis, Konstanz University, 01 2018.

[HFL+19] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Trans. Embed. Comput. Syst.*, 2019.

[IWA+19] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 169–178, 2019.

[JMD+07] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, 2007.

[Lan08] T. Langer. *On Generalized Barycentric Coordinates and Their Applications in Geometric Modeling*. PhD thesis, University of Saarlandes, 2008.

[LKCOL07] Y. Lipman, J. Kopf, D. Cohen-Or, and D. Levin. Gpu-assisted positive mean value coordinates for mesh deformations. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, page 117–123, 2007.

[Mat20] J. Matt. Analyze n-dimensional polyhedra in terms of vertices or (in)equalities. *MATLAB Central File Exchange*, 2020. Retrieved March 20, 2020.

[NGSM12] A. Narkawicz, J. Garloff, A. Smith, and C. Muñoz. Bounding the range of a rational functiom over a box. *Reliable Computing*, 17, 12 2012.

[Rus07] R. M. Rustamov. Boundary element formulation of harmonic coordinates. Technical report, Purdue University, 2007.

[Tel17] M. Telgarsky. Neural networks and rational functions. *CoRR*, 2017.

[TG17] J. Titi and J. Garloff. Matrix methods for the simplicial bernstein representation and for the evaluation of multivariate polynomials. *Applied Mathematics and Computation*, 315:246–258, 12 2017.

[THG15] J. Titi, T. Hamadneh, and J. Garloff. Convergence of the simplicial rational bernstein form. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 433–441, Cham, 2015. Springer International Publishing.

[Vlč12] M. Vlček. Chebyshev polynomial approximation for activation sigmoid function. *Neural Network World*, 22:387–393, 2012.

[Wal11] S. Waldron. Generalised affine barycentric coordinates. *Jaen Journal on Approximation*, 3, 2011.

[WSHD07] J. Warren, S. Schaefer, A. Hirani, and M. Desbrun. Barycentric coordinates for convex sets. *Adv. Comput. Math.*, 27:319–338, 10 2007.