Neural Network Training under Input-Output Set Constraints using Reachability and Differentiable Collision Checking

CS229 FINAL PROJECT REPORT

Adam Dai
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
addai@stanford.edu

Derek Knowles
Department of Mechanical Engineering
Stanford University
Stanford, CA 94305
dcknowles@stanford.edu

1 Introduction

Neural networks are commonly used to approximate highly nonlinear functions and create a mapping between data inputs and data outputs. However, standard neural network training methods do not allow for specifying input-output constraints of the system. Neural network verification methods can be deployed to check input-output constraints (1), but generally are only applied after a neural network has finished training and do not provide real-time feedback for the neural network during training. This paper introduces a method for constrained neural network training, in which the network attempts to minimize an objective over some dataset which also ensuring that outputs do not lie in some unsafe output space.

Providing strict performance guarantees of a neural network will allow for neural networks to be implemented in safety-critical systems such as autonomous vehicles. For example, it might be useful to be able to provably guarantee that a neural network controller for an autonomous vehicle can receive position and velocity as inputs and will never command a velocity that exceeds the physical capability of the vehicle's motor (unsafe set in the output space).

This project presents a method for training a neural network under input-output constraints by using a differentiable zonotope intersection check to move an output zonotope out of collision with a zonotope representing an unsafe set. While our method generalizes to arbitrarily dimensional input and output spaces, we provide results for a 2D input to 2D output function approximation problem so that both the input and output spaces can be easily visualized.

Figure 1 illustrates the input space on the left and the output of the nonlinear function on the right which we will approximate with our neural network. Additionally, the red square in the plot on the right represents an unsafe set that we constrain the neural network approximation output to avoid.

2 Related Works

2.1 Neural Network Verification

As previously mentioned, there are a number of neural network verification methods that can be used for a trained neural network (1). Given a set of uncertain inputs, the Neural Network Verification (NNV) software tool is able to compute the over-approximation of the reachable set for a feedforward neural network (2). Although this work also combines neural networks and zonotopes, it differs from our approach in that they use zonotopes to represent the layer-by-layer output space of a trained neural network whereas we use zonotopes to compute real-time feedback for a neural network during training.

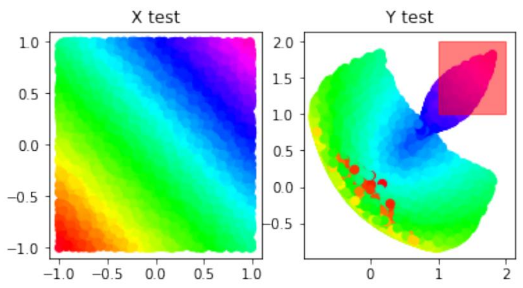


Figure 1: Input data on the left and nonlinear function output on the right. Color is used to help identify which input points correspond to which output points. The right plot shows a red square representing an unsafe set that we constrain the neural network approximation output to avoid.

Safe-by-repair is another alternate neural network validation method which uses a systematic process of small changes to an existing network to conform to constraints (3). Again, this method is suited for a trained neural network and provides no guarantees of actually reaching a provably safe solution.

Another recent neural network verification approach uses an anytime algorithm to return unsafe cells while enumerating polyhedral cells in the input space (rather than iterating layer-by-layer through the network) (4). Again, this paper is a tool for trained neural networks but does not provide real-time feedback for a neural network during training.

2.2 Constrained Optimization

There have been a few related approaches that augment neural networks with constrained optimization (5). One approach is to use a constrained optimization layer, such as Conditional Random Field (CRF) or Non Linear Programming (NLP) solver. These layers use the output of the network as a potential function for optimization while enforcing constriants. Another approach is to use a constraint violation penalty which is added to the objective loss function, and penalizes violation of the constraint. However, our method is the first to employ a differentiable zonotope intersection constraint, which can be backpropagated in order to update the parameters of the network to drive it towards constraint satisfaction. Using this approach, we are able to guarantee constraint satisfaction upon convergence of the training, which existing methods are unable to do.

3 Method

3.1 Zonotopes and Constrained Zonotopes

Throughout our project, zonotopes are used to represent sets in the input, output, and neural network layer space. A *zonotope* is defined by a center, $c \in \mathbb{R}^n$, and generators, $G \in \mathbb{R}^{n \times m}$ where m is the number of generators.

$$\mathcal{Z}(c, G) := \left\{ c + G\beta \mid \|\beta\|_\infty \le 1 \right\} \tag{1}$$

A *constrained zonotope* is defined similarly but adds linear constraints determined by A and b such that $A\beta = b$

$$\mathcal{CZ}(c, G, A, b) := \left\{ c + G\beta \mid \|\beta\|_\infty \le 1,\ A\beta = b \right\} \tag{2}$$

Note: for the remainder of this report, we will use the term zonotope to refer to constrained zonotopes for brevity.

3.2 Constrained Zonotope Neural Network Reachability

Given a constrained zonotope $\mathcal{CZ}(c, G, A, b)$ as an input and a neural network f, we can propagate this input zonotope through f to obtain a set of output constrained zonotopes. We perform this propagation with *layer-by-layer reachability*. Each layer of the network consists of a linear transform followed by ReLU activation, or linear transform alone. Thus, we can compute the output set of constrained zonotopes by applying each layer's operation.

We describe in the following sections how to perform the neural network layer operations as set operations on constrained zonotopes:

3.2.1 Linear layer

A linear layer $\mathcal{L}$ parameterized by weight matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$ and bias vector $\mathbf{b} \in \mathbb{R}^n$ applied to a constrained zonotope $Z_{\text{in}} = \mathcal{CZ}(c, G, A, b)$ simply corresponds to an affine transformation of the constrained zonotope:

$$\mathcal{L}(Z_{\text{in}}, \mathbf{W}, \mathbf{b}) = \mathcal{CZ}(\mathbf{W}c + \mathbf{b}, \mathbf{W}G, A, b) \tag{3}$$

3.2.2 ReLU activation

In order to apply a ReLU activation, $\text{ReLU}(x) = \max(0, x)$, as a set operation on a constrained zonotope $Z_{\text{in}} = \mathcal{CZ}(c, G, A, b)$, we make use of halfspace intersections. Intuitively, when the ReLU is applied to a high-dimensional set of points, points in negative regions will be compressed to 0. Mathematically, we can write

$$\text{ReLU}(Z_{\text{in}}) = \bigcup_{i=1}^{2^n-1} Z^{(i)}, \text{ where} \tag{4}$$

$$Z^{(i)} = \mathcal{CZ}\left(c^{(i)}, G^{(i)}, A^{(i)}, b^{(i)}\right) \tag{5}$$

and each of the zonotopes $Z^{(i)}$ is given by

$$d^{(i)} = \tfrac{1}{2}\left(G_{+}\mathbf{1}_{m\times 1} - \text{diag}\left(\mathbf{1}_{n\times 1} - 2u^{(i)}\right)c\right), \tag{6}$$

$$G^{(i)} = [\text{diag}\left(u^{(i)}\right)G,\ \mathbf{0}_{n\times n}], \tag{7}$$

$$A^{(i)} = \left[\text{diag}(\mathbf{1}_{n\times 1} - 2u_i)\,G,\ \text{diag}\left(d^{(i)}\right)\right], \tag{8}$$

$$c^{(i)} = \text{diag}\left(u^{(i)}\right)c, \tag{9}$$

$$b^{(i)} = -\text{diag}\left(\mathbf{1}_{n\times 1} - 2u^{(i)}\right)c - d^{(i)}, \text{ and} \tag{10}$$

where n is the dimension of the zonotope, m is the number of generators, $G_{+}^{(i)} \in \mathbb{R}^{n\times m}$ denotes a matrix containing the absolute value of each element of the generator matrix G, and $u^{(i)} \in \mathbb{R}^{n\times 1}$ is the i^{th} combination of the n-tuples defined over the set $\{0, 1\}$, excluding $\mathbf{0}_{n\times 1}$. Intuitively, $Z^{(i)}$ is the output zonotope for the i^{th} "quadrant" of n-dimensional space of the original zonotope Z.

Then, for a neural network f and input zonotope Z_{in}, we can compose these operations in order to compute the set of output "reachable" zonotopes $Z_{\text{out}} = \{Z_{\text{out}}^{(1)}, \ldots, Z_{\text{out}}^{(N)}\}$. We call the computation of this reachable set the *input set forward pass*, and denote by $F(Z_{\text{in}}) = Z_{\text{out}}$.

3.3 Zonotope Collision Check

In order to enforce the input-output constraints for our neural network, we need to be able to determine if two zonotopes are intersecting (to check if the output set of the network intersects with the unsafe set). For a pair of constrained zonotopes, $Z_1 = \mathcal{CZ}(c_1, G_1, A_1, b_1)$ and $Z_2 = \mathcal{CZ}(c_2, G_2, A_2, b_2)$, the intersection of Z_1 and Z_2 is given by (6, Prop. 1) as

$$Z_1 \cap Z_2 = \mathcal{CZ}\left(c_1, [G_1, 0], \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \\ G_1 & -G_2 \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \\ c_2 - c_1 \end{bmatrix}\right). \tag{11}$$

Let $Z_\cap = Z_1 \cap Z_2 = \mathcal{CZ}(\mathbf{c}, \mathbf{G}, \mathbf{A}, \mathbf{b})$. Z_1 and Z_2 collide if $Z_\cap$ is nonempty. We can check if $Z_\cap$ is nonempty by solving the convex optimization problem:

$$\min_{\mathbf{z}, v}.\quad v \tag{12}$$

$$\text{s.t.}\quad \mathbf{A}\mathbf{z} = \mathbf{b} \tag{13}$$

$$\|\mathbf{z}\|_1 \le v \tag{14}$$

Then, the intersection $Z_\cap$ is empty if and only if v is less than or equal to 1,

$$Z_\cap \neq \emptyset \iff v \leq 1. \tag{15}$$

We can treat this optimization problem as a function of Z_1 and Z_2 which outputs the value v, which call the *collision check value* of Z_1 and Z_2. Then, we can define a "constraint loss" function $L_{\text{con}}(Z_1, Z_2) = (1 - v)^2$. Minimizing this function drives v to 1, which is the point at which Z_1 and Z_2 are just barely not in collision (their boundaries are touching).

In the case of Z_{out}, we have a set of zonotopes, so we compute the collision check value $v^{(i)}$ for each zonotope $Z_{\text{out}}^{(i)} \in Z_{\text{out}}$ with an unsafe set represented by zonotope Z_{unsafe}. Then for all $v^{(i)} < 1$ (i.e. collision checks which result in collision), we compute $L_{\text{con}}^{(i)}(Z_{\text{out}}^{(i)}, Z_{\text{unsafe}})$ and sum these constraint losses to obtain a total constraint loss. Thus the total constraint loss can be expressed as

$$L_{\text{con}}(Z_{\text{out}}, Z_{\text{unsafe}}) = \sum_i 1\{v^{(i)} < 1\} L_{\text{con}}^{(i)}(Z_{\text{out}}^{(i)}, Z_{\text{unsafe}}) \tag{16}$$

3.4 Constrained neural network training

We can formulate the constrained neural network training problem as follows. Assume we have a N-layer fully connected feedforward ReLU network, represented by the function f, which maps inputs $x \in \mathcal{D}_x \subseteq \mathbb{R}^{d_x}$ to output $y \in \mathcal{D}_y \subseteq \mathbb{R}^{d_y}$ (i.e. $f : \mathcal{D}_x \to \mathcal{D}_y$). The network is parameterized by a set of N weights $\mathbf{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_N\}$ and N bias vectors $\mathbf{b} = \{\mathbf{b}_1, \ldots, \mathbf{b}_N\}$. In addition, we are given

- Training dataset $(x^{(i)}, y^{(i)}), i = 1, \ldots, n$ of training examples $x^{(i)}$ and labels $y^{(i)}$
- Objective loss function $L_{\text{obj}}(f(x^{(i)}), y^{(i)}) = \frac{1}{n} \sum_{i=1}^n \left\| f(x^{(i)}) - y^{(i)} \right\|^2$
- Input set $Z_{\text{in}} \subseteq \mathcal{D}_x$ represented by a constrained zonotope
- Unsafe set $Z_{\text{unsafe}} \subseteq \mathcal{D}_y$ represented by a constrained zonotope

We wish to find the set of weights and biases which minimize the objective loss over the training dataset while obeying the input-output constraint:

$$\begin{aligned} \min_{\mathbf{W}, \mathbf{b}}. \quad & L_{\text{obj}}(f(x^{(i)}), y^{(i)}) \\ \text{s.t.} \quad & f(x) \notin Z_{\text{unsafe}} \; \forall x \in Z_{\text{in}} \end{aligned} \tag{17}$$

Alternatively, if we consider $Z_{\text{out}} = f(Z_{\text{in}})$, we can rewrite the constraint as

$$Z_{\text{out}}^{(i)} \cap Z_{\text{unsafe}} = \emptyset \quad \forall Z_{\text{out}}^{(i)} \in Z_{\text{out}}$$

In order to solve this problem, we make use of the input set forward pass and collision check. We start with the standard neural network training setup. Since we are using a small toy dataset, we compute the loss across the entire dataset, and use it backpropagate and update the weights with gradient steps using the pytorch SGD optimizer. These updates train the network to learn the data $(x^{(i)}, y^{(i)})$.

In addition, each iteration we also perform an input set forward pass to compute the set of output zonotopes Z_{out} under the current state of the network parameters, then compute the constraint loss with Z_{unsafe} from Equation 16, then backpropagate and step the constraint optimizer to train the network to satisfy the constraint.

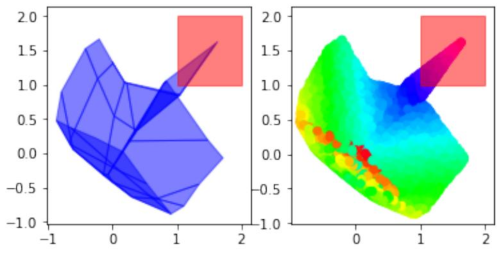
4 Results

We pick the following nonlinear function to train the network to approximate:

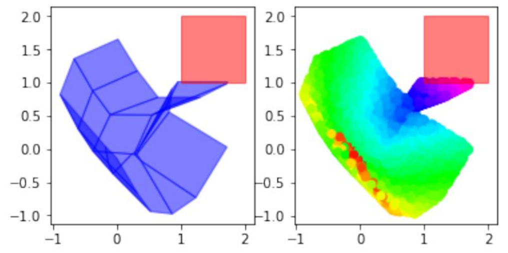
$$g(x_1, x_2) = \begin{bmatrix} x_1^2 + \sin(x_2) \\ x_2^2 + \sin(x_1) \end{bmatrix} \tag{18}$$

which maps inputs vectors in $\mathbb{R}^2$ to output vectors in $\mathbb{R}^2$ as shown in Figure 1. We choose $Z_{\text{in}} = \mathcal{Z}(\mathbf{0}_{2\times1}, \mathbf{I}_{2\times2})$ and generate training data by sampling 10000 points $x^{(i)}$ from Z_{in} and computing corresponding output points $y^{(i)} = g(x^{(i)})$. We use a 2-layer ReLU network with layer sizes 10×2 and 2×10 which also maps from $\mathbb{R}^2$ to $\mathbb{R}^2$ to approximate the function from the data. The unsafe zonotope is chosen as $Z_{\text{unsafe}} = \mathcal{Z}\left(\begin{bmatrix}1.5\\1.5\end{bmatrix}, 0.5\mathbf{I}_{2\times 2}\right)$ and is shown in red in the output space.

The results from both unconstrained and constrained training are shown in Figure 2. We trained both the unconstrained and constrained for 1000 iterations. The unconstrained training is able to successfully learn a function approximation for the nonlinear function $g(x^{(i)})$ and the constrained training is able to learn the an approximate function while also avoiding the unsafe set, Z_{unsafe}.



(a) Unconstrained training with zonotope representation on the left and output points on the right.



(b) Constrained training with zonotope representation on the left and output points on the right.

Figure 2: Neural network learns to approximate the nonlinear function, while avoiding the unsafe set represented by the red square.

Table 1: Experimental Results

	Unconstrained Optimization	Constrained Optimization
Objective Loss, $L_{\text{obj}}(f(x^{(i)}), y^{(i)})$	**0.00386**	0.0115
Constraint Loss, $L_{\text{con}}(Z_{\text{out}}, Z_{\text{unsafe}})$	0.0575	**0.00312**

Quantitative experimental results in Table 1 show that the constrained training has a lower constraint loss than the unconstrained training. Our proposed approach is able to move the output out of collision with the unsafe set, Z_{unsafe}. However, the constrained training is only able to achieve lower constraint loss by increasing the objective loss. The unconstrained training is sensibly able to achieve a lower objective loss than the constrained training since the unconstrained version simply approximates the nonlinear function as best as possible while completely ignoring the obstacle zonotope.

5 Conclusion/Future Work

For our project, we developed a method of computing reachable sets of neural networks using constrained zonotopes, as well as performing differentiable collision checking of those zonotopes with an unsafe set, and combined these two techniques to yield a novel method of training neural networks under output unsafe set constraints. We tested our approach for a simple 2D nonlinear function approximation example, and saw that the network learns to fit the function while avoiding the unsafe set.

In the future, we will speed up the forward pass step of the constrained training to allow for quicker training. We will also extend the proposed method to larger networks beyond simple 2D input to 2D output function approximators. We will also apply to a real-world problem such as aircraft collision avoidance so that we can compare our constrained results with benchmark results that use traditional network verification methods. We would also like to compare to existing approaches for constrained optimization of neural networks, in particular the constraint violation penalty approach, as that one is most similar to our approach.

6 Contributions

Adam Dai worked on the zonotope classes, collision check loss derivations, neural network framework, unconstrained training, and constrained training. Derek Knowles worked on plotting visualizations, conversion from CPU to GPU training, and constrained training. Edgar Chung (labmate not in CS229) converted the ReLU activation and linear layers (see Section 3.2) to numpy python functions. Shreyas Kousik (labmate not in CS229) provided advising on zonotope reachability methods and proofs.

References

[1] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *arXiv preprint arXiv:1903.06758*, 2019.

[2] H. D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12224 LNCS. Springer, jul 2020, pp. 3–17. [Online]. Available: https://doi.org/10.1007/978-3-030-53288-8_1

[3] U. S. Cruz, J. Ferlez, and Y. Shoukry, "Safe-by-Repair: A Convex Optimization Approach for Repairing Unsafe Two-Level Lattice Neural Network Controllers," apr 2021. [Online]. Available: http://arxiv.org/abs/2104.02788

[4] J. A. Vincent and M. Schwager, "Reachable Polyhedral Marching (RPM): A Safety Verification Algorithm for Robotic Systems with Deep Neural Network Components," *arXiv*, nov 2020. [Online]. Available: http://arxiv.org/abs/2011.11609

[5] D. Jindal, "Augmenting Neural Networks with Constrained Optimization," 2019. [Online]. Available: https://towardsdatascience.com/augmenting-neural-networks-with-constraints-optimization-ac747408432f

[6] J. K. Scott, D. M. Raimondo, G. R. Marseglia, and R. D. Braatz, "Constrained zonotopes: A new tool for set-based estimation and fault detection," *Automatica*, vol. 69, pp. 126–136, 2016.