Towards Reliable Neural Specifications

Chuqin Geng [*1 2] **Nham Le** [*3] **Xiaojie Xu** [1 2] **Zhaoyue Wang** [1 2] **Arie Gurfinkel** [3] **Xujie Si** [2 4]

Abstract

Having reliable specifications is an unavoidable challenge in achieving verifiable correctness, robustness, and interpretability of AI systems. Existing specifications for neural networks are in the paradigm of *data as specification*. That is, the local neighborhood centering around a reference input is considered to be correct (or robust). While existing specifications contribute to verifying adversarial robustness, a significant problem in many research domains, our empirical study shows that those verified regions are somewhat tight, and thus fail to allow verification of test set inputs, making them impractical for some real-world applications. To this end, we propose a new family of specifications called neural representation as specification. This form of specifications uses the intrinsic information of neural networks, specifically neural activation patterns (NAPs), rather than input data to specify the correctness and/or robustness of neural network predictions. We present a simple statistical approach to mining neural activation patterns. To show the effectiveness of discovered NAPs, we formally verify several important properties, such as various types of misclassifications will never happen for a given NAP, and there is no ambiguity between different NAPs. We show that by using NAP, we can verify a significant region of the input space, while still recalling 84% of the data on MNIST. Moreover, we can push the verifiable bound to 10 times larger on the CIFAR10 benchmark. Thus, we argue that NAPs can potentially be used as a more reliable and extensible specification for neural network verification.

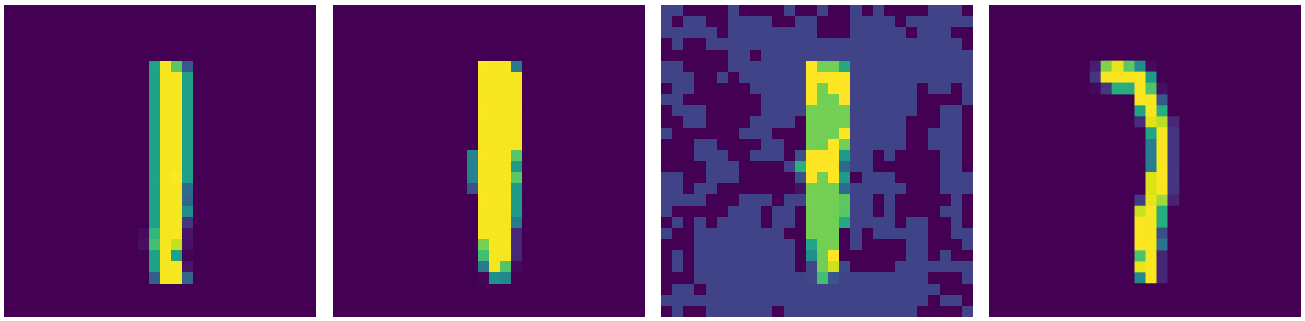
[*]Equal contribution [1]McGill University [2]Mila Quebec AI Institute [3]University of Waterloo [4]University of Toronto. Correspondence to: Chuqin Geng <chuqin.geng@mail.mcgill.ca>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

1. Introduction

The advances in deep neural networks (DNNs) have brought a wide societal impact in many domains such as transportation, healthcare, finance, e-commerce, and education. This growing societal-scale impact has also raised some risks and concerns about errors in AI software, their susceptibility to cyber-attacks, and AI system safety (Dietterich & Horvitz, 2015). Therefore, the challenge of verification and validation of AI systems, as well as, achieving trustworthy AI (Wing, 2021), has attracted much attention of the research community. Existing works approach this challenge by building on *formal methods* – a field of computer science and engineering that involves verifying properties of systems using rigorous mathematical specifications and proofs (Wing, 1990). Having a formal specification — a precise, mathematical statement of what AI system is supposed to do is critical for formal verification. Most works (Katz et al., 2017; 2019; Huang et al., 2017; 2020; Wang et al., 2021) use the specification of adversarial robustness for classification tasks that states that the NN correctly classifies an image as a given adversarial label under perturbations with a specific norm (usually L_∞). Generally speaking, existing works use a paradigm of *data as specification* — the robustness of local neighborhoods of reference data points with ground-truth labels is the only specification of correct behaviors. However, from a learning perspective, this would lead to *overfitted* specification, since only local neighborhoods of reference inputs get certified and the generalization to unseen data points may not be guaranteed.

As a concrete example, Figure 1 illustrates the fundamental limitation of such overfitted specifications. Specifically, a testing input like the one shown in Figure 1a can hardly be verified even if all local neighborhoods of all training images have been certified using the L_∞ norm. This is because adversarial examples like Figure 1c fall into a much closer region compared to testing inputs (e.g., Figure 1a), as a result, the truly verifiable region for a given reference input like Figure 1b can only be smaller. All neural network verification approaches following such data-as-specification paradigm inherit this limitation *regardless* of their underlying verification techniques. In order to avoid such a limitation, a new paradigm for specifying what is correct or wrong is necessary. The intrinsic challenge is that manually giving a proper specification on the input space is no easier than



(a) A testing image from MNIST, classified as 1

(b) The closest training image in MNIST, whose L_∞ distance to Fig. 1a is 0.5294

(c) An adversarial example misclassified as 8, whose L_∞ distance to Fig. 1b is 0.2

(d) A testing image, on which our verified NAP (for digit 1) disagrees with the ground truth (i.e., 7)

Figure 1. The limitation of "data-as-specification": First three images show that a test input can be much further away (in L_∞) from its closest train input compared to adversarial examples (the upper bound of a verifiable local region). The last image shows that even data itself can be imperfect.

directly programming a solution to the machine learning problem itself. We envision that a promising way to address this challenge is developing specifications directly on top of, instead of being agnostic to, the learned model.

We propose a new family of specifications, *neural representation as specification*, where neural activation patterns form specifications. The key observation is that inputs from the same class often share a neural activation pattern (NAP) – a carefully chosen subset of neurons that are expected to be activated (or not activated) for the majority of inputs in a class. Although two inputs are distant in a certain norm in the input space, the neural activations exhibited when the same prediction is made are very close. For instance, we can find a *single* NAP that is shared by *nearly all* training and testing images (including Figure 1a and Figure 1b) in the same class but not the adversarial example like Figure 1c. We can further formally *verify* that *all possible* inputs following this particular NAP can never be misclassified. Specifications based on NAP enable successful verification of a broad region of inputs, which would not be possible if the data-as-specification paradigm were used. For the MNIST dataset, a verifiable NAP *mined* from the training images could cover up to 84% testing images, a significant improvement in contrast to 0% when using neighborhoods of training images as the specification. To our best knowledge, this is the first time that a significant fraction of *unseen* testing images have been formally verified.

This unique advantage of using NAPs as specification is enabled by the intrinsic information (or neural representation) embedded in the neural network model. Furthermore, such information is a simple byproduct of a prediction and can be collected easily and efficiently. Besides serving as reliable specifications for neural networks, we foresee other important applications of NAPs. For instance, verified NAPs may serve as proofs of correctness or certificates for predictions. We hope our initial findings shared in this paper would inspire new interesting applications. We summarize our contribution as follows:

- We propose a new family of formal specifications for neural networks, *neural representation as specification*, which use activation patterns (NAPs) as specifications. We also introduce a tunable parameter to specify the level of abstraction of NAPs.
- We propose a simple yet effective approximate method to mine NAPs from neural networks and training datasets.
- We show that NAPs can be easily checked by out-of-the-box neural network verification tools used in VNNCOMP – the annual neural network verification competition, such as Marabou.
- We conduct thorough experimental evaluations from both statistical and formal verification perspectives. Particularly, we show that a single NAP is sufficient for certifying a significant fraction of unseen inputs.

2. Background

2.1. Neural Networks for Classification Tasks

In this paper, we focus on feed-forward neural networks for classification (including convolutional neural nets). A neural network $\mathcal{N}$ of L layers is a set $\{(\boldsymbol{W}^i, \boldsymbol{b}^i) \mid 1 \leqslant i \leqslant L\}$, where $\boldsymbol{W}^i$ and $\boldsymbol{b}^i$ are the weight matrix and the bias for layer i, respectively. The neural network $\mathcal{N}$ defined a function $F_N : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ (d_0 and d_L represent the input and output dimension, respectively), defined as $F_{\mathcal{N}}(x) = z^L(x)$, where $z^0(x) = x$, $z^i(x) = \boldsymbol{W}^i\sigma(z^{i-1}(x)) + \boldsymbol{b}^i$ and σ is

the activation function. Neurons are indexed linearly by $v_0, v_1, \cdots$. In this paper we focus only on the ReLU activation function, i.e., $\sigma(x) = \max(x, 0)$ element-wise, but the idea and techniques can be generalized for different activation functions and architectures as well. The i^{th} element of the prediction vector $F_{\mathcal{N}}(x)[i]$ represents the score or likelihood for the i^{th} label, and the one with the highest score ($\arg\max_i F_{\mathcal{N}}(x)[i]$) is often considered as the predicted label of the network $\mathcal{N}$. We denote this output label as $\mathcal{O}_{\mathcal{N}}(x)$. When the context is clear, we omit the subscript $\mathcal{N}$ for simplicity.

2.2. Adversarial Attacks against Neural Networks and the Robustness Verification Problem

Given a neural network $\mathcal{N}$, the aim of adversarial attacks is to find a perturbation v of an input x, such that x and $x + v$ are "similar" according to some domain knowledge, yet $\mathcal{O}(x) \neq \mathcal{O}(x + v)$. In this paper, we use the common formulation of "similarity" in the field: two inputs are similar if the L_∞ norm of v is small. Under this formulation, finding an adversarial example can be defined as solving the following optimization problem:[1]

$$\min ||v||_\infty \; s.t. \; \mathcal{O}(x) \neq \mathcal{O}(x + v) \tag{1}$$

In practice, it is very hard to formally define "similar": should an image and a crop of it be "similar"? Should two sentences differ by one synonym be the same? We refer curious readers to the survey (Xu et al., 2020) for a comprehensive review of different formulations.

One natural defense against adversarial attacks, called *robustness verification*, is to prove that $\min ||v||_\infty$ must be greater than some user-specified threshold ϵ. Formally, given that $\mathcal{O}(x) = i$, we verify

$$\forall x' \in B(x, \epsilon) \cdot \forall j \neq i \cdot F(x')[i] - F(x')[j] > 0 \tag{2}$$

where $B(x, \epsilon)$ is a L_∞ norm-ball of radius ϵ centered at x: $B(x, \epsilon) = \{x' \mid ||x - x'||_\infty \leqslant \epsilon\}$. If Equation (2) holds, we say that x is ϵ-robust.

3. Neural Activation Patterns

In this section, we discuss in detail neural activation patterns (NAPs), what we consider as NAPs and how to relax them, and what interesting properties of NAPs can be checked using neural network verification tools like Marabou (Katz et al., 2019).

[1]While there are alternative formulations of adversarial robustness (see Xu et al. (2020)), in this paper, we use adversarial attacks as a black box, thus, stating one formulation is sufficient.

3.1. NAPs and Their Relaxation

In our setting (Section 2), the output of each neuron is passed to the ReLU function before going to neurons of the next layer, i.e., $z^i(x) = \boldsymbol{W}^i \sigma(z^{i-1}(x)) + \boldsymbol{b}^i$. We abstract each neuron into two states: *activated* (if its output is positive) and *deactivated* (if its output is non-positive). Clearly, for any given input, each neuron can be either activated or deactivated.

Definition 3.1 (Neural Activation Pattern)**.** A *Neural Activation Pattern (NAP)* of a neural network is a tuple $\mathcal{P} := (A, D)$, where A and D are two disjoint subsets of activated and deactivated neurons, respectively. Note that $\mathcal{P}$ excludes the neurons that could be either activated or deactivated. An example is shown in Table 2. Hence A $\bigcup$ D is only a subset of all neurons in a neural network.

Definition 3.2 (Partially ordered NAP)**.** For any given two NAPs $\bar{\mathcal{P}} := (\bar{A}, \bar{D})$ and $\mathcal{P} := (A, D)$. We say $\bar{\mathcal{P}}$ subsumes $\mathcal{P}$ iff A, D are subsets of $\bar{A}, \bar{D}$ respectively. Formally, this can be defined as:

$$\bar{\mathcal{P}} \preccurlyeq \mathcal{P} \iff \bar{A} \supseteq A \; and \; \bar{D} \supseteq D \tag{3}$$

Moreover, two NAPs $\bar{\mathcal{P}}$ and $\mathcal{P}$ are equivalent if $\bar{\mathcal{P}} \preccurlyeq \mathcal{P}$ and $\mathcal{P} \preccurlyeq \bar{\mathcal{P}}$.

Definition 3.3 (NAP Extraction Function)**.** A NAP Extraction Function E takes a neural network $\mathcal{N}$ and an input x as parameters, and returns a NAP $\mathcal{P} := (A, D)$ where A and D represent all the activated and deactivated neurons of $\mathcal{N}$ respectively when passing x through $\mathcal{N}$.

With the above definitions in mind, we are able to describe the relationship between an input and a specific NAP. An input x *follows* a NAP $\mathcal{P}$ of a neural network $\mathcal{N}$ if:

$$E(\mathcal{N}, x) \preccurlyeq \mathcal{P} \tag{4}$$

For a given neural network $\mathcal{N}$ and an input x, it is possible x follows multiple NAPs. In addition, there are some trivial NAPs such as $(\varnothing, \varnothing)$ that can be followed by any input. From the representational learning point of view, these trivial NAPs are the least specific abstraction of inputs, which fails to represent data with different labels. Thus, we are prone to study more specific NAPs due to their rich representational power. Moreover, an ideal yet maybe impractical scenario is that all inputs with a specific label follow the same NAP. Given a label ℓ, and let S be the training dataset, and S_ℓ be the set of data labeled as ℓ, Formally, this scenario can be described as:

$$\forall x \in S_\ell \cdot \; E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_\ell \iff \mathcal{O}(x) = \ell \tag{5}$$

This can be viewed as a condition for perfectly solving classification problems. In our view, $\mathcal{P}_\ell$, the NAP with respect to ℓ, if exists, can be seen as a certificate for the prediction

Algorithm 1 NAP Mining Algorithm

Input: relaxing factor δ, neural network $\mathcal{N}$, dataset S_ℓ
Initialize a counter c_k for each neuron v_k
for $x \in S_\ell$ **do**
  compute $E(\mathcal{N}, x)$
  if v_k is activated **then**
    c_k += 1
  end if
end for
$A_\ell \leftarrow \{v_k \mid \frac{c_k}{|S_\ell|} \geqslant \delta\}$, $D_\ell \leftarrow \{v_k \mid \frac{c_k}{|S_\ell|} \leqslant 1-\delta\}$
$\mathcal{P}_\ell^\delta \leftarrow (A_\ell, D_\ell)$

of a neural network: inputs following $\mathcal{P}_\ell$ can be provably classified as ℓ by $\mathcal{N}$. However, in most cases, it is infeasible to have a perfect $\mathcal{P}_\ell$ that captures the exact inputs for a given class. On the one hand, there is no access to the ground truth of all possible inputs; on the other hand, DNNs are not guaranteed to precisely learn the ideal patterns. Thus, to accommodate standard classification settings in which Type I and Type II Errors are non-negligible, we relax $\mathcal{P}_\ell$ in such a way that only a portion of the input data with a specific label ℓ follows the relaxed NAP. The formal relaxation of NAPs is defined as follows.

Definition 3.4 (δ-relaxed NAP)**.** We introduce a relaxing factor $\delta \in [0, 1]$. We say a NAP is δ-relaxed with respect to the label ℓ, denoted as $\mathcal{P}_\ell^\delta := (A_\ell^\delta, D_\ell^\delta)$, if it satisfies the following condition:

$$\exists S'_\ell \subseteq S_\ell \; s.t. \; \frac{|S'_\ell|}{|S_\ell|} \geqslant \delta \; \textit{and} \; \forall x \in S'_\ell, \; E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_\ell^\delta \quad (6)$$

Intuitively, the δ-relaxed factor controls the level of abstraction of NAP. When $\delta = 1.0$, not only $\mathcal{P}^{\delta=1.0}$ is the most precise (as all inputs from S_ℓ follow it) but also the least specific. In this sense, $\mathcal{P}^{\delta=1.0}$ can be viewed as the highest level of abstraction of the common neural representation of inputs with a specific label. However, being too abstract is also a sign of under-fitting, this may also enhance the likelihood of Type II Errors for NAPs. By decreasing δ, the likelihood of a neuron being chosen to form a NAP increases, making NAPs more specific. This may help alleviate Type II Errors, yet may also worsen the recall rate by producing more Type I Errors.

In order to effectively mine δ-relaxed NAPs, we propose a simple statistical method shown in Algorithm 1 [2]. Table 1 reports the effect of δ on the precision recall trade-off for mined δ-relaxed NAPs on the MNIST dataset. The table shows how many test images from a label ℓ follow $\mathcal{P}_\ell^\delta$, together with how many test images from other labels that also follow the same $\mathcal{P}_\ell^\delta$. For example, there are 980 images in the test set with label 0 (second column). Among them, 967 images follow $\mathcal{P}_{\ell=0}^{\delta=1.0}$. In addition to that, there are 20 images from the other 9 labels that also follow $\mathcal{P}_{\ell=0}^{\delta=1.0}$.

With the decrease of δ, we can see that in both cases, both numbers decrease, suggesting that it is harder for an image to follow $\mathcal{P}_{\ell=0}^{\delta=.99}$ without being classified as 0 (the NAP is more precise), at the cost of having many images classified as 0 fail to follow $\mathcal{P}_{\ell=0}^{\delta=.99}$ (the NAP recalls worse). In short, the usefulness of NAPs largely depends on their precision-recall trade-off. Thus, choosing the right δ or the right level of abstraction becomes crucial in using NAPs as specifications in verification. We will discuss this matter further in Section 4.

3.2. Interesting NAP Properties

We expect that NAPs can serve as the key component in more reliable specifications of neural networks. As the first study on this topic, we introduce here three important ones.

The non-ambiguity property of NAPs We want our NAPs to give us some confidence about the predicted label of an input, thus a crucial sanity check is to verify that no input can follow two different NAPs of two different labels. Formally, we want to verify the following:

$$\forall x \cdot \forall i \neq j \cdot E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_{\ell=i} \implies E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_{\ell=j} \quad (7)$$

Note that this property doesn't hold if either $A_{\ell=i} \bigcap D_{\ell=j}$ or $A_{\ell=j} \bigcap D_{\ell=i}$ is non-empty as a single input cannot activate and deactivate the same neuron. If that's not the case, we can encode and verify the property using verification tools.

NAP robustness property The intuition of using neural representation as specification not only accounts for the internal decision-making process of neural networks but also leverages the fact that NAPs themselves map to regions of our interests in the whole input space (Hanin & Rolnick, 2019a;b). In contrast to canonical ϵ-balls, these NAP-derived regions are more flexible in terms of size and shape. We explain this insight in more detail in Section 3.3. Concretely, we formalize this NAP robustness verification problem as follows: given a neural network $\mathcal{N}$ and a NAP $\mathcal{P}_{\ell=i}$, we want to check:

$$\forall x \in R \cdot \forall j \neq i \cdot F(x)[i] - F(x)[j] > 0 \quad (8)$$

in which

$$R = \{x \mid E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_{\ell=i}\} \quad (9)$$

NAP-augmented robustness property Instead of only having the activation patterns as specification, we can still specify ϵ-balls in the input space for verification. This conjugated form of specification has two advantages: First, it

[2] Note that this algorithm is an approximate method for mining δ-relaxed NAP, whereas δ should be greater than 0.5, otherwise, $A_\ell^\delta \bigcap D_\ell^\delta \neq \varnothing$. We leave more precise algorithms for future work.

Table 1. The number of the test images in MNIST that follow a given NAP$^{\delta}$. For a label i, $\overline{i}$ represents images with labels other than i yet follow NAP$^{\delta}_{\ell=i}$. The leftmost column is the values of δ. The top row indicates how many images in the test set are of a label.

	0 (980)		1 (1135)		2 (1032)		3 (1010)		4 (982)		5 (892)		6 (958)		7 (1028)		8 (974)		9 (1009)	
	0	$\overline{0}$	1	$\overline{1}$	2	$\overline{2}$	3	$\overline{3}$	4	$\overline{4}$	5	$\overline{5}$	6	$\overline{6}$	7	$\overline{7}$	8	$\overline{8}$	9	$\overline{9}$
1.00	967	20	1124	8	997	22	980	13	959	25	874	32	937	26	1003	28	941	22	967	12
0.99	775	1	959	0	792	4	787	2	766	3	677	1	726	4	809	2	696	3	828	4
0.95	376	0	456	0	261	1	320	0	259	0	226	0	200	0	357	0	192	0	277	0
0.90	111	0	126	0	43	0	92	0	76	0	24	0	45	0	144	0	44	0	73	0

focuses on the verification of valid test inputs instead of adversarial examples. Second, the constraints on NAPs are likely to make verification tasks effortless by refining the search space of the original verification problem, in most cases, allowing the verification on much larger ϵ-balls. We formalize the NAP-augmented robustness verification problem as follows: given a neural network $\mathcal{N}$, an input x, and a mined $\mathcal{P}_{\ell=i}$, we check:

$$\forall x' \in B^{+}(x, \epsilon, \mathcal{P}_{\ell=i}) \cdot \forall j \neq i \cdot F(x')[i] - F(x')[j] > 0 \tag{10}$$

in which $\mathcal{O}(x) = i$ and

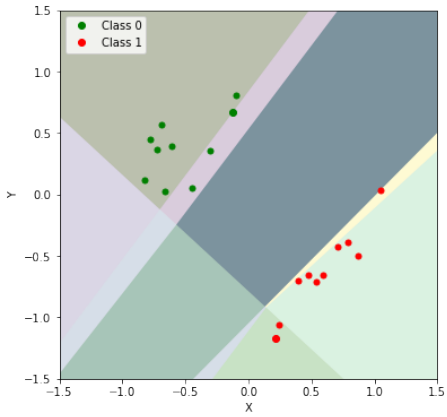
$$B^{+}(x, \epsilon, \mathcal{P}_{\ell=i}) = \{x' \mid ||x - x'||_{\infty} \leqslant \epsilon, E(\mathcal{N}, x') \preccurlyeq \mathcal{P}_{\ell=i}\} \tag{11}$$

Working with NAPs using Marabou In this paper, we use Marabou (Katz et al., 2019), a dedicated state-of-the-art NN verifier. Marabou extends the Simplex (Nelder & Mead, 1965) algorithm for solving linear programming with special mechanisms to handle non-linear activation functions. Internally, Marabou encodes both the verification problem and the adversarial attacks as a system of linear constraints (the weighted sum and the properties) and non-linear constraints (the activation functions). Same as Simplex, at each iteration, Marabou tries to fix a variable so that it doesn't violate its constraints. While in Simplex, a violation can only happen due to a variable becoming out-of-bound, in Marabou a violation can also happen when a variable doesn't satisfy its activation constraints.

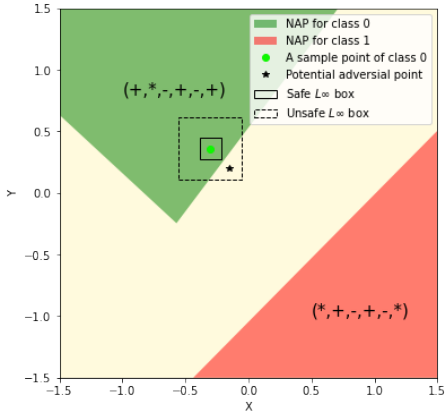
NAPs and NAP properties can be encoded using Marabou with little to no changes to Marabou itself. To force a neuron to be activated or deactivated, we add a constraint for its output. To improve performance, we infer ReLU's phases implied by the NAPs, and change the corresponding constraints[3].For example, given a ReLU $v_i = max(v_k, 0)$, to enforce v_k to be activated, we remove the constraint from Marabou and add two new ones: $v_i = v_k$, and $v_k \geqslant 0$.

[3]Marabou has a similar optimization, but the user cannot control when or if it is applied.

3.3. Case Study: Visualizing NAPs of A Simple Neural Network



(a) Linear regions in different colors are determined by weights and biases of the neural network. Points colored either red or green constitute the training set.



(b) NAPs are more flexible than L_{∞} norm-balls (boxes) in terms of covering verifiable regions.

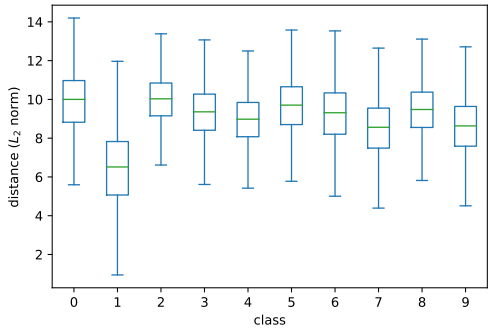
Figure 2. Visualization of linear regions and NAPs as specifications compared to L_{∞} norm-balls.

We show the advantages of NAPs as specifications using a simple example of a three-layer feed-forward neural network that predicts a class of 20 points located on a 2D plane. We trained a neural network consisting of six neurons that

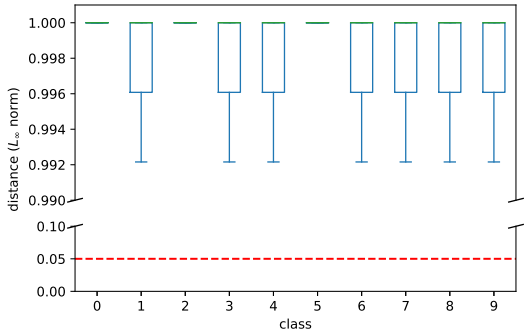
achieves 100% accuracy in the prediction task. The resulting linear regions as well as the training data are illustrated in Figure 2a. Table 2 summarizes the frequency of states of each neuron based on the result of passing all input data through the network, and NAPs for labels 0 and 1. Figure 2b visualizes NAPs for labels 0 and 1, and the unspecified region which provides no guarantees on data that fall into it. The green dot is so close to the boundary between $\mathcal{P}_{\ell=0}$ and the unspecified region that some L_∞ norm-balls (boxes) such as the one drawn in the dashed line may contain an adversarial example from the unspecified region. Thus, what we could verify ends up being a small box within $\mathcal{P}_{\ell=0}$. However, using $\mathcal{P}_{\ell=0}$ as a specification allows us to verify a much more flexible region than just boxes, as suggested by the NAP-augmented robustness property in Section 3.2. This idea generalizes beyond the simple 2D case, and we will illustrate its effectiveness further with a critical evaluation in Section 4.3.

Table 2. The frequency of each ReLU and the NAPs for each label. Activated and deactivated neurons are denoted by $+$ and $-$, respectively, and $*$ denotes an arbitrary neuron state.

Label	Neuron states	#samples	NAP
0 (Green)	$(+,-,-,+,-,+)$	8	$(+,*,-,+,-,+)$
	$(+,+,-,+,-,+)$	2	
1 (Red)	$(+,+,-,-,+,-)$	7	$(*,+,-,+,-,*)$
	$(-,+,-,-,+,-)$	2	
	$(+,+,-,-,+,+)$	1	



(a) The distribution of L_2-norms between any two images from the same label. Images of digit (label) 1 are much similar than that of other digits.



(b) The distribution of L_∞-norms between any two images from the same label. The red line is drawing at 0.05 – the largest ϵ used in VNNCOMP (2021).

Figure 3. Distances between any two images from the same label (class) are quite significant under different metrics of norm.

4. Evaluation

In this section, we validate our observation about the distance between inputs, as well as evaluate our NAPs and NAP properties on networks and datasets from VNNCOMP-21.

4.1. Experiment Setup

Our experiments are based on benchmarks from VNNCOMP (2021) – the annual neural network verification competition. We use 2 of the datasets from the competition: MNIST and CIFAR10. For MNIST, we use the two largest models `mnistfc_256x4` and `mnistfc_256x6`, a 4- and 6-layers fully connected network with 256 neurons for each layer, respectively. For CIFAR10, we use the convolutional neural net `cifar10_small.onnx` with 2568 ReLUs. Experiments are done on a machine with an Intel(R) Xeon(R) CPU E5-2686 and 480GBs of RAM. Timeouts for MNIST and CIFAR10 are 10 and 30 minutes, respectively.

4.2. L_2 and L_∞ Maximum Verified Bounds

We empirically find that the L_2 and L_∞ maximum verifiable bounds are much smaller than the distance between real data, as illustrated in Figure 1. We plot the distribution of distances in L_2 [4] and L_∞ norm between all pairs of images with the same label from the MNIST dataset, as shown in Figure 3. For each class, the smallest L_∞ distance of any two images is significantly larger than 0.05, which is the largest perturbation used in VNNCOMP (2021).

This suggests that the *data as specification* paradigm (i.e., using reference inputs with perturbations bounded in L_2 or L_∞ norm) is not sufficient to verify test set inputs or unseen data. The differences between training and testing data of each class are usually significantly larger than the perturbations allowed in specifications using L_∞ norm-balls.

4.3. The NAP Robustness Property

We conduct two sets of experiments with MNIST and CIFAR10 to demonstrate the NAP and NAP-augmented robustness properties. The results are reported in Tables 3 to 5. For label ℓ from 0 to 9, 'Y' ('N') indicates that the network is (not) robust (i.e., no adversarial example of label ℓ exists).

[4] The L_2 metric is not commonly used by the neural network verification research community as it is less computationally efficient than the L_∞ metric.

Table 3. Robustness of the illustrative example in Figure 1

	0	1	2	3	4	5	6	7	8	9
$\epsilon = 0.2$, no NAP	N	-	N	T/o	T/o	T/o	T/o	T/o	N	T/o
$\epsilon = 0.2, \delta = 1.0$	N	-	N	Y	T/o	T/o	Y	T/o	N	N
$\epsilon = 0.2, \delta = 0.99$	Y	-	Y	Y	Y	Y	Y	Y	Y	Y
$\epsilon = 1, \delta = 0.99$ (NAP robustness property)	Y	-	Y	Y	Y	Y	Y	Y	Y	Y

Table 4. Inputs that are not robust can be augmented with a NAP to be robust. With $\delta = 0.99$, all inputs can be verified to be robust at $\epsilon = 0.05$ – the largest checked ϵ in VNNCOMP-21(not shown)

	0	1	2	3	4	5	6	7	8	9
$\mathcal{O}(\mathbf{x_0}) = \mathbf{0}$										
$\epsilon = 0.05, \delta = 1.0$	-	Y	Y	Y	Y	Y	Y	Y	Y	Y
$\epsilon = 0.3, \delta = 0.99$	-	Y	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_1}) = \mathbf{1}$										
$\epsilon = 0.05, \delta = 1.0$	Y	-	Y	Y	Y	Y	Y	Y	N	Y
$\epsilon = 0.3, \delta = 0.99$	Y	-	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_2}) = \mathbf{0}$										
$\epsilon = 0.05, \delta = 1.0$	-	T/o	T/o	Y	T/o	T/o	Y	N	T/o	T/o
$\epsilon = 0.3, \delta = 0.99$	-	Y	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_3}) = \mathbf{7}$										
$\epsilon = 0.05, \delta = 1.0$	N	T/o	Y	Y	T/o	T/o	Y	-	N	T/o
$\epsilon = 0.3, \delta = 0.99$	Y	Y	Y	Y	Y	Y	Y	-	Y	Y
$\mathcal{O}(\mathbf{x_4}) = \mathbf{9}$										
$\epsilon = 0.05, \delta = 1.0$	T/o	Y	Y	Y	Y	Y	N	Y	N	-
$\epsilon = 0.3, \delta = 0.99$	Y	T/o	T/o	Y	N	Y	T/o	T/o	T/o	-
$\mathcal{O}(\mathbf{x_5}) = \mathbf{1}$										
$\epsilon = 0.05, \delta = 1.0$	Y	-	N	Y	Y	Y	Y	N	N	N
$\epsilon = 0.3, \delta = 0.99$	Y	-	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_6}) = \mathbf{9}$										
$\epsilon = 0.05, \delta = 1.0$	T/o	T/o	T/o	T/o	T/o	T/o	T/o	T/o	T/o	-
$\epsilon = 0.3, \delta = 0.99$ (mnistfc_256x6)	Y	Y	Y	Y	Y	Y	Y	Y	Y	-

'T/o' means the verification of robustness timed out.

MNIST with fully connected NNs In Figure 1, we show an illustrative image $\mathcal{I}$ (of digit 1) and its adversarial example within the distance of $L_\infty = 0.2$. As shown in Table 3, three different kinds of counter-example can be found within this distance. In contrast, the last row shows that all input images in the *entire input space* following the mined NAP specification $\mathcal{P}_{\ell=1}^{\delta=0.99}$ can be safely verified. It is worth noting that this specification covers 84% (959/1135) of the test set inputs (Table 1). To our best knowledge, this is the first specification for MNIST dataset that covers a substantial fraction of testing images. To some extent, it serves as a reasonable machine-checkable definition of digit 1 in MNIST.

The second and third rows of Table 3 show the robustness of combining L_∞ norm perturbation and NAPs as the specification. The third row is well expected as the last row has shown that the network is robust against NAP itself (without L_∞ norm constraint). It is interesting to see that when we increase δ to 1.0, the mined NAP specification $\mathcal{P}_{\ell=1}^{\delta=1.0}$ becomes too general and covers a much larger region that includes more than 99% (1124/1135) testing images as shown in Table 1. As a result, together with $L_\infty = 0.2$ constraint, only two classes of adversarial examples can be safely verified, which is still better than only using $L_\infty = 0.2$ perturbation as the specification.

Table 5. Augmented robustness with CIFAR10 and CNN.

ϵ	0.012			0.024			0.12		
δ	0.99	0.95	0.9	0.99	0.95	0.9	0.99	0.95	0.9
$\mathcal{O}(x_0) = 8$	Y	Y	Y	N	T/o	Y	T/o	Y	Y
$\mathcal{O}(x_1) = 6$	T/o	N	Y	N	N	Y	N	N	Y
$\mathcal{O}(x_2) = 0$	Y	Y	Y	Y	Y	Y	N	N	N
$\mathcal{O}(x_3) = 1$	N	N	N	N	N	N	N	N	N
$\mathcal{O}(x_4) = 9$	N	Y	Y	N	N	N	N	N	N
$\mathcal{O}(x_5) = 7$	Y	Y	Y	N	T/o	Y	N	Y	Y
$\mathcal{O}(x_6) = 3$	Y	Y	Y	Y	Y	Y	N	N	N

We further study how NAP-augmented specification helps to improve the verifiable bound. Specifically, we collect all (x, ϵ) tuples in VNNCOMP-21 MNIST benchmarks that are known to be not robust (an adv. example is found in $B(x, \epsilon)$). Among them, the first six tuples correspond to `mnistfc_256x4` and the last one corresponds to `mnistfc_256x6`. Table 4 reports the verification results with NAP augmented specification.

For the first six instances, using the NAP augmented specifications $B^+(\cdot, \epsilon = 0.05, \mathcal{P}^{\delta=1.0})$ enables the verification against more labels, outperforming using only L_∞ perturbation as the specification. By slightly relaxing the NAP ($\delta = 0.99$) , *all* of the chosen inputs can be proven to be robust. Furthermore, with $\delta = 0.99$, we can verify the robustness for 6 of the 7 inputs (Table 4) with $\epsilon = 0.3$, which is *an order of magnitude* bigger bound than before. Note that decreasing δ specifies a smaller region, usually allowing verification with bigger ϵ, but a smaller region tends to cover fewer testing inputs. Thus, choosing an appropriate δ is crucial for having useful NAPs.

CIFAR10 with CNN To show that our insights and methods can be applied to more complicated datasets and network topologies, we conduct the second set of experiments using convolutional neural nets trained on the CIFAR10 dataset. We extract all (x, ϵ) tuples in the CIFAR10 dataset that are known to be not robust from VNNCOMP-21 (an adv. example is found in $B(x, \epsilon)$) and verify them using augmented NAP. For CIFAR10, $\mathcal{P}^{\delta=1.0}$ does not exist, thus we use $\mathcal{P}^{\delta=.99}$, $\mathcal{P}^{\delta=.95}$ and $\mathcal{P}^{\delta=.90}$. We follow the scenario used in VNNCOMP-21 and test the robustness against $(\textit{correctLabel} + 1) \mod 10$, which is from Marabou-cifar10 in VNN-COMP 2021. This property checks whether an image with label N may be misclassified as $N + 1$ after undergoing some perturbation. The results are reported in Table 5. As with MNIST, we observe that by relaxing δ, we were able to verify more examples at every ϵ. Even with $\epsilon = 0.12$ ($10\times$ the verifiable bound, which translates to an input space $10^{3072}\times$ bigger!), by slightly relaxing δ to 0.9, we can verify 3 out of 7 inputs.

Note there are both garbage and good images within this epsilon. This is the reason NAPs look at the activation pattern first. If an input has no counterexamples in that epsilon it is a strong indication of robustness (albeit limited to a given activation pattern). On the other hand, a counterexample of NAP can be either a good adversarial example (if it is a good image), or an example to refine the NAP specification itself (if it is a garbage image). We always aim to verify the specification with the largest epsilon possible and decrease the epsilon as necessary.

4.4. The Non-ambiguity Property of Mined NAPs

We evaluate the non-ambiguity property of our mined NAP at different δs on MNIST. At $\delta = 1.0$, we can construct inputs that follow any pair of NAP, indicating that $\mathcal{P}^{\delta=1.0}$s do not satisfy the property. However, by setting $\delta = 0.99$, we are able to prove the non-ambiguity for *all* pairs of NAPs, through both trivial cases and invoking Marabou. This is because relaxing δ leaves more neurons in NAPs, making it more difficult to violate the non-ambiguity property.

The non-ambiguity property of NAPs holds an important prerequisite for neural networks to achieve a sound classification result. Otherwise, the final prediction of inputs with two different labels may become indistinguishable. We argue that mined NAPs should demonstrate strong non-ambiguity properties and ideally, all inputs with the same label i should follow the same $\mathcal{P}_{\ell=i}$. However, this strong statement may fail even for an accurate model when the training dataset itself is problematic, as what we observed in Figure 1d as well as many examples in Appendix C. These examples are not only similar to the model but also to humans despite being labeled differently. The experiential results also suggest our mined NAPs do satisfy the strong statement proposed above if excluding these noisy samples.

5. Related Work and Future Directions

5.1. Abstract Interpretation in Verifying Neural Networks

The software verification problem is undecidable in general (Rice, 1953). Given that a Neural Network can also be considered a program, verifying any non-trivial property of a Neural network is also undecidable. Prior work on neural network verification includes specifications that are linear functions of the output of the network: Abstract Interpretation (AbsInt) (Cousot & Cousot, 1977) pioneered a happy middle ground: by sacrificing completeness, an AbsInt verifier can find proof much quicker, by over-approximating reachable states of the program. Many NN-verifiers have adopted the same technique, such as DeepPoly (Singh et al., 2019), CROWN (Wang et al., 2021), NNV (Tran et al., 2021), etc. They all share the same insight: the biggest bottleneck in verifying Neural Networks is the non-linear activation functions. By abstracting the activation into linear functions as much as possible, the verification can be many orders of magnitude faster than complete methods such as Marabou. However, there is no free lunch: Abstract-based verifiers are inconclusive and may not be able to verify properties even when they are correct.[5] On the other hand, the *neural representation as specification* paradigm proposed in this work can be naturally viewed as a method of Abstract Interpretation, in which we abstract the state of each neuron to only activated and deactivated by leveraging NAPs. We would like to explore more refined abstractions of the values of the neurons. For example, we could consider abstractions such as $(-\infty, 0], (0, 1], (1, +\infty)$, which represent a more detailed characterization of neuron values, in contrast to our current NAPs that use the abstractions $(-\infty, 0], (0, +\infty)$. We leave the exploration of such refined abstractions for future work.

5.2. Neural Activation Pattern in Interpreting Neural Networks

There are many attempts aimed to address the black-box nature of neural networks by highlighting important features in the input, such as Saliency Maps (Simonyan et al., 2014; Selvaraju et al., 2016) and LIME(Ribeiro et al., 2016). But these methods still pose the question of whether the prediction and explanation can be trusted or even verified. Another direction is to consider the internal decision-making process of neural networks such as Neural Activation Patterns (NAP). One popular line of research relating to NAPs is to leverage them in feature visualization (Yosinski et al., 2015; Bäuerle et al., 2022; Erhan et al., 2009), which investigates what kind of input images could activate certain neurons in the model. Those methods also have the ability to visualize the internal working mechanism of the model to help with transparency. This line of methods is known as activation maximization. While being great at explaining the prediction of a given input, activation maximization methods do not provide a specification based on the activation pattern: at best they can establish a correlation between seeing a pattern and observing an output, but not causality. Moreover, moving from reference sample to revealing neural network activation pattern is limiting as the portion of NAP uncovered is dependent on the input data. This means that it might not be able to handle cases of unexpected test data. Conversely, our method starts from the bottom up: from the activation pattern, we uncover what region of input can be verified. This property of our method grants the capability

[5] Methods such as alpha-beta CROWN (Wang et al., 2021) claim to be complete even when they are Abstract-based because the abstraction can be controlled to be as precise as the original activation function, thus reducing the method back to a complete one.

to be generalized. Motivated by our promising results, we would like to generalize our approach to modern deep learning models such as Transformers (Vaswani et al., 2017), whose activation patterns have been proven to play a crucial role in understanding the essence of given tasks, rather than merely learning surface statistics (Li et al., 2023). Gopinath et al. also focus on neural network explanation by studying input properties and layer properties (Gopinath et al., 2019). They collect the activations of all neurons in a specific layer and use them as features to learn a decision tree, which serves as a formal interpretation for that layer. In contrast, our work aims to identify a dominant neural activation pattern that captures a significant portion of desired inputs from a specific class while excluding adversarial inputs.

6. Conclusion

We propose a new paradigm of neural network specifications, which we call *neural representation as specification*, as opposed to the traditional *data as specifications*. Specifically, we leverage neural network activation patterns (NAPs) to specify the correct behaviors of neural networks. We argue this could address two major drawbacks of "data as specifications". First, NAPs incorporate intrinsic properties of networks which data fails to do. Second, NAPs could cover much larger and more flexible regions compared to L_∞ norm-balls centered around reference points, making them appealing to real-world applications in verifying unseen data. Moreover, we introduce a relaxation factor that specifies the abstraction level of NAPs, which plays an essential role in determining the effectiveness of NAPs as the specification. We also propose a simple method to mine relaxed NAPs and show that working with NAPs can be easily supported by modern neural network verifiers such as Marabou. Through a simple case study and thorough valuation on the MNIST and CIFAR benchmarks, we show that using NAPs as the specification not only addresses major drawbacks of *data as specifications*, but also demonstrates important properties such as non-ambiguity and one order of magnitude stronger verifiable bounds. We foresee that NAPs have the great potential of serving as simple, reliable, and efficient certificates for neural network predictions.

Acknowledgement

We thank the anonymous reviewers for their insightful comments. This work was supported, in part, by Individual Discovery Grants from the Natural Sciences and Engineering Research Council of Canada and the Canada CIFAR AI Chair Program.

References

Bäuerle, A., Jönsson, D., and Ropinski, T. Neural activation patterns (naps): Visual explainability of learned concepts, 2022. URL https://arxiv.org/abs/2206.10611.

Cousot, P. and Cousot, R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 238–252, 1977.

Dietterich, T. G. and Horvitz, E. Rise of concerns about AI: reflections and directions. *Commun. ACM*, 58(10):38–40, 2015.

Erhan, D., Bengio, Y., Courville, A. C., and Vincent, P. Visualizing higher-layer features of a deep network. 2009.

Gopinath, D., Converse, H., Pasareanu, C. S., and Taly, A. Property inference for deep neural networks. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pp. 797–809. IEEE, 2019. doi: 10.1109/ASE.2019.00079. URL https://doi.org/10.1109/ASE.2019.00079.

Hanin, B. and Rolnick, D. Complexity of linear regions in deep networks, 2019a.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns, 2019b.

Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. Safety verification of deep neural networks. In Majumdar, R. and Kunčak, V. (eds.), *Computer Aided Verification*, pp. 3–29, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63387-9.

Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., and Yi, X. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2020.100270. URL https://www.sciencedirect.com/science/article/pii/S1574013719302527.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In Majumdar, R. and Kunčak, V. (eds.), *Computer Aided Verification*, pp. 97–117, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63387-9.

Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D. L., Kochenderfer, M. J., and Barrett, C. W. The marabou framework for verification and analysis of deep neural networks. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pp. 443–452. Springer, 2019.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent world representations: Exploring a sequence model trained on a synthetic task, 2023.

Nelder, J. A. and Mead, R. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

Ribeiro, M. T., Singh, S., and Guestrin, C. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144, 2016.

Rice, H. G. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953. ISSN 00029947. URL http://www.jstor.org/stable/1990888.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL http://arxiv.org/abs/1610.02391.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Bengio, Y. and LeCun, Y. (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6034.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019. doi: 10.1145/3290354. URL https://doi.org/10.1145/3290354.

Tran, H.-D., Pal, N., Musau, P., Lopez, D. M., Hamilton, N., Yang, X., Bak, S., and Johnson, T. T. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I*, pp. 263–286, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-81684-1. doi: 10.1007/978-3-030-81685-8_12. URL https://doi.org/10.1007/978-3-030-81685-8_12.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, pp. 5998–6008, 2017.

VNNCOMP. Vnncomp, 2021. URL https://sites.google.com/view/vnn2021.

Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.-J., and Kolter, J. Z. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34, 2021.

Wing, J. M. A specifier's introduction to formal methods. *Computer*, 23(9):8–24, 1990.

Wing, J. M. Trustworthy AI. *Commun. ACM*, 64(10):64–71, 2021.

Xu, H., Ma, Y., Liu, H., Deb, D., Liu, H., Tang, J., and Jain, A. K. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17:151–178, 2020.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. Understanding neural networks through deep visualization, 2015. URL https://arxiv.org/abs/1506.06579.

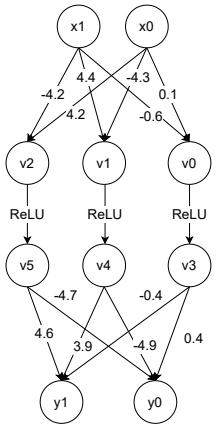
A. A running example

To help with illustrating later ideas, we present a two-layer feed-forward neural network XNET (Figure 4a) to approximate an analog XOR function $f(x_0, x_1) : [[0, 0.3] \cup [0.7, 1]]^2 \rightarrow \{0, 1\}$ such that $f(x_0, x_1) = 1$ iff $(x_0 \leqslant 0.3 \wedge x_1 \geqslant 0.7)$ or $(x_0 \geqslant 0.7 \wedge x_1 \leqslant 0.3)$. The network computes the function

$$F_{\text{XNET}}(x) = \boldsymbol{W}^1 \max(\boldsymbol{W}^0(x) + \boldsymbol{b}^0, 0) + \boldsymbol{b}^1$$

where $x = [x_0, x_1]$, and values of $\boldsymbol{W}^0, \boldsymbol{W}^1, \boldsymbol{b}^0, \boldsymbol{b}^1$ are shown in edges of Figure 4a. $\mathcal{O}(x) = 0$ if $F_{\text{XNET}}(x)[0] > F_{\text{XNET}}(x)[1]$, $\mathcal{O}(x) = 1$ otherwise.

Note that the network is not arbitrary. We have obtained it by constructing two sets of 1 000 randomly generated inputs, and training on one and validating on the other until the NN achieved a perfect F1-score of 1.



(a) XNET: A NN that computes the analog XOR function.

$$
\begin{aligned}
v_0 &= 0.1x_0 - 0.6x_1 \\
v_1 &= -4.3x_0 + 4.4x_1 \\
v_2 &= 4.2x_0 - 4.2x_1 \\
v_3 &= max(v_0, 0) \\
v_4 &= max(v_1, 0) \\
v_5 &= max(v_2, 0) \\
y_0 &= 0.4v_3 - 4.9v_4 + 3.9v_5 + 6.7 \\
y_1 &= -0.4v_3 + 3.9v_4 + 4.6v_5 - 7.4 \\
x_0 &\leqslant 0.1 \wedge x_0 \geqslant 0.02 \\
x_1 &\leqslant 0.1 \wedge x_1 \geqslant 0.02 \\
0 &< y_0 - y_1
\end{aligned}
$$

(b) Marabou's system of constraints for verifying that XNET is 0.04-robust at (0.06, 0.06)

$$
\begin{aligned}
v_0 &= 0.1x_0 - 0.6x_1 \\
v_1 &= -4.3x_0 + 4.4x_1 \\
v_2 &= 4.2x_0 - 4.2x_1 \\
v_3 &= v_0 \\
v_4 &= max(v_1, 0) \\
v_5 &= 0 \\
y_0 &= 0.4v_4 - 4.9v_5 + 3.9v_6 + 6.7 \\
y_1 &= -0.4v_4 + 3.9v_5 + 4.6v_6 - 7.4 \\
x_0 &\leqslant 0.3 \wedge x_0 \geqslant 0 \\
x_1 &\leqslant 0.3 \wedge x_1 \geqslant 0 \\
v_0 &\geqslant 0 \\
v_2 &\leqslant 0
\end{aligned}
$$

(c) Check if $\mathcal{P}_{\ell=1} = ((z_0), ())$ and $\mathcal{P}_{\ell=0} = ((), (z_2))$ are non-ambiguous in the first quadrant using Marabou

Figure 4. Using Marabou to verify NAP properties of XNET.

B. Other Evaluations

B.1. L_1-norms of distance

Figure 5 shows the distributions of L_1-norms of all image pairs from the same class, similar to Figure 3, the distances between image pairs from class 1 are much smaller compare to other classes.

B.2. Overlap ratio

Figure 6 shows the heatmap of the overlap ratio between any two classes for 6 δ values. For the grid in each column in a heatmap, the overlap ratio is calculated by the number of overlapping neurons of the NAPs of the class labelled for the row and the column divided by the number of neurons in the NAP of the class labelled for the column, which is why the values in the heatmaps are not symmetric along the diagonal. Based on the shade of the colors in our heapmap, we can see that, during the process of decreasing δ, the overlapping ratios decrease first and then increase in general, it might because that, with the loose of restriction on when a neuron is considered as activated/inactivated, more neurons are included in the NAP, which means more constrains, but at the same time, for two NAPs of any two classes, it is more likely that they have more neurons appearing in both NAPs.

Table 6 shows the maximum overlap ratio for one class, that is, for one reference class, the maximum overlap ratio between this reference class and any other class. This table is basically extracting the maximum values of each column other than the 1 on the diagonal in our heatmap in Figure 6, in the each column of our table, it also follows the pattern that the value of overlap ratio decreases first and then increase with the decrease of δ.

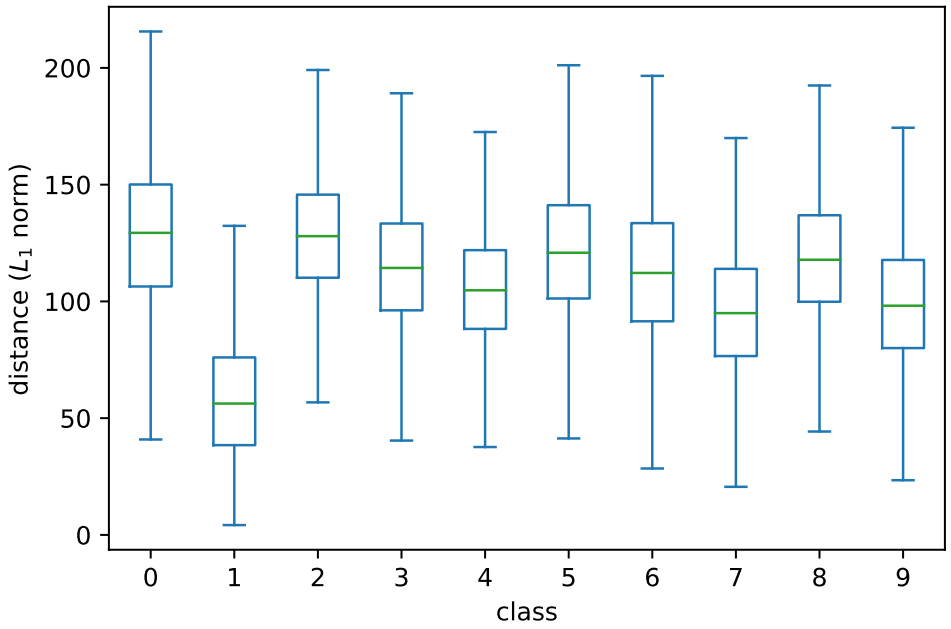


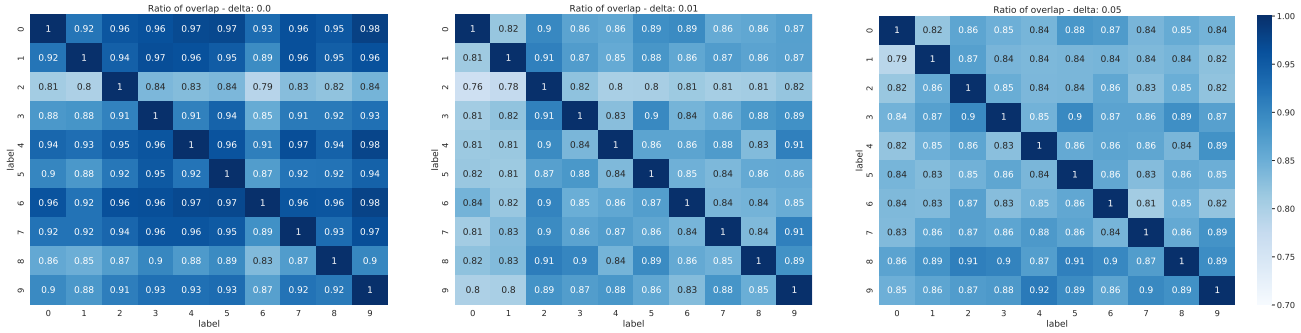
Figure 5. The distribution of L_1-norms of all image pairs for each class.

Table 6. The maximum overlap ratio for each label (class) on a given NAP$^\delta$ for MNIST. Each cell is obtained by $\max_i |N^\delta_{col} \bigcap N^\delta_i|/|N^\delta_{col}|$ where N^δ_{col} is the set of neurons in the dominant pattern for the label (class) in the header of the column of the selected cell with the given δ, N_i is the set of neurons in the dominant pattern for the label (class) i with the given δ.

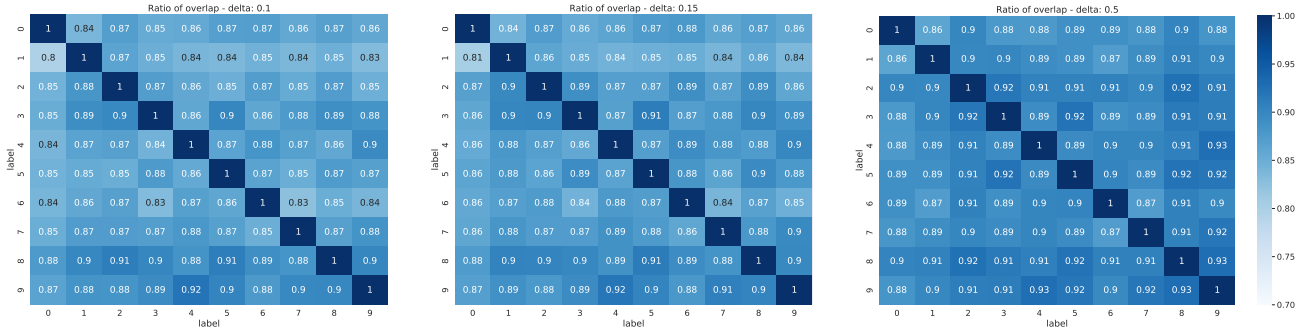
	0	1	2	3	4	5	6	7	8	9
1.00	0.959	0.928	0.963	0.966	0.972	0.973	0.930	0.965	0.957	0.981
0.99	0.844	0.834	0.911	0.901	0.881	0.898	0.895	0.884	0.880	0.908
0.95	0.864	0.885	0.909	0.904	0.915	0.908	0.899	0.897	0.890	0.893
0.90	0.877	0.900	0.910	0.901	0.921	0.910	0.890	0.899	0.900	0.901
0.85	0.876	0.904	0.904	0.900	0.919	0.913	0.893	0.907	0.904	0.900
0.75	0.893	0.922	0.913	0.912	0.928	0.925	0.905	0.916	0.916	0.913
0.50	0.903	0.905	0.925	0.923	0.926	0.923	0.907	0.918	0.927	0.927

C. Misclassification Examples

In this section, we display some interesting exmaples from the the MNIST test set that follow the NAP of some class other than their ground truth, which means these images are misclassified. We consider these samples interesting because, instead of misclassification, it is more reasonable to say that these images are given wrong ground truth from human perspective.

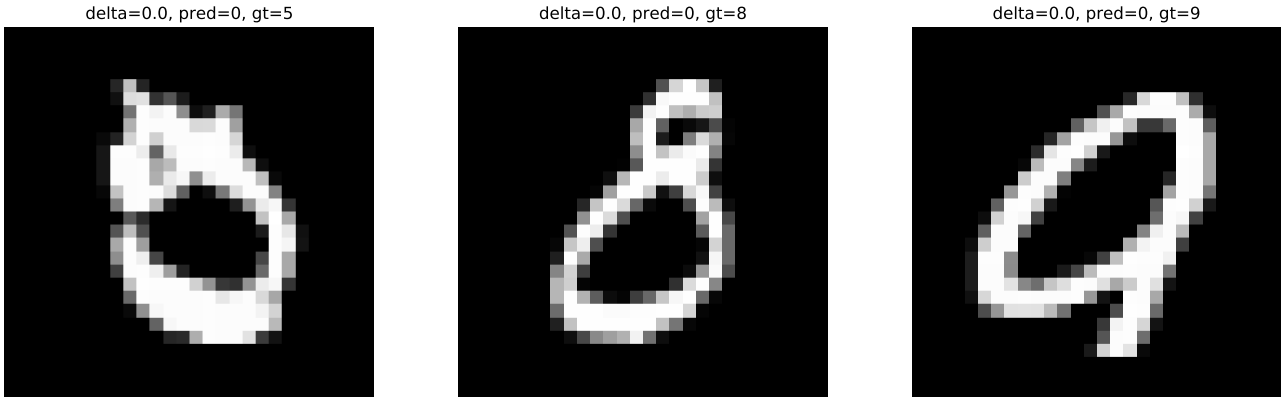


(a) $\delta = 1.00$ (b) $\delta = 0.99$ (c) $\delta = 0.95$



(d) $\delta = 0.90$ (e) $\delta = 0.85$ (f) $\delta = 0.50$

Figure 6. Overlap ratio of the dominant pattern of two labels (classes) on a given NAP$^{\delta}$. Values in each grid are obtained by $|N^{\delta}_{col} \bigcap N^{\delta}_{row}|/|N^{\delta}_{col}|$ where N^{δ}_{col} is the set of neurons in the dominant pattern for the label (class) of the column of the selected grid with the given δ, N_{row} is the set of neurons in the dominant pattern for the label (class) of the row of the selected grid with the given δ.

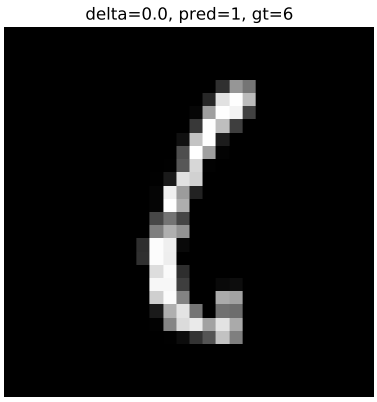


(a) A testing image from MNIST with ground truth 5, classified as 0

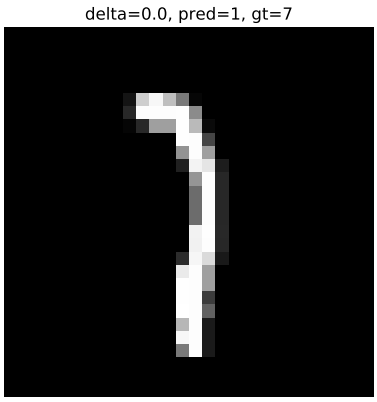
(b) A testing image from MNIST with ground truth 8, classified as 0

(c) A testing image from MNIST with ground truth 9, classified as 0

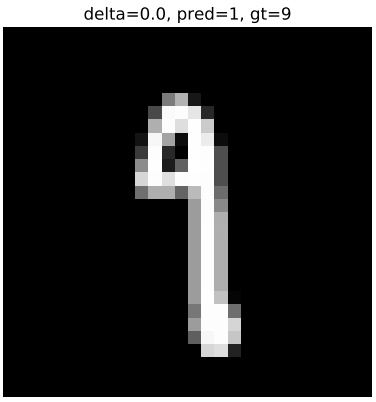
Figure 7. Some interesting test images from MNIST that are misclassified as 0 and also follow the NAP of class 0.



(a) A testing image from MNIST with ground truth 6, classified as 1

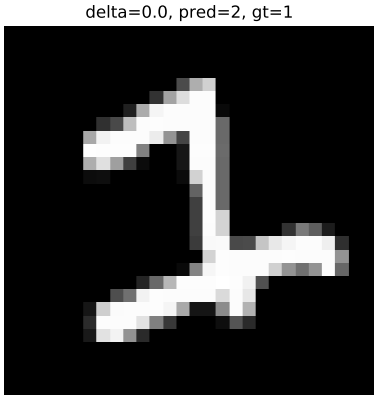


(b) A testing image from MNIST with ground truth 7, classified as 1

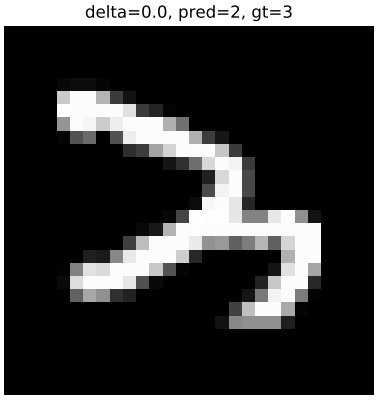


(c) A testing image from MNIST with ground truth 9, classified as 1

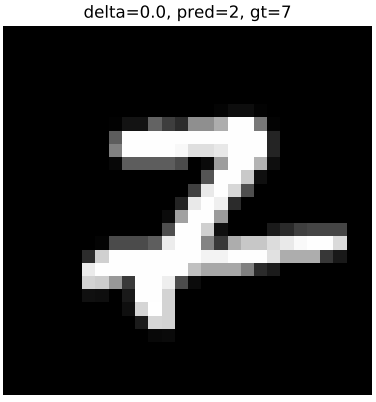
Figure 8. Some interesting test images from MNIST that are misclassified as 1 and also follow the NAP of class 1.



(a) A testing image from MNIST with ground truth 1, classified as 2

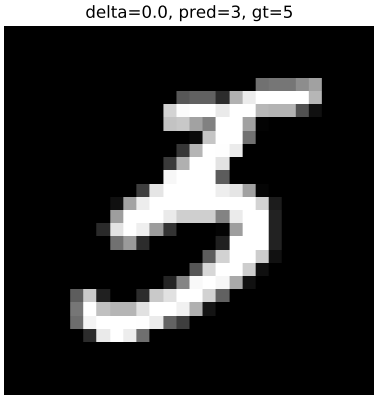


(b) A testing image from MNIST with ground truth 3, classified as 2

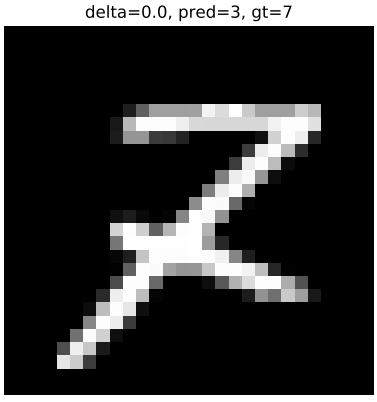


(c) A testing image from MNIST with ground truth 7, classified as 2

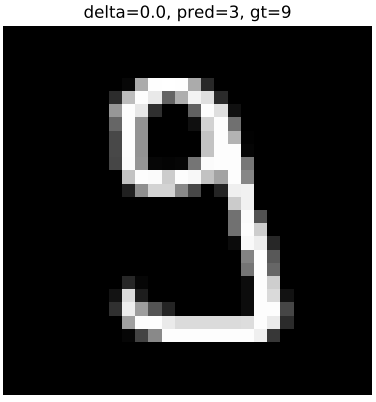
Figure 9. Some interesting test images from MNIST that are misclassified as 2 and also follow the NAP of class 2.



(a) A testing image from MNIST with ground truth 5, classified as 3

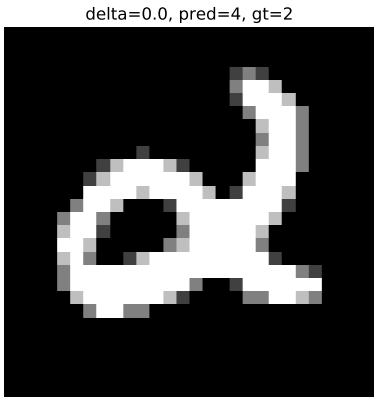


(b) A testing image from MNIST with ground truth 7, classified as 3

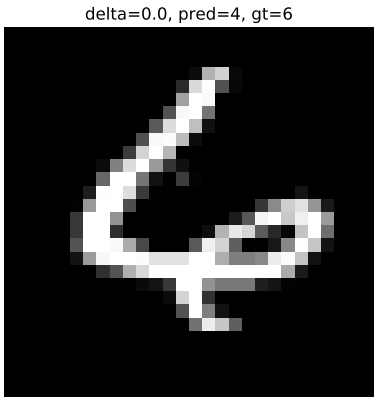


(c) A testing image from MNIST with ground truth 9, classified as 3

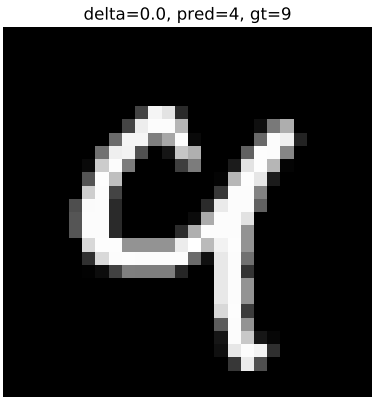
Figure 10. Some interesting test images from MNIST that are misclassified as 3 and also follow the NAP of class 3.



(a) A testing image from MNIST with ground truth 2, classified as 4

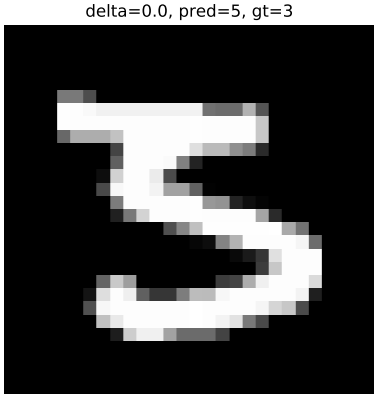


(b) A testing image from MNIST with ground truth 6, classified as 4

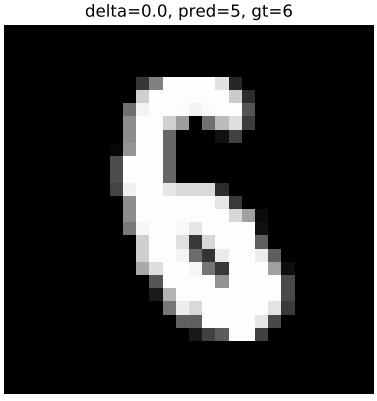


(c) A testing image from MNIST with ground truth 9, classified as 4

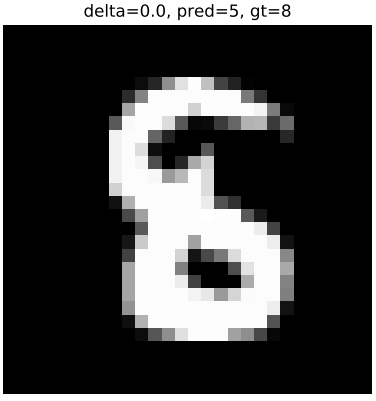
Figure 11. Some interesting test images from MNIST that are misclassified as 4 and also follow the NAP of class 4.



(a) A testing image from MNIST with ground truth 3, classified as 5

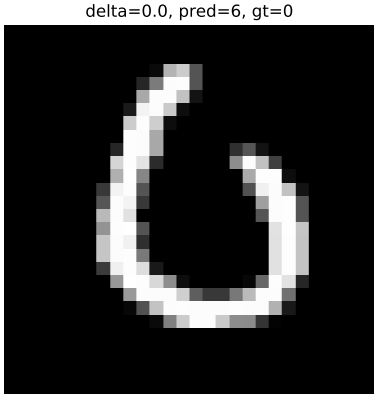


(b) A testing image from MNIST with ground truth 6, classified as 5

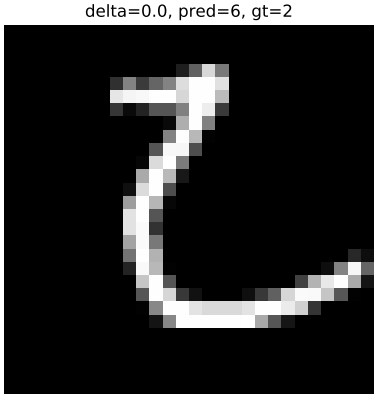


(c) A testing image from MNIST with ground truth 8, classified as 5

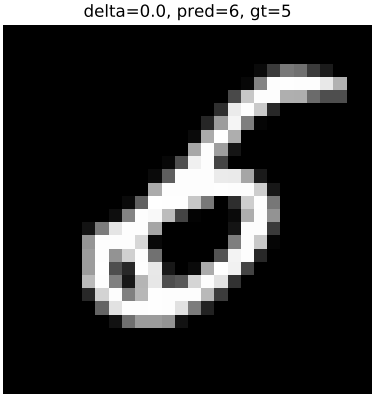
Figure 12. Some interesting test images from MNIST that are misclassified as 5 and also follow the NAP of class 5.



(a) A testing image from MNIST with ground truth 0, classified as 6



(b) A testing image from MNIST with ground truth 3, classified as 6



(c) A testing image from MNIST with ground truth 5, classified as 6

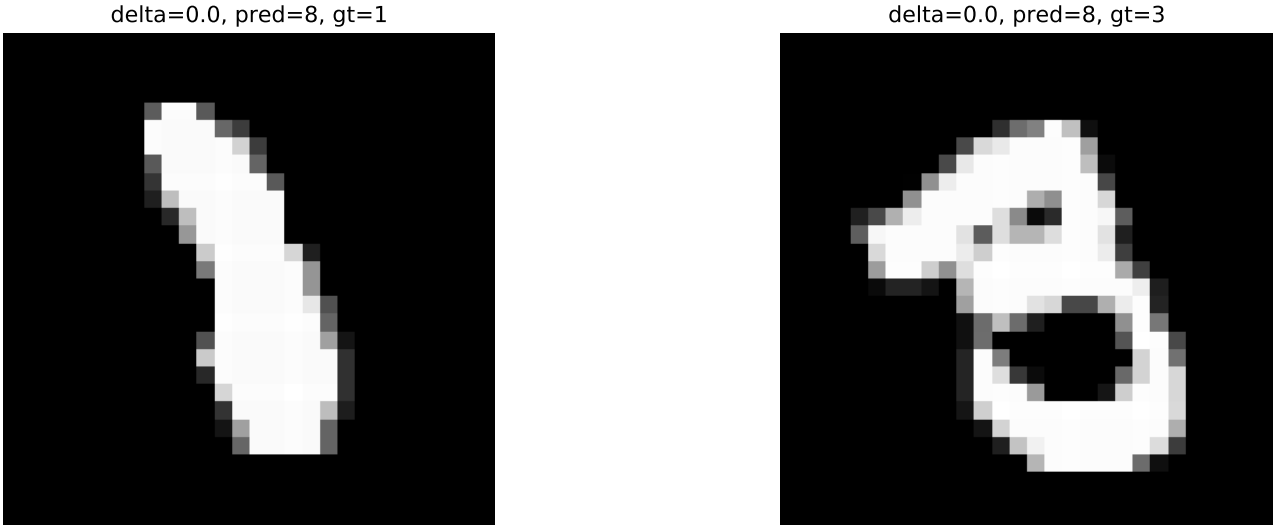
Figure 13. Some interesting test images from MNIST that are misclassified as 6 and also follow the NAP of class 6.

(a) A testing image from MNIST with ground truth 2, classified as 7

(b) A testing image from MNIST with ground truth 3, classified as 7

(c) A testing image from MNIST with ground truth 9, classified as 7

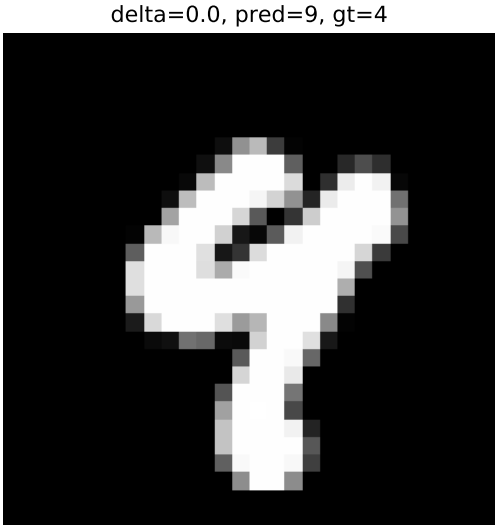
Figure 14. Some interesting test images from MNIST that are misclassified as 7 and also follow the NAP of class 7.



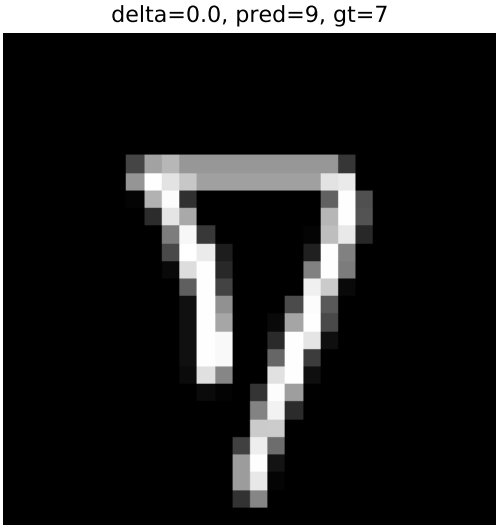
(a) A testing image from MNIST with ground truth 1, classified as 8

(b) A testing image from MNIST with ground truth 3, classified as 8

Figure 15. Some interesting test images from MNIST that are misclassified as 8 and also follow the NAP of class 8.



(a) A testing image from MNIST with ground truth 4, classified as 9



(b) A testing image from MNIST with ground truth 7, classified as 9

Figure 16. Some interesting test images from MNIST that are misclassified as 9 and also follow the NAP of class 9.