Verification of Binarized Deep Neural Networks - An Overview

Mădălina Erașcu and Andreea Postovan

West University of Timișoara, Romania

madalina.erascu@e-uvt.ro

December 7th, 2022

Outline

Contents

Motivation

- Deep learning is used on safety-critical systems, however, the main criticism and concern is on the lack of understanding the decision process behind the networks.

Motivation

- Deep learning is used on safety-critical systems, however, the main criticism and concern is on the lack of understanding the decision process behind the networks.
- Question: Can we trust the decisions that neural networks make?

Motivation

- Deep learning is used on safety-critical systems, however, the main criticism and concern is on the lack of understanding the decision process behind the networks.
- Question: Can we trust the decisions that neural networks make?

Define the properties that the neural networks should have and verify if they hold for the network.

Litarature

- Neural Networks Verification

Litarature

- Neural Networks Verification
- Binarized Neural Networks Verification

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
- Binarized Neural Networks Verification

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
  - VNN-COMP `https://sites.google.com/view/vnn20/vnncomp`
- Binarized Neural Networks Verification

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
  - VNN-COMP `https://sites.google.com/view/vnn20/vnncomp`
- Binarized Neural Networks Verification
  - N. Narodytska et al. *Verifying properties of binarized deep neural networks*, 2018 [9]

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
  - VNN-COMP `https://sites.google.com/view/vnn20/vnncomp`
- Binarized Neural Networks Verification
  - N. Narodytska et al. *Verifying properties of binarized deep neural networks*, 2018 [9]
  - N. Narodytska *Formal Analysis of Deep Binarized Neural Networks*, 2018 [8]

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
  - VNN-COMP `https://sites.google.com/view/vnn20/vnncomp`
- Binarized Neural Networks Verification
  - N. Narodytska et al. *Verifying properties of binarized deep neural networks*, 2018 [9]
  - N. Narodytska *Formal Analysis of Deep Binarized Neural Networks*, 2018 [8]
  - G. Amir et al. *An SMT-based approach for verifyig binarized neural networks*, 2021

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
  - VNN-COMP `https://sites.google.com/view/vnn20/vnncomp`
- Binarized Neural Networks Verification
  - N. Narodytska et al. *Verifying properties of binarized deep neural networks*, 2018 [9]
  - N. Narodytska *Formal Analysis of Deep Binarized Neural Networks*, 2018 [8]
  - G. Amir et al. *An SMT-based approach for verifyig binarized neural networks*, 2021
  - K. Jia and M. Rinard. *Efficient Exact Verification of Binarized Neural Networks*, 2020

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
  - VNN-COMP `https://sites.google.com/view/vnn20/vnncomp`
- Binarized Neural Networks Verification
  - N. Narodytska et al. *Verifying properties of binarized deep neural networks*, 2018 [9]
  - N. Narodytska *Formal Analysis of Deep Binarized Neural Networks*, 2018 [8]
  - G. Amir et al. *An SMT-based approach for verifyig binarized neural networks*, 2021
  - K. Jia and M. Rinard. *Efficient Exact Verification of Binarized Neural Networks*, 2020
  - M. Lechner et al. *Quantization-aware Interval Bound Propagation for Training Certifiably Robust Quantized*, 2022 Neural Networks

Litarature

- Neural Networks Verification
  - State-of-the-art AAAI2022: `https://neural-network-verification.com`
  - VNN-COMP `https://sites.google.com/view/vnn20/vnncomp`
- Binarized Neural Networks Verification
  - N. Narodytska et al. *Verifying properties of binarized deep neural networks*, 2018 [9]
  - N. Narodytska *Formal Analysis of Deep Binarized Neural Networks*, 2018 [8]
  - G. Amir et al. *An SMT-based approach for verifyig binarized neural networks*, 2021
  - K. Jia and M. Rinard. *Efficient Exact Verification of Binarized Neural Networks*, 2020
  - M. Lechner et al. *Quantization-aware Interval Bound Propagation for Training Certifiably Robust Quantized*, 2022 Neural Networks
  - It seems BNN (verification) is on ascending trend (cf. Google Scholar): 2016 - 40 entries, 2017 - 127, 2018 - 376, 2019 - 529, 2020 - 676, 2021 - 756, 2022 - 737

Why Verification/Analysis of Deep Neural Networks (DNNs) is important?

- Is there a way to analyze deep neural networks?

Why Verification/Analysis of Deep Neural Networks (DNNs) is important?

- Is there a way to analyze deep neural networks?
- Can we explain their decisions?

Why Verification/Analysis of Deep Neural Networks (DNNs) is important?

- Is there a way to analyze deep neural networks?
- Can we explain their decisions?
- How robust are these networks to perturbations of inputs?

Why Verification/Analysis of Deep Neural Networks (DNNs) is important?

- Is there a way to analyze deep neural networks?
- Can we explain their decisions?
- How robust are these networks to perturbations of inputs?
- How critical is the choice of one architecture over an other?

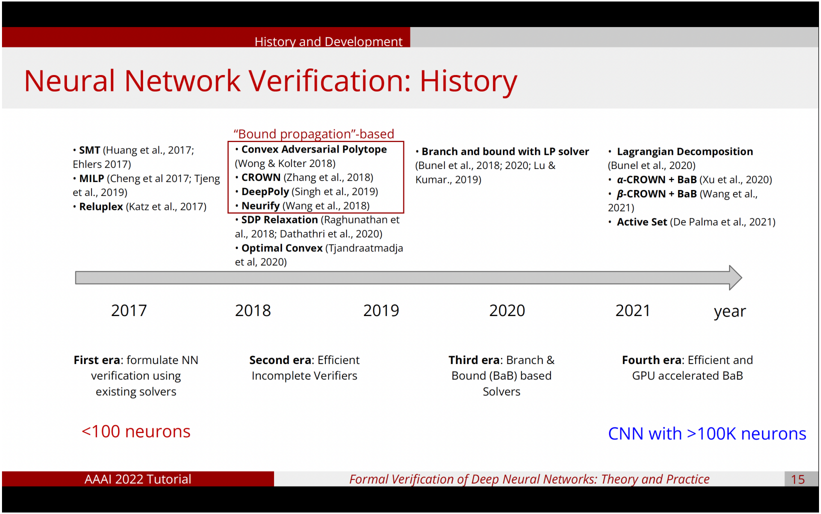
Why Verification/Analysis of Deep Neural Networks (DNNs) is important?

- Is there a way to analyze deep neural networks?
- Can we explain their decisions?
- How robust are these networks to perturbations of inputs?
- How critical is the choice of one architecture over an other?

Verification problem: encode the (exact representation of the) network and the property we aim to verify as a formal statement, using, e.g. ILP, SMT or SAT.

Methods for the Verification of DNNs

Techniques for verification methods for DNNs can not be used for BNNs [3]



Why Binarized Neural Networks (BNNs)?

Features useful in resource constrained environments (embedded devices or mobile phones):

Why Binarized Neural Networks (BNNs)?

Features useful in resource constrained environments (embedded devices or mobile phones):

- memory efficient (weights and activations are primarily binary)

Why Binarized Neural Networks (BNNs)?

Features useful in resource constrained environments (embedded devices or mobile phones):

- memory efficient (weights and activations are primarily binary)
- computationally efficient (activations are binary $\rightsquigarrow$ algorithms for fast binary matrix multiplication.

Why Binarized Neural Networks (BNNs)?

Features useful in resource constrained environments (embedded devices or mobile phones):

- memory efficient (weights and activations are primarily binary)
- computationally efficient (activations are binary $\rightsquigarrow$ algorithms for fast binary matrix multiplication.

Why Binarized Neural Networks (BNNs)?

Features useful in resource constrained environments (embedded devices or mobile phones):

- memory efficient (weights and activations are primarily binary)
- computationally efficient (activations are binary $\rightsquigarrow$ algorithms for fast binary matrix multiplication.

Weights and activations are represented using 1 bit (quantization).

Why Binarized Neural Networks (BNNs)?

Features useful in resource constrained environments (embedded devices or mobile phones):

- memory efficient (weights and activations are primarily binary)
- computationally efficient (activations are binary $\rightsquigarrow$ algorithms for fast binary matrix multiplication.

Weights and activations are represented using 1 bit (quantization).
Performance of BNNs is comparable to that of DNNs (real-value parameters) [2].

Contents

Notations

- $[s] = \{1, 2, ..., s\}$
- $\mathbf{v} = (v_1, ..., v_m), \ \mathbf{v} \in \mathbb{R}$
- $\|v\|_p, \ p \geq 1$ is the L_p-norm $\mathbf{v}$, $\|v\|_p = \sqrt[p]{\sum_{i=1}^{m} |v_i|^p}$

Notations

- $[s] = \{1, 2, ..., s\}$
- $\mathbf{v} = (v_1, ..., v_m),\ \mathbf{v} \in \mathbb{R}$
- $\|v\|_p,\ p \geq 1$ is the L_p-norm $\mathbf{v}$, $\|v\|_p = \sqrt[p]{\sum_{i=1}^{m} |v_i|^p}$
- Formula A is satisfiable/unsatisfiable ...

Notations

- $[s] = \{1, 2, ..., s\}$
- $\mathbf{v} = (v_1, ..., v_m),\ \mathbf{v} \in \mathbb{R}$
- $\|v\|_p,\ p \geq 1$ is the L_p-norm $\mathbf{v}$, $\|v\|_p = \sqrt[p]{\sum_{i=1}^{m} |v_i|^p}$
- Formula A is satisfiable/unsatisfiable ...
- *Image classification problem*: we are given (1) a set of training images drawn from an unknown distribution ν over $X = \mathbb{Z}^n$, where n is the size of individual images, (2) a label for each image $L : \mathbb{Z}^n \rightarrow [s]$.

Notations

- $[s] = \{1, 2, ..., s\}$
- $\mathbf{v} = (v_1, ..., v_m),\ \mathbf{v} \in \mathbb{R}$
- $\|v\|_p,\ p \geq 1$ is the L_p-norm $\mathbf{v}$, $\|v\|_p = \sqrt[p]{\sum_{i=1}^{m} |v_i|^p}$
- Formula A is satisfiable/unsatisfiable ...
- *Image classification problem*: we are given (1) a set of training images drawn from an unknown distribution ν over $X = \mathbb{Z}^n$, where n is the size of individual images, (2) a label for each image $L : \mathbb{Z}^n \to [s]$.
  - *Training*: given a labeled training set, learn a neural network classifier that can be used as inference engine.
  - During the inference the network is *fixed*.

Contents

Properties of neural networks

Let F be a general feedforward neural, $F(\mathbf{x})$ be the output of F on input $\mathbf{x}$ and $l_{\mathbf{x}} = L(\mathbf{x})$ be the ground truth label of $\mathbf{x}$.

Properties of neural networks

Let F be a general feedforward neural, $F(\mathbf{x})$ be the output of F on input $\mathbf{x}$ and $l_{\mathbf{x}} = L(\mathbf{x})$ be the ground truth label of $\mathbf{x}$.

Properties:

1. Robustness: small perturbations on inputs do not affect the output.
   - global robustness: for any valid input, there is no small perturbation that can change the decision of the network on this input.

Definition (Global Robustness)

A feedforward neural network F is globally-robust if for any input $\mathbf{x}$, $\mathbf{x} \in X$ and τ, $\|\tau\|_\infty \leq \epsilon$ we have that $F(\mathbf{x} + \tau) = l_{\mathbf{x}}$.

- local robustness: is defined for a single input x.

Definition (Local Robustness)

A feedforward neural network F is locally-robust for an input $\mathbf{x}$, $\mathbf{x} \in X$, if there does not exist τ, $\|\tau\|_\infty \leq \epsilon$ such that $F(\mathbf{x} + \tau) \neq l_x$.

Properties of neural networks

Let F be a general feedforward neural, $F(\mathbf{x})$ be the output of F on input $\mathbf{x}$ and $l_{\mathbf{x}} = L(\mathbf{x})$ be the ground truth label of $\mathbf{x}$.

Properties:

1. Robustness: small perturbations on inputs do not affect the output.
2. Invertibility: explore a set of inputs that map to a given output (example: what the inputs of the network are, if exist, that map to a given output).

Definition (Local Invertibility)

A feedforward neural network F is locally invertible for an output $\mathbf{s}$ if there exists $\mathbf{x}$, $\mathbf{x} \in C(X)$, such that $F(\mathbf{x}) = \mathbf{s}$, where $C(X)$ denotes the constrained domain of inputs.

> Related problem: how to enumerate multiple, preferably diverse by some measure, inputs of the network that map to a given output.

Properties of neural networks

Let F be a general feedforward neural, $F(\mathbf{x})$ be the output of F on input $\mathbf{x}$ and $l_{\mathbf{x}} = L(\mathbf{x})$ be the ground truth label of $\mathbf{x}$.

Properties:

1. Robustness: small perturbations on inputs do not affect the output.
2. Invertibility: explore a set of inputs that map to a given output (example: what the inputs of the network are, if exist, that map to a given output).
3. Network equivalence: two networks F_1 and F_2 are equivalent if they generate same outputs on all inputs drawn from the domain X.

Definition (Network Equivalence)

Two feedforward neural networks F_1 and F_2 are equivalent if for all $\mathbf{x} \in X$, $F_1(\mathbf{x}) = F_2(\mathbf{x})$.

Application: network alteration.

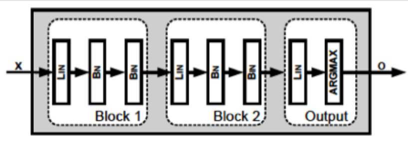
Contents

Binarized Neural Networks (BNNs) [2, 8, 9, 1]

Definition (Binarized Neural Network)

A binarized neural network $BNN : \{-1,1\}^n \to [s]$ is a feedforward network that is composed of d blocks, BLK_1, ..., BLK_{d-1}, O. Formally, given an input $\mathbf{x}$, $BNN(\mathbf{x}) = O(BLK_{d-1}, ...(BLK_1(\mathbf{x})))$

Schematic view of a binarized neural network



Structure of internal and outputs blocks, which stacked together form a BNN. A_k and b_k – parameters of the *LIN* layer; $\alpha_{k_i}, \gamma_{k_i}, \mu_{k_i}, \sigma_{k_i}$ – parameters of the *BN* layer. μ and σ correspond to mean and standard deviation computed in the training phase. The *BIN* layer is parameter free.

Structure of kth Internal block, $\mathrm{BLK}_k : \{-1,1\}^{n_k} \to \{-1,1\}^{n_{k+1}}$ on input $\mathbf{x}_k \in \{-1,1\}^{n_k}$	
LIN	$\mathbf{y} = A_k \mathbf{x}_k + \mathbf{b}_k$, where $A_k \in \{-1,1\}^{n_{k+1} \times n_k}$ and $\mathbf{b}_k \in \mathbb{R}^{n_{k+1}}$
BN	$z_i = \alpha_{k_i}\left(\frac{y_i - \mu_{k_i}}{\sigma_{k_i}}\right) + \gamma_{k_i}$, where $\mathbf{y} = (y_1, \ldots, y_{n_{k+1}})$, and $\alpha_{k_i}, \gamma_{k_i}, \mu_{k_i}, \sigma_{k_i} \in \mathbb{R}$. Assume $\sigma_{k_i} > 0$.
BIN	$\mathbf{x}_{k+1} = \mathrm{sign}(\mathbf{z})$ where $\mathbf{z} = (z_1, \ldots, z_{n_{k+1}}) \in \mathbb{R}^{n_{k+1}}$ and $\mathbf{x}_{k+1} \in \{-1,1\}^{n_{k+1}}$

Structure of Output Block, $O : \{-1,1\}^{n_d} \to [s]$ on input $\mathbf{x}_d \in \{-1,1\}^{n_d}$	
LIN	$\mathbf{w} = A_d \mathbf{x}_d + \mathbf{b}_d$, where $A_d \in \{-1,1\}^{s \times n_d}$ and $\mathbf{b}_d \in \mathbb{R}^s$
ARGMAX	$o = \mathrm{argmax}(\mathbf{w})$, where $o \in [s]$

Contents

Observations

- There is no public repository associated to the papers.
- Few details in the papers about the architecture of the underlying networks so we could not reproduce their results.
- They claim they use binary values only but from the encoding one could observe real values for some layers in the blocks encoding (see next slides).

Contents

Encodings of BNNs

- BNN encoding into Boolean formulae.
- The encoding of the BNN is a conjunction of encodings of its blocks.
- $BINBLK_k(\mathbf{x}_k, \mathbf{x}_{k+1})$ a Boolean function that encodes the kth block (BLK_k) with an input $\mathbf{x}_k$ and an output $\mathbf{x}_{k+1}$.
- $BINO(\mathbf{x}_d, o)$ be a Boolean function that encodes O that takes an input $\mathbf{x}_d$ and outputs $\mathbf{o}$.
- The entire BNN on input $\mathbf{x}$ can be encoded as a Boolean formula, with $\mathbf{x}_1$ (first layer) $= \mathbf{x}$ (input):

$$\left(\bigwedge_{k=1}^{d-1} BINBLK_k(x_k, x_{k+1}) \right) \wedge BINO(x_d, o)$$

- Encodings: MILP $\rightsquigarrow$ ILP $\rightsquigarrow$ SAT.

Mixed Integer Linear Program (MILP) Encoding

- *Encoding of BLK_k*: encode each layer in BLK_k to MILP separately. Let a_i be the i-th row of the matrix A_k, $x_k \in \{-1, 1\}^{n_k}$ denote the input to BLK_k.
- *Linear Transformation*. Transformation for fully connected layer (suitable for convolutions also as they are linear operations). We have:

$$y_i = \langle \mathbf{a}_i, \mathbf{x}_k \rangle + b_i, \quad i = 1, \ldots, n_{k+1} \tag{1}$$

  where $\mathbf{y} = (y_1, \ldots, y_{n_{k+1}}) \in \mathbb{R}^{n_{k+1}}$.
- *Batch Normalization*: takes the output of the linear layer as an input. By definition, we have:

$$\begin{aligned} z_i &= \alpha_{k_i} \left(\frac{y_i - \mu_{k_i}}{\sigma_{k_i}} \right) + \gamma_{k_i}, \quad i = \overline{1, n_{k+1}} \\ \sigma_{k_i} z_i &= \alpha_{k_i} y_i - \alpha_{k_i} \mu_{k_i} + \sigma_{k_i} \gamma_{k_i} \end{aligned} \tag{2}$$

- *Binarization*. For the BIN operation, which implements a sign function, we need to deal with conditional constraints.

$$z_i \geq 0 \Rightarrow v_i = 1 \tag{3}$$

$$z_i < 0 \Rightarrow v_i = -1 \quad i = \overline{1, n_{k+1}} \tag{4}$$

Mixed Integer Linear Program Encoding (cont'd)

- *Encoding of O*: $w_i = \langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i$, $i = 1, .., s$, where $\mathbf{a}_i$ represents the i-th column in A_d and $\mathbf{w} = (w_1, ..., w_s)$. To encode ARGMAX, an ordering relation between w_i's must be imposed:

$$
\begin{aligned}
& w_i \geq w_j \iff d_{ij} = 1 && d_{ij} \text{ newly introduced vars} \\
& \sum_{j=1}^{s} d_{ij} = s \implies o = i && i, j = \overline{1, s}
\end{aligned}
\tag{5}
$$

Mixed Integer Linear Program Encoding (cont'd)

- *Encoding of O*: $w_i = \langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i$, $i = 1, .., s$, where $\mathbf{a}_i$ represents the i-th column in A_d and $\mathbf{w} = (w_1, ..., w_s)$. To encode ARGMAX, an ordering relation between w_i's must be imposed:

$$\begin{aligned} w_i \geq w_j \iff d_{ij} = 1 & \qquad d_{ij} \text{ newly introduced vars} \\ \sum_{j=1}^{s} d_{ij} = s \implies o = i & \qquad i, j = \overline{1, s} \end{aligned} \tag{5}$$

Example

Consider an internal block with two inputs and one output. Suppose we have the following parameters: $A_k = [1, -1]$, $b_k = [-0.5]$, $\alpha_k = [0.12]$, $\mu_k = [-0.1]$, $\sigma_k = [2]$, $\delta_k = [0.1]$.

1. apply the linear transformation: $y_1 = x_{k_1} - x_{k_2} - 0.5$
2. apply batch normalization: $2z_1 = 0.12y_1 + (-0.12) * (-0.1) + 2 * 0.1$
3. apply binarization: $z1 \geq 0 \implies v_1 = 1 \wedge z_1 < 0 \implies v_1 = -1$. $(x_{k+1} = v_1)$.

Mixed Integer Linear Program Encoding (cont'd)

- *Encoding of O*: $w_i = \langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i$, $i = 1, .., s$, where $\mathbf{a}_i$ represents the i-th column in A_d and $\mathbf{w} = (w_1, ..., w_s)$. To encode ARGMAX, an ordering relation between w_i's must be imposed:

$$w_i \geq w_j \iff d_{ij} = 1 \qquad d_{ij} \text{ newly introduced vars}$$
$$\sum_{j=1}^{s} d_{ij} = s \implies o = i \qquad i, j = \overline{1, s} \tag{5}$$

Example

Consider an output block with two inputs and two outputs. We have the following parameters for this block $A_d = [1, -1; -1, 1]$ and $b = [-0.5, 0.2]$.

1. encoding of the linear transformation

$$w_1 = x_{d_1} - x_{d_2} - 0.5 \qquad w_2 = -x_{d_1} + x_{d_2} + 0.2$$

2. Two outputs $\rightsquigarrow$ 4 Boolean variables d_{ij}, $i, j = 1, 2$; $d_{11} = d_{22} = 1 \rightsquigarrow$ consider only non-diagonal variables.

$$w_1 \geq w_2 \iff d_{12} = 1 \quad \wedge \quad w2 < w1 \iff d_{21} = 1$$

3. we compute the output o of the neural network as:

$$d_{11} + d_{12} = 2 \implies o = 1 \quad \wedge \quad d_{21} + d_{22} = 2 \implies o = 2$$

Integer Linear Programming (ILP) Encoding

ILP encoding is smaller than the MILP one.

- *Encoding of BLK_k*: z and y are functional variables of $\mathbf{x}_k$. We can substitute them in (3) and (4) based on (1) and (2) respectively:

$$\frac{\alpha_{k_i}}{\sigma_{k_i}}(\langle \mathbf{a}_i, \mathbf{x}_k \rangle + b_i) - \frac{\alpha_{k_i}}{\sigma_{k_i}}\mu_{k_i} + \gamma_{k_i} \implies v_i = 1$$

- *Linear and batch normalization:*
  - Case $\alpha_{k_i} > 0$, we have:

    $\langle \mathbf{a}_i, \mathbf{x}_k \rangle \geq -\frac{\sigma_{k_i}}{\alpha_{k_i}}\gamma_{k_i} + \mu_{k_i} - b_i \implies x_i' = 1$(see p. 6618 right column, bottom)

    Consider $C_i = \left\lceil -\frac{\sigma_{k_i}}{\alpha_{k_i}}\gamma_{k_i} + \mu_{k_i} - b_i \right\rceil$. We encode (3) and (4):

$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle \geq C_i \Rightarrow v_i = 1$$
$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle < C_i \Rightarrow v_i = -1 \quad i = \overline{1, n_{k+1}}$$

  - Case $\alpha_{k_i} < 0$. Same as above but $C_i = \left\lfloor -\frac{\sigma_{k_i}}{\alpha_{k_i}}\gamma_{k_i} + \mu_{k_i} - b_i \right\rfloor$
  - Case $\alpha_{k_i} = 0$, we have: $\gamma_{k_i} \implies v_i = 1$

Integer Linear Programming (ILP) Encoding (cont'd)

- *Encoding of O*: introduce the Boolean variables d_{ij} avoiding the intermediate variables w_i

$$\langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i \geq \langle \mathbf{a}_j, \mathbf{x}_d \rangle + b_j \iff d_{ij} = 1, \quad i, j = \overline{1, s}$$

$$\iff$$

$$\langle \mathbf{a}_i - \mathbf{a}_j, \mathbf{x}_d \rangle \geq \lceil b_j - b_i \rceil \iff d_{ij} = 1$$

where $\mathbf{a}_i$ and $\mathbf{a}_j$ denote the ith and jth rows in the matrix A_d.

Further, constraints (5) can be used.

Integer Linear Programming (ILP) Encoding (cont'd)

- *Encoding of O*: introduce the Boolean variables d_{ij} avoiding the intermediate variables w_i

$$\langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i \geq \langle \mathbf{a}_j, \mathbf{x}_d \rangle + b_j \iff d_{ij} = 1, \quad i, j = \overline{1, s}$$

$$\iff$$

$$\langle \mathbf{a}_i - \mathbf{a}_j, \mathbf{x}_d \rangle \geq \lceil b_j - b_i \rceil \iff d_{ij} = 1$$

where $\mathbf{a}_i$ and $\mathbf{a}_j$ denote the ith and jth rows in the matrix A_d.

Further, constraints (5) can be used.

Example

Recall the internal block with two inputs and one output: $A_k = [1, -1]$, $b_k = [-0.5]$, $\alpha_k = [0.12]$, $\mu_k = [-0.1]$, $\sigma_k = [2]$, $\delta_k = [0.1]$. We have:

$$x_{k_1} - x_{k_2} \geq \left\lceil \frac{-2}{0.1} * 0.1 - 0.1 - (-0.5) \right\rceil = -1 \Rightarrow v_1 = 1$$

$$x_{k_1} - x_{k_2} < \left\lceil \frac{-2}{0.1} * 0.1 - 0.1 - (-0.5) \right\rceil = -1 \Rightarrow v_1 = -1$$

Integer Linear Programming (ILP) Encoding (cont'd)

- *Encoding of O*: introduce the Boolean variables d_{ij} avoiding the intermediate variables w_i

$$\langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i \geq \langle \mathbf{a}_j, \mathbf{x}_d \rangle + b_j \iff d_{ij} = 1, \quad i, j = \overline{1, s}$$

$$\iff$$

$$\langle \mathbf{a}_i - \mathbf{a}_j, \mathbf{x}_d \rangle \geq \lceil b_j - b_i \rceil \iff d_{ij} = 1$$

where $\mathbf{a}_i$ and $\mathbf{a}_j$ denote the ith and jth rows in the matrix A_d.

Further, constraints (5) can be used.

Example

Recall the output block with two inputs and two outputs: $A_d = [1, -1; -1, 1]$ and $b = [-0.5, 0.2]$. We have

$$x_{d_1} - x_{d_2} - 0.5 \geq -x_{d_1} + x_{d_2} + 0.2 \iff d_{12} = 1 \text{ equiv. to } x_{d_1} - x_{d_2} \geq \left\lceil \frac{0.7}{2} \right\rceil \iff d_{12} = 1$$

$$x_{d_2} - x_{d_1} + 0.2 \geq -x_{d_2} + x_{d_1} - 0.5 \iff d_{21} = 1 \text{ equiv. to } x_{d_1} - x_{d_2} \leq \left\lceil \frac{0.7}{2} \right\rceil \iff d_{21} = 1$$

Contents

SAT Encoding

Naive approach: encode the BLK and O blocks into CNF is to directly translate their ILP encoding defined above into SAT.

SAT Encoding

Naive approach: encode the BLK and O blocks into CNF is to directly translate their ILP encoding defined above into SAT.
Drawback: inefficient – the resulting encoding will be very large.

SAT Encoding

Naive approach: encode the BLK and O blocks into CNF is to directly translate their ILP encoding defined above into SAT.
Drawback: inefficient – the resulting encoding will be very large.
Solution: exploit the properties of BNNs.

SAT Encoding

Naive approach: encode the BLK and O blocks into CNF is to directly translate their ILP encoding defined above into SAT.
Drawback: inefficient – the resulting encoding will be very large.
Solution: exploit the properties of BNNs.

Sequential counters for encoding cardinality constraints

SAT Encoding

Naive approach: encode the BLK and O blocks into CNF is to directly translate their ILP encoding defined above into SAT.
Drawback: inefficient – the resulting encoding will be very large.
Solution: exploit the properties of BNNs.

Sequential counters for encoding cardinality constraints

Consider a *cardinality constraint*: $\sum_{i=1}^{m} l_i \geq C$, where $l_i \in \{0, 1\}$ is a Boolean variable and C is a constant. This can be compiled into CNF using *sequential counters* $SQ(l, C)$, $l = (l_1, ..., l_m)$.

SAT Encoding

Naive approach: encode the BLK and O blocks into CNF is to directly translate their ILP encoding defined above into SAT.
Drawback: inefficient – the resulting encoding will be very large.
Solution: exploit the properties of BNNs.

Sequential counters for encoding cardinality constraints

Consider a *cardinality constraint*: $\sum_{i=1}^{m} l_i \geq C$, where $l_i \in \{0, 1\}$ is a Boolean variable and C is a constant. This can be compiled into CNF using *sequential counters* $SQ(l, C)$, $l = (l_1, ..., l_m)$.
Then $SQ(l, C)$ is equivalent to:

$$(l_1 \Leftrightarrow r_{(1,1)}) \wedge (\neg r_{(1,j)},\ j = \overline{2, C})$$
$$r_{(i,1)} \Leftrightarrow l_i \vee r_{(i-1,1)} \quad \wedge$$
$$r_{(i,j)} \Leftrightarrow l_i \wedge r_{(i-1,j-1)} \vee r_{(i-1,j)},\ j = \overline{2, C}$$

where $i = \overline{2, m},\ r_{(j,p)} = \mathbb{T} \iff \sum_{i=1}^{j} l_i \geq p$

SAT Encoding (cont'd)

- *Encoding of BLK_k*:

$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle \geq C_i \iff \sum_{j=1}^{n_k} a_{ij} x_{k_j} \geq C_i \iff$$

Variable replacement: $x_{k_j} \in \{0,1\} \rightsquigarrow x_{k_j}^{(b)} \in \{0,1\}$ with $x_{k_j} = 2x_{k_j}^{(b)} - 1$. We have:

$$\sum_{j=1}^{n_k} a_{ij}(2x_{k_j}^{(b)} - 1) \geq C_i \Rightarrow v_i = 1 \iff$$

Denote: $\mathbf{a}_i^+ = \{j | a_{ij} = 1\}$ and $\mathbf{a}_i^- = \{j | a_{ij} = -1\}$. We have:

$$\sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} x_{k_j}^{(b)} \geq \left\lceil \frac{C_i}{2} + \sum_{j=1}^{n_k} \frac{a_{ij}}{2} \right\rceil \iff \sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} (1 - \overline{x_{k_j}^{(b)}}) \geq C_i' \iff$$

$$\sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} \overline{x_{k_j}^{(b)}} \geq C_i' + |a_i^-|.$$ Hence $\sum_{j=1}^{n_k} l_{k_j} \geq D_i \Rightarrow v_i^{(b)} = 1, \; i = \overline{1, n_{k+1}}$

Further we have: $\bigwedge_{i=1}^{n_{k+1}} SQ(l, D_i) \wedge \bigwedge_{i=1}^{n_{k+1}} \left(r_{i(n_k, D_i)} \iff v_i^{(b)} \right)$

Similarly for

$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle < C_i \Rightarrow v_i = -1 \quad i = \overline{1, n_{k+1}}$$

SAT Encoding (cont'd)

- *Encoding of O*:

$$\langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i \geq \langle \mathbf{a}_j, \mathbf{x}_d \rangle + b_j \iff \langle \mathbf{a}_i - \mathbf{a}_j, \mathbf{x}_d \rangle \geq \lceil b_j - b_i \rceil \iff$$

$$\iff$$

$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle \geq C_i \iff \sum_{j=1}^{n_k} a_{ij} x_{k_j} \geq C_i \iff$$

Variable replacement: $x_{k_j} \in \{0,1\} \rightsquigarrow x_{k_j}^{(b)} \in \{0,1\}$ with $x_{k_j} = 2x_{k_j}^{(b)} - 1$. We have:

$$\sum_{j=1}^{n_k} a_{ij}(2x_{k_j}^{(b)} - 1) \geq C_i \Rightarrow v_i = 1 \iff$$

Denote: $\mathbf{a}_i^+ = \{j | a_{ij} = 1\}$ and $\mathbf{a}_i^- = \{j | a_{ij} = -1\}$. We have:

$$\sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} x_{k_j}^{(b)} \geq \left\lceil \frac{C_i}{2} + \sum_{j=1}^{n_k} \frac{a_{ij}}{2} \right\rceil \iff \sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} (1 - \overline{x_{k_j}^{(b)}}) \geq C_i' \iff$$

$$\sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} \overline{x_{k_j}^{(b)}} \geq C_i' + |a_i^-|.$$ Hence $$\sum_{j=1}^{n_k} l_{k_j} \geq D_i \Rightarrow v_i^{(b)} = 1, \; i = \overline{1, n_{k+1}}$$

Further we have: $$\bigwedge_{i=1}^{n_{k+1}} SQ(l, D_i) \wedge \bigwedge_{i=1}^{n_{k+1}} \left(r_{i(n_k, D_i)} \iff v_i^{(b)} \right)$$

SAT Encoding (cont'd)

Example

Recall the internal block with two inputs and one output: $A_k = [1, -1]$, $b_k = [-0.5]$, $\alpha_k = [0.12]$, $\mu_k = [-0.1]$, $\sigma_k = [2]$, $\delta_k = [0.1]$. We have:

$$x_{k_1} - x_{k_2} \geq -1 \Rightarrow v_1 = 1 \iff 2x_{k_1}^{(b)} - 1 - (2x_{k_2}^{(b)} - 1) \geq -1 \Rightarrow v_1^{(b)} = 1 \iff$$
$$x_{k_1}^{(b)} - x_{k_2}^{(b)} \geq \lceil 0.5 \rceil$$
$$x_{k_1} - x_{k_2} < -1 \Rightarrow v_1 = -1$$

Speeding-up the SAT Encoding

- Takes advantage of the modular structure of BNNs.

Speeding-up the SAT Encoding

- Takes advantage of the modular structure of BNNs.
- The approach works for all properties, we exemplify for adversarial robustness.

Speeding-up the SAT Encoding

- Takes advantage of the modular structure of BNNs.
- The approach works for all properties, we exemplify for adversarial robustness.
- The network can be encoded as a conjunction of two Boolean formulas: *Gen* (generator) encodes the first block of the network, and *Ver* (verifier) encodes the rest of the network:

$$BNN_{A_d}(\mathbf{x}+\tau, o, l_{\mathbf{x}}) = Gen(\mathbf{x}+\tau, \mathbf{y}) \wedge Ver(\mathbf{y}, \mathbf{z}, o, l_{\mathbf{x}})$$

where

$$Gen(\mathbf{x}+\tau, \mathbf{y}) = CNF(||\tau||_\infty \leq \epsilon) \wedge \bigwedge_{i=1}^{n} CNF(\mathbf{x}_i + \tau_i) \in [LB, UB]) \wedge$$
$$BINBLK_1(\mathbf{x}+\tau, \mathbf{y})$$

$$Ver(\mathbf{y}, \mathbf{z}, o, l_{\mathbf{x}}) = BINBLK_2(\mathbf{y}, \mathbf{z}) \wedge BINO(\mathbf{z}, o) \wedge CNF(o \neq l_{\mathbf{x}}).$$

Speeding-up the SAT Encoding

- Takes advantage of the modular structure of BNNs.
- The approach works for all properties, we exemplify for adversarial robustness.
- The network can be encoded as a conjunction of two Boolean formulas: *Gen* (generator) encodes the first block of the network, and *Ver* (verifier) encodes the rest of the network:

  $$BNN_{A_d}(\mathbf{x}+\tau, o, l_{\mathbf{x}}) = Gen(\mathbf{x}+\tau, \mathbf{y}) \wedge Ver(\mathbf{y}, \mathbf{z}, o, l_{\mathbf{x}})$$

  where

  $$Gen(\mathbf{x}+\tau, \mathbf{y}) = CNF(||\tau||_\infty \leq \epsilon) \wedge \bigwedge_{i=1}^{n} CNF(\mathbf{x}_i + \tau_i) \in [LB, UB]) \wedge \\ BINBLK_1(\mathbf{x}+\tau, \mathbf{y})$$

  $$Ver(\mathbf{y}, \mathbf{z}, o, l_{\mathbf{x}}) = BINBLK_2(\mathbf{y}, \mathbf{z}) \wedge BINO(\mathbf{z}, o) \wedge CNF(o \neq l_{\mathbf{x}}).$$

- *Gen* and *Ver* share only $\mathbf{y}$ $\rightsquigarrow$ use *Craig interpolants* to build efficient search procedure.

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $\mathit{vars}(I) = \mathit{vars}(A) \cap \mathit{vars}(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $\mathit{Gen}(\mathbf{x} + \tau, \mathbf{y})$.

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $Gen(\mathbf{x} + \tau, \mathbf{y})$.
2. Let $\tilde{\mathbf{y}}$ denote this assignment to $\mathbf{y}$.

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $Gen(\mathbf{x} + \tau, \mathbf{y})$.
2. Let $\tilde{\mathbf{y}}$ denote this assignment to $\mathbf{y}$.
3. Check if the assignment $\mathbf{y} = \tilde{\mathbf{y}}$ can be extended to a satisfying assignment for the Ver formula.

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $\mathit{vars}(I) = \mathit{vars}(A) \cap \mathit{vars}(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $\mathit{Gen}(\mathbf{x} + \tau, \mathbf{y})$.
2. Let $\tilde{\mathbf{y}}$ denote this assignment to $\mathbf{y}$.
3. Check if the assignment $\mathbf{y} = \tilde{\mathbf{y}}$ can be extended to a satisfying assignment for the Ver formula.
4. If yes (assign. makes Ver SAT) $\rightsquigarrow$ adversarial perturbation τ found

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $Gen(\mathbf{x} + \tau, \mathbf{y})$.
2. Let $\tilde{\mathbf{y}}$ denote this assignment to $\mathbf{y}$.
3. Check if the assignment $\mathbf{y} = \tilde{\mathbf{y}}$ can be extended to a satisfying assignment for the Ver formula.
4. If yes (assign. makes Ver SAT) $\rightsquigarrow$ adversarial perturbation τ found
5. If no $\rightsquigarrow$ generate an interpolant I of $Gen(\mathbf{x} + \tau, \mathbf{y}) \wedge Ver(\mathbf{y} = \tilde{\mathbf{y}}, \mathbf{z}, o, l_{\mathbf{x}})$ by extracting an UNSAT core of $Ver(\mathbf{y} = \tilde{\mathbf{y}}, z, o, l_{\mathbf{x}})$

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $Gen(\mathbf{x} + \tau, \mathbf{y})$.
2. Let $\tilde{\mathbf{y}}$ denote this assignment to $\mathbf{y}$.
3. Check if the assignment $\mathbf{y} = \tilde{\mathbf{y}}$ can be extended to a satisfying assignment for the Ver formula.
4. If yes (assign. makes Ver SAT) $\rightsquigarrow$ adversarial perturbation τ found
5. If no $\rightsquigarrow$ generate an interpolant I of $Gen(\mathbf{x} + \tau, \mathbf{y}) \wedge Ver(\mathbf{y} = \tilde{\mathbf{y}}, \mathbf{z}, o, I_{\mathbf{x}})$ by extracting an UNSAT core of $Ver(\mathbf{y} = \tilde{\mathbf{y}}, z, o, I_{\mathbf{x}})$
6. Use assumptions, which are assignments of $\tilde{\mathbf{y}}$, in the SAT solver to obtain a core. Since none of the satisfying assignments to I can be extended to a valid satisfying assignment of $BNN_{A_d}(I_{\mathbf{x}} + \tau, o, I_{\mathbf{x}})$, we block them all in Gen by redefining $Gen := Gen \wedge \neg I$.

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $Gen(\mathbf{x} + \tau, \mathbf{y})$.
2. Let $\tilde{\mathbf{y}}$ denote this assignment to $\mathbf{y}$.
3. Check if the assignment $\mathbf{y} = \tilde{\mathbf{y}}$ can be extended to a satisfying assignment for the Ver formula.
4. If yes (assign. makes Ver SAT) $\rightsquigarrow$ adversarial perturbation τ found
5. If no $\rightsquigarrow$ generate an interpolant I of $Gen(\mathbf{x} + \tau, \mathbf{y}) \wedge Ver(\mathbf{y} = \tilde{\mathbf{y}}, \mathbf{z}, o, l_{\mathbf{x}})$ by extracting an UNSAT core of $Ver(\mathbf{y} = \tilde{\mathbf{y}}, z, o, l_{\mathbf{x}})$
6. Use assumptions, which are assignments of $\tilde{\mathbf{y}}$, in the SAT solver to obtain a core. Since none of the satisfying assignments to I can be extended to a valid satisfying assignment of $BNN_{A_d}(l_{\mathbf{x}} + \tau, o, l_{\mathbf{x}})$, we block them all in Gen by redefining $Gen := Gen \wedge \neg I$.
7. Repeat from step 1. The procedure terminates since the solution space is reduced.

Speeding-up the SAT Encoding (cont'd)

Definition (Craig Interpolants)

Let A and B be Boolean formulas such that the formula $A \wedge B$ is UNSAT. Then there exists a formula I, called interpolant, such that $vars(I) = vars(A) \cap vars(B)$, $B \wedge I$ is UNSAT and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.

Idea of the approach:

1. first generate a satisfying assignment to variables τ and $\mathbf{y}$ for $Gen(\mathbf{x}+\tau, \mathbf{y})$.
2. Let $\tilde{\mathbf{y}}$ denote this assignment to $\mathbf{y}$.
3. Check if the assignment $\mathbf{y} = \tilde{\mathbf{y}}$ can be extended to a satisfying assignment for the Ver formula.
4. If yes (assign. makes Ver SAT) $\rightsquigarrow$ adversarial perturbation τ found
5. If no $\rightsquigarrow$ generate an interpolant I of $Gen(\mathbf{x}+\tau, \mathbf{y}) \wedge Ver(\mathbf{y} = \tilde{\mathbf{y}}, \mathbf{z}, o, l_{\mathbf{x}})$ by extracting an UNSAT core of $Ver(\mathbf{y} = \tilde{\mathbf{y}}, z, o, l_{\mathbf{x}})$
6. Use assumptions, which are assignments of $\tilde{\mathbf{y}}$, in the SAT solver to obtain a core. Since none of the satisfying assignments to I can be extended to a valid satisfying assignment of $BNN_{A_d}(l_{\mathbf{x}}+\tau, o, l_{\mathbf{x}})$, we block them all in Gen by redefining $Gen := Gen \wedge \neg I$.
7. Repeat from step 1. The procedure terminates since the solution space is reduced.
8. If the formula $Gen(\mathbf{x}+\tau, \mathbf{y})$ becomes UNSAT, then there is no valid perturbation τ , i.e., the network is ϵ-robust on image $\mathbf{x}$.

Contents

Encoding the Properties

1. Adversarial Constraint:
   $BNN_{A_d}(\mathbf{x} + \tau, o, l_{\mathbf{x}}) = CNF(||\tau||_\infty \leq \epsilon) \vee \bigwedge_{i=1}^{n} CNF((\mathbf{x}_i + \tau_i) \in [L, U]) \wedge$
   $BNN(\mathbf{x} + \tau, o) \wedge CNF(o \neq l_{\mathbf{x}})$
2. Verifying Universal Adversarial Robustness

$$\bigwedge_{i=1}^{|S|} BNN_{A_d}(\mathbf{x}_i + \tau, o_i, l_{\mathbf{x}_i}) \iff q_j \wedge CNF\left(\sum_{i=1}^{|S|} q_j \geq \rho|S|\right)$$

3. Verifying Network Equivalence: if

$$\bigwedge_{i=1}^{n} CNF(x_i \in [L, U]) \wedge BNN_1(\mathbf{x}, o_1) \wedge BNN_2(\mathbf{x}, o_2) \wedge o_1 \neq o_2$$

   is UNSAT then the networks are equivalent. If SAT then we obtain a witness image $\mathbf{x}$.

Experimental Results

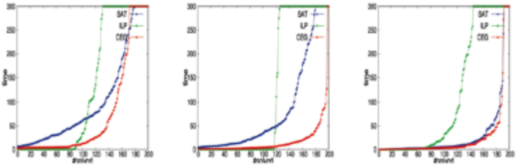
- Torch framework; Tital Pascal X GPU
- Datasets: MNIST, MNIST-rot (MINIST where the digits were rotated uniformly in $[0, 2\pi]$ radians), MNIST-back-image (MINIST with a patch from a black-and-white image was used as the background for the digit image)
- focus on adversarial robustness
- Architecture
  - 4 internal blocks with each block containing a linear layer (LIN) and a final output block.
  - LIN layer in the first block contains 200 neurons, the LIN layers in other blocks contain 100 neurons
  - BN and BIN layers in each block were used; additionally a hard *tanh* layer in each internal block was used only during training
  - For inputs processing, two layers (BN and BIN) were added to the BNN, as the first 2 layers in the network to perform binarization of the grayscale inputs $\rightsquigarrow$ (+) network architecture simplification and search space reduction; (-) lower accuracy of the original BNN by approx.1 %
- Accuracy of the resulting network on the MNIST, MNISTrot, and MNIST-back-image datasets were 95.7%, 71%, resp. 70%

Experimental Results (cont'd)

Checking adversarial robustness:

- from each dataset randomly picked 20 images correctly classified by the network for each of the 10 classes (coresponding to digits) $\rightsquigarrow$ 200 images
- for search space reduction, focus on important pixels as defined by saliency map: perturb the top 50% of highly salient pixels in an image; if a valid perturbation that leads to misclassification among this set of pixels can not be found then search again over all pixels of the image.
- experimented with 3 different maximum perturbation values $\epsilon \in \{1, 3, 5\}$
- timeout: 300 seconds for each instance
- compare three methods of searching for adversarial perturbations. (1) ILP method with SCIP solver, (2) pure SAT method for the sequential counters method using Glucose SAT solver, (3) the SAT menthod from (2) augumented with the counter-example-guided.
- complexity of the SAT formulae: 1.4 million variables and 5 million clauses: MNIST-rot – approx 7 million clauses; MNIST and MNIST-back – approx 5 and 3 million clauses on average, respectively.
- the largest instance contains: 3 million variables and 12 million clauses.

Experimental Results (cont'd)



(a) MNIST, $\epsilon = 1$ (b) MNIST-rot, $\epsilon = 1$ (c) MNIST-back-image, $\epsilon = 1$

	Solved instances (out of 200)									Certifiably ϵ-robust		
	MNIST			MNIST-rot			MNIST-back-image					
	SAT	ILP	CEG	SAT	ILP	CEG	SAT	ILP	CEG	SAT	ILP	CEG
	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#	#	#
$\epsilon = 1$	180 (77.3)	130 (31.5)	171 (34.1)	179 (57.4)	125 (10.9)	197 (13.5)	191 (18.3)	143 (40.8)	191 (12.8)	138	96	138
$\epsilon = 3$	187 (77.6)	148 (29.0)	181 (35.1)	193 (61.5)	155 (9.3)	198 (13.7)	107 (43.8)	67 (52.7)	119 (44.6)	20	5	21
$\epsilon = 5$	191 (79.5)	165 (29.1)	188 (36.3)	196(62.7)	170(11.3)	198(13.7)	104 (48.8)	70 (53.8)	116 (47.4)	3	-	4

- Advantages of the method: complete search procedure $\rightsquigarrow$ certify ϵ-robustness $\rightsquigarrow$ there exists no adversarial perturbation technique that can fool the network on these images.
- existing methods: incomplete
- with increasing ϵ, the number of images on which the network is ϵ-robust decreases as the adversary can leverage the larger value to construct

Contents

Summary of paper [1]

- [1] uses both binary and non-binary parameters

Summary of paper [1]

- [1] uses both binary and non-binary parameters
- In fact, ablation study pursued by Andreea Postovan on traffic signs dataset shows that we need a layer with real values otherwise we loose too much information (low accuracy); also linear layer is not sufficient, we need convolutional neural networks (CNNs).

Summary of paper [1]

- [1] uses both binary and non-binary parameters
- In fact, ablation study pursued by Andreea Postovan on traffic signs dataset shows that we need a layer with real values otherwise we loose too much information (low accuracy); also linear layer is not sufficient, we need convolutional neural networks (CNNs).
- For MNIST dataset pure binarized network could be sufficient because the features to be detected are not complex.

Summary of paper [1]

- [1] uses both binary and non-binary parameters
- In fact, ablation study pursued by Andreea Postovan on traffic signs dataset shows that we need a layer with real values otherwise we loose too much information (low accuracy); also linear layer is not sufficient, we need convolutional neural networks (CNNs).
- For MNIST dataset pure binarized network could be sufficient because the features to be detected are not complex.
- For Fashion dataset they used XNOR-net (binary and non-binary layers).

Summary of paper [1]

- [1] uses both binary and non-binary parameters
- In fact, ablation study pursued by Andreea Postovan on traffic signs dataset shows that we need a layer with real values otherwise we loose too much information (low accuracy); also linear layer is not sufficient, we need convolutional neural networks (CNNs).
- For MNIST dataset pure binarized network could be sufficient because the features to be detected are not complex.
- For Fashion dataset they used XNOR-net (binary and non-binary layers).
- The approach is based on Reluplex [5] (verification of DNNs) whose calculus was extended to handle `sign` function.

Summary of paper [1]

- [1] uses both binary and non-binary parameters
- In fact, ablation study pursued by Andreea Postovan on traffic signs dataset shows that we need a layer with real values otherwise we loose too much information (low accuracy); also linear layer is not sufficient, we need convolutional neural networks (CNNs).
- For MNIST dataset pure binarized network could be sufficient because the features to be detected are not complex.
- For Fashion dataset they used XNOR-net (binary and non-binary layers).
- The approach is based on Reluplex [5] (verification of DNNs) whose calculus was extended to handle `sign` function.
- Paper shows that the extension with `sign` function is sufficient to verify BNNs.
- Reluplex = ReLU + Simplex

Summary of paper [1]

- [1] uses both binary and non-binary parameters
- In fact, ablation study pursued by Andreea Postovan on traffic signs dataset shows that we need a layer with real values otherwise we loose too much information (low accuracy); also linear layer is not sufficient, we need convolutional neural networks (CNNs).
- For MNIST dataset pure binarized network could be sufficient because the features to be detected are not complex.
- For Fashion dataset they used XNOR-net (binary and non-binary layers).
- The approach is based on Reluplex [5] (verification of DNNs) whose calculus was extended to handle `sign` function.
- Paper shows that the extension with `sign` function is sufficient to verify BNNs.
- Reluplex = ReLU + Simplex
- ReLU(x) = max(0,x) which translated to constraints gives a lot of disjunctions $\rightsquigarrow$ infeasible to be solve using brute force $\rightsquigarrow$ various improvements.

Summary of paper [1]

- [1] uses both binary and non-binary parameters
- In fact, ablation study pursued by Andreea Postovan on traffic signs dataset shows that we need a layer with real values otherwise we loose too much information (low accuracy); also linear layer is not sufficient, we need convolutional neural networks (CNNs).
- For MNIST dataset pure binarized network could be sufficient because the features to be detected are not complex.
- For Fashion dataset they used XNOR-net (binary and non-binary layers).
- The approach is based on Reluplex [5] (verification of DNNs) whose calculus was extended to handle `sign` function.
- Paper shows that the extension with `sign` function is sufficient to verify BNNs.
- Reluplex = ReLU + Simplex
- ReLU(x) = max(0,x) which translated to constraints gives a lot of disjunctions $\rightsquigarrow$ infeasible to be solve using brute force $\rightsquigarrow$ various improvements.
- Paper has a public repository available and even the machine learning models were not available, the information from the repo+paper+email exchange with authors was sufficient in order to reproduce the ML models from the paper.

Contents

Summary of paper [6]

The paper allows verification of networks with both binary and non-binary weights and activations (i.e. output of the activation functions).

Summary of paper [6]

The paper allows verification of networks with both binary and non-binary weights and activations (i.e. output of the activation functions).

The paper presents a set of rules from transforming a network with different layers into logical formulae (linear, batch normalization, sign) – similar to [8, 9].

Summary of paper [6]

The paper allows verification of networks with both binary and non-binary weights and activations (i.e. output of the activation functions).

The paper presents a set of rules from transforming a network with different layers into logical formulae (linear, batch normalization, sign) – similar to [8, 9].

The paper compares verification tasks, like robustness, for BNNs, resp. DNNs, models trained for MINIS and ACAS controler datasets.

Summary of paper [6]

The paper allows verification of networks with both binary and non-binary weights and activations (i.e. output of the activation functions).

The paper presents a set of rules from transforming a network with different layers into logical formulae (linear, batch normalization, sign) – similar to [8, 9].

The paper compares verification tasks, like robustness, for BNNs, resp. DNNs, models trained for MINIS and ACAS controler datasets.

The paper has a similar approach as [8, 9] (translation of the network into formulae) but the underlying problem is similar to [1] (binary and non-binary parameters); the authors are co-authors of papers on Reluplex and Marabou.

Summary of paper [6]

- The paper allows verification of networks with both binary and non-binary weights and activations (i.e. output of the activation functions).
- The paper presents a set of rules from transforming a network with different layers into logical formulae (linear, batch normalization, sign) – similar to [8, 9].
- The paper compares verification tasks, like robustness, for BNNs, resp. DNNs, models trained for MINIS and ACAS controler datasets.
- The paper has a similar approach as [8, 9] (translation of the network into formulae) but the underlying problem is similar to [1] (binary and non-binary parameters); the authors are co-authors of papers on Reluplex and Marabou.
- My impression is that this paper just tested [8, 9] approach to compare with [1].

Contents

Summary of paper [4]

Efficient Exact Verification (EEV) tool:

Summary of paper [4]

Efficient Exact Verification (EEV) tool:

- a novel SAT solver that speeds up BNN verification by natively handling the reified cardinality constraints arising in BNN encodings;

Summary of paper [4]

Efficient Exact Verification (EEV) tool:

- a novel SAT solver that speeds up BNN verification by natively handling the reified cardinality constraints arising in BNN encodings;
- strategies to train solver-friendly robust BNNs by inducing balanced layer-wise sparsity* and low cardinality bounds, and adaptively cancelling the gradients.

* Maybe weights prunning can handle this issue already during training?

Summary of paper [4]

Efficient Exact Verification (EEV) tool:

- a novel SAT solver that speeds up BNN verification by natively handling the reified cardinality constraints arising in BNN encodings;
- strategies to train solver-friendly robust BNNs by inducing balanced layer-wise sparsity* and low cardinality bounds, and adaptively cancelling the gradients.

Contributions of the paper:

* Maybe weights prunning can handle this issue already during training?

Summary of paper [4]

Efficient Exact Verification (EEV) tool:

- a novel SAT solver that speeds up BNN verification by natively handling the reified cardinality constraints arising in BNN encodings;
- strategies to train solver-friendly robust BNNs by inducing balanced layer-wise sparsity* and low cardinality bounds, and adaptively cancelling the gradients.

Contributions of the paper:

- first exact verification results for l_∞-bounded adversarial robustness of nontrivial convolutional BNNs on the MNIST and CIFAR10 datasets.

* Maybe weights prunning can handle this issue already during training?

Contents

Summary of paper [7]

Uses formal techniques during BNN training to ensure robustness:

Summary of paper [7]

Uses formal techniques during BNN training to ensure robustness:

- quantization-aware interval bound propagation (QA-IBP) method for training robust quantized neural networks (QNNs)

Summary of paper [7]

Uses formal techniques during BNN training to ensure robustness:

- quantization-aware interval bound propagation (QA-IBP) method for training robust quantized neural networks (QNNs)
- complete verification procedure for verifying the adversarial robustness of QNNs

Summary of paper [7]

Uses formal techniques during BNN training to ensure robustness:

- quantization-aware interval bound propagation (QA-IBP) method for training robust quantized neural networks (QNNs)
- complete verification procedure for verifying the adversarial robustness of QNNs
- the verification procedure has the advantage that it runs entirely on GPU or other accelerator devices.

Conclusions

- Papers [1, 8, 9] provide complete algorithms, i.e. if a property is true then it is identified as such.
- Papers [8, 9] formalize the BNN from scratch (no public repository).
- Techniques from paper [1] are implemented in Reluplex which is an extension of Marabou (`https://github.com/NeuralNetworkVerification/Marabou`) - code available and up to date.

References I

[1] Guy Amir, Haoze Wu, Clark Barrett, and Guy Katz. An smt-based approach for verifying binarized neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 203–222. Springer, 2021.

[2] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to $+1$ or -1. *CoRR*, abs/1602.02830, 2016.

[3] Thomas A. Henzinger, Mathias Lechner, and Dorde Zikelic. Scalable verification of quantized neural networks (technical report). *CoRR*, abs/2012.08185, 2020.

[4] Kai Jia and Martin Rinard. Efficient exact verification of binarized neural networks. *Advances in neural information processing systems*, 33:1782–1795, 2020.

[5] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: a calculus for reasoning about deep neural networks. *Formal Methods in System Design*, pages 1–30, 2021.

[6] Christopher Lazarus and Mykel J. Kochenderfer. A mixed integer programming approach for verifying properties of binarized neural networks. 2022.

References II

[7] Mathias Lechner, Đorđe Žikelić, Krishnendu Chatterjee, Thomas A. Henzinger, and Daniela Rus. Quantization-aware interval bound propagation for training certifiably robust quantized neural networks, 2022.

[8] Nina Narodytska. Formal analysis of deep binarized neural networks. In *IJCAI*, pages 5692–5696, 2018.

[9] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.