MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

# Verification of binarised neural networks using ASP

Bachelor’s Thesis

JINDŘICH MATUŠKA

Advisor: RNDr. Samuel Pastva, PhD.

Department of Computer Systems and Communications

Brno, Spring 2024

MUNI
FI

## Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Jindřich Matuška

**Advisor:** RNDr. Samuel Pastva, PhD.

# Acknowledgements

foo bar

## Abstract

foo bar

## Keywords

bnn, verification, binarized neural networks answer set programming, asp

# Contents

# List of Tables

# List of Figures

# 1 Introduction

Deep neural networks (DNN) are state of the art technology. They are used in many real-world applications e.g. medicine, self-driving cars, automatic systems, many of which are critical. State of the art deep neural network have up to hundreads of bilions parameters [https://arxiv.org/pdf/2109.01652.pdf]. That is too many for people to comprehend. We need tools for automation of verification of these.

While DNNs generally base their computations in floating point numbers, the computation of output is computationaly hard. For this, quantization, a new branch of DNN development, where instead of 32 or 64-bits long floating point numbers, low-bit-width (eg. 4) fixed point numbers are used instead. This mitigates the high cost of computation and need for high-end devices. Extreme example of quantization are Binary neural networks (BNN). These use binary parameters for their computations, resulting in much cheaper computational price per parameter on specialised devices.

Verification of neural networks is split into two categories. First, qualitative verification, searches through input space looking for any adversarial input, e.g. input which results in bad output. Second, quantitative verification, searches through input space counting number of adversarial inputs. State of the art verificators are usually based on satisfability theories (SAT) or satisfability modulo theories (SMT) paradigm. Even on BNNs these verificators scale only up to and hundreads of parameters for quantitative verification [BNNQuanalyst]. In this thesis I have implemented quantitative BNN validator using ASP paradigm in framework Clingo from Potassco. I have also evaluated speed of this validator agnist state of the art implementation [BNNQuanalysist] on mnist and fashionmnist dataset.

# 2 Used tools

## 2.1 Answer set programming

Answer set programming (ASP) is a form of declarative programming oriented towards difficult, primarily NP-hard, search problems. [Lifschitz, Vladimir. Answer set programming. Vol. 3. Heidelberg: Springer, 2019.] ASP is particularly suited for solving difficult combinatorial search problems. [A Glimpse of Answer Set Programming Christian Anger and Kathrin Konczak and Thomas Linke and Torsten Schaub] ASP is somewhat closely related to propositional satisfability checking (SAT) in sense that the problem is represented as logic program. Difference is in computational mechanism of finding solution.

### 2.1.1 Syntax and semantics of ASP

Logic program $\Pi$ in ASP is a set of rules. Each rule $r_i \in \Pi$ has form of

$$\text{head}(r_i) \leftarrow \text{body}(r_i) \tag{2.1}$$

$\text{head}(r_i)$ is then a single atom $p_0$, while $\text{body}(r_i)$ is a set of zero or more atoms $\{p_1, \ldots, p_m, \text{not}\, p_{m+1}, \ldots \text{not}\, p_n\}$.

$$p_0 \leftarrow p_1, \ldots, p_m, \text{not}\, p_{m+1}, \ldots, \text{not}\, p_n \tag{2.2}$$

Further the $\text{body}(r_i)$ can be split between $\text{body}^+(r_i) = \{p_1, \ldots, p_m\}$ and $\text{body}^-(r_i) = \{p_{m+1} \ldots p_n\}$. If $\forall r_i \in \Pi.\, \text{body}^-(r_i) = \emptyset$, then $\Pi$ is called *basic*.

Semantically rule (2.2) means *If all atoms from* $\text{body}^+(r_i)$ *are included in answer set and no atom from* $\text{body}^-(r_i)$ *is in answer set, then* $\text{head}(r_i)$ *has to be in the set.*

Set of atoms $X$ is closed under a basic program $\Pi$ if for any $r_i \in \Pi.\, \text{head}(r_i) \in X \iff \text{body}(r_i) \subseteq X$. For general case the concept of *reduct of a program* $\Pi$ *relative to a set* $X$ *of atoms* is needed.

$$\Pi^X = \{\text{head}(r_i) \leftarrow \text{body}^+(r_i) \mid r_i \in \Pi, \text{body}^-(r_i) \cap X = \emptyset\} \tag{2.3}$$

This program is always basic as it only contains positive atoms.

Let's denote $\mathrm{Cn}(\Pi)$ the minimal set of atoms closed under a basic program $\Pi$, that is

$$\mathrm{Cn}(\Pi) \supseteq \{\mathrm{head}(r_i) \mid r_i \in \Pi, \mathrm{body}(r_i) \subseteq \mathrm{Cn}(\Pi)\} \tag{2.4}$$

such set is said to constitute program $\Pi$. Set $X$ of atoms is said to be answer set of $\Pi$ iff

$$\mathrm{Cn}(\Pi^X) = X \tag{2.5}$$

In other words, the answer set is a model of $\Pi$ such that its every atom is grounded in $\Pi$.

Let's illustrate this property on an example.

$$\Pi = \{p \leftarrow p, q \leftarrow \mathrm{not}\, p\}$$

There are 4 subsets of set of all atoms. First the reduct relative to the subset is made, on it the calculation of constituting set is made. If $X = \mathrm{Cn}(\Pi^X)$, then $X$ is one of answer sets.

| $X$ | $\Pi^X$ | $\mathrm{Cn}(\Pi^X)$ |
|---|---|---|
| {} | $p \leftarrow p$<br>$q \leftarrow$ | $\{q\}$ |
| {p} | $p \leftarrow p$ | {} |
| {q} | $p \leftarrow p$<br>$q \leftarrow$ | $\{q\}$ |
| {p, q} | $p \leftarrow p$ | {} |

There is only a single answer set of $\Pi$, that is $\{q\}$. There is an interesting type of rule in the table, $p \leftarrow$. This type of rule is commonly reffered as *fact* and ensures that atom $p$ is always included in the answer set.

Another toy example is $\Pi = \{p \leftarrow \mathrm{not}\, q, q \leftarrow \mathrm{not}\, p\}$. Again, there are 4 subsets of set of all atoms.

| $X$ | $\Pi^X$ | $\mathrm{Cn}(\Pi^X)$ |
|---|---|---|
| {} | $p \leftarrow$<br>$q \leftarrow$ | $\{p, q\}$ |
| {p} | $p \leftarrow$ | $\{p\}$ |
| {q} | $q \leftarrow$ | $\{q\}$ |
| {p, q} | | {} |

This time there are two answer sets of $\Pi$, $\{p\}$ and $\{q\}$. As we can see, the double not forms a XOR - exactly one of atoms $p, q$ has to be in the answer set.

### 2.1.2 Language extensions

# 3 Můj přínos

## 3.1 Binary neural network

I have been working with deterministic binarized neural networks (BNN) as defined in [Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In Proceedings of the Annual Conference on Neural Information Processing Systems. 4107–4115.] and [BNNQuanalys]. For convinience I have rewritten the output as a natural number instead of one-hot vector. This article defines deterministic BNN as a convolution of layers. The input is encoded as vector $\mathbb{B}_{\pm 1}^{n_1}$. The BNN is then encoded as tuple of blocks $(t_1, t_2, \dots, t_d, t_{d+1})$.

$$\mathcal{N} : \mathbb{B}_{\pm 1}^{n_1} \to \mathbb{N}_0$$

$$\mathcal{N} = t_{d+1} \circ t_d \circ \cdots \circ t_1, \tag{3.1}$$

where for each $i \in \{1, 2, \dots d\}$ $t_i$ is inner block consisting of LIN layer $t_i^{lin}$, BN layer $t_i^{bn}$ and BIN layer $t_i^{bin}$.

$$t_i : \mathbb{B}_{\pm 1}^{n_i} \to \mathbb{B}_{\pm 1}^{n_{i+1}}$$

$$t_i = t_i^{bin} \circ t_i^{bn} \circ t_i^{lin} \tag{3.2}$$

Output block $t_{d+1}$ then consists of LIN layer $t_{d+1}^{lin}$ and ARGMAX layer $t_{d+1}^{am}$.

$$t_{d+1} : \mathbb{B}_{\pm 1}^{n_{d+1}} \to \mathbb{N}_0$$

$$t_{d+1} = t_{d+1}^{am} \circ t_{d+1}^{lin} \tag{3.3}$$

Each layer is a function with parameters defined in table [table num].

I have transformed the parameters to lower their count and make them integer only.

$$t_i(\vec{x}) = (t_i^{bin} \circ t_i^{bn} \circ t_i^{lin})(\vec{x}) = y$$

$$y_j = \text{sign}_{\pm 1} \left( \alpha_j \cdot \frac{\langle \vec{x}, \mathbf{W}_{:,j} \rangle + b_j - \vec{\mu}_j}{\vec{\sigma}_j} - \vec{\gamma}_j \right)$$

| Layer | Function | Parameters | Definition |
|---|---|---|---|
| LIN | $t_i^{lin} : \mathbb{B}_{\pm1}^{n_i} \rightarrow \mathbb{R}^{n_{i+1}}$ | Weight matrix: $\mathbf{W} \in \mathbb{B}_{\pm1}^{n_i \times n_{i+1}}$<br>Bias (row) vector: $\vec{b} \in \mathbb{R}^{n_{i+1}}$ | $t_i^{bn}(\vec{x}) = \vec{y}$, where $\forall j \in [n_{i+1}]$,<br>$\vec{y}_j = \langle \vec{x}, \mathbf{W}_{:,j} \rangle + \vec{b}_j$ |
| BN | $t_i^{bn} : \mathbb{R}^{n_{i+1}} \rightarrow \mathbb{R}^{n_{i+1}}$ | Weight vector: $\vec{\alpha} \in \mathbb{R}^{n_{i+1}}$<br>Bias vector: $\vec{\gamma} \in \mathbb{R}^{n_{i+1}}$<br>Mean vector: $\vec{\mu} \in \mathbb{R}^{n_{i+1}}$<br>Std. dev. vector: $\vec{\sigma} \in \mathbb{R}^{n_{i+1}}$ | $t_i^{bn}(\vec{x}) = \vec{y}$, where $\forall j \in [n_{i+1}]$,<br>$\vec{y}_j = \vec{\alpha}_j \cdot \frac{\vec{x}_j - \vec{\mu}_j}{\vec{\sigma}_j} + \vec{\gamma}_j$ |
| BIN | $t_i^{bin} : \mathbb{R}^{n_{i+1}} \rightarrow \mathbb{B}_{\pm1}^{n_{i+1}}$ | - | $t_i^{bin}(\vec{x}) = \vec{y}$, where $\forall j \in [n_{i+1}]$,<br>$\vec{y}_j = \begin{cases} +1, & \text{if } \vec{x}_j \geq 0; \\ -1, & \text{otherwise} \end{cases}$ |
| ARGMAX | $t_{d+1}^{am} : \mathbb{R}^{n_{d+1}} \rightarrow \mathbb{N}_0$ | - | $t_{d+1}^{am}(\vec{x}) = \arg\max(\vec{x})$ |

The argument of $\text{sign}_{\pm1}$ can be further analysed.

$$\vec{\alpha}_j \cdot \frac{\langle \vec{x}, \mathbf{W}_{:,j} \rangle + \vec{b}_j - \vec{\mu}_j}{\vec{\sigma}_j} - \vec{\gamma}_j \geq 0$$

There are three possible cases of value $\frac{\vec{\alpha}_j}{\vec{\sigma}_j}$:

$$\begin{gathered} \frac{\vec{\alpha}_j}{\vec{\sigma}_j} \geq 0 : \\ \langle \vec{x}, \mathbf{W}_{:,j} \rangle + \vec{b}_j - \vec{\mu}_j - \frac{\vec{\sigma}_j}{\vec{\alpha}_j} \cdot \vec{\gamma}_j \geq 0 \\ \mathbf{W}'_{:,j} = \mathbf{W}_{:,j}, \vec{b}'_j = \vec{b}_j - \vec{\mu}_j - \frac{\vec{\sigma}_j}{\vec{\alpha}_j} \cdot \vec{\gamma}_j \\ \langle \vec{x}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j \geq 0 \end{gathered}$$

$$\begin{gathered} \frac{\vec{\alpha}_j}{\vec{\sigma}_j} \leq 0 : \\ \langle \vec{x}, \mathbf{W}_{:,j} \rangle + \vec{b}_j - \vec{\mu}_j - \frac{\vec{\sigma}_j}{\vec{\alpha}_j} \cdot \vec{\gamma}_j \leq 0 \\ \langle \vec{x}, -\mathbf{W}_{:,j} \rangle - \vec{b}_j + \vec{\mu}_j + \frac{\vec{\sigma}_j}{\vec{\alpha}_j} \cdot \vec{\gamma}_j \geq 0 \\ \mathbf{W}'_{:,j} = -\mathbf{W}_{:,j}, \vec{b}'_j = -\vec{b}_j + \vec{\mu}_j + \frac{\vec{\sigma}_j}{\vec{\alpha}_j} \cdot \vec{\gamma}_j \\ \langle \vec{x}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j \geq 0 \end{gathered}$$

$$\begin{gathered}\vec{\alpha}_j = 0: \\ -\vec{\gamma}_j \geq 0 \\ \mathbf{W}'_{:,j} = \vec{0},\ \vec{b}'_j = -\vec{\gamma}_j \\ \langle \vec{x}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j \geq 0\end{gathered}$$

This shows that each inner layer can be computed using weight matrix and one bias parameter. However if there were some parameter $\alpha_j = 0$, $\mathbf{W}'$ would not have values only from $\pm 1$ but from $\{-1, 0, 1\}$. This is not an issue as Clingo works by default above integers. If 0-values were issue, they could be removed either by offsetting bias parameter or by modifying the BNN structure to remove the constant node. (This has not been implemented in my work.)

$$\vec{\alpha}_j = 0: \ \vec{b}'_j = \begin{cases} \langle \vec{1}, \vec{1} \rangle + 1 & \text{if } -\vec{\gamma}_j \geq 0; \\ -\langle \vec{1}, \vec{1} \rangle - 1 & \text{otherwise} \end{cases}$$

In my implementation I am also encoding inputs as binary values $\{0, 1\}$.

$$\begin{gathered}\vec{x} = 2\vec{x}^{(b)} - \vec{1} \\ \langle \vec{x}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j \geq 0 \\ \langle 2\vec{x}^{(b)} - \vec{1}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j \geq 0 \\ 2\langle \vec{x}^{(b)}, \mathbf{W}'_{:,j} \rangle - \langle \vec{1}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j \geq 0 \\ \langle \vec{x}^{(b)}, \mathbf{W}'_{:,j} \rangle + \tfrac{-\langle \vec{1}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j}{2} \geq 0\end{gathered}$$

Still each block could be computed using one weight matrix and one bias vector. Additionally, as $\langle \vec{x}^{(b)}, '_{:,j} \rangle$ is an integer value, equation is equivalent to

$$\langle \vec{x}^{(b)}, \mathbf{W}'_{:,j} \rangle + \left\lfloor \frac{-\langle \vec{1}, \mathbf{W}'_{:,j} \rangle + \vec{b}'_j}{2} \right\rfloor \geq 0$$

As for the output block, it has already just one weight matrix and one bias vector. The implementation of ARGMAX layer is mainly Clingo-based. The only needed transformation is to split bias to whole and decimal part.

$$\vec{b} = \vec{b}' + \vec{p}, \vec{b}'_j = \lfloor \vec{b}_j \rfloor$$

$\vec{p}$ is then used as main order of outputs.

Implementation of these transformations in Python is added as [appendix]

## 3.2 Encoding of BNN in Clingo

Implementation of BNN computation is avalible as [appendix]

## 3.3 Encoding of input regions

An input region of BNN is some subset of inputs on which the analysis is performed. I have implemented both of input region types from [BNNQuanalys], that is input region based on Hamming distance ($R_H$) and input region with fixed indices ($R_I$).

The quantitative analysis of said BNN is a problem to say how many inputs

# 4 Evaluation

## 5 Discussion