Principled Rewriting of ONNX Operators for Reluctant Solvers

Alban Grastien, Guilhem Ardouin, and Julien Girard-Satabin

Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
firstname.lastname@cea.fr
julien.girard2@cea.fr

Abstract. Neural networks (NN) are often represented using the Open Neural Network Exchange (ONNX) format that provides a rich set of operators. This richness implies that many tools support only a subset of ONNX operators and, consequently, a subset of NNs. However, some of these operators can be expressed equivalently as combinations of simpler, already supported, operators. In this work, we present an extension of CAISAR—a platform for verification of NN that uses external solvers—that automatically transforms these redundant operators into simpler ones before calling existing solvers, thus extending the range of these solvers. This work focuses on the ArgMax operator, proposing a transformation, proving its correctness, and presenting experimental results.

Keywords: Formal Specification and Verification, Machine Learning

1 Introduction

As machine learning, and artificial intelligence in general, has become more and more prominent, there has been an increasing need for verification of such systems [6]. This has led to the introduction of standard representations, both of a dedicated exchange format for neural networks, ONNX [7], and of the problems to solve with the VNNLIB format [8].

While many solvers are being developped, the efforts are not always matched with appropriate resources. This is in particular true for contributions from the academic world. The ONNX format is comprehensive and includes no fewer than 196 operators. Some of these operators are redundant, in that they can be defined as a combination of simpler operators. While this is beneficial for expressivity—they make it easier to understand what their purpose is in

Solver	Nb op. (/196)
nnenum [5]	17
α-β-CROWN [12]	22
Marabou [9]	16
Maraboupy [13]	24
PyRAT [10]	54

Table 1: Number of operators supported by the most common solvers.

the architecture—they represent a burden for the developers of solvers who are more interested in testing new solving techniques than supporting an ever-growing list of operators. This is illustrated on Tab. 1 where we see that many solvers support a very narrow list of operators. Furthermore, a prover may support an operator that is unsupported by others. For instance, prover nnenum supports vector concatenation which Marabou does not, while the latter supports vector transposition. We note that those operations are rather common in day-to-day machine learning practice. It is therefore unlikely that provers will be able to handle a standard neural network out-of-the-box, forcing the expert to do manual alterations to the architecture, which are error-prone and need to be done on a per-prover basis.

In this context, CAISAR [1] has been introduced as a verification platform that offers various tools to extend the capabilities of existing solvers to a larger array of problems. Of interest here, we introduce in this paper a new feature of CAISAR that allows it to reformulate existing neural networks into equivalent ones that use a "simpler" (or different) set of operators. Thanks to this reformulation, any solver that supports the restricted set of operators can be used to solve problems that include the more complicated operator, even if this operator is not supported natively by the solver. Similar reformulations, called *simplifications*, have been proposed in DNNV [11] that, as far as we know, is no longer maintained. Tensor compilers also often rewrite neural networks before sending them to the architecture where computation will take place to improve performance; there too, we see an effort to prove that these *rewrites* are valid [3], but notice that their objective is to improve computation time and not to increase support.

In particular, we focus on the ArgMax operator, which computes the index of the elements in certain rows of a tensor that have maximal value. This choice is motivated by ArgMax ubiquity in standard image classification settings, and hyper-properties specifications [2]. We propose a systematic transformation for any neural network that includes this operator, show the correctness of this transformation, and present an implementation of the transformation. We demonstrate its applicability on a test case where we simplify the expression of a neural network property by using an ArgMax operator inside the model to be verified and then transform it to verify the property using existing solvers.

Section 2 introduces the problem and our contributions: the translation of a stripped-down version of ArgMax into a different set of operators (support for the more advanced features is presented in Appendix A) and its integration in CAISAR. Section 3 provides a proof of the correctness of the translation. This extension is used experimentally in Section 4 before concluding in Section 5.

2 Preliminaries & Contributions

In this section, we provide the results in a slightly informal manner. The formal results are provided in Section 3. We first introduce the notions of tensors, neu-

ral networks, and ONNX. We then discuss how we propose to reformulate the ArgMax operator, and finally discuss its integration in CAISAR.

2.1 Tensors, Neural Networks, and ONNX

We use the notation $[N]$ to represent the set $\{0, \ldots, N-1\}$.

A *tensor* $\mathtt{t} \in \mathtt{T}$ is a multi-dimensional array of *primitive elements* such as integers or numbers in floating-point format. Tensor $\mathtt{t}$ has *shape* $sh(\mathtt{t}) = (\dim_1, \ldots, \dim_r)$ where r is the *rank* of $\mathtt{t}$ and each $\dim_\ell \in \mathbf{N}$ is the *size* of the ℓ-th *dimension*. The elements of tensor $\mathtt{t}$ are accessed by $\mathtt{t}[\mathbf{i}]$ where $\mathbf{i} = [i_1, \ldots, i_r] \in dom(sh(\mathtt{t})) \stackrel{def}{=} [\dim_1] \times \cdots \times [\dim_r]$ is a vector of positions for each dimension.

A tensor $\mathtt{t}$ of shape $sh(\mathtt{t}) = (\dim_1, \ldots, \dim_r)$ contains

$$|dom(sh(\mathtt{t}))| = \dim_1 \times \cdots \times \dim_r$$

elements. Given a vector $\mathbf{i} = [i_1, \ldots, i_r]$, we write $\mathbf{i}_{\ell \mapsto j}$ to refer to the same vector where the ℓ-th position is replaced with j, i.e., $[i_1, \ldots, i_{\ell-1}, j, i_{\ell+1}, \ldots, i_r]$.

An n-ary *ONNX operator* $\textsc{Op} : \mathtt{T}^n \mapsto \mathtt{T}$ of arity n is an operator that takes n tensors as input and returns a tensor. Throughout the paper, we distinguish the operator and its semantics: the operator is denoted with small caps (Op) and its mathematical semantics with italics ((op)). In this paper, we only consider the following operators:

1. $\textsc{Const}_{\mathtt{t}}$ (which returns $\mathtt{t}$),
2. Add (element-wise sum of two tensors),
3. Sign (element-wise sign of a tensor, that returns 1, 0, -1 if the element is strictly positive, zero, or strictly negative, respectively),
4. ReLU (element-wise application of the rectified linear unit activation function that, for any x, returns $\max(0, x)$),
5. $\textsc{Transpose}_p$ (change of indexing of elements in the tensor following the permutation p),
6. MatMul (standard matrix multiplication),
7. $\textsc{ArgMax}_\ell$ (function that returns the element with highest value across dimension ℓ),
8. SoftMax (smooth approximation of a one-hot vector).

The semantics of each operator is formally defined in Section 3, but we clarify $\textsc{ArgMax}_\ell$ as it is necessary to understand the translation. For a given vector $\mathbf{i} \in dom(sh(\mathtt{t}))$, for two indices $j, k \in \dim_\ell$, the element at position $\mathbf{i}_{\ell \mapsto j}$ (or, when clear from the context, simply *at position* j) is considered *higher* than the one at position k if the following property holds:

$$\begin{cases} \mathtt{t}[\mathbf{i}_{\ell \mapsto j}] > \mathtt{t}[\mathbf{i}_{\ell \mapsto k}] & \text{if } k < j, \\ \mathtt{t}[\mathbf{i}_{\ell \mapsto j}] \geq \mathtt{t}[\mathbf{i}_{\ell \mapsto k}] & \text{if } k > j, \end{cases} \tag{1}$$

i.e., the value $\mathtt{t}[\mathbf{i}_{\ell \mapsto j}]$ at position j is greater than that at position k, where tie breaking favours the smaller index. The $argmax_\ell$ semantics specifies that the size

of the output tensor at dimension ℓ would be one, and that the value at position $\mathbf{i}_{\ell \mapsto 0}$ should be the index $\hat{\jmath}$ of highest value (amongst all the elements at position $\mathbf{i}_{\ell \mapsto k}$).

A *neural network* (NN) described in ONNX format is a dataflow (essentially a graph) of tensors where nodes are labelled with ONNX operators, as in Fig. 1 (here, limited to a non-branching graph). The input tensor is passed to the first node which performs its computation and passes it to the next node(s) until the end of the network is reached.

2.2 CAISAR

CAISAR is a platform that aims at helping practitioners verify NNs. For instance, it supports multiple solvers via a single interface, and makes it easy to switch between these solvers to build on the strengths of the different technologies. It also allows the practitioner to define high-level goals through the use of the rich specification language WhyML, as we illustrate in Section 4. Most solvers require a NN as input, which is provided by CAISAR in ONNX format.

In this paper, we introduce a new functionality to CAISAR, which is the ability to replace certain complex ONNX operators that are not supported by some solvers. In particular, we show in the next subsection that ArgMax can be rewritten as a semantically equivalent network using simpler operators, so that any property that is satisfied by a NN that features this operator should be satisfied by a NN where this operator has been transformed (assuming all operators do satisfy their specification). This means that a solver that does not support ArgMax can still be used to verify properties on NN that include this operator as long as it supports the operators of the translation.

CAISAR supports all the solvers mentionned in Table 1. Each solver can be run in different modes, based on the command arguments. For each of these *variants*, a *driver* needs to be provided that indicates what preprocessing needs to be performed. We specify the list of transformations in this driver file; before calling the solver, CAISAR applies these transformations to the ONNX.

In a future implementation, we hope to automatically determine which transformations are necessary. Notice however that there might be different ways to transform a complex operator with varying impact on performance; choosing the preferred transformations might be a delicate exercise.

2.3 Transforming the ArgMax

We use Figure 1 as a reference. First, the ℓ-th dimension is swapped with the r-th one via a Transpose operation so that the MatMul operation is applied on dimension r; the Transpose operation is reversed at the end of the translation. The permutation is

$$p_{\ell,r} \stackrel{def}{=} [1, \ldots, \ell - 1, r, \ell + 1, \ldots, r - 1, \ell].$$

(If $\ell = r$, no transpose is performed.) Notice that Transpose can be a non-trivial operation when applied to a tensor; here however, the solver reasons about

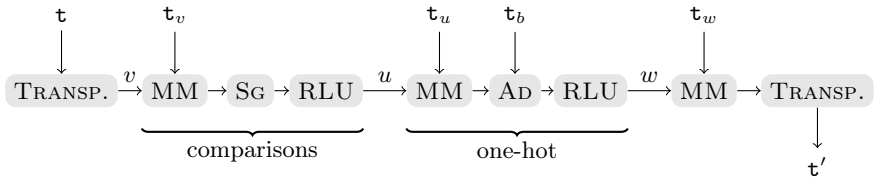

Fig. 1: Sub-network replacing the ARGMAX operator. The tensors and the semantics of v, u, and w are described in the text.

the effects of this operation but does not apply it in practice. At this point, the tensor is simply a collection of $|dom(sh(\mathtt{t}))|/\dim_\ell$ vectors of dimension $\dim_\ell$; we simplify the discussion by considering one such vector that we denote v (i.e., $v[j] \stackrel{def}{=} \mathtt{t}[\mathbf{i}_{\ell \mapsto j}]$ for the $\mathbf{i}$ of interest).

In a second step, the sub-network computes a vector u of size $|C^2_{\dim_\ell}| = \frac{\dim_\ell(\dim_\ell - 1)}{2}$ (where $C^2_{\dim_\ell}$ is *"choose* 2 *from* $[\dim_\ell]$*"*). Each element of vector u is associated with a pair $p \in C^2_{\dim_\ell}$ and we write $\xi(p)$ the index of this element. In other words:

1. $\forall p \in C^2_{\dim_\ell}.\ \xi(p) \in [\ |C^2_{\dim_\ell}|\ ]$ and
2. $\forall \{p, p'\} \subseteq C^2_{\dim_\ell}.\ p \neq p' \Rightarrow \xi(p) \neq \xi(p')$.

For $j < k$, the sub-network ensures $u[\xi(\{j,k\})] = 1$ if $v[k]$ is higher than $v[j]$, and $u[\xi(\{j,k\})] = 0$ otherwise. This is done by computing $v[k] - v[j]$ through a MATMUL operation followed with a SIGN and a RELU operations.

In a third step, it computes a one-hot vector w where $w[\hat{\jmath}]$ is 1 if $\hat{\jmath}$ is the index with highest value in v, and all other $w[k]$ are 0. This is done via a MATMUL operation that adds up $u[\xi(\{j,k\})]$ for all $j > k$ and $(1 - u[\xi(\{j,k\})])$ for all $k > j$ (this should equate $\dim_\ell - 1$ for $\hat{\jmath}$ and less for other indices) and subtracts $\dim_\ell - 2$ to the result, and followed with a RELU.

In the fourth step, the one-hot vector is multiplied (via a MATMUL) with vector $[0, 1, \ldots, \dim_\ell - 1]$ so that the last dimension has size one and its value is the position of the 1 in w (starting from index 0).

Description of the intermediate tensors For completeness, we describe the tensors used in the translation.

Tensor $\mathtt{t}_v$ is a tensor of shape $(\dim_\ell, C^2_{\dim_\ell})$ defined by

$$\forall \{j,k\} \in C^2_{\dim_\ell}.\ \forall i \in [\dim_\ell].\ \mathtt{t}_v[i, \xi(\{j,k\})] = \begin{cases} 1 & \text{if } i = \max(j,k) \\ -1 & \text{if } i = \min(j,k) \\ 0 & \text{otherwise.} \end{cases}$$

Tensor $\mathtt{t}_u$ is the transpose of $\mathtt{t}_v$, i.e., a tensor of shape $(C^2_{\dim_\ell}, \dim_\ell)$ such that

$$\forall\{j,k\} \in C^2_{\dim_\ell}.\ \forall i \in [\dim_\ell].\ \mathtt{t}_u[\xi(\{j,k\}), i] = \begin{cases} 1 & \text{if } i = \max(j,k) \\ -1 & \text{if } i = \min(j,k) \\ 0 & \text{otherwise.} \end{cases}$$

Tensor $\mathtt{t}_b$ is a tensor of shape $(\dim_1, \ldots, \dim_{r-1}, \dim_\ell)$ such that

$$\forall \mathbf{i} \in \mathit{dom}(\mathit{sh}(\mathtt{t}_b)).\ \forall j \in [\dim_\ell].\quad \mathtt{t}_b(\mathbf{i}_{r \mapsto j}) = j - 1.$$

Tensor $\mathtt{t}_w$ is a tensor of shape $(\dim_\ell, 1)$ such that

$$\forall j \in [\dim_\ell].\quad \mathtt{t}_w[j, 0] = j.$$

3 Proof of Correctness

In this proof, we demonstrate that the sub-NN described in Fig. 1 satisfies the *argmax* specification, i.e., that an ArgMax operator can be replaced with the sequence of operators from the figure.

3.1 Specification

We provide the specification for each operator. Each ONNX operator Op is assumed to satisfy the corresponding specification *op*. While *softmax* is not used in the proof, we included it nonetheless as it is relevant for Section 4.

Notice that, for simplicity, we ignore or enforce certain aspects of these operators such as the broadcasting of binary operations. We also assume a simplified version of *argmax* which we extend in the Appendix. In what follows, a *shape-preserving* operator refers to an operator whose output tensor has the same shape as its input tensor(s).

- $\mathit{const}_\mathtt{t}$ is a 0-ary operator that outputs tensor $\mathtt{t}$: $\mathit{const}_\mathtt{t}() = \mathtt{t}$.
- *add* is a shape-preserving binary operator s.t., for all $\mathbf{i} \in \mathit{dom}(\mathit{sh}(\mathtt{t}_1))$, $\mathit{add}(\mathtt{t}_1, \mathtt{t}_2)[\mathbf{i}] = \mathtt{t}_1[\mathbf{i}] + \mathtt{t}_2[\mathbf{i}]$ holds.
- *sign* and *relu* are shape-preserving unary operators s.t., for all $\mathbf{i} \in \mathit{dom}(\mathit{sh}(\mathtt{t}))$, hold

$$\mathit{sign}(\mathtt{t})[\mathbf{i}] = \begin{cases} 1 & \text{if } \mathtt{t}[\mathbf{i}] > 0 \\ -1 & \text{if } \mathtt{t}[\mathbf{i}] < 0 \\ 0 & \text{if } \mathtt{t}[\mathbf{i}] = 0, \end{cases} \quad \text{and} \quad \mathit{relu}(\mathtt{t})[\mathbf{i}] = \begin{cases} \mathtt{t}[\mathbf{i}] & \text{if } \mathtt{t}[\mathbf{i}] \geq 0 \\ 0 & \text{if } \mathtt{t}[\mathbf{i}] < 0. \end{cases}$$

- $\mathit{transpose}_p$ where p is a permutation of the list of indices $[1, \ldots, r]$ is a unary operator over tensors of rank r. Given a tensor $\mathtt{t}$ of shape $\mathit{sh}(\mathtt{t}) = (\dim_1, \ldots, \dim_r)$, it generates a tensor of shape $(\dim_{p[1]}, \ldots, \dim_{p[r]})$ such that, $\forall [i_1, \ldots, i_r] \in \mathit{dom}(\mathit{sh}(\mathtt{t}))$, holds

$$\mathit{transpose}_p(\mathtt{t})[i_{p[1]}, \ldots, i_{p[r]}] = \mathtt{t}[i_1, \ldots, i_r].$$

– *matmul* is a binary operator that represents the standard matrix multiplication over tensors of same rank. Specifically, let $\mathtt{t}_1$ be a tensor of shape $(\dim_1^1, \ldots, \dim_r^1)$. Tensor $\mathtt{t}_2$ must be a tensor of rank 2 of shape $(\dim_1^2, dim_2^2)$ such that $\dim_r^1 = \dim_1^2$. Then, tensor $\mathtt{t}' = \mathit{matmul}(\mathtt{t}_1, \mathtt{t}_2)$ has rank r with shape $sh(\mathtt{t}') =$

$$(\dim_1^1, \ldots, \dim_{r-2}^1, \dim_{r-1}^1, \dim_2^2)$$

that satisfies $\forall \mathbf{i} = [i_1, \ldots, i_r] \in dom(sh(\mathtt{t}'))$, $\mathit{matmul}(\mathtt{t}_1, \mathtt{t}_2)[\mathbf{i}] =$

$$\sum_{j \in [\dim_r^1]} \mathtt{t}_1[\mathbf{i}_{r \mapsto j}] \times \mathtt{t}_2[j, i_r].$$

– argmax_ℓ is a unary operator that applies to tensors of rank ℓ or more and that indicates the index of the highest value across the ℓ-th dimension. In case of a tie, argmax_ℓ reports the lowest index. If tensor $\mathtt{t}$ has shape $(\dim_1, \ldots, \dim_r)$, then tensor $\mathtt{t}' = \mathit{argmax}_\ell(\mathtt{t})$ has shape

$$(\dim_1, \ldots, \dim_{\ell-1}, 1, \dim_{\ell+1}, \ldots, \dim_r),$$

i.e., identical to that of $\mathtt{t}$ except that the ℓ-th dimension collapses to 1. Furthermore, $\forall \mathbf{i} \in dom(sh(\mathtt{t}'))$, $\mathtt{t}'[\mathbf{i}_{\ell \mapsto 0}]$ is the position of the highest element of $[\mathtt{t}[\mathbf{i}_{\ell \mapsto 0}], \ldots, \mathtt{t}[\mathbf{i}_{\ell \mapsto \dim_\ell - 1}]]$.

– *softmax* is a shape-preserving unary operator that performs the soft-max overall the last layer of the tensor. Specifically, $\forall \mathbf{i} = [i_1, \ldots, i_r]$, $\mathit{softmax}(\mathtt{t})[\mathbf{i}]$ equals

$$\frac{e^{\mathtt{t}[\mathbf{i}]}}{\sum_{k \in [\dim_r]} e^{\mathtt{t}[\mathbf{i}_{r \mapsto k}]}}$$

3.2 Proof

Consider a vector $\mathbf{i} \in dom(sh(\textsc{ArgMax}_\ell(\mathtt{t})))$ and index $\hat{\jmath} \stackrel{def}{=} \textsc{ArgMax}_\ell(\mathtt{t})[\mathbf{i}]$. With reference to Figure 1, our transformation replaces ArgMax with a sub-NN that contains 9 ONNX operators. The proof consists in tracking properties of the tensor at different steps of this sub-NN in relation with $\mathbf{i}$ and $\hat{\jmath}$. To this end, for each step $s \in \{0, \ldots, 9\}$, we use $\mathtt{t}_s$ to refer to the tensor after the s-th operation. For instance, $\mathtt{t}$ is also called $\mathtt{t}_0$ and the tensor after the *sign* operator is $\mathtt{t}_3$. Our goal is to demonstrate that $\mathtt{t}_9[\mathbf{i}]$ evaluates to $\textsc{ArgMax}_\ell(\mathtt{t})[\mathbf{i}]$, i.e., $\hat{\jmath}$.

Properties of $\mathtt{t}_0$ By definition of $\hat{\jmath}$, we have:

– $\forall k \in [\dim_\ell]$. $\mathtt{t}_0[\mathbf{i}_{\ell \mapsto \hat{\jmath}}] \geq \mathtt{t}_0[\mathbf{i}_{\ell \mapsto k}]$ and
– $\forall k \in [\dim_\ell]$. $k < \hat{\jmath} \implies \mathtt{t}_0[\mathbf{i}_{\ell \mapsto \hat{\jmath}}] > \mathtt{t}_0[\mathbf{i}_{\ell \mapsto k}]$.

Properties of $\mathtt{t}_1 = \mathit{transpose}_{p_{\ell,r}}(\mathtt{t}_0)$ Let $\mathbf{i}'$ be the vector $\mathbf{i}$ after the permutation has been applied; vector $\mathbf{i}$ will not be used again until the end of the proof. We can rewrite the previous properties with the new vector:

– $\forall k \in [\dim_\ell]$. $\mathtt{t}_1[\mathbf{i}'_{r \mapsto \hat{\jmath}}] \geq \mathtt{t}_1[\mathbf{i}'_{r \mapsto k}]$ and
– $\forall k \in [\dim_\ell]$. $k < \hat{\jmath} \implies \mathtt{t}_1[\mathbf{i}'_{r \mapsto \hat{\jmath}}] > \mathtt{t}_1[\mathbf{i}'_{r \mapsto k}]$.

Properties of $\mathtt{t}_4 = relu(sign(matmul(\mathtt{t}_1, \mathtt{t}_v)))$ The following properties hold:

- The values in tensor $\mathtt{t}_4$ are all in $\{0, 1\}$. This is due to the application of *sign* followed with *relu*.
- $\forall k < \hat{\jmath}.\ \mathtt{t}_4[\mathbf{i}'_{r\mapsto\xi(k,j)}] = 1$. This is because $\mathtt{t}_1[\mathbf{i}'_{r\mapsto\hat{\jmath}}]$ is strictly greater than $\mathtt{t}_1[\mathbf{i}'_{r\mapsto k}]$, so that their difference is positive, the *sign* operation returns 1, and the *relu* operation returns 1.
- $\forall k > \hat{\jmath}.\ \mathtt{t}_4[\mathbf{i}'_{r\mapsto\xi(k,j)}] = 0$. This is because $\mathtt{t}_1[\mathbf{i}'_{r\mapsto\hat{\jmath}}]$ is greater than or equal to $\mathtt{t}_1[\mathbf{i}'_{r\mapsto k}]$, so that their difference is 0 or less, the *sign* operation returns 0 or -1, and the *relu* operation returns 0.

We do not need to consider the values associated with pair $\{j, k\} \not\ni \hat{\jmath}$ (except for the fact that they belong to $\{0, 1\}$, first property), because the comparison of k with $\hat{\jmath}$ will suffice to infer relevant information about this index k.

Properties of $\mathtt{t}_5 = matmul_u(\mathtt{t}_4)$ The *matmul* operation between $\mathtt{t}_4$ and $\mathtt{t}_u$ implies that, for any $j \in [\dim_\ell]$, the following holds:

$$\mathtt{t}_5[\mathbf{i}'_{r\mapsto j}] \quad = \quad \sum_{k<j} \mathtt{t}_4[\mathbf{i}'_{r\mapsto\xi(k,j)}] \times 1 \quad + \quad \sum_{k>j} \mathtt{t}_4[\mathbf{i}'_{r\mapsto\xi(k,j)}] \times (-1)$$

where k ranges over $[\dim_\ell]$. Thus we can show:

- $\mathtt{t}_5[\mathbf{i}'_{r\mapsto\hat{\jmath}}] = \hat{\jmath}$. From the previous paragraph, we know that the value on $\mathtt{t}_4$ is 1 when k is lower than $\hat{\jmath}$ and 0 otherwise. Thus, the sum adds up exactly to $\sum_{k<\hat{\jmath}} 1 \times 1 + \sum_{k>\hat{\jmath}} 0 \times (-1) = \hat{\jmath}$.
- $\forall j < \hat{\jmath}.\ \mathtt{t}_5[\mathbf{i}'_{r\mapsto j}] \leq j - 1$. Because the values of tensor $\mathtt{t}_4$ are between 0 and 1, an upper bound to this sum is j; however, the index $k = \hat{\jmath} > j$ yields a value $\mathtt{t}_4[\mathbf{i}'_{r\mapsto\xi(k,j)}] = 1$, so that the upper bound for the sum is $\sum_{k<j} 1 \times 1 + \sum_{k>j, k\neq j} 0 \times (-1) + 1 \times (-1) = j - 1$.
- $\forall j > \hat{\jmath}.\ \mathtt{t}_5[\mathbf{i}'_{r\mapsto j}] \leq j - 1$. A similar reasoning applies except that, this time, the known value is for index $k = \hat{\jmath} < j$: $\mathtt{t}_4[\mathbf{i}'_{r\mapsto\xi(k,j)}] = 0$. This implies that the upper bound to the sum is $\sum_{k<j, k\neq\hat{\jmath}} 1\times1+0\times1+\sum_{k>j} 0\times(-1) = j-1$.

Properties of $\mathtt{t}_7 = relu(add(\mathtt{t}_5, \mathtt{t}_b))$ The operations between $\mathtt{t}_5$ and $\mathtt{t}_7$ are such that, for any $j \in [\dim_\ell]$, the following holds:

$$\mathtt{t}_7[\mathbf{i}_{r\mapsto j}] = relu(\mathtt{t}_5[\mathbf{i}'_{r\mapsto j}] + 1 - j).$$

Thus, we can show:

- $\mathtt{t}_7[\mathbf{i}'_{r\mapsto\hat{\jmath}}] = relu(\hat{\jmath} + 1 - \hat{\jmath}) = 1$.
- $\forall j \in [\dim_\ell].\ j \neq \hat{\jmath} \implies \mathtt{t}_7[\mathbf{i}'_{r\mapsto j}] = 0$. This is because the value before *relu* is at most $(j - 1) + 1 - j = 0$.

Properties of $\mathtt{t}_8 = matmul(\mathtt{t}_7, \mathtt{t}_w)$ The *matmul* operation is such that

$$\mathtt{t}_8[\mathbf{i}'_{r\mapsto 0}] = \sum_{j\in dom(\dim_\ell)} j \times \mathtt{t}_7[\mathbf{i}'_{r\mapsto j}].$$

Since all $\mathtt{t}_7[\mathbf{i}'_{r\mapsto j}]$ equal 0 except for $j = \hat{\jmath}$ (for which the value is 1), the sum is $\mathtt{t}_8[\mathbf{i}'_{r\mapsto 0}] = \hat{\jmath}$.

Properties of $\mathtt{t}' = \mathtt{t}_9 = \mathit{transpose}_{p_{\ell,r}}(\mathtt{t}_8)$ This is a simple transpose: $\mathtt{t}_9[\mathbf{i}_{\ell \mapsto 0}] = \hat{\jmath}$.

This concludes our proof of correctness for the transformation of the ArgMax operator.

4 Application: Confidence-Based Properties for Provers

We demonstrate the benefits of our work on the verification of robustness of NN.

4.1 Confidence-based global robustness

Robustness is one of the most studied properties of NN. It specifies that a small change to the input should not modify dramatically the output of the network. When testing this property globally (i.e., over the complete range of possible inputs rather than on a few selected points), this property is never satisfied for classification NNs: in the region close to the frontier of decision, tiny changes to the input *will* modify the output. Thus, we are interested in verifying the property only when the decision is associated with a certain confidence.

The notion of *confidence-based safety properties* has been introduced by Athavale et al. [4]. In such a property, the *confidence* refers to the confidence level of the prediction made by the NN. The idea is to add a constraint to the original safety property of interest: robustness is only required when the confidence score of the input is above a given threshold. From this general concept, confidence-based properties can be derived to confidence-based global robustness, which allows some non-robust behaviour for low-confidence inputs. Its formal definition is given in Definition 1.

Definition 1 (Confidence-based global robustness). *Let* $f : \mathbb{R}^m \to \mathbb{R}^n$ *be a NN over tensors of shape* (m), $\mathcal{X} \subseteq \mathbb{R}^m$ *be a subspace of input tensors, and* $\varepsilon > 0$ *and* $\kappa \in (0, 1]$ *be two constants. NN* f *is* globally ε-robust for confidence κ over $\mathcal{X}$ *if and only if it satisfies*

$$\forall \mathtt{x}, \mathtt{x}' \in \mathcal{X}. \bigwedge_{j \in [m]} |\mathtt{x}[j] - \mathtt{x}'[j]| \leq \varepsilon \ \wedge \ \mathit{CONF}(f(\mathtt{x})) > \kappa \Rightarrow \mathit{CLASS}(f(\mathtt{x})) = \mathit{CLASS}(f(\mathtt{x}'))$$

where $\mathit{CONF}(f(\mathtt{x})) \stackrel{def}{=} \max_{j \in [n]} (\mathrm{softmax}(f(\mathtt{x}))[j])$ *and* $\mathit{CLASS}(f(\mathtt{x})) \stackrel{def}{=} \mathrm{argmax}\,(f(\mathtt{x}))$.

We note that the softmax operator in Def. 1 makes the problem very hard to verify. We now discuss two implementations of confidence-based global robustness based on the way the class equality is checked.

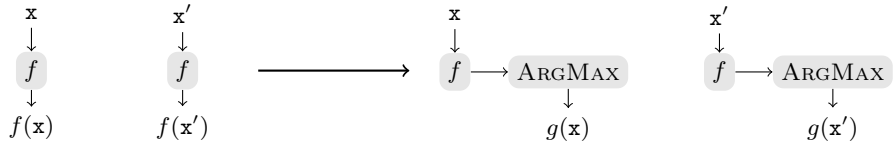

Fig. 2: g is obtained by adding an ArgMax operator after f

Logical Implementation A practical implementation of confidence-based global robustness, `LOGICAL`, is available in CAISAR [2].

To check if the predicted class of two vectors $\mathtt{x}, \mathtt{x}' \in \mathbb{R}^m$ is the same with respect to f, one needs to verify if $\text{argmax}(f(\mathtt{x})) = \text{argmax}(f(\mathtt{x}'))$. Checking this condition using first-order logic requires the identification of the label of the class with the highest score. Given a label $l \in [n]$, $\text{CLASS}(f(\mathtt{x})) = l$ implies that for all $l' \in [n]$, we have $f(x)[l] \geq f(x)[l']$. Thus, the constraint $\text{CLASS}(f(\mathtt{x})) = l$ can be encoded using a predicate $\text{pred}_{f,\mathtt{x},l}$ defined such that

$$\text{pred}_{f,\mathtt{x},l} \quad \stackrel{def}{=} \quad \forall l' \in [n].\ f(\mathtt{x})[l] \geq f(\mathtt{x})[l'].$$

We note that this implementation ignores the tie situations, which is a very minor issue. Therefore, checking $\text{CLASS}(f(\mathtt{x})) = \text{CLASS}(f(\mathtt{x}'))$ can be reduced to checking the satisfiability of the predicate

$$\text{eq}_{f,\mathtt{x},\mathtt{x}'} \quad \stackrel{def}{=} \quad \forall l \in [n].\ \text{pred}_{f,\mathtt{x},l} \Leftrightarrow \text{pred}_{f,\mathtt{x}',l}.$$

The constraint $\neg\ \text{eq}_{f,\mathtt{x},\mathtt{x}'}$ is then directly passed to the solver (PyRAT). The logical implementation is therefore applicable because the *argmax* operation is not part of the input NN but is, instead, a constraint related to our specific *Robustness* problem.

ArgMax operators It is also possible to verify the class equality of two executions of f by using the ArgMax operator, `ARGMAX`. Figure 2 represents the two original executions that are compared and their reformulation using $g \stackrel{def}{=} \text{ArgMax} \circ f$, obtained by adding an ArgMax operator after f.

In this setting, checking the class equality between $f(\mathtt{x})$ and $f(\mathtt{x}')$ is reduced to checking $g(\mathtt{x}) = g(\mathtt{x}')$. This predicate is lighter than $\text{eq}_{f,\mathtt{x},\mathtt{x}'}$. In practice, to obtain the graph presented in Fig. 2, we use CAISAR. CAISAR edits the original computation graph by adding the ArgMax operator at the end. Since the operator ArgMax is unsupported by NN verification solvers, CAISAR transforms it with the reformulation given in Fig. 1.

4.2 Experimental evaluation

To assess the relevance of this reformulation, we experimentally compared the two implementations, LOGICALand ARGMAX, with PyRAT.[1] We note that this is not a direct comparison between using a solver that supports ArgMax and using our reformulation. The goal of this section is to check whether the use of transformations is a viable method to handle the lack of support for some operators. To this end, we use several instances of neural network verification benchmarks.

COMPAS is a neural network that, based on a selection of features, assesses the probability that criminal defendants will commit future crimes; here, we use a feedforward neural network with 53 parameters. *Adult* is a neural network that estimates, using a person's information, whether this person has an income above $50,000 per year; here, we use a feedforward neural network with 34 parameters. *Industrial* is a neural network used in an undisclosed industrial setting; here, we use a feedforward neural network with 1630 parameters.

For these experiments, all instances are set with $\varepsilon = 0.001$, only the confidence threshold varies. The time taken for each instance is the mean across all executions of the same instance (10 executions) with the prover PyRAT. Note that in this study we do not focus on the efficient verification of confidence-based global robustness, but rather on performance differences between two implementations for class equality.

From Table 2, we observe that the ARGMAX implementation is significantly faster on both *COMPAS* and *Industrial*, but it is not for *Adult*, where the verification is slightly slower. It can be argued that, for a neural network with fewer than three output classes, the ArgMax encoding is heavier to verify than the logical formula that is small (its size grows with the number of output classes). But with more output classes, performance difference is observed in favour of the ARGMAX implementation. Therefore, using the ARGMAX approach yields a faster verification for neural networks with more than two output classes.

This experimentation highlights two positive aspects of using the ARGMAX representation. First, it speeds up verification for neural networks with more than two output classes. Second, it helps to have an easier representation of the verification problem: the first-order logic encoding is replaced by an equality test on the output of two neural networks, as shown in Figure 2.

5 Conclusion

In this paper, we propose a transformation of the ArgMax ONNX operator into an equivalent sequence of simpler ones. While this could be used in any context, we are particularly interested in using this for verification of neural networks where solvers generally support a restricted set of operators that excludes ArgMax. We implemented this transformation in the CAISAR plateform and

[1] Attempts to use other solvers 'out-of-the-box' failed, meaning that these solvers would require careful parameter-tuning to solve these problems in reasonable time.

		Validity	LOGICAL	ARGMAX	Time difference	
COMPAS	($\kappa = 0.9$)	✓	17.4s	12.3s	−5.1s	(−29.3%)
COMPAS	($\kappa = 0.8$)	✓	18.8s	13.5s	−5.3s	(−28.2%)
COMPAS	($\kappa = 0.7$)	✓	38.3s	30.0s	−8.3s	(−21.7%)
COMPAS	($\kappa = 0.6$)	✓	564.6s	493.7s	−70.9s	(−12.6%)
COMPAS	($\kappa = 0.2$)	✗	63.8s	34.2s	−29.6s	(−46.4%)
COMPAS	($\kappa = 0.1$)	✗	64.0s	34.4s	−29.6s	(−46.2%)
Adult	($\kappa = 0.9$)	✓	13.4s	13.5s	+0.1s	(+0.7%)
Adult	($\kappa = 0.8$)	✓	21.8s	23.0s	+1.2s	(+5.5%)
Adult	($\kappa = 0.7$)	✓	123.2s	138.4s	+15.2s	(+12.3%)
Adult	($\kappa = 0.2$)	✗	78.2s	86.7s	+8.5s	(+10.9%)
Adult	($\kappa = 0.1$)	✗	78.6s	86.8s	+8.2s	(+10.4%)
Industrial	($\kappa = 0.999$)	✓	87.8s	61.9s	−25.9s	(−29.5%)
Industrial	($\kappa = 0.99$)	✓	274.6s	188.1s	−86.5s	(−31.5%)
Industrial	($\kappa = 0.9$)	✓	2058.9s	1389.5s	−669.4s	(−32.5%)
Industrial	($\kappa = 0.2$)	✗	115.9s	68.8s	−47.1s	(−40.6%)
Industrial	($\kappa = 0.1$)	✗	116.2s	68.6s	−47.6s	(−41.0%)

Table 2: Runtime comparison on three benchmarks for different confidence thresholds ✓ stands for "Valid", ✗ for "Invalid"

showed that it not only allows the use of solvers that do not support this operator but also yields better performance than an ad-hoc reformulation for the problem of confidence-based global robustness.

This type of contribution means that developers of dedicated solvers can now concentrate on the interesting aspects of their technology. Having access to this transformation means that they do not need to explicitly support ArgMax, which implies less work and simpler, less error-prone, and easier-to-maintain code on their side. We recognise however that our implementation, being a generic one, could lead to a performance drop depending on the inner workings of the solver. For instance, an ArgMax operator allows a solver to immediately infer the range $[\dim_\ell]$ of values of the output network; inferring the same range for the network from Figure 1 requires non-trivial reasoning.

Our goal is to expand this work and to provide additional transformations. For instance, ArgMax automatically gives us ArgMin for free. The SoftMax operator can be easily derived from the Exp one. We want to provide the largest possible array of transformations to broaden the class of NN verification problems that can be solved by existing solvers. Ideally, we would like to define a minimal subset of supported operators and a partially-ordered set from "higher level" operators to the minimally-supported subset.

Another interesting extension is to propose multiple transformations, when possible, and study which one is the most efficient for which solvers. For instance,

the currently proposed transformation requires to apply a SIGN operator over a tensor whose size is quadratic in the input (this is because the transformation computes the difference between every pair of elements in the vector). A different transformation could require only a linear number of SIGN operations. This can be done by implementing a single-elimination procedure comprised of a log number of rounds in each of which half the elements are discarded. It is unclear whether this implementation would lead to better performance but there is an argument to be made that the SIGN operation is a critical path in the search.

A Variants of *argmax*

The specification of *argmax* includes two options that we have not yet discussed.

The first option indicates whether the dimension upon which the *argmax* is performed is removed altogether. This means that the output shape is not

$$(\mathrm{dim}_1, \ldots, \mathrm{dim}_{\ell-1}, 1, \mathrm{dim}_{\ell+1}, \mathrm{dim}_r)$$

but

$$(\mathrm{dim}_1, \ldots, \mathrm{dim}_{\ell-1}, \mathrm{dim}_{\ell+1}, \mathrm{dim}_r).$$

This can be implemented adding a RESHAPE operator that allows one to access the list of atomic elements in the tensor differently. For instance, a tensor of shape $(2, 3, 4)$ with 24 elements could be interpreted as a tensor of shape $(1, 6, 1, 4)$. In this case, because the reshape occurs on a dimension of size 1, the reshaping is trivial.

The second option pertains to the tie breaking rule between elements of identical value. When the *select last index* option is set, the tie breaking rule favours the element at highest index instead of the lowest index.

This interpretation can be achieved by modifying the constant tensors involved in the transformation. For instance, tensor $\mathtt{t}_v$ is defined so that the difference $v[k] - v[j]$ is computed, where $k > j$. The reason for this order is this operation is followed by a *sign* and a *relu*: the case $v[k] = v[j]$ is treated similarly as the case $v[k] < v[j]$. When the tie breaking rule is reversed, we want to compute $v[j] - v[k]$ instead. Thus, the values -1 and 1 are swapped in $\mathtt{t}_u$ (and in $\mathtt{t}_v$).

This implies that the values of $\mathtt{t}_b$ also need updating. The value associated with $\hat{\jmath}$ in $\mathtt{t}_5$ is now $(\mathrm{dim}_\ell - \hat{\jmath} - 1)$. The bias associated with j is therefore $2 - \mathrm{dim}_\ell + j$, so that applying this bias associates $\hat{\jmath}$ in $\mathtt{t}_6$ to 1.

References

1. Alberti, M., Bobot, F., Chihani, Z., Grastien, A., Girard-Satabin, J.: The CAISAR platform: extending the reach of Machine Learning specification and verification. In: International Conference on Integrated Formal Methods (IFM). pp. 290–309 (2025)

2. Ardouin, G., Alberti, M., Girard-Satabin, J.: La confiance avec le contrôle : spécification et vérification d'hyperpropriétés sur réseaux de neurones. In: Journées francophones des langages applicatifs (JFLA). pp. 1–21 (2026)
3. Arora, J., Lu, S., Jain, D., Xu, T., Houshmand, F., Phothilimthana, P.M., Lesani, M., Narayanan, P., Murthy, K.S., Bodik, R., Sabne, A., Mendis, C.: TensorRight: automated verification of tensor graph rewrites. In: Proceedings of the ACM on Programming Languages. vol. 9, pp. 832–863 (2025)
4. Athavale, A., Bartocci, E., Christakis, M., Maffei, M., Nickovic, D., Weissenbacher, G.: Verifying global two-safety properties in neural networks with confidence. In: International Conference on Computer Aided Verification (CAV). pp. 329–351 (2024)
5. Bak, S.: nnenum: verification of ReLU neural networks with optimized abstraction refinement. In: NASA Formal Methods Symposium (NFM). pp. 19–36
6. Brix, C., Bak, S., Johnson, T.T., Wu, H.: The fifth international verification of neural networks competition (VNN-COMP 2024): summary and results (2024)
7. Community: ONNX: Open Neural Network eXchange Format standard for machine learning interoperability (2022)
8. Demarchi, S., Guidotti, D., Pulina, L., Tacchella, A.: Supporting standardization of neural networks verification with VNN-LIB and CoCoNet. In: Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS). pp. 47–58 (2023)
9. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The Marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification (CAV). pp. 443–452 (2019)
10. Lemesle, A., Lehmann, J., Gall, T.L., Chihani, Z.: Neural Network Verification with PyRAT. In: International Static Analysis Symposium (SAS). pp. 11–33 (2025)
11. Shriver, D., Elbaum, S., Dwyer, M.B.: DNNV: a framework for deep neural network verification. In: Computer Aided Verification. pp. 137–150. Lecture Notes in Computer Science (2021)
12. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for complete and incomplete neural network robustness verification. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 29909–29921 (2021)
13. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., Huang, P., Lahav, O., Wu, M., Zhang, M., Komendantskaya, E., Katz, G., Barrett, C.: Marabou 2.0: a versatile formal analyzer of neural networks. In: International Conference on Computer Aided Verification (CAV). pp. 249–264 (2024)