ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 463-468

Extracting Interface Assertions from Neural Networks in Polyhedral Format

Stephan Breutel, Frédéric Maire, Ross Hayward
Smart Devices Laboratory
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane Qld 4001, Australia

Abstract. It is difficult to have confidence in software components if they have no clear and valid interface description. In this paper we describe a new method that determines informative interface assertions for the function f, learned by a feed-forward neural network or sigmoidal support vector machine from training data. The interface assertions we are considering are of the form "if $\mathbf{y} \in \mathcal{R}_o$ then $f^{-1}(\mathbf{y}) \subseteq \mathcal{R}_i$", where $\mathbf{y}$ is an output vector in the region $\mathcal{R}_o$ of the output vector space and $f^{-1}(\mathbf{y})$ is a set of input vectors $\mathbf{x}$, such that $f(\mathbf{x}) = \mathbf{y}$. The most refined interface assertions are obtained when $\mathcal{R}_i$ is the reciprocal image of $\mathcal{R}_o$ by f. The method introduced in this paper computes a polyhedron containing $\mathcal{R}_i$ from a polyhedral output region $\mathcal{R}_o$. Empirical results indicate that the complexity of the proposed method scales well with the number of nodes per layer.

1 Introduction

The safety requirements for software in areas such as airplane control and medical applications highlight the importance of software verification in computer science. For a neural network application, one might, for example, forbid special output regions. In this case it is important to show that these output regions are not reached by the neural network operating within a specified input range.

To validate neural networks, we can provide a set of conditions on the input under which a set of output conditions are valid (and vice versa). Let f be the function a feed-forward neural network has learned from the training data. The interface assertions we are considering here are of the form: [1] "if $\mathbf{y} \in \mathcal{R}_o$ then $f^{-1}(\mathbf{y}) \subseteq \mathcal{R}_i$", where $\mathbf{y}$ is an output vector in the region $\mathcal{R}_o$ of the output vector-space and $f^{-1}(\mathbf{y}) = \{\mathbf{x} | f(\mathbf{x}) = \mathbf{y}\}$. Historically people have used axis-parallel hypercubes for $\mathcal{R}_i$ and $\mathcal{R}_o$ to extract rules from neural networks [3]. However, the most refined interface assertions are obtained when $\mathcal{R}_i$ is the reciprocal image of $\mathcal{R}_o$ by f. We approximate the reciprocal image by successively back-propagating finite unions of polyhedra through all layers of a feed-forward neural network, as unions of polyhedra are capable of concisely

[1] We also work on forward-propagating regions, i.e. extracting knowledge of the form: "if $\mathbf{x} \in \mathcal{R}_i$ then $f(\mathbf{x}) \in \mathcal{R}_o$". In both cases, we use the terminology interface assertions.

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 463-468

describing arbitrary regions of higher dimensional vector spaces.
The idea of propagating regions through a neural network was first introduced in [2] by considering axis-parallel hypercubes. The approach, called Validity Interval Analysis, is based on a refinement process that is performed by forward- and backward propagating hypercubes through the neural network. However, hypercubes are not closed under affine transformations, whereas polyhedra are [1]. The algorithm presented in [1] works perfectly for linear transformations, but the complexity of the method for non-linear transformations does not scale well with the number of nodes per layer. In this paper we follow a different approach that scales better.
The paper is organized as follows: Section 2 describes how to back-propagate a polyhedron through a neural network. Section 3 provides empirical results. We conclude this paper and describe future work in Section 4.

2 Backpropagating Polyhedra

In this section, we explain how to backpropagate a polyhedron through a single weight layer network. We successively apply the algorithm to each layer of a multi-layer network, starting from the output layer back to the input layer.
A polyhedron $\mathcal{P} = \{\mathbf{x} \mid \mathbf{A}\,\mathbf{x} \leq \mathbf{b}\}$ is the intersection of a finite number of half-spaces. We assume that our polyhedron are bounded. From a practical point of view this assumption is justified as neural networks normally deal with finite inputs. A box $\mathcal{B}$ is an axis-parallel hypercube, a subscript indicates if the box is in the input (x) or output (y) space. A superscript denotes an iteration index.

Affine transformation phase It is easy to show [1] that the reciprocal image of a polyhedron $\mathcal{P}_y = \{\mathbf{y}|\mathbf{A}\mathbf{y} \leq \mathbf{b}\}$ under an affine transformation $\mathbf{x} \mapsto \Gamma(\mathbf{x}) = \mathbf{W}\mathbf{x} + \boldsymbol{\theta}$ is given by: $\mathcal{P}_x = \{\mathbf{x}|\mathbf{A}\mathbf{W}\mathbf{x} \leq \mathbf{b} - \mathbf{A}\boldsymbol{\theta}\}$. Linear programming methods are used to remove redundant inequalities.

Transfer function phase In [1], a piece-wise linear approximation of the sigmoid function was used to approximate the true reciprocal image of a back-propagated polyhedron. This idea is practical only for low dimensional cases, i.e. small number of neurons, as it splits the vector space into m^n boxes, where m is the number of piece-wise linear functions and n is the number of neurons, which results in an exponential time and space complexity.
Our approach avoids this problem by computing a polyhedron containing the true reciprocal image. The direction vector $\mathbf{g}$ of each hyperplane (facet) of this *wrapping* polyhedron has to be determined. The gradient vector of a point on the manifold of the true reciprocal image is possibly the best choice, as this leads to the exact solution for linear transfer functions. Our problem can be stated as follows: given a polyhedron $\mathcal{P}_y = \{\mathbf{y}|\mathbf{A}\mathbf{y} \leq \mathbf{b}\}$ in the output space (y-space) of a transfer function layer we want to compute the smallest polyhedron, with respect to the chosen directions $\mathbf{g}$'s, that contains the reciprocal image $\mathcal{R} = \sigma^{-1}(\mathcal{P}_y) = \{\mathbf{x}|\mathbf{A}\sigma(\mathbf{x}) \leq \mathbf{b}\}$ in the input space (x-space). Once we have chosen a direction vector $\mathbf{g}$ for a hyperplane we have to determine the

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 463-468

optimal position, i.e. a position tangent to the region $\mathcal{R}$. This problem can be expressed as a non-linear optimization problem in x-space [2]:

$$max \quad \mathbf{g^T x} \qquad \text{subject to} \qquad \mathbf{A}\sigma(\mathbf{x}) \leq \mathbf{b}$$

The solution of this optimization problem defines the optimal position for a hyperplane $\mathcal{H} = \{\mathbf{x}|\mathbf{g}^T\mathbf{x} = \beta\}$, such that $\mathcal{R} \subseteq \mathcal{H}^- = \{\mathbf{x}|\mathbf{g}^T\mathbf{x} < \beta\}$.
We use a binary search method to find for a hyperplane $\mathcal{H}$ a position as close as possible to the region $\mathcal{R}$= $\{\mathbf{x}|\mathbf{A}\sigma(\mathbf{x}) \leq \mathbf{b}\}$, such that $\mathcal{R} \subseteq \mathcal{H}^-$. At the beginning, $\mathcal{H}$ is positioned with the midpoint between a point $\mathbf{q}_x$ on the manifold of $\mathcal{R}$, and the vertex $\mathbf{p}_x$, the corner with the maximum value for the cost-function $\mathbf{g}^T\mathbf{x}$ subject to $\mathbf{x} \in \mathcal{B}_x^0 = \square(\mathcal{R})$, where $\square(\mathcal{R})$ denotes the smallest axis-parallel hypercube containing the region $\mathcal{R}$. The hyperplane is moved closer towards $\mathcal{R}$ if $\mathcal{R} \cap \mathcal{H}^+$ is empty. To determine whether $\mathcal{R} \cap \mathcal{H}^+$ is empty, a box-refinement process is applied (similarly [2] used a box-refinement process to refine axis-parallel rules). If there is no intersection we can move the hyperplane closer towards $\mathcal{R}$, otherwise we move the hyperplane further outside. We stop the algorithm if the distance between two consecutive hyperplane positions is less than a small value ϵ, or if the volume of the refined box $\mathcal{B}_x$ is very small. In the last case we know that there is an intersection between $\mathcal{B}_x$ and the region $\mathcal{R}$. We can position $\mathcal{H}$ to a corner of $\mathcal{B}_x$ outside of $\mathcal{R}$. It can be shown that the upper bound for the distance between $\mathcal{R}$ and $\mathcal{H}$ is then given by the diagonal of the box.

binary search - main algorithm

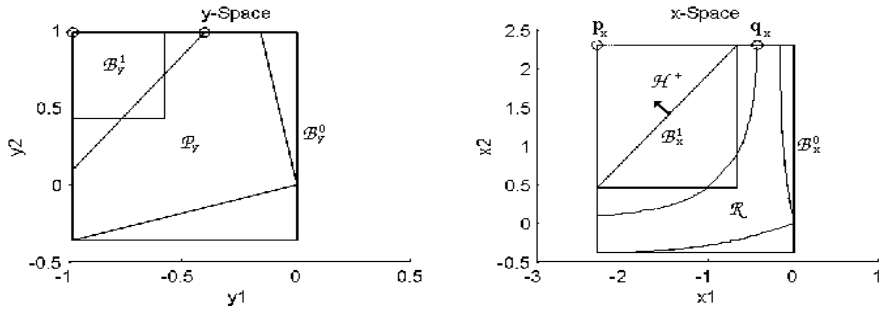
$//InitPhase$
$\mathcal{P}_y = \{\mathbf{y}|\mathbf{A}\mathbf{y} \leq \mathbf{b}\}; \mathcal{B}_y^0 = \square(\mathcal{P}_y); \mathcal{B}_x^0 = \square(\sigma^{-1}(\mathcal{B}_y^0))$
$\mathbf{p}_x = argmax \quad \mathbf{g}^T\mathbf{x} \quad s.to \quad \mathbf{x} \in \mathcal{B}_x^0;$
$\mathbf{q}_y = argmax \quad \mathbf{g}^T\mathbf{y} \quad s.to \quad \mathbf{A}\mathbf{y} \leq \mathbf{b};$
$\mathbf{q}_x = \sigma^{-1}(\mathbf{q}_y); \triangle\lambda = 0.5; \lambda = 0.5;$
$//MainLoop$
do
⌈
$\mathbf{m}_x = \mathbf{p}_x + \lambda(\mathbf{q}_x - \mathbf{p}_x); \quad \mathcal{H} = \{\mathbf{x}|\mathbf{g}^T\mathbf{x} = \mathbf{g}^T\mathbf{m}_x\};$
$\mathcal{B}_x = \text{refine}(\mathcal{B}_x^0, \mathcal{B}_y^0, \mathcal{P}_y, \mathcal{H}); \quad \triangle\lambda = 0.5\ \triangle\lambda;$
if $((V(\mathcal{B}_x) \stackrel{?}{=} 0) \quad \lambda = \lambda + \triangle\lambda$
else $\lambda = \lambda - \triangle\lambda$
⌋**while** $((V(\mathcal{B}_x) > \epsilon) \wedge (\triangle\lambda > \epsilon))$

The crucial part of our algorithm is the box refinement process. The refinement process is a succession of a box refinement in x-space followed by a forward-propagation of the refined box to y-space, and a refinement process in y-space

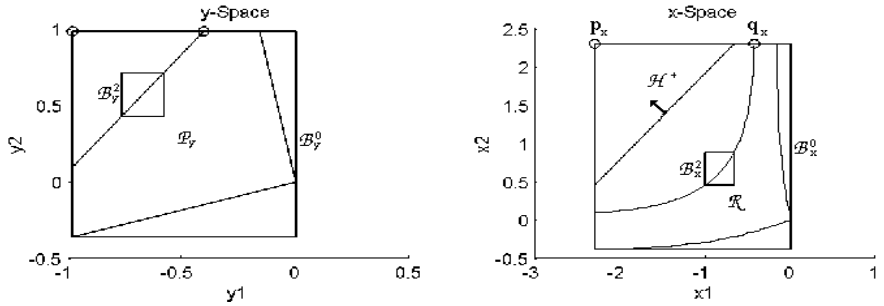
[2] A nice feature of our method is that the calculation can be distributed among different CPU's, according to the chosen number of hyperplane directions.

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 463-468

followed by a back-propagation of the refined box to x-space. We will say we send a box from x-space to y-space and vice versa. The refinement for the $k+1$ iteration in x-space is calculated by: $\mathcal{B}_x^{k+1} = \Box(\mathcal{B}_x^k \cap \mathcal{H}^+)$ and in y-space by: $\mathcal{B}_y^{k+1} = \Box(\mathcal{B}_y^k \cap \mathcal{P}_y)$. The forward-propagation of a box from x-space to y-space is a component-wise application of the sigmoid function on the box, similarly the backward-propagation of a box from y-space to x-space is a component-wise application of the inverse sigmoid function on the box. We repeat this process until we know if there is an intersection between the half-space $\mathcal{H}^+$ and the region $\mathcal{R}$. In Figure 1 we depict an example for the refinement process.



(a) On the right hand side of the above figure: Previously $\mathcal{H}$ was already positioned closer to $\mathcal{R}$. At this stage $\lambda = 0.75$ and $\triangle\lambda = 0.125$. We send the box $\mathcal{B}_x^1 = \Box(\mathcal{B}_x^0 \cap \mathcal{H}^+)$ to y-space and compute $\mathcal{B}_y^1 = \sigma(\mathcal{B}_x^1)$, which is illustrated on the left hand side.



(b) As depicted on the left hand side, we calculated in y-space $\mathcal{B}_y^2 = \Box(\mathcal{B}_y^1 \cap \mathcal{P}_y)$. After calculating the new box $\mathcal{B}_x^2 = \sigma^{-1}(\mathcal{B}_y^2)$, we detected that there is no intersection between the half-space $\mathcal{H}^+$ and the region $\mathcal{R}$, because $\mathcal{B}_x^2$ is completely in $\mathcal{H}^-$. This means: $\mathcal{B}_x^3 = (\mathcal{B}_x^2 \cap \mathcal{H}^+) = \emptyset$ and therefore: $\mathcal{H}$ can be moved again closer towards $\mathcal{R}$.

Figure 1: Example for the refinement process.

The refinement algorithm relies on the following two lemmas:
Lemma 1: $\mathcal{B}_x \cap \sigma^{-1}(\mathcal{P}_y) \neq \emptyset$ iff $\sigma(\mathcal{B}_x) \cap \mathcal{P}_y \neq \emptyset$
Lemma 2: $\Box(\mathcal{B} \cap \mathcal{R}) \subseteq \mathcal{B}$, where $\mathcal{B}$ is a box and $\mathcal{R}$ is an arbitrary region.
We start with the box $\mathcal{B}_x^1 = \Box(\mathcal{B}_x^0 \cap \mathcal{H}^+)$, i.e. $\mathcal{B}_x^1$ is the box of the intersection of the wrapping hypercube $\mathcal{B}_x^0$ and the half-space $\mathcal{H}^+$. If this box intersects

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 463-468

the region $\mathcal{R}$ then the corresponding box $\mathcal{B}_y^1 = \sigma(\mathcal{B}_x^1)$ intersects $\mathcal{P}_y$ in y-space (see Lemma 1). In y-space we determine the box of the intersection between $\mathcal{B}_y^1$ and $\mathcal{P}_y$, i.e. $\mathcal{B}_y^2 = \Box(\mathcal{B}_y^1 \cap \mathcal{P}_y)$. $\mathcal{B}_y^2$ is either (see Lemma 2) an empty box, i.e. $\mathcal{B}_y^1 \cap \mathcal{P}_y = \emptyset$, an unchanged box, i.e. $\Box(\mathcal{B}_y^1 \cap \mathcal{P}_y) = \mathcal{B}_y^1$ or a refined box, i.e. $\Box(\mathcal{B}_y^1 \cap \mathcal{P}_y) \subset \mathcal{B}_y^1$. We send the refined box back to x-space and have again this three cases when intersecting the box with $\mathcal{H}^+$. In the third case we have to repeat the refinement process if the volume of the refined box (in x-space) is bigger than a predefined small value ϵ, for all other cases we can stop the refinement process (note that a box is considered as unchanged, if the volume of the set difference between two consecutive boxes, denoted as $V(\mathcal{B}_x^k \Delta \mathcal{B}_x^{k+1})$, is less than a small value ϵ).

$$\mathcal{B}_x = \text{refine}(\mathcal{B}_x^0, \mathcal{B}_y^0, \mathcal{P}_y, \mathcal{H})$$

$k = 0; \mathcal{B}_x^{k+1} = \Box(\mathcal{B}_x^k \cap \mathcal{H}^+)$

do

$\lceil //forward, i.e. \quad X \to Y$

$k = k + 1; \quad \mathcal{B}_y^k = \sigma(\mathcal{B}_x^k); \quad \mathcal{B}_y^{k+1} = \Box(\mathcal{B}_y^k \cap \mathcal{P}_y);$

$//backward, i.e. \quad Y \to X$

$k = k + 1; \quad \mathcal{B}_x^k = \sigma^{-1}(\mathcal{B}_y^k); \quad \mathcal{B}_x^{k+1} = \Box(\mathcal{B}_x^k \cap \mathcal{H}^+);$

$\lfloor$**while**$(V(\mathcal{B}_x^{k+1}) \neq 0 \quad \wedge \quad V(\mathcal{B}_x^k \Delta \mathcal{B}_x^{k+1}) > \epsilon \quad \wedge \quad V(\mathcal{B}_x^{k+1}) > \epsilon)$

$\mathcal{B}_x = \mathcal{B}_x^{k+1};$

3 Empirical Results

We tested the binary search algorithm to position a single hyperplane close to the region $\mathcal{R} = \sigma^{-1}(\mathcal{P}_y) = \{\mathbf{x} | \mathbf{A}\sigma(\mathbf{x}) \leq \mathbf{b}\}$ for randomly constructed polyhedra (approximately 100 per dimension) and randomly chosen direction vectors $\mathbf{g}$'s. Before using a binary search method, we tried a branch and bound approach to solve the optimization problem:

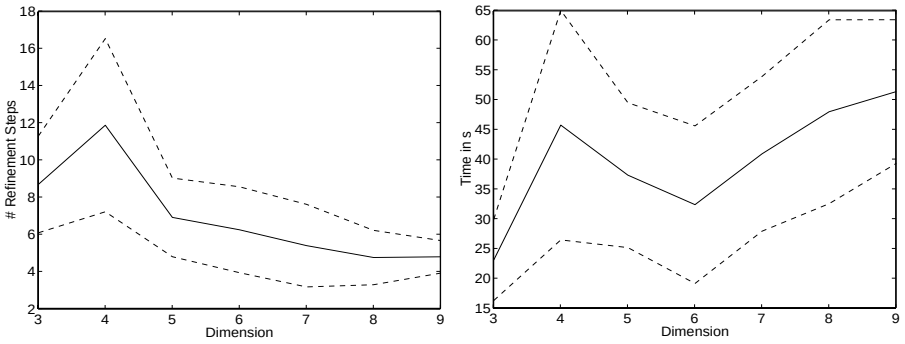
$$max \quad \mathbf{g^T x} \qquad \text{subject to} \qquad \mathbf{A}\sigma(\mathbf{x}) \leq \mathbf{b}$$

We compared the binary search algorithm and the branch and bound method with the same data. To assure comparable solutions between the two methods the binary search algorithm stopped once the solution was better or similar to the corresponding branch and bound solution. The table below contains the 95% confidence intervals of the computation times.

Dimension	Branch and Bound, time in s.	Binary search, time in s.
3	[24.1 , 30.8]	[16.1 , 29.8]
4	[58.9 , 81.9]	[26.3 , 64.9]
5	[98.8 , 151.4]	[25.0 , 49.4]
6	[150.8 , 243.4]	[19.0 , 45.5]
7	[277.5 , 464.6]	[27.8 , 53.8]
8	[569.8 , 1195.3]	[32.4 , 63.3]
9	[945.2 , 2147.4]	[39.2 , 63.3]

ESANN'2003 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 463-468

In Figure 2 we plotted the number of refinement steps and the time in seconds together with the lower and upper bounds of the 95% confidence interval.



(a) Average number of refinement steps.

(b) Average time in s.

Figure 2: Average number of refinement steps and time complexity.

4 Conclusion and future work

Given a set of polyhedral constraints on the output space of a feed-forward neural network our algorithm produces an approximation of the corresponding true reciprocal image in the input space. We approximate the true reciprocal image for the non-linear phase from outside, i.e. the true reciprocal image is completely contained in the polyhedral approximation. Empirical results showed that the proposed binary search method scales well with the dimension, and in addition, the proposed binary search strategy can be applied to any invertible function.
Future research will include the forward-propagation of a polyhedron through a neural network, which would allow interface assertions of the form: "if $\mathbf{x} \in \mathcal{R}_i$ then $f(\mathbf{x}) \in \mathcal{R}_o$". We also work on better refined approximations of a region $\mathcal{R}$ by using multiple polyhedra instead of a single polyhedral approximation.

References

[1] F. Maire. Rule-extraction by backpropagation of polyhedra. *Neural networks*, 12:717–725, 1998.

[2] S. B. Thrun. Extracting Provably Correct Rules from Artificial Neural Networks. Technical Report IAI-TR-93-5, Department of Computer Science III, University of Bonn, 1993.

[3] A. Tickle, R. Andrews, Mostefa Golea, and J. Diederich. The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6):1057–1068, 1998.