Neural Networks for High-Resolution State Evaluation in General Game Playing

Daniel Michulke
Department of Computer Science
Dresden University of Technology
daniel.michulke@mailbox.tu-dresden.de

Abstract

$C-IL^2P$ is an algorithm that transforms a propositional domain theory to a neural network that correctly represents the domain theory and is ready-to-use without prior training. Its original intention was to transform explicit symbolic knowledge into a neural network to allow for learning.

The game playing agent system presented in (Michulke and Thielscher 2009) uses the algorithm differently: By transforming the symbolic description of the goal of a game to a neural network it obtains an evaluation function for states of that game. Much like fuzzy logic, the network can be used for graded inference while retaining correctness. But in contrast to fuzzy logic, the network is able to learn and may consequently improve with experience, which is unique among competing agents and arguably an important advantage in a game playing setting. However, since not intended for this use, the transformation algorithm produces networks that cannot correctly represent complex domain theories without losing their ability to distinguish some input vectors that ought to have a different evaluation.

In this paper we introduce a generalization of $C-IL^2P$ that addresses the above issue. It structures the formerly monolithic approach of logic-to-network transformation to allow for lower weights in the network. This increases the output resolution by several orders of magnitude, as experiments demonstrate, while maintaining correctness.

1 Introduction

$C-IL^2P$ (d'Avila Garcez and Gabbay 2002) is an algorithm that transforms a set of propositional rules to a neural network. In this way, it is possible to create a neural network from a propositional domain theory without prior training or search for good initialization parameters. At the same time the neural network correctly represents the domain theory and can be trained using standard algorithms such as backpropagation.

A different application of such a network is as an inference tool that works on real numbers representing truth values. The network input is a vector of truth values and the network evaluates to what extent the domain theory holds on the input. This approach is used in (Michulke and Thielscher 2009) to generate a state evaluation function for a general game playing agent. The propositionalized goal of the game (as stated in a game rules) is considered the domain theory and a game state the input. The output of the network for any given state corresponds to the output of the goal definition of the game and is therefore useful for distinguishing goal states from non-goal states. Moreover, the output is at same time monotonic, meaning that states that correspond more to the goal of the game produce a higher output. The network thus extrapolates from goal states to non-goal states and is a good evaluation function that can be used to guide search to the most favorable among the states searched.

Additionally, it can also be trained, e.g. using standard backpropagation, which especially in the game playing environment is an important advantage. However, in their paper they also state that $C-IL^2P$ sets tight constraints on weights in order to ensure correctness. A by-product of this is that the network loses its ability to distinguish states from each other under certain circumstances.

For this reason we present a generalization of $C-IL^2P$ that imposes much weaker constraints on the weight settings than its algorithmic predecessor. As a result, much more complex domain theories can be represented by the resulting network while at the same time the output of the network is descriptive (different states produce different outputs) and correct. We demonstrate the benefit of our new approach in the domain of General Game Playing where experiments in more than 160 games show that the ability to correctly distinguish states from each other increases by several orders of magnitude when compared to $C-IL^2P$.

Note that the approach is *not* a fuzzy inference engine in the strict sense. The constructed networks do, however, possess important properties such as correctness and monotonicity and therefore behave similar to fuzzy inference engines.

The remainder of this paper is organized as follows: In the next section we give the necessary background for this paper and show a simple case where a network constructed using $C-IL^2P$ cannot distinguish between different inputs. In section 3 we introduce a structured approach on rule-to-neuron transformation and, based on this, present an algorithmic description of our approach in 4. Finally, we demonstrate that our new approach improves the abilities of neural networks to represent complex domain theories(section 5) and conclude in section 6.

2 Background

2.1 State Evaluation in General Game Playing

In General Game Playing (GGP), the rules of a game are given minutes before the corresponding match starts. These rules are encoded in the Game Description Language (GDL, (Love et al. 2008)) and define the roles (e.g. white and black in Chess), initial state, legal moves, state transitions, terminal states and goals for each role. Agents thus cannot rely on any preprogrammed behaviors but have to autonomously develop a strategy for the game.

A core part of the strategy is the state evaluation function. It maps states to values and indicates how good a state is. In any state, the agent then takes the move that leads to the state with maximum state value. While there are several agents with different approaches on what to use as an evaluation function, the approach of (Schiffel and Thielscher 2007), called Fluxplayer, seems particularly intriguing as it does not exhibit the problems of other approaches such as assuming opponent random behavior, as e.g. in (Finnsson and Björnsson 2008), or time-consuming feature guessing and evaluation mechanisms, as in (Clune 2007).

Instead, it derives an evaluation function directly from the goal conditions of the game. These goal conditions are predicate logic formulas that are transformed by first ground-instantiating them and then evaluating the resulting propositional logic formulas using t-norm fuzzy logics. The approach is successful with Fluxplayer constantly being among the top 4 agents in the annual worldwide championship in GGP.

A similar approach by (Michulke and Thielscher 2009) uses neural networks instead of t-norm fuzzy logics, based on the idea that its ability to learn should prove an important edge against other agents, given that human-level intelligence builds on both, logic and learning. This article deals with the resolution problem outlined in their GGP agent system and shows how the problem's impact can be greatly reduced. While there are other possibilities to address the resolution problem (e.g. using high-precision arithmetic), we know of none that does not imply higher computational cost when used for evaluation. This, however, is prohibitive in a game playing setting where the time needed for state evaluation is critical property of the algorithm.

2.2 Neuro-Symbolic Integration

Neuro-Symbolic Integration (NSI) was originally intended to unify neural networks and symbolic representations of knowledge to combine the capability to learn of neural networks with the logic reasoning facilities of an explicit representation. It is basically concerned with encoding a domain theory into a network, training the network and possibly extracting knowledge afterwards again. This process constitutes a cycle that enables transforming both systems into each other, see (Bader and Hitzler 2005) for an overview.

In the light of our different application of NSI concepts, KBANN (Knowledge-Based Artificial Neural Network, (Towell, Shavlik, and Noordenier 1990)) is the only comparable algorithm. It transforms an "approximately correct" propositional domain theory to a unipolar neural network. Training was then used to correct the errors in the initial theory. When applied as classifier, KBANN learned faster and qualitatively performed better than other learning algorithms. However, the theoretical framework restricted its application to domain theories with only small numbers of rules and antecedents per rule.

$C - IL^2P$ (d'Avila Garcez and Gabbay 2002) addresses this problem using bipolar networks, but the networks still require weights too high to be used for correctly fuzzy-evaluating inputs while at the same time retaining the ability to distinguish among all possible inputs.

2.3 The $C - IL^2P$ algorithm

$C - IL^2P$ represents the propositional values for truth and falsity as intervals of the output o of a bipolar neuron. If a propositional variable is true, the corresponding neuron gives a value of $o \in [A_{min}, 1]$ while an output $o \in [-1, A_{max}]$ is interpreted as falsity. Outputs $o < -1$ or $o > 1$ lie outside of the co-domain of the activation function $h(x) = \frac{2}{1+e^{-x}} - 1$ while outputs in the non-empty interval (A_{max}, A_{min}) are guaranteed to not occur by restricting the weights of connections to the neuron (parameter W, equation (4)). Usually, $-A_{max} = A_{min}$ is set, allowing propositional negation to be encoded as arithmetic negation. For simplicity we describe the algorithm in a slightly modified form. $C - IL^2P$ transforms rules of the form

$$q \Leftarrow \bigotimes_{1 \leq i \leq k} p_i \text{ with } \bigotimes \in \{\bigwedge, \bigvee\} \tag{1}$$

where q is an atom and the p_i are positive or negative literals. A rule is represented by $k+1$ neurons where k neurons represent the literals p_i that have an outgoing connection to the $k+1st$ neuron representing the head. Negative literals are represented just as positive literals, but with their connection weight negated. In addition to these k connections, a connection to the bias unit (which has constant output 1) is added that works as a threshold θ. The weight of this connection is set depending on the number k of children and the operator $\bigotimes \in \{\bigwedge, \bigvee\}$:

$$\theta_{\bigwedge}(k) = -\theta_{\bigvee}(k) = \frac{(1 + A_{min}) * (1 - k)}{2} * W \tag{2}$$

The parameter A_{min} determines the truth threshold above which a neural output value represents truth and W the standard weight for any connection within the neural network. Both can be chosen freely, but are subject to the following restrictions:

$$1 > A_{min} > \frac{k_{max} - 1}{k_{max} + 1} \tag{3}$$

$$W \geq 2 * \frac{ln(1 + A_{min}) - ln(1 - A_{min})}{k_{max}(A_{min} - 1) + A_{min} + 1} \tag{4}$$

k_{max} is the maximum number of antecedents a rule has.

$C - IL^2P$ for Fuzzy Evaluation Due to the monotonicity of the activation function, an input that corresponds better to the domain theory also produces a higher output of the network. Given that the network at the same time correctly represents the domain theory, it can be used for fuzzy inference.

However, with the steepness $h'(x) = 1 - h(x)^2$ of the activation function approximating zero for high absolute values of x, neurons encoding a conjunction (disjunction) with few (many) fulfilled antecedents may produce the same network output as the following example demonstrates:

Example 1. Consider a neuron z representing a conjunction of the output of 4 preceding neurons $z \Leftarrow \bigwedge_{i=1}^{4} y_i$. We assume the maximal number of children of a node to be $k = 4$ and set $A_{min} = -A_{max} = 0.9 > 0.6$ and $W = 4 \geq 3.93$, fulfilling thus equations (3) and (4). The following table shows the neural activation $a_z = W * (\theta_\wedge(k) + t)$ and output $o_z = h(a_z)$ of neuron z for t of the 4 y_i representing *true* (having output 1).

true antecedents t	activation a_z	output o_z
0	-27.4	-1.000000
1	-19.4	-1.000000
2	-11.4	-0.999978
3	-3.4	-0.935409
4	4.6	0.980096

While the neuron correctly encodes a conjunction, it is not able to distinguish the cases where none or one antecedent is fulfilled. This is due to the resolution imposed (10^{-6}), but occurs similarly in all finite floating-point representations such as 32-bit computers. This lack of distinction becomes worse when the output values are propagated through the network.

The main reason for this behavior is the weight restriction in equation (4). The higher the absolute weights are, the more likely the activation of a neuron is far away from zero. This is a consequence of the global setting of W and A_{min} that depend on the maximum number of antecedents of a rule k_{max} and implies weights higher than necessary for nodes with $k < k_{max}$ children. Moreover, the minimum and maximum output values of each neuron are fixed to -1 and 1. Possible lower values are thus not considered for weight determination.

3 Local Neuron Transformation

We reduce these effects in our more general version of $C - IL^2P$. We begin by defining a *standard neuron*.

Definition 1 (Standard Neuron)**.** Let z be an input layer neuron with no predecessors and the output value $o_z \in O \subset [-1, 1]$ *OR* a neuron with

- a real weight w_z,
- a real bias $bias_z$,
- and the unbiased input $\hat{i}_z = \sum_{y \in pred(z)} o_y$ where $pred(z) \neq \emptyset$ is the non-empty set of preceding neurons to z,
- the biased input $i_z = \hat{i}_z + bias_z$,
- the bipolar activation function $h(x) = \frac{2}{1+e^{-x}} - 1$,
- the output $o_z = h(w_z * i_z)$.

Then we call z a standard neuron.

Like in $C - IL^2P$, we represent truth and falsity of a propositional variable by a neuron with an output value in a specified interval. However, instead of using the intervals $[-1, A_{max}]$ and $[A_{min}, 1]$ with A_{max} and A_{min} as global parameters, we generalize by parameterizing the minimum false and maximum true output value (-1 and 1 respectively) and by setting all parameters *locally*, that is, for each neuron individually. Let a propositional variable q be represented by a neuron z with output o_z. Then we define:

$$q \Leftrightarrow o_z \in [o_z^+, o_z^{++}] \qquad \neg q \Leftrightarrow o_z \in [o_z^{--}, o_z^-]$$

We will use the term *limits* to refer to the quadruple $(o_z^{--}, o_z^-, o_z^+, o_z^{++})$ of output parameters of a neuron z.

In order to interpret a neuron output value as a propositional variable, we must ensure that the output intervals for $true$ and $false$, are distinct, that is, they do not overlap $o_z^- < o_z^+$. We furthermore constrain propositional truth to positive values and falsity to negative values $o_z^- < 0 < o_z^+$. This will allow us later to define propositional negation as arithmetic negation.

We call a neuron fulfilling this constraint *representative*.

Definition 2 (Representative Neuron)**.** Let z be a standard neuron with the output value o_z.

Then we call z representative if the set of possible output values $O_z = \{o_z\}$ is identical to the set of real values $O_z = [o_z^{--}, o_z^-] \cup [o_z^+, o_z^{++}]$ and $o_z^- < 0 < o_z^+$.

With $o_z^{--} \leq o_z^-$ and $o_z^+ \leq o_z^{++}$ the two intervals of a representative neuron z are not empty: $[o_z^{--}, o_z^-] \neq \emptyset \neq [o_z^+, o_z^{++}]$. Along with the output range $[-1, 1]$ of the activation function the following inequalities hold:

$$-1 \leq o_z^{--} \leq o_z^- < 0 < o_z^+ \leq o_z^{++} \leq 1$$

Note that any neuron is representative regardless of its absolute weight: Since from $o_z^- < 0 < o_z^+$ follows $h(w_z * i_z^-) < 0 < h(w_z * i_z^+)$ and $h(x)$ is an odd function ($h(-x) = -h(x)$), a representative neuron remains representative if we change its current weight w_z to another weight w_z' as long as the sign of the weight is the same $sgn(w_z) = sgn(w_z')$. This enables us to set the weight freely without confusing truth values.

Our line of argumentation now goes as follows: Consider the rule $q \Leftarrow \bigotimes_{1 \leq i \leq k} p_i$ with $\bigotimes \in \{\bigwedge, \bigvee\}$. We assume that the positive form of all literals p_i has already been translated to the neurons y_i and that these neurons $\{y_i : 0 \leq i \leq k\}$ are representative. Then we create a neuron z that represents q by doing the following:

Output-Input Mapping map the output values o_y of all neurons y to the input value $\hat{i}_z$ such that $\hat{i}_z \in [\hat{i}_z^+, \hat{i}_z^{++}] \Leftrightarrow q$ and $\hat{i}_z \in [\hat{i}_z^{--}, \hat{i}_z^-] \Leftrightarrow \neg q$

Distinct Input increase the weights w_y of all neurons y such that $\hat{i}_z^- < \hat{i}_z^+$

Representative Neuron set the bias $bias_z$ such that z is representative

We obtain a complete representation of the rule without having to impose a constraint on the weight w_z. Instead, we set the weights w_y of the predecessors $y \in pred(z)$. The three steps will be explained subsequently.

Output-Input Mapping The output-input mapping defines how to derive the input limits of a neuron z from a set of output limits of its predecessors $y \in pred(z)$. These limits determine the input intervals that represent truth and falsity depending on the type of rule (conjunction or disjunction) and the output limits of the preceding neurons.

$$\hat{i}_z^{++} = \sum_{y \in pred(z)} o_y^{++} \qquad \hat{i}_z^{--} = \sum_{y \in pred(z)} o_y^{--} \tag{5}$$
$$\hat{i}_{z,and}^{+} = \sum_{y \in pred(z)} o_y^{+} \qquad \hat{i}_{z,and}^{-} = \hat{i}_z^{++} - \min_{y \in pred(z)} o_y^{++} - o_y^{-}$$
$$\hat{i}_{z,or}^{-} = \sum_{y \in pred(z)} o_y^{-} \qquad \hat{i}_{z,or}^{+} = \hat{i}_z^{--} + \min_{y \in pred(z)} o_y^{+} - o_y^{--}$$

The limits are calculated according to four scenarios: The maximum truth and minimum false values $\hat{i}_z^{++}$ and $\hat{i}_z^{--}$ are simply the sum of the corresponding output values and can be considered the *best-case* values as they are the values with the maximum possible distance to the ambiguous value 0.

The *worst-case* values depend on the operator: Conjunctions represent a worst-case truth if their predecessors represent worst-case truth. However, they become false if all predecessors give even the best-case truth value with the exception of one that gives the maximum false (worst-case false) value. The truth values for disjunctions are calculated similarly.

Finally, negation is represented by arithmetical negation. If z represents a rule where the neuron y is a negative antecedent then substitute y by a neuron y' with the negated output limits:

$$o_{y'}^{++} = -o_y^{--} \qquad o_{y'}^{+} = -o_y^{-} \tag{6}$$
$$o_{y'}^{--} = -o_y^{++} \qquad o_{y'}^{-} = -o_y^{+}$$

Note that neither weight nor bias of z are set at this stage but the propositional meaning defined instead.

Since the resulting intervals may overlap, i.e. $\hat{i}_z^- < \hat{i}_z^+$ cannot be guaranteed, we "separate" the two intervals by setting the weights w_y in the following step.

Distinct Input We show that a weight w_y for the predecessors y of a neuron z exists such that the input of z is distinct. We use the parameter $\Delta_z \stackrel{!}{=} \hat{i}^+ - \hat{i}^-$ to refer to the difference by which the input intervals of z are distinct. For simplicity, we assume $-i^- = i^+$, consequently it holds $-o^- = o^+$. This assumption will be justified in theorem 2.

Theorem 1. Let z be a standard neuron representing a conjunction or disjunction of its predecessors $y \in pred(z)$ as given in equation (5). Furthermore, let all predecessors be representative standard neurons with $-i^- = i^+$. Then for an arbitrary $\Delta_z < 2$ there exists a weight w_y for all neurons y such that the input of z is distinct by at least Δ_z, i.e. $\hat{i}_z^+ \geq \hat{i}_z^- + \Delta_z$.

$$w_y \geq -\frac{1}{i_{min}^+} ln \frac{2 - \Delta_z}{\Delta_z + 2k} \tag{7}$$

min refers to the neuron that has the minimum worst-case truth value i_{min}^+ of all neurons $y \in pred(z)$.

Proof for Conjunctive Case (Sketch).

$$\begin{aligned}
\hat{i}_z^+ &\stackrel{!}{\geq} \hat{i}_z^- + \Delta_z \\
\sum_{y \in pred(z)} o_y^+ &\geq \sum_{y \in pred(z)} o_y^{++} - \min_{y \in pred(z)} (o_y^{++} - o_y^-) + \Delta_z \\
k * \min_{y \in pred(z)} o_y^+ &\geq k - \min_{y \in pred(z)} (1 + o_y^+) + \Delta_z \\
(k+1) * (o_{min}^+ - 1) &\geq \Delta_z - 2 \\
o_{min}^+ &\geq \frac{\Delta_z - 2}{k+1} + 1 \\
w_{min} &\geq -\frac{1}{i_{min}^+} ln \frac{2 - \Delta_z}{\Delta_z + 2k}
\end{aligned}$$

Since i_{min}^+ is smallest by definition, the weight bounds of any neuron $y \in pred(z)$ is less or equal to w_{min}. □

The proof is similar for the disjunctive case. The result is the worst-case lower bound for the weights w_y such that the input of z is distinct by at least Δ_z. In extreme cases, this bound corresponds to the weight bound in $C - IL^2P$. However, our algorithm calculates these parameters locally and benefits thus from any situation that does not represent a worst-case.

Besides, the result shows that $2 > \Delta$ is a precondition for all successor neurons, ensuring thereby that Δ is smaller than the maximum possible output difference of a neuron $o^{++} - o^{--} = 2$.

Representative Neurons Now we just set the bias such that z itself is representative.

Theorem 2. Let z be a standard neuron with distinct input. By setting

$$bias_z = \frac{-\hat{i}_z^- - \hat{i}_z^+}{2} \tag{8}$$

z is representative.

Proof(Sketch). It holds $i_z^- = -i_z^+ \neq 0$. Since the activation function $h(x)$ is odd and strictly monotonically increasing for $w > 0$, the input interval directly translates to a representative output interval. □

Note that by setting the bias in between the limits $\hat{i}^-$ and $\hat{i}^+$, we obtain $-i^- = i^+$ and $-o^- = o^+$.

4 Transformation Algorithm

A basic algorithm can now be constructed that translates a rule of the form $q \Leftarrow \bigotimes_{1 \leq i \leq k} p_i$ with $\bigotimes \in \{\bigwedge, \bigvee\}$ to a neuron. We consider the p_i already transformed to the neurons y_i. Thus their biased input limits (i_y) are set while the weights w_y are not. The algorithm sets these weights w_y for each y and the input limit $\hat{i}_z$ by doing the following:

1. calculate the output limits o_y of each predecessor neuron $y \in pred(z)$ for a weight w_y

2. calculate the input limits $\hat{i}_z$ (equation (5))
3. if the input limit is distinct then calculate the bias (equation (8)), else increase w_y and return to step 1

The algorithm can be applied to generate a complete network by calling it on a set of rules. When transforming to a neuron z, it sets the $bias_z$ and the input limits i_z together with the weights w_y of its preceding neurons. The transformation works thus simultaneously on two levels of the network and we must pay attention to the input layer and output layer neurons. Since output layer neurons have no successor, they need not guarantee an output distinct by Δ and their weight can thus be set to $w = 1$. On the other hand, input layer neurons need no bias and weight settings as they have well-defined output values.

4.1 Soundness

Since all input layer neurons are defined such that they are representative and we set the bias of each neuron such that it is representative it holds that every neuron is representative. To show that the transformation of a neuron is sound, we show that the output of a neuron z correctly represents a conjunction or disjunction of the variables represented by its predecessor neurons.

Theorem 3. If the neurons y_i represent the variables p_i in the rule $q \Leftarrow \bigotimes_{1 \leq i \leq k} p_i$ with $\bigotimes \in \{\bigwedge, \bigvee\}$ then we can construct a neuron z such that it represents q.

Proof. Due to the monotonicity of the activation function, we limit the proof to border cases. The proof for the disjunction is omitted as it is similar.

We show that if all predecessor neurons represent $true$, the conjunction neuron z also represents $true$: $\forall y \in pred(z) : o_y \geq o_y^+ \rightarrow o_z \geq o_z^+$. It follows:

$$\sum_y o_y \geq \sum_y o_y^+ = \hat{i}_z^+$$
$$\hat{i}_z \geq \hat{i}_z^+$$

As all neurons y are representative, we set a positive weight such that $\hat{i}_z^+ > \hat{i}_z^-$, and from monotonicity of the activation function $h(x)$ follows that $o_z \geq o_z^+$.

If one predecessor neuron $y*$ represents $false$, the conjunction neuron z also represents $false$: $\exists y* \in pred(z) : o_{y*} \leq o_{y*}^- \rightarrow o_z < o_y^-$. The maximum unbiased input $\hat{i}_{z,max}$ is then

$$\hat{i}_{z,max} = \sum_{y \in pred(z)} o_y^{++} - o_{y*}^{++} - o_{y*}^-$$
$$\hat{i}_{z,max} \leq \sum_{y \in pred(z)} o_y^{++} - \min_{y \in pred(z)} o_y^{++} - o_y^- = \hat{i}_z^-$$

Again, we set a positive weight w_y such that $\hat{i}_z^+ > \hat{i}_z^-$, and from monotonicity follows that $o_z \leq o_z^-$. □

4.2 Open Problems

When transforming a graph to a network, the weights of a node y are processed by all of its successors and might be increased. If a node z sets the weight w_y of one of his predecessors and later a sibling z' of z increases this weight w_y, the absolute output values of y are increased and might decrease the difference between i_z^- and i_z^+ under certain circumstances.

There is no straightforward way of handling this problem due to the setting of w_y and the input limits i_z for each neuron z. The neuron transformation therefore implicitly ranges over two levels of the neural network. The simplest way to handle the scenario is to recalculate the neuron parameters for all previously transformed parent nodes of y and their successors if the weight w_y is increased. This can be time-consuming in the worst-case. Therefore we propose a heuristics that transforms the rules with the highest number of antecedents first as they are more likely to lead to higher weights. In addition, if the propositional domain theory has a layered structure (e.g. when in Disjunctive Normal Form), it is possible to transform it layer by layer, starting with the lowest. Once a layer is transformed, the weights of the preceding layer need not be recomputed anymore.

A second issue is that, unlike the original $C - IL^2P$, our approach is not directly designed to work on cyclic domain theories such as $a \Leftarrow b, b \Leftarrow a$. Although we consider such a scenario unlikely, cyclic domains do not invalidate our theoretical foundation of a rule-to-neuron transformation. Instead, the top-level algorithm would have to be modified, and one might e.g. opt to start from a $C - IL^2P$ constructed network and gradually reduce weights and see whether neurons remain representative and their input distinct by Δ_z.

In these scenarios, however, we consider it unlikely that resolution and the capability to distinguish between different input vectors play an important role. However, in case they do, simple modifications to the transformation algorithm

5 Experiments

While having proved that our algorithm is correct and as such suitable for translating a set of propositional rules to a neural network, we now demonstrate that it gives a higher output resolution than standard $C - IL^2P$. We do this in the domain of General Game Playing since it is our intended domain of application. Recall that due to the problem outlined in Example 1 different input vectors (in GGP: game states) are mapped to the same value, though one state fulfills more propositions of the domain theory (in GGP: the goal function) than the other.

Therefore we tested the networks generated by standard $C - IL^2P$ and by our generalized version on (by the time of the evaluation) 197 valid game descriptions submitted to the GGP server at Dresden University of Technology[1]. 36 of these games have a trivial goal condition without conjunctions or disjunctions, consequently no network can be constructed. We transformed the goal description of the remaining 161 games to a propositional domain theory using

[1] http://euklid.inf.tu-dresden.de:8180/ggpserver

the same techniques as in (Michulke and Thielscher 2009). We then transformed this domain theory to neural networks and evaluated all states of the game that are reachable from the initial state by one move. Our hypotheses are that 1. the networks constructed with our algorithm evaluate these states to a bigger output interval than $C - IL^2P$ and 2. that our approach can distinguish more of these states from each other. The latter is a direct consequence of the first hypothesis as a higher resolution implies a better treatment of situations as given in example 1.

To avoid superposition of outputs of different networks in the evaluation function, we evaluate a state by using only the output of the network of the first role in the game description and only for the goal condition winning the game. In this way, we keep results comparable for games that have a higher number of roles and goals. Figure 5 depicts our evaluation results. The x-axis represents the number of the game name in alphabetical order. Though game names have no linear relationship, we chose a line chart as different variants of the same game often have an alphabetically similar name.

Bigger Output Interval In the lower part of figure 5 we see the logarithm (base 10) of the size of the output interval of all 161 nontrivial games. We determine the size of the output interval by evaluating all states of depth 1 with our network and subtracting the minimum value from the maximum value obtained on these states. If minimum and maximum are equal, we consider (for visual reasons) the size of the output interval to be the machine precision, that is $2^{-53} \approx 10^{-16}$ for 64 bit floating-point precision. Since the network gives an output in the interval $[-1, 1]$, the logarithm is mostly negative. We can see that the resolution of our version (gray dotted line) is higher in 81 games and equal in 77 games to that of $C - IL^2P$. In three games, the precision is lower which is a side-effect of our lowest-weight approach in networks that consists of exactly one rule. In these cases, however, we could set an arbitrary weight, since our approach does not impose any constraints on the weight of connections to the output neuron.

The 77 games with equal resolution are games where no atomic part of the goal is affected by the first move. An example is the game Checkers where the winner is determined in terms of the number of pieces of both players. As it is impossible to remove any stone in the first move, all states are evaluated to the same value.

The logarithmic output size over all games increases in average from -12.92 to -10.13, while for all games with an output size bigger than machine precision it increases from -9.91 to -4.38. Since both values are biased, we roughly expect our algorithm to halve the logarithmic output size, that is, increase the size of the output interval of a neuron to the size of its square root.

Number of Distinct States The upper part of figure 5 shows the number of states that were evaluated to distinct values. The x-axis is aligned with the lower figure and we can see that in 41 of the cases with a higher resolution, also a higher number of states with distinct state value was identified. Since our algorithm is correct, we can conclude that in these 41 games our evaluation can now distinguish states it could not distinguish before.

6 Summary

We have presented a generalization of $C - IL^2P$ algorithm that correctly represents a set of propositional rules as neural network. It structures the formerly monolithic transformation process of a propositional rule to a neuron into several smaller parts and allows for potentially lower weights. This proves advantageous when using the algorithm to create a neural network that must correctly represent a domain theory and at the same time be able to distinguish between several input vectors. As an example we showed that the evaluation function of a game-playing agent can correctly fuzzy-evaluate and distinguish states for even complex games with an increase in output resolution by several orders of magnitude. The main disadvantage of the algorithm is its higher worst-case run-time requirements. However, for GGP these aspects do not play a significant role due to the layered structure of propositionalized goal conditions. Moreover, the proposed heuristic reduces the probability of a neuron parameter recalculation.

6.1 Future Work

With the approach presented we can derive an evaluation function for General Game Playing based on neural networks. If further enhanced with features, the evaluation function should be able to compete with state-of-the-art systems, given that the typical problems of neural networks (no initialization, low resolution) are addressed. Specifically, the evaluation function should outperform in setups that allow for learning since, unlike other agents, it builds on logic and learning. Currently, however, such a setup is not part of the GGP championship.

Neural networks also implicitly solve the feature weighting problem: Equipped with a feature generation mechanism, features can be inserted in the network and after enough training episodes the connection weights indicate the utility of the feature. A feature with near-zero weights can consequently be removed. Such a feature addition and subtraction mechanism combined with a reasonable evaluation function would also allow for self-play and function evolution.

On the technical level, with parameter Δ fixed to 1, the chance to adapt the local Δ-values is not used for further maximization of the output resolution. An interesting possibility is also to use first-order logic as domain theory as considered in (Bader and Hitzler 2005).

References

Bader, S., and Hitzler, P. 2005. P.: Dimensions of neural-symbolic integration - a structured survey. In *We Will Show Them: Essays in Honour of Dov Gabbay*, 167–194. College Publications.

Clune, J. 2007. Heuristic evaluation functions for general game playing. In *AAAI*. Vancouver: AAAI Press.

d'Avila Garcez, A. B. K. B., and Gabbay, D. 2002. *Neural-Symbolic Learning Systems*. Springer.

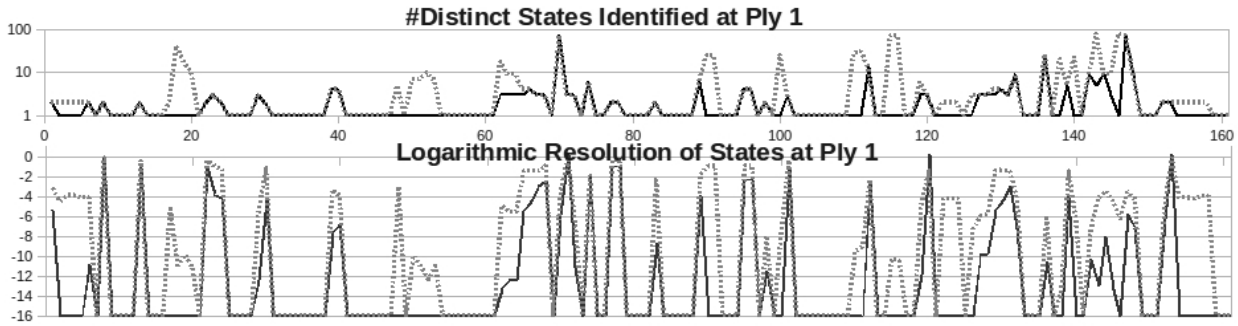

Figure 1: Above: Number of Distinct States Found; Below: Logarithmic Resolution of States

Finnsson, H., and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *AAAI*. AAAI Press.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2008. General game playing: Game description language specification. Technical Report March 4. The most recent version should be available at http://games.stanford.edu/.

Michulke, D., and Thielscher, M. 2009. Neural networks for state evaluation in general game playing. In *ECML PKDD '09: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, 95–110. Berlin, Heidelberg: Springer-Verlag.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In *Proceedings of the National Conference on Artificial Intelligence*, 1191–1196. Vancouver: AAAI Press.

Towell, G. G.; Shavlik, J. W.; and Noordenier, M. O. 1990. Refinement of approximate domain theories by knowledge based neural network. volume 2, 861–866.