City Research Online

City St George's, University of London

Citation: Garcez, A., Gabbay, D. M. & Lamb, L. C. (2004). Argumentation Neural Networks: Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems (TR/2004/DOC/01). .

This is the published version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/4062/

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

Argumentation Neural Networks: Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems

A.S. d'Avila Garcez D.M. Gabbay L.C. Lamb

Department of Computing

City University

Technical Report Series

TR/2004/DOC/01

ISSN 1364-4009

Argumentation Neural Networks: Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems

A. S. d'Avila Garcez [1] and **D. M. Gabbay** [2] and **L. C. Lamb** [3]

Abstract. While neural networks have been successfully used in a number of machine learning applications [15, 11], logical languages have been the standard for the representation of legal and argumentative reasoning [3, 9]. In this paper, we present a new hybrid model of computation that allows for the deduction and learning of argumentative reasoning. We do so by using Neural-Symbolic Learning Systems, where non-classical reasoning is representable. We propose a *Neural Argumentation Algorithm* to translate argumentation networks into standard neural networks. We then show a correspondence between the semantics of the two networks. The algorithm works not only for acyclic argumentation networks but also for circular networks. The approach enables cummulative argumentation through learning, as the strength of the arguments change over time. **Keywords**: Neural-Symbolic Systems, Value-based Argumentation Frameworks, Hybrid Systems, Argumentation.

1 Introduction

Neural-Symbolic integration concerns the application of problem-specific symbolic knowledge within the connectionist paradigm [4]. While neural networks have been successfully used in a number of machine learning applications [15, 11], logical languages have been the standard for the representation of legal and argumentative reasoning [3, 9]. In this paper, we present a new hybrid model of computation that allows for the deduction and learning of argumentative reasoning [2]. We do so by using Neural-Symbolic Learning Systems where nonmonotonic reasoning and a number of other non-classical logics are representable [6]. Neural-Symbolic Learning Systems use simple, single hidden layer neural network structures to represent and learn nonmonotonic, epistemic or temporal symbolic knowledge with the use of off-the-shelf neural learning algorithms [7, 8].

Our goal is to facilitate learning capabilities in value-based argumentation frameworks, as arguments may evolve over time, with certain arguments being strengthened and others weakened. Although we do not perform learning experiments in this paper, we show how the object language of Bench-Capon's argumentation networks can be translated into standard neural networks for learning. The networks are set up by a *Neural Argumentation Algorithm,* introduced in this paper. The proof that the neural network computes the stable model semantics of the logic program associated with the argumentation network is then given, thus showing that the two representations are equivalent. The algorithm works not only for acyclic argumentation networks but also for circular networks. As a result, we provide a way to tackle the problem of circularity in arguments by offering learning as an alternative to enable multi-part, cummulative argumentation. As new information comes along, and arguments in the cycle are strengthened, the circularity is overcome.

In Section 2, we briefly present the basic concepts of neural-symbolic systems used throughout this paper. In Section 3, we introduce the *Neural Argumentation Algorithm* and prove that the neural network computes the stable models of the given argumentation network, thus proving the correctness of the algorithm. Section 4 concludes the paper and discusses directions for future work.

2 Neural-Symbolic Learning Systems

An artificial neural network is a directed graph. A unit in this graph is characterised, at time t, by its input vector $I_i(t)$, its input potential $U_i(t)$, its activation state $A_i(t)$, and its output $O_i(t)$. The units (neurons) of the network are interconnected via a set of directed and weighted connections. If there is a connection from unit i to unit j, then $W_{ji} \in \mathbb{R}$ denotes the *weight* associated with such a connection.

The *activation state* of a neuron i at time t ($A_i(t)$) is a bounded real or integer number. The input potential of neuron i at time t ($U_i(t)$) is obtained by computing a weighted sum for neuron i such that $U_i(t) = \sum_j W_{ij}x_j(t)$, where $x_j(t)$ is the input signal from neuron j to neuron i at time t, and W_{ij} denotes the weight vector $(W_{i1}, W_{i2}, ..., W_{in})$ to neuron i. In addition, θ_i (an extra weight with input always fixed at 1) is known as the *threshold* of neuron i. The neuron's new activation state $A_i(t + \Delta t)$ is given by its *activation rule* h_i, which is a function of the neuron's current activation state and input potential, i.e. $A_i(t + \Delta t) = h_i(A_i(t), U_i(t))$. The neuron's output value $O_i(t + \Delta t)$ is given by $O_i(t + \Delta t) = f_i(A_i(t + \Delta t))$. Usually, f_i is the identity function.

The units of a neural network can be organised in layers. A *n-layer feedforward network* N is an acyclic graph. N consists of a sequence of layers and connections between successive layers, containing one input layer, $n - 2$ hidden layers and

[1] Department of Computing, City University London, UK
[2] Department of Computer Science, King's College London, UK
[3] Instituto de Informatica, UFRGS, Porto Alegre, Brazil

one output layer, where $n \geq 2$. When $n = 3$, we say that N is a *single hidden layer network*. When each unit occurring in the *i-th* layer is connected to each unit occurring in the $i + 1$-*st* layer, we say that N is a *fully-connected network*.

Let r and s be the number of units occurring in the input and output layer, respectively. A multilayer feedforward network N computes a function $f : \mathbb{R}^r \rightarrow \mathbb{R}^s$ as follows. The input vector is presented to the input layer at time t_1 and propagated through the hidden layers to the output layer. At each time point, all units update their input potential and activation state synchronously. At time t_n the output vector is read off the output layer. In addition, most neural models have a *learning rule*, responsible for changing the weights of the network so that it learns to approximate f given a number of *training examples* (input vectors and their respective target output vectors).

C-ILP [6] is a massively parallel computational model based on an artificial neural network that integrates inductive learning from examples and background knowledge with deductive learning from logic programming. Following [12] (see also [13]), a *Translation Algorithm* maps a logic program $\mathcal{P}$ into a single hidden layer neural network $\mathcal{N}$ such that $\mathcal{N}$ computes the least fixed-point of $\mathcal{P}$. This provides a massively parallel model for computing the widely used stable model semantics of $\mathcal{P}$ [10]. In addition, $\mathcal{N}$ can be trained with examples using a neural learning algorithm [16], having $\mathcal{P}$ as background knowledge. The knowledge acquired by training can then be extracted [5], closing the learning cycle [17].

Let us exemplify how C-ILP*'s Translation Algorithm* works. Each rule (r_l) of $\mathcal{P}$ is mapped from the input layer to the output layer of $\mathcal{N}$ through one neuron (N_l) in the single hidden layer of $\mathcal{N}$. Intuitively, the *Translation Algorithm* from $\mathcal{P}$ to $\mathcal{N}$ has to implement the following conditions: (c_1) The input potential of a hidden neuron (N_l) can only exceed N_l's threshold (θ_l), activating N_l, when all the positive antecedents of r_l are assigned the truth-value *true* while all the negative antecedents of r_l are assigned $false$; and (c_2) The input potential of an output neuron (A) can only exceed A's threshold (θ_A), activating A, when at least one hidden neuron N_l that is connected to A is activated.

Example 1 *(*C-ILP*) Consider the logic program* $\mathcal{P} = \{BC \sim D \rightarrow A; EF \rightarrow A; \rightarrow B\}$ *where* $\sim$ *stands for* default negation *[14]. The* Translation Algorithm *derives the network* $\mathcal{N}$ *of Figure 1, setting weights* ($W's$) *and thresholds* ($\theta's$) *in such a way that conditions* (c_1) *and* (c_2) *above are satisfied. Note that, if* $\mathcal{N}$ *ought to be fully-connected, any other link (not shown in Figure 1) should receive weight zero initially. Each input and output neuron of* $\mathcal{N}$ *is associated with an atom of* $\mathcal{P}$*. As a result, each input and output vector of* $\mathcal{N}$ *can be associated with an interpretation for* $\mathcal{P}$*. Note also that each hidden neuron* N_l *corresponds to a rule* r_l *of* $\mathcal{P}$*. In order to compute the stable models of* $\mathcal{P}$*, output neuron B should feed input neuron B such that* $\mathcal{N}$ *is used to iterate the fixed-point operator of* $\mathcal{P}$ *[6].* $\mathcal{N}$ *will eventually converge to a stable state which is identical to the stable model of* $\mathcal{P}$ *provided that* $\mathcal{P}$ *is an acceptable program [1].*

In the case of argumentation networks it will be sufficient to consider definite logic programs (i.e. programs without $\sim$). In this case, the neural network will contain only positive weights (W). We will then expand such a positive network to represent attacks using negative weights from the network's hidden layer to its output layer.

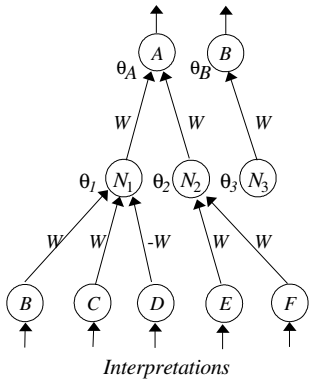

Figure 1: A neural network for program $\mathcal{P}$.

In order to use the network $\mathcal{N}$ of Figure 1 as a massively parallel model for Logic Programming, we just have to follow two steps: (i) add neurons to the input and output layers of $\mathcal{N}$, allowing it to be recurrently connected; and (ii) add the corresponding recurrent connections with fixed weight $W_r = 1$, so that the activation of output neuron A feeds back into the activation of input neuron A, the activation of output neuron B feeds back into the activation of input neuron B, and so on. For instance, given any initial activation in the input layer of $\mathcal{N}_r$ (network of Figure 1 recurrently connected), it always converges to the following stable state: $A = false, B = true, C = false, D = false, E = false$, and $F = false$, that represents the unique fixed-point of $\mathcal{P}$.

3 Argumentation Neural Networks

In the value-based argumentation framework defined in [2], argumentation networks are used to model arguments and counter-arguments. A typical example in the area is the following *moral debate* example.

Hal, a diabetic, loses his insulin in an accident through no fault of his own. Before collapsing into a coma, he rushes to the house of Carla, another diabetic. She is not at home, but Hal breaks into her house and uses some of her insulin. Was Hal justified? Does Carla have a right to compensation? The following are some of the arguments involved in the example.

A: Hal is justified, he was trying to save his life;
B: It is wrong to infringe the property rights of another;
C: Hal compensates Carla;
D: Hal is endangering Carla's life;
E: Carla has abundant insulin; and
F: If Hal is too poor to compensate Carla he should be allowed to take the insulin as no one should die because they are poor.

In [2], arguments and counter-arguments are arranged in an argumentation network, as in Figure 2, where an arrow from argument X to argument Y indicates that X attacks Y. For example, the fact that it is wrong to infringe Carla's right of property (**B**) attacks Hal's justification (**A**).

In the argumentation network of Figure 2, some aspects may change as the debate progresses and actions are taken, with the strength of an argument in attacking another changing in time. This is a learning process that can be implemented using a neural network in which the weights encode

the strength of the arguments. The neural network for the set of arguments {**A**, **B**, **D**} is depicted in Figure 3. The network is an auto-associative single hidden layer network with input (A,B,D), output (A,B,D) and hidden layer (h_1,h_2,h_3). Solid arrows represent positive weights and dotted arrows represent negative weights. Arguments are supported by positive weights and attacked by negative ones. Argument **A** (input neuron A), for example, supports itself (output neuron A) with the use of hidden neuron h_1. Similarly, argument **B** supports itself (via h_2), and so does argument **D** (via h_3). From the argumentation network, **B** attacks **A**, and **D** attacks **A**. The attacks are implemented in the neural network by the negative weights (see dotted lines in Figure 3) with the use of h_2 and h_3.

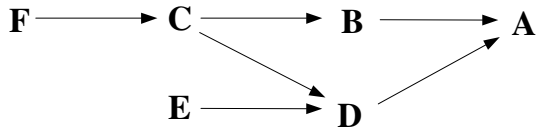

Figure 2: The moral debate argumentation network.

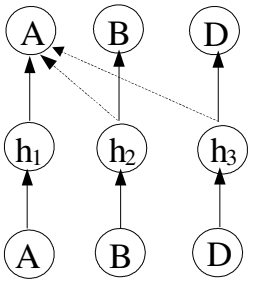

Figure 3: A neural network for arguments A, B, D.

The network of Figure 3 is a standard feedforward neural network that can be trained with the use of a standard neural learning algorithm. Training would change the initial weights of the network (the initial belief on the strength of arguments and counter-arguments), according to examples of input/output patterns, i.e. examples of the relationship between arguments **A**, **B** and **D**. If the absolute value of the weight from neuron h_1 to output neuron A is greater than the sum of the absolute values of the weights from neurons h_2 and h_3 to A, one could say that argument **A** prevails (in which case output neuron A should be *activated* in the neural network).

We shall implement the above behaviour using C-ILP networks. The *Neural Argumentation Algorithm* introduced below takes an argumentation network as input and produces a C-ILP neural network as output. These networks use a semi-linear activation function $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ and inputs in $\{-1, 1\}$ to enable effective learning.[4] This has to be taken into account by the algorithm. We do so by defining $A_{min} \in (0, 1)$ as the minimum activation for a neuron to be considered *active* (or *true*), and $A_{max} \in (-1, 0)$ as the maximum activation for a neuron to be considered *non active* (or *false*). We assume, for mathematical convenience and without loss of generality, that $A_{max} = -A_{min}$. Now, we need to define the values of A_{min}, $W's$ and $\theta's$ such that the neural network can compute the behaviour of the argumentation framework. The values are set by the algorithm, and come from the proof of Theorem 1, which shows that the neural network indeed computes the behaviour of the argumentation framework. The network can then be run, as exemplified in the sequel, to compute the prevailing arguments.

Definition 1 *An argumentation network has the form* $\mathcal{A} = < \alpha, attack, v >$, *where* α *is a set of arguments,* $attack \subseteq \alpha^2$ *is a relation indicating which arguments attack which other arguments, and* v *is a function from* $attack$ *to* $\{0, 1\}$, *which gives the relative strength of an argument. If* $v(\alpha_i, \alpha j) = 1$ *then* α_i *is said to be stronger than* α_j. *Otherwise,* α_i *is said to be weaker than* α_j.

Given a definite logic program $\mathcal{P}$, consider that the atoms of $\mathcal{P}$ are numbered from 1 to η such that the input and output layers of $\mathcal{N}$ are vectors of length η, where the i-th neuron represents the i-th atom of $\mathcal{P}$.

Neural Argumentation Algorithm

1. Given an argumentation network $\mathcal{A}$ with arguments $\alpha_1, \alpha_2, ..., \alpha_n$, make $\mathcal{P} = \{r_1 : \alpha_1 \rightarrow \alpha_1, r_2 : \alpha_2 \rightarrow \alpha_2, ..., r_3 : \alpha_n \rightarrow \alpha_n\}$;
2. Let $A_{min} > 0$, $W > 0$ and $W' < 0$;
3. Calculate $W \geq (1/\beta A_{\min}) \cdot (ln\,(1 + A_{\min}) - ln\,(1 - A_{\min}))$;
4. For each rule r_l of $\mathcal{P}$ $(1 \leq l \leq n)$:
   (a) Add a neuron N_l to the hidden layer of $\mathcal{N}$;
   (b) Connect neuron α_l in the input layer to N_l and set the connection weight to W;
   (c) Connect N_l to neuron α_l in the output layer and set the connection weight to W;
   (d) Set the threshold of all hidden and output neurons to zero;
5. Set $g(x) = x$ as the activation function of the neurons in the input layer of $\mathcal{N}$.[5]
6. Set $h(x)$ as the activation function of the neurons in the hidden and output layers of $\mathcal{N}$.[6]
7. For each $(\alpha_i, \alpha_j) \in attack$, do:
   (a) Connect hidden neuron N_i to ouptut neuron α_j;
   (b) If $v(\alpha_i, \alpha_j) = 0$ then
      i. Set connection weight $W' > h^{-1}(A_{\min}) - WA_{\min}$;
   (c) If $v(\alpha_i, \alpha_j) = 1$ then
      i. Set the connection weight $W' < (h^{-1}(-A_{\min})) - W)/A_{\min}$;
8. If $\mathcal{N}$ ought to be fully-connected, set all other connections to zero.

Note that the programs $\mathcal{P}$ obtained from an argumentation network will always have the form of a set of rules $r_i : \alpha_i \rightarrow \alpha_i$.

[4] A differentiable function such as $h(x)$ is necessary, e.g., if a gradient descent learning algorithm is to be used. The parameter β is responsible for defining the slope of the function. Typically, $\beta = 1$.

[5] In this way, the activation of the neurons in the input layer of $\mathcal{N}$, given by each input vector **i**, will represent an interpretation for $\mathcal{P}$.

[6] In this way, a gradient-based learning algorithm, such as *Back-propagation*, can be applied on $\mathcal{N}$.

As a result, differently from in the C-ILP *Translation Algorithm* [6], in which rules having more than one antecedent are accounted for, here there is always a single antecedent per rule (α_i). This allows us to use $A_{\min} > 0$ and $\theta = 0$ when using the algorithm to calculate W and W'. In addition, the fact that $W > 0$ and $W' < 0$ fits very well with the idea of arguments having strengths (W), and attacks also having strengths (W'). Of course, one could restrict W and W' to intervals. In the above algorithm, we leave this open to the user. The values of W and W' could, for example, be defined by an audience using any voting system [2]. In this system, at some point, an accumulation of attacks with different strengths - neither being individually stronger than the argument being attacked - might produce a value $\sum_i W'_i$ that overcomes W. This is naturally the way that neural networks work. All we need to make sure is that the neural network *computes* the prevailing arguments of an argumentation framework, according to the following definition.

Definition 2 *($\mathcal{N}$ computes $\mathcal{A}$) Let $(\alpha_i, \alpha_j) \in$ attack. We say that a neural network $\mathcal{N}$ computes the prevailing arguments of an argumentation framework $\mathcal{A}$ if (i) and (ii) below hold. (i) If α_i is stronger than α_j then output neuron α_j will not be activated when input neurons α_i and α_j are both activated, and (ii) If α_i is weaker than α_j then the activation of input neuron α_i will not be individually responsible for output neuron α_j being deactivated when input neuron α_j is activated.*

Theorem 1 *(*Correctness of Argumentation Algorithm*) For each argumentation network $\mathcal{A}$, there exists a feedforward neural network $\mathcal{N}$ with exactly one hidden layer and semi-linear neurons such that $\mathcal{N}$ computes $\mathcal{A}$.*

Proof. *First, we need to show that the positive neural network computes $\mathcal{P}$. When $r_i : \alpha_i \rightarrow \alpha_i \in \mathcal{P}$, we need to show that (a) if $\alpha_i \geq A_{\min}$ in the input layer then $\alpha_i \geq A_{\min}$ in the output layer. We also need to show that (b) if $\alpha_i \leq -A_{\min}$ in the input layer then $\alpha_i \leq -A_{\min}$ in the output layer. (a) In the worst case, the input potential of hidden neuron N_i is $WA_{\min}$, and the output of N_i is $h(WA_{\min})$. We want $h(WA_{\min}) \geq A_{\min}$. Then, again in the worst case, the input potential of output neuron α_i will be $WA_{\min}$, and we want $h(WA_{\min}) \geq A_{\min}$. As a result, $W \geq h^{-1}(A_{\min})/A_{\min}$ needs to be verified, which gives $W \geq (1/\beta A_{\min}) \cdot (ln\,(1 + A_{\min}) - ln\,(1 - A_{\min}))$, as in the algorithm. The proof of (b) is analogous to the proof of (a). Now, we need to show that the addition of negative weights to the neural network implements the attacks in the argumentation network. When $v(\alpha_i, \alpha_j) = 1$, we want to ensure that the activation of output neuron α_j is smaller than $-A_{\min}$ whenever both hidden neurons N_i and N_j are activated. In the worst case scenario, N_i presents activation $A_{\min}$ while N_j presents activation 1. We have $h(W + A_{\min}W') < -A_{\min}$. Thus, we need $W' < (h^{-1}(-A_{\min})) - W)/A_{\min}$; this is obtained directly from the* ***Argumentation Algorithm****. Similarly, when $v(\alpha_i, \alpha_j) = 0$, we want to ensure that the activation of output neuron α_j is larger than $A_{\min}$ whenever both hidden neurons N_i and N_j are activated. In the worst case scenario, now N_i presents activation 1 while N_j presents activation $A_{\min}$. We have $h(A_{\min}W + W') > A_{\min}$. Thus, we need $W' > h^{-1}(A_{\min}) - WA_{\min}$; again, this is obtained directly from the* ***Argumentation Algorithm****. This completes the proof.* □

Our next step is to run the neural network to find out which arguments prevail in a situation. The key to running the network properly is to connect output neurons to their corresponding input neurons using weights fixed at 1, so that the activation of output neuron A, for example, is fed into the activation of input neuron A the next time round. This implements chains such as A attacks B, B attacks C, C attacks D, and so on, by propagating activations around the network. The following example illustrates the dynamics of argumentation neural networks.

Example 2 *(*Argument Computation*) Take the case in which an argument A attacks an argument B, and B attacks an argument C, which in turn attacks A in a cycle. In order to implement this in a neural network, we need three hidden neurons (h_1, h_2, h_3), positive weights to explicitly represent the fact that A supports itself (via h_1), B supports itself (via h_2), and so does C (via h_3). In addition, we need negative weights from h_1 to B, from h_2 to C and from h_3 to A to implement attacks (see Figure 4). If the value of argument A (i.e. the weight from h_1 to A) is stronger than the value of argument C (the weight from h_3 to C, which is expected to be the same in absolute terms as the weight from h_3 to A), C cannot attack and defeat A. As a result, A is activated. Since A and B have the same value, B is not activated, since the weights from h_1 and h_2 to B will both have the same absolute value. Finally, if B is not activated then C will be activated, and a stable state {A,C} will be reached in the network. In Bench-Capon's model [2], this is precisely the case in which colour* blue *is assigned to A and B, and colour* red *is assigned to C with* blue *being stronger than* red*. Note that the order in which we reason does not affect the final result (the stable state reached). For example, if we started from B successfully attacking C, C would not be able to attack A, but then A would successfully attack B, which would this time round not be able to successfully attack C, which in turn would be activated in the final stable state {A,C}. If, however, all the weights are the same in absolute terms, no argument is supposed to win. In this case, neither of output neurons A,B,C will be activated for any input given, indicating precisely this situation. The neural network does not loop, but always stabilises in [-1,-1,...,-1], which is what one would expect.*

The implementation of the network's behaviour (weights and biases) must be such that, when we start form a number of positive arguments (input vector [1,1,...1]), weights with the same absolute values cancel each other producing zero as the output neuron's input potential. Generally speaking, a neuron with zero or less input potential is then deactivated, while a neuron with positive input potential is activated. We conclude by discussing two alternative implementations of the moral debate example.

Example 3 *(*Moral Debate Neural Network*) We apply the* Neural Argumentation Algorithm *to the argumentation network of Figure 2, and obtain the neural network of Figure 5. From the* Translation Algorithm*, we know that $A_{min} > 0$ and $W > 0$. Let us take $A_{min} = 0.5$ and $W = 5$ (recall that W is the weight of solid arrows in the network). Following [9], we reason about the problem by grouping arguments according to the features of* life, property *and* fact*. Arguments* ***A****,* ***D*** *and* ***F*** *are related to the right of life, arguments* ***B*** *and* ***C*** *are related*

to property rights, and argument ***E*** *is a fact. We may argue whether* property *is stronger than* life *but facts are always the strongest. If property is stronger than life then* $v(B, A) = 1$, $v(D, A) = 1$, $v(C, B) = 1$, $v(C, D) = 1$, $v(E, D) = 1$, *and* $v(F, C) = 0$. *From the* Neural Argumentation Algorithm, *when* $v(\alpha_i, \alpha_j) = 0$ *we must have* $W' > -1.4$, *and when* $v(\alpha_i, \alpha_j) = 1$ *we must have* $W' < -12.2$. *The actual value of each attack may depend on the audience. Nevertheless, provided the above condition on* W' *are satisfied, according to Theorem 1, the network will compute the expected prevailing arguments, as follows: F does not defeat C, C defeats B, E defeats D and, as a result, we obtain {A,C,E} as the acceptable set of arguments. Nonetheless, if life is considered stronger than property then* $v(F, C) = 1$ *now, and as a result, F defeats C and, since C is defeated, it cannot defeat B, which in turn cannot defeat A (because life is stronger than property). Thus, we obtain the set {A,B,E,F} of acceptable arguments.*[7] *This shows that two different lines of argumentation will provide the same answer to the question of whether Hall was justified (A), but two different answers to the question of whether Carla has the right to compensation (C).*

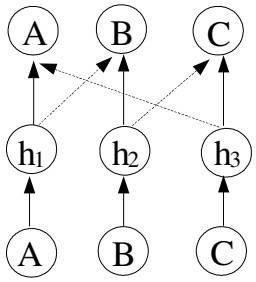

Figure 4: A circular argumentation neural network.

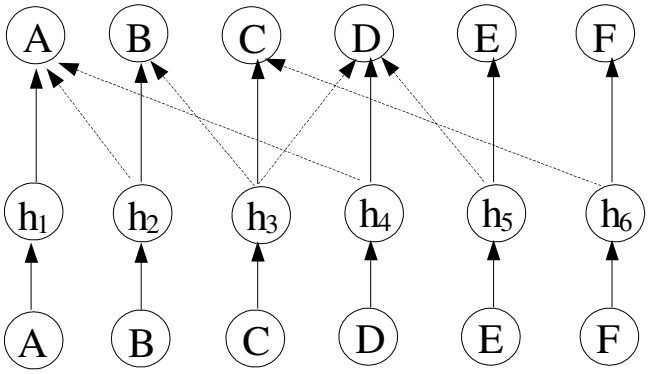

Figure 5: The moral-debate example as a neural network.

4 Conclusion and Future Work

In this paper, we have presented a new hybrid model of computation that allows for the deduction and learning of argumentative reasoning. The model combines value-based argumentation frameworks and neural-symbolic learning systems by providing a translation from argumentation networks to C-ILP neural networks, and a theorem showing that such a translation is correct. The model works not only for acyclic argumentation networks but also for circular networks and enables cummulative argumentation through learning.

Experiments on learning argumentation neural networks capable of evolving over time are currently being conducted. Complexity issues regarding the parallel computation of argumentation neural networks in contrast with standard value-based argumentation frameworks are also being investigated. We believe that a neural implementation of this reasoning process may, in fact, be advantageous from a purely computational point of view due to neural networks' parallel nature.

[7] The complete set of argument values in this case is: $v(B, A) = 0$, $v(D, A) = 1$, $v(C, B) = 1$, $v(C, D) = 0$, $v(E, D) = 1$, and $v(F, C) = 1$. The values of W' are calculated in the same way as before.

REFERENCES

[1] K. R. Apt and D. Pedreschi. Reasoning about termination of pure prolog programs. *Information and Computation*, 106:109–157, 1993.

[2] T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13:429–448, 2003.

[3] A. Bondarenko, P. Dung, R. Kowalski, and F. Toni. An abstract, argumentation theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

[4] I. Cloete and J. M. Zurada, editors. *Knowledge-Based Neurocomputing*. The MIT Press, 2000.

[5] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

[6] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Perspectives in Neural Computing. Springer-Verlag, 2002.

[7] A. S. d'Avila Garcez and L. C. Lamb. Reasoning about time and knowledge in neural-symbolic learning systems. In S. Thrun, L. Saul, and B. Schoelkopf, editors, *Advances in Neural Information Processing Systems 16*, Proceedings of the NIPS 2003 Conference, Vancouver, Canada, To appear 2004. MIT Press.

[8] A. S. d'Avila Garcez, L. C. Lamb, K. Broda, and D. M. Gabbay. Applying connectionist modal logics to distributed knowledge representation problems. *International Journal of Artificial Intelligence Tools*, (to appear) 2004.

[9] D. M. Gabbay and J. Woods. The law of evidence and labelled deduction: A position paper. *Phi News*, 4, October 2003.

[10] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the fifth Logic Programming Symposium*, pages 1070–1080, 1988.

[11] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[12] S. Holldobler and Y. Kalinke. Toward a new massively parallel computational model for logic programming. In *Proceedings of the Workshop on Combining Symbolic and Connectionist Processing, ECAI 94*, pages 68–77, 1994.

[13] S. Holldobler, Y. Kalinke, and H. P. Storr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, 11(1):45–58, 1999.

[14] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[15] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

[17] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, 1994.