

---

# Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems

ARTUR S. D'AVILA GARCEZ, *Department of Computing, City University London, London EC1V 0HB, UK.*  
Email: [aag@soi.city.ac.uk](mailto:aag@soi.city.ac.uk)

DOV M. GABBAY, *Department of Computer Science, King's College London, Strand, London WC2R 2LS, UK.*  
Email: [dg@dcs.kcl.ac.uk](mailto:dg@dcs.kcl.ac.uk)

LUIS C. LAMB, *Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, 91501-970, Brazil.*  
Email: [LuisLamb@acm.org](mailto:LuisLamb@acm.org)

## Abstract

While neural networks have been successfully used in a number of machine learning applications, logical languages have been the standard for the representation of argumentative reasoning. In this paper, we establish a relationship between neural networks and argumentation networks, combining reasoning and learning in the same argumentation framework. We do so by presenting a new *neural argumentation algorithm*, responsible for translating argumentation networks into standard neural networks. We then show a correspondence between the two networks. The algorithm works not only for acyclic argumentation networks, but also for circular networks, and it enables the accrual of arguments through learning as well as the parallel computation of arguments.

*Keywords:* Neural-symbolic systems, value-based argumentation frameworks, hybrid systems.

## 1 Introduction

The study of formal models of argumentation has long been a subject of intensive investigation in several areas, notably in logic, philosophy, decision making, artificial intelligence, and law [5, 6, 8, 9, 15, 17, 22, 27, 28, 34]. In artificial intelligence, models of argumentation have been one of the approaches used in the representation of commonsense, nonmonotonic reasoning. They have been particularly successful when modelling chains of *defeasible arguments* so as to reach a conclusion [25, 26]. Although logic-based models have been the standard for the representation of argumentative reasoning [6, 16], such models are intrinsically related to artificial neural networks, as we will show in this paper.

Neural networks have been successfully used in a number of computational learning applications [20, 24]. By establishing a relationship between neural networks and argumentation networks, we aim to provide a model in which the learning of arguments can be combined with reasoning capabilities within the same framework. In more general terms, the integration of reasoning and learning has been advocated by Valiant as a key challenge for computer science [31].

In this paper, we introduce a *neural argumentation algorithm*, which is responsible for translating value-based argumentation networks into standard neural networks with the use of neural-symbolic

## 2 Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems

systems [11]. Neural-symbolic systems concern the application of problem-specific symbolic knowledge within the neurocomputing paradigm. They have been used to combine neural network-based learning systems with nonmonotonic, epistemic, and temporal symbolic knowledge representation and reasoning [13, 14].

We show that the neural network created by the neural argumentation algorithm executes a sound computation of the prevailing arguments in the argumentation network. This shows that the two representations are equivalent. However, arguments will frequently attack one another in such a way that cycles are formed. In such cases, a notion of relative strength of the arguments may be required to decide which arguments should prevail. Still, in some cases, circularities may lead to an infinite loop in the computation. To tackle this problem, we propose the use of a learning mechanism. Learning can be used to resolve circularities by the iterative change of the strength of arguments as new information becomes available. Learning and its relation to accrual, cumulative argumentation [32, 27] in neural networks will also be discussed.

Our long-term 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. At the same time, we seek to enable the parallel computation of argumentation frameworks by making use of the machinery of neural networks.

The remainder of the paper is organised as follows. In Section 2, we present the basic concepts of value-based argumentation, neural networks, and neural-symbolic systems used throughout the paper. In Section 3, we introduce the *neural argumentation algorithm*, and prove that the neural network executes a sound computation of the argumentation network, and therefore that the translation is correct. In Section 4, we investigate how learning may help overcome circularities, and how neural networks may support accrual, cumulative argumentation. Section 5 concludes the paper and discusses directions for future work.

## 2 Background

In this section, we present the basic concepts of value-based argumentation frameworks used throughout this paper. We also define neural networks and introduce neural-symbolic learning systems.

### 2.1 Value-based Argumentation Frameworks

We start by describing the notion of value-based argumentation frameworks, following Bench-Capon's work [4]. In order to record the values associated with arguments, Bench-Capon has extended Dung's argumentation framework [15] by adding to it a set of values and a function mapping arguments to values. A notion of relative strength of arguments may then be defined by an audience, which essentially creates an ordering on the set of values. As a result, if an argument  $A$  attacks an argument  $B$  and the value of  $B$  is preferred over the value of  $A$  then it may be the case that  $A$  is accepted, and yet  $A$  is not able to defeat  $B$ . Therefore, a distinction between attacks and *defeats* - which are typically defined as successful attacks - is used [9, 26].

DEFINITION 2.1 ([15])

An argumentation network has the form  $\mathcal{A} = \langle \alpha, attack \rangle$ , where  $\alpha$  is a set of arguments, and  $attack \subseteq \alpha^2$  is a relation indicating which arguments attack which other arguments.

DEFINITION 2.2 ([4])

A value-based argumentation framework is a 5-tuple  $VAF = \langle \alpha, attacks, V, val, P \rangle$ , where  $\alpha$  is a finite set of arguments,  $attacks$  is an irreflexive binary relation on  $\alpha$ ,  $V$  is a non-empty set

of values,  $val$  is a function mapping elements in  $\alpha$  to elements in  $V$ , and  $P$  is a set of possible audiences, where we may have as many audiences as there are orderings on  $V$ . For every  $A \in \alpha$ ,  $val(A) \in V$ .

Bench-Capon also uses the notion of *colouring* in order to define preferences among audiences. For example, consider audience *red* (which prefers red over blue) and audience *blue* (which prefers blue over red), and the following value-based argumentation:  $A$  (coloured red) attacks  $B$  (coloured blue) which attacks  $C$  (coloured blue). As a result,  $\{A, C\}$  are the prevailing arguments for audience *red*; however, for audience *blue* the prevailing arguments are  $\{A, B\}$ . It is also important to note that Bench-Capon defines the notions of *objective* and *subjective* acceptability of arguments. The first are arguments acceptable no matter the choice of preferred values for every audience, whereas the second are acceptable to some audiences. Arguments which are neither objectively nor subjectively acceptable are called *undefensible*. Following Bench-Capon and the extension to argumentation networks given in [3], we model the strength of arguments by defining a function  $v$  from *attack* to  $\{0, 1\}$ , which gives the relative strength of an argument. Given  $\alpha_i, \alpha_j \in \alpha$ , if  $v(\alpha_i, \alpha_j) = 1$  then  $\alpha_i$  is said to be stronger than  $\alpha_j$ . Otherwise,  $\alpha_i$  is said to be weaker than  $\alpha_j$ .

Let us briefly consider the relationship between argumentation frameworks and neural networks informally. We can think of a neural network as a graph in which the vertices represent *neurons* and the edges indicate the *connections* between neurons. In a neural network, each edge is labelled with a real number indicating the relative *weight* of the connection. If we represent an argument as a neuron then a connection from neuron  $i$  to neuron  $j$  can be used to indicate that argument  $i$  either attacks or supports argument  $j$ . The weight of the connection can be seen as corresponding to the strength of the attack or the support. Any real number can be assigned to the weight of a connection in a neural network, and thus we will associate negative weights with attacks, and positive weights with supporting arguments<sup>1</sup>, as detailed later on.

In order to compute the prevailing arguments in a neural network, one needs to take into consideration the relative strength of the attacks as given, for example, by an audience. Since the strength of the different arguments is represented by the weights of the network, and since learning is the process of progressively changing the weights, it seems natural to use neural learning algorithms to change the network as new information becomes available. We will investigate this in more detail in Section 4. First, let us define neural networks precisely and introduce neural-symbolic systems.

## 2.2 Neural-Symbolic Learning Systems

An artificial neural network is a directed graph with the following structure: a unit (or neuron) in the 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 of the network are interconnected via a set of directed and weighted connections such that if there is a connection from unit  $i$  to unit  $j$  then  $W_{ji} \in \mathbb{R}$  denotes the *weight* of this connection. The input potential of neuron  $i$  at time  $t$  ( $U_i(t)$ ) is obtained by computing a weighted sum for neuron  $i$  such that  $U_i(t) = \sum_j W_{ij} I_j(t)$  (see Figure 1). The activation state  $A_i(t)$  of neuron  $i$  at time  $t$  - a bounded real or integer number - is then given by the neuron's *activation function*  $h_i$  such that  $A_i(t) = h_i(U_i(t))$ . Typically,  $h_i$  is either a linear function, a non-linear (step) function, or a sigmoid function (e.g.:  $\tanh(x)$ ). In addition,  $\theta_i$  (an extra weight

<sup>1</sup>Generally speaking, an argument  $i$  supports an argument  $j$  if the coordination of  $i$  and  $j$  reduces the likelihood of  $j$  being defeated. There are different ways in which an argument may support another. For example, argument  $i$  may support argument  $j$  by attacking an argument  $k$  that attacks  $j$ ; or argument  $i$  may support  $j$  directly, e.g. by strengthening the value of  $j$  [33]. In this paper, we use the terms *attack* and *support* in a loose way, since it will be sufficient to define precisely just the notion of *defeat*.

#### 4 Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems

with input always fixed at 1) is known as the *threshold* of neuron  $i$ . We say that neuron  $i$  is *active* at time  $t$  if  $A_i(t) > \theta_i$ . Finally, the neuron's output value  $O_i(t)$  is given by its output function  $f_i(A_i(t))$ . Usually,  $f_i$  is the identity function.

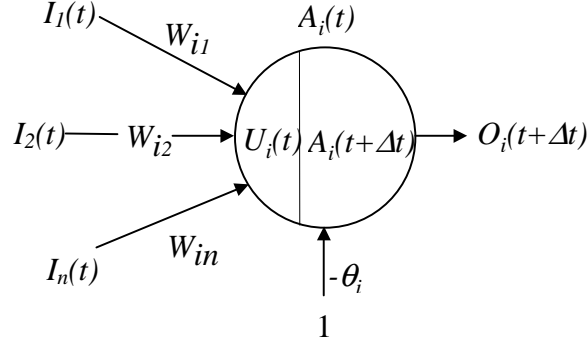


FIGURE 1. The neuron or processing unit.

The units of a neural network can be organised in layers. A  $n$ -layer *feedforward network* is an acyclic graph. It consists of a sequence of layers and connections between successive layers, containing one input layer,  $n - 2$  hidden layers, and one output layer, where  $n \geq 2$ . When  $n = 3$ , we say that the network 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 the network is *fully-connected*.

A multilayer feedforward network computes a function  $\varphi : \mathbb{R}^r \rightarrow \mathbb{R}^s$ , where  $r$  and  $s$  are the number of units occurring, respectively, in the input and output layers of the network. In the case of single hidden layer networks, the computation of  $\varphi$  occurs as follows: at time  $t_1$ , the input vector is presented to the input layer. At time  $t_2$ , the input vector is propagated through to the hidden layer, and the units in the hidden layer update their input potential and activation state. At time  $t_3$ , the hidden layer activation state is propagated to the output layer, and the units in the output layer update their input potential and activation state. At time  $t_4$ , 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 progressively so that it learns to approximate  $\varphi$  given a number of *training examples* (input vectors and their respective target output vectors).

In the case of *backpropagation* - the neural learning algorithm most successfully applied in industry [29] - an *error* is calculated as the difference between the network's actual output vector and the target vector, for each input vector in the set of examples. This error  $\mathbf{E}$  is then propagated back through the network, and used to calculate the variation of the weights  $\Delta \mathbf{W}$ . This calculation is such that the weights vary according to the *gradient* of the error, i.e.  $\Delta \mathbf{W} = -\eta \nabla \mathbf{E}$ , where  $0 < \eta < 1$  is called the *learning rate*. The process is repeated a number of times in an attempt to minimise the error, and thus approximate the network's actual output to the target output, for each example. In order to try and avoid shallow local minima in the error surface, a common extension of the learning algorithm above takes into account, at any time  $t$ , not only the gradient of the error function, but also the variation of the weights at time  $t - 1$ , so that  $\Delta \mathbf{W}_t = -\eta \nabla \mathbf{E} + \mu \Delta \mathbf{W}_{t-1}$ , where  $0 < \mu < 1$  is called the *term of momentum*. Typically, a subset of the set of examples available for training is left out of the learning process so that it can be used for checking the network's *generalisation* ability, i.e. its ability to respond well to examples not seen during training.

The *Connectionist Inductive Learning and Logic Programming System* C-ILP [11] is a massively parallel computational model based on an artificial neural network that integrates inductive learning

and deductive reasoning. In C-ILP, 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  $\mathcal{P}$  [23]. This provides a massively parallel model for computing the stable model semantics of  $\mathcal{P}$  [18]. In addition,  $\mathcal{N}$  can be trained with examples using a neural learning algorithm [29], having  $\mathcal{P}$  as background knowledge. The knowledge acquired by training can then be extracted [10], closing the learning cycle, as advocated in [30].

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<sub>1</sub>) the input potential of a hidden neuron  $N_l$  can only exceed its threshold  $\theta_l$ , activating  $N_l$ , when all the positive antecedents of  $r_l$  are assigned truth-value *true* while all the negative antecedents of  $r_l$  are assigned *false*; and (c<sub>2</sub>) the input potential of an output neuron  $A$  can only exceed its threshold ( $\theta_A$ ), activating  $A$ , when at least one hidden neuron  $N_l$  that is connected to  $A$  is activated.

#### EXAMPLE 2.3 (C-ILP)

Consider the logic program  $\mathcal{P} = \{B \wedge C \wedge \sim D \rightarrow A; E \wedge F \rightarrow A; B\}$ , where  $\sim$  stands for *negation* [23]. From  $\mathcal{P}$ , the C-ILP *translation algorithm* produces the network  $\mathcal{N}$  of Figure 2, setting weights ( $W$ ) and thresholds ( $\theta$ ) in a way that conditions (c<sub>1</sub>) and (c<sub>2</sub>) above are satisfied. Note that, if  $\mathcal{N}$  ought to be fully-connected, any other link (not shown in Figure 2) 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  $\mathcal{P}$  such that neuron  $N_1$  will be activated if neurons  $B$  and  $C$  are activated while neuron  $D$  is not; output neuron  $A$  will be activated if either  $N_1$  or  $N_2$  is activated; and output neuron  $B$  will be activated if  $N_3$  is, while  $N_3$  is always activated regardless of the input vector (i.e.  $B$  is a *fact*). To compute the stable models of  $\mathcal{P}$ , the output vector is recursively given as the next input to the network such that  $\mathcal{N}$  is used to iterate the fixed-point operator of  $\mathcal{P}$  [11]. For example, output neuron  $B$  should feed input neuron  $B$ .  $\mathcal{N}$  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 [2]. For example, given any initial activation in the input layer of  $\mathcal{N}_r$  (network of Figure 2 recurrently connected), it always converges to a stable state in which neuron  $B$  is activated and all the other neurons are not. We associate this with literal  $B$  being assigned truth-value *true*, while all the other literals are assigned truth-value *false*, which represents the unique fixed-point of  $\mathcal{P}$ .

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 extend such a positive network to represent attacks by using negative weights from the network's hidden layer to its output layer, as explained in detail in what follows.

### 3 Argumentation Neural Networks

In this section, we introduce the algorithm that allows us to translate value-based argumentation networks into neural networks. Firstly, let us see how the translation works in a typical argumentation example, namely, the *moral debate* example [4].

*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;

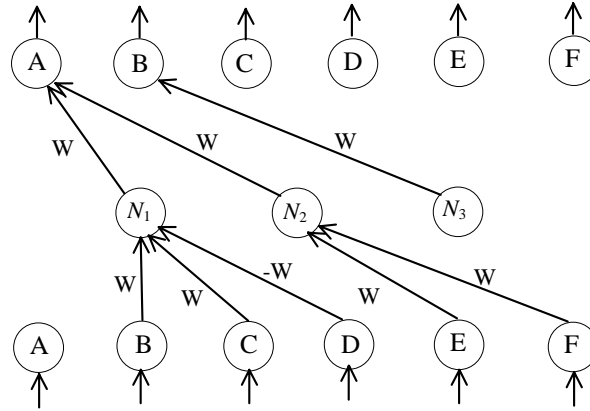


FIGURE 2. A neural network for logic program  $\mathcal{P}$ .

- $B$ : 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;
- $F$ : 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.

Arguments and counter-arguments can be arranged in an argumentation network, as in Figure 3, 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 3, some aspects may change as the debate progresses and actions are taken, with the strength of an argument in attacking another changing in time. We see this as 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 4. The network is single hidden layer with inputs  $(A, B, D)$ , outputs  $(A, B, D)$  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 4) with the use of  $h_2$  and  $h_3$ , respectively.

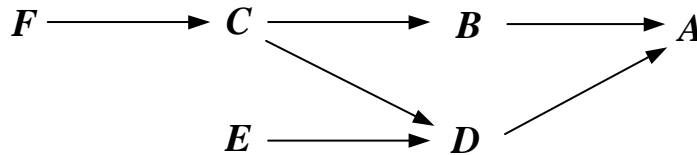
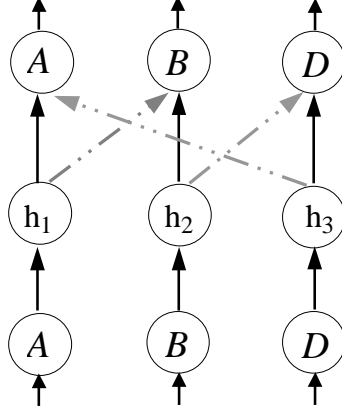


FIGURE 3. The moral debate argumentation network.

The network of Figure 4 is a standard feedforward neural network that can be trained with the use

FIGURE 4. A neural network for arguments  $A$ ,  $B$ ,  $D$ .

of a standard neural learning algorithm. Learning would change the initial weights of the network (or the initial beliefs on the strength of arguments and counter-arguments), according to examples (input and output patterns) of the relationship between arguments  $A$ ,  $B$  and  $D$ . This will become clearer in Section 4, where we shall give examples of learning argumentation.

In Figure 4, generally speaking, if the absolute value of the weight from neuron  $h_1$  to output neuron  $A$  (i.e. the strength of  $A$ ) is greater than the sum of the absolute values of the weights from neurons  $h_2$  and  $h_3$  to  $A$  (i.e. the strength of the attacks on  $A$ ), one should be able to say that argument  $A$  prevails (in which case output neuron  $A$  should be *active* in the neural network). Let us implement this form of reasoning using C-ILP neural networks.

The *neural argumentation algorithm* introduced below takes a value-based argumentation framework as input and produces a C-ILP neural network as output. These networks use a semi-linear activation function  $h(x) = \frac{2}{1+e^{-x}} - 1$  and inputs in  $\{-1, 1\}$ , where 1 represents *true* and  $-1$  represents *false*.<sup>2</sup> In addition, parameter  $A_{min}$  ( $0 < A_{min} < 1$ ) indicates the minimum activation for a neuron to be considered *active*. The algorithm then defines the set of weights of the neural network as a function of  $A_{min}$  such that the neural network computes the prevailing arguments according to the argumentation framework. The values of the weights derive from the proof of Theorem 3.2, which shows that the neural network indeed executes a sound computation of the argumentation framework.

### Neural Argumentation Algorithm

1. Given a value-based argumentation framework  $\mathcal{A}$  with arguments  $\alpha_1, \alpha_2, \dots, \alpha_n$ , let:  
 $\mathcal{P} = \{r_1 : \alpha_1 \rightarrow \alpha_1, r_2 : \alpha_2 \rightarrow \alpha_2, \dots, r_n : \alpha_n \rightarrow \alpha_n\}$ .
2. Number each atom of  $\mathcal{P}$  from 1 to  $n$  and create the input and output layers of a neural network  $\mathcal{N}$  such that the  $i$ -th neuron represents the  $i$ -th atom of  $\mathcal{P}$ .
3. Given  $0 < A_{min} < 1$ , calculate  $W \geq (1/A_{min}) \cdot (\ln(1 + A_{min}) - \ln(1 - A_{min}))$ .

<sup>2</sup>A differentiable function such as  $h(x)$  is needed for a gradient descent learning algorithm such as backpropagation. The use of  $\{-1, 1\}$  has been proved more efficient for learning than the use of  $\{0, 1\}$ , since values close to zero imply in very small changes of the weights during learning [7].

## 8 Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems

4. For each rule  $r_l$  of  $\mathcal{P}$  ( $1 \leq l \leq n$ ) do:
  - (a) Add a neuron  $N_l$  to the hidden layer of  $\mathcal{N}$ ;
  - (b) Connect neuron  $\alpha_l$  in the input layer of  $\mathcal{N}$  to hidden neuron  $N_l$  and set the connection weight to  $W$ ;
  - (c) Connect hidden neuron  $N_l$  to neuron  $\alpha_l$  in the output layer of  $\mathcal{N}$  and set the connection weight to  $W$ .
5. For each  $(\alpha_i, \alpha_j) \in \text{attack}$ ,<sup>3</sup> do:
  - (a) Connect hidden neuron  $N_i$  to output neuron  $\alpha_j$ ;
  - (b) If  $v(\alpha_i, \alpha_j) = 0$  then set the connection weight to  $W' > h^{-1}(A_{min}) - W A_{min}$ ;
  - (c) If  $v(\alpha_i, \alpha_j) = 1$  then set the connection weight to  $W' < (h^{-1}(-A_{min}) - W)/A_{min}$ .
6. Set the threshold of each neuron in  $\mathcal{N}$  to zero.
7. Set  $g(x) = x$  as the activation function of the neurons in the input layer of  $\mathcal{N}$ .<sup>4</sup>
8. Set  $h(x) = \frac{2}{1+e^{-x}} - 1$  as the activation function of the neurons in the hidden and output layers of  $\mathcal{N}$ .<sup>5</sup>

Note that, differently from the general C-ILP translation algorithm, in which rules may have any number of literals in the antecedent [11], here there is always a single literal  $\alpha_i$  in the antecedent of each rule  $r_i$ . This allows us to use threshold *zero* in the algorithm above. Note also that, in the algorithm,  $W > 0$  and  $W' < 0$ . This fits well with the idea of arguments having strengths ( $W$ ), and attacks also having strengths ( $W'$ ). In practice, the values of  $W$  and  $W'$  could be defined, e.g., by an audience using some form of voting system [4].

The notion of an argument that *supports* another seems natural in argumentation neural networks. If argument  $\alpha_i$  supports argument  $\alpha_j$ , this may be implemented easily in the neural network by the addition of a rule of the form  $\alpha_i \rightarrow \alpha_j$  to program  $\mathcal{P}$ .<sup>6</sup> We need to make sure that the neural network *computes* the prevailing arguments of the argumentation framework. For example, if an argument  $\alpha_i$  attacks an argument  $\alpha_j$ , and  $\alpha_i$  is stronger than  $\alpha_j$ , then neuron  $\alpha_i$  should be able to deactivate neuron  $\alpha_j$ . Conversely, if  $\alpha_i$  is weaker than  $\alpha_j$ , and no other argument attacks  $\alpha_j$ , then neuron  $\alpha_i$  should not be allowed to deactivate neuron  $\alpha_j$ . The following definition captures this.

### DEFINITION 3.1 ( $\mathcal{N}$ computes $\mathcal{A}$ )

Let  $(\alpha_i, \alpha_j) \in \text{attacks}$ . Let  $A_\alpha(t)$  denote the activation state of neuron  $\alpha$  at time  $t$ . We say that a neural network  $\mathcal{N}$  computes an argumentation framework  $\mathcal{A}$  if (i) whenever  $v(\alpha_i, \alpha_j) = 1$ , if  $A_{\alpha_i}(t) > A_{min}$  and  $A_{\alpha_j}(t) > A_{min}$  then  $A_{\alpha_j}(t+1) < -A_{min}$ ;<sup>7</sup> and (ii) whenever  $v(\alpha_i, \alpha_j) = 0$ , if  $A_{\alpha_i}(t) > A_{min}$  and  $A_{\alpha_j}(t) > A_{min}$  and for every  $\alpha_k$  ( $k \neq i, j$ ) such that  $(\alpha_k, \alpha_j) \in \text{attacks}$ ,  $A_{\alpha_k}(t) < -A_{min}$  then  $A_{\alpha_j}(t+1) > A_{min}$ .

<sup>3</sup>Recall that if  $v(\alpha_i, \alpha_j) = 1$  then  $\alpha_i$  should defeat  $\alpha_j$ , and if  $v(\alpha_i, \alpha_j) = 0$  then  $\alpha_i$  should not defeat  $\alpha_j$ . The notion of defeat will be defined precisely in the sequel.

<sup>4</sup>In this way, the activation of the neurons in the input layer of  $\mathcal{N}$ , given by each input vector  $\mathbf{i} \in \{-1, 1\}^{\mathcal{P}}$ , will represent an interpretation for  $\mathcal{P}$ , where 1 represents *true* and  $-1$  represents *false*.

<sup>5</sup>In this way, a gradient-based learning algorithm, such as *backpropagation*, can be applied to  $\mathcal{N}$ .

<sup>6</sup>In this way, an accumulation of arguments  $(\alpha_1, \dots, \alpha_n)$ , neither being individually stronger than the argument they attack ( $\alpha_{n+1}$ ), might produce an input potential  $n \cdot W'$  that overcomes the strength  $W$  of  $\alpha_{n+1}$ . This is naturally the way that neural networks work, and it relates to the accrual of arguments. We will discuss this in more detail in the next section.

<sup>7</sup>We use  $-A_{min}$  for mathematical convenience, so that neuron  $\alpha$  is said to be not active if  $A_\alpha(t) < -A_{min}$ , and active if  $A_\alpha(t) > A_{min}$ .

We now show that the translation from argumentation frameworks to neural networks is correct.

**THEOREM 3.2 (Correctness of Argumentation Algorithm)**

For each argumentation network  $\mathcal{A}$  there exists a 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 neural network computes  $\mathcal{P}$ . When  $r_i : \alpha_i \rightarrow \alpha_i \in \mathcal{P}$ , we need to show that (a) if  $\alpha_i > A_{min}$  in the input layer then  $\alpha_i > A_{min}$  in the output layer. We also need to show that (b) if  $\alpha_i < -A_{min}$  in the input layer then  $\alpha_i < -A_{min}$  in the output layer. (a) In the worst case, the input potential of hidden neuron  $N_i$  is  $W \cdot A_{min}$ , and the output of  $N_i$  is  $h(W \cdot A_{min})$ . We want  $h(W \cdot A_{min}) > A_{min}$ . Then, again in the worst case, the input potential of output neuron  $\alpha_i$  will be  $W \cdot A_{min}$ , and we want  $h(W \cdot A_{min}) > A_{min}$ . As a result,  $W > h^{-1}(A_{min})/A_{min}$  needs to be verified, which gives  $W > (1/A_{min}) \cdot (\ln(1 + A_{min}) - \ln(1 - A_{min}))$ , 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 framework. When  $v(\alpha_i, \alpha_j) = 1$ , we want to ensure that the activation of output neuron  $\alpha_j$  is smaller than  $-A_{min}$  whenever both hidden neurons  $N_i$  and  $N_j$  are active. In the worst case scenario,  $N_i$  has activation  $A_{min}$  while  $N_j$  has activation 1. We have  $h(W + A_{min}W') < -A_{min}$ . Thus, we need  $W' < (h^{-1}(-A_{min}) - W)/A_{min}$ ; this is satisfied by the *argumentation algorithm*. Similarly, when  $v(\alpha_i, \alpha_j) = 0$ , we want to ensure that the activation of output neuron  $\alpha_j$  is larger than  $A_{min}$  whenever both hidden neurons  $N_i$  and  $N_j$  are active. In the worst case scenario, now  $N_i$  has activation 1 while  $N_j$  has 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 satisfied by the *argumentation algorithm*. This completes the proof.

## 4 Argument Computation and Learning

In this section, we consider the computation of arguments in neural networks. We start by giving an example. We then consider the case in which arguments attack each other forming a cycle in the argumentation network. This may result in an infinite computation in the neural network. To tackle this problem, we propose the use of learning as a way of breaking the circularity. Learning can be seen as a way of implementing the accrual of arguments. We conclude the section by discussing this issue. There is interesting further research on each of these topics. In this section, our purpose is to introduce a range of issues for argumentation neural networks, but we are far from exhausting the subject.

Once we have translated argumentation networks into neural networks, our next step is to run the neural network to find out which arguments prevail. To run the network, we connect output neurons to their corresponding input neurons using weights  $W_r = 1$  (see Figure 5) so that, for example, the activation of output neuron  $A$  is fed into input neuron  $A$  at the next time round [21]. This implements chains such as  $A$  attacks  $B$ ,  $B$  attacks  $C$ ,  $C$  attacks  $D$ , and so on, by propagating activation around the network. The following example illustrates this computation in the case of the moral debate example introduced above.

**EXAMPLE 4.1 (Moral Debate Neural Network)**

We apply the *neural argumentation algorithm* to the argumentation network of Figure 3, and obtain the neural network of Figure 5. From the algorithm, we know that we should have  $A_{min} > 0$  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 [16], let us consider 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 an audience. Nevertheless, provided that the above conditions on  $W'$  are satisfied, the network will compute the expected prevailing arguments according to Theorem 3.2, 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. Now, if *life* is considered stronger than *property* then  $v(F, C) = 1$ . 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, the network converges to the state  $\{A, B, E, F\}$  of acceptable arguments.<sup>8</sup> This shows that two different lines of value-based 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 ( $C$ ).

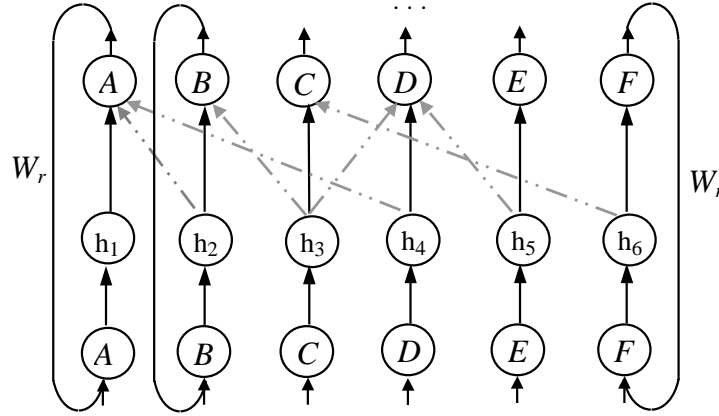


FIGURE 5. The moral-debate example as a neural network.

#### 4.1 Circular Argumentation

Arguments may frequently attack one another in such a way that cycles are formed. In such cases, the relative strength of the arguments will decide which of them should prevail, if any. In [3], as part of a study of the dynamics of argumentation networks [8], Barringer, Gabbay and Woods discuss how to handle loops during the computation of arguments. They differentiate between *syntactic* and *semantic* loops, in that the former occurs as cycles in the argumentation network (e.g., when argument  $A$  attacks argument  $B$  and vice-versa), while the latter also depends on the strength of the arguments involved in the loop.

In this way, if  $A$  is considerably stronger than  $B$  or vice-versa, no semantic loop will exist despite the fact that there is a (syntactic) loop in the network; the relative strength of the arguments resolves

<sup>8</sup>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 constraints on  $W'$  are calculated in the same way as before.

the loop. Even if  $A$  and  $B$  both have similar strengths, one possible interpretation is that neither argument should prevail. This would also resolve the loop and, as we shall see in the sequel, the dynamics of argumentation neural networks follows this interpretation. Still, there are situations in which the network oscillates between stable states, which indicates the existence of alternative, conflicting sets of arguments. This problem may be resolved by changing the strength of certain arguments. Such a change may either be due to the fact that new information has become available, or it may come from the investigation of the oscillating behaviour of the argumentation network itself, as exemplified below.

EXAMPLE 4.2 (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, as shown in Figure 6. 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 7). 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 active and succeeds in attacking  $B$  (since we assume that the weights from  $h_1$  and  $h_2$  to  $B$  have the same absolute value). Since  $B$  is not active,  $C$  will be active, and a stable state  $\{A, C\}$  will be reached. In Bench-Capon’s model [4], 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 had started with  $B$  successfully attacking  $C$ ,  $C$  would not have been able to defeat  $A$ , but then  $A$  would successfully defeat  $B$ , which would, this time round, not be able to successfully defeat  $C$ , which in turn would be active in the final stable state  $\{A, C\}$ .

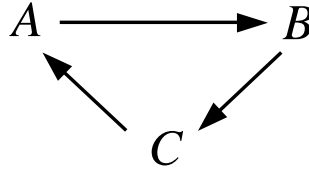


FIGURE 6. Circular arguments.

In Example 4.2, a *syntactic loop* exists in that the attacks in the argumentation network form a loop in Figure 6. However, there is no *semantic loop*, as the computation of the arguments converges to a stable state, as exemplified above. Even if the strength of the arguments were all the same, the neural network of Figure 7 would converge to  $\{ \}$ , as follows: assume that solid arrows have weight  $W$ , and dotted arrows have weight  $-W$  in the network of Figure 7. Let  $(A, B, C) = [1, 1, 1]$  denote the network’s input vector. Thus,  $h(W)$  will be the activation state of each hidden neuron, where  $h$  is the activation function of such neurons. Then,  $W \cdot h(W) - W \cdot h(W) = 0$  will be the input potential of each output neuron, and thus  $h(0) = 0$  (recall that  $\theta = 0$ ) will be the activation state of each output neuron  $A, B$ , and  $C$ . Now, given input vector  $(A, B, C) = [0, 0, 0]$ , the activation state of each hidden neuron will be zero, and then the activation state of each output neuron will be zero. As a result, the network converges to a stable state  $\{ \}$  in which no argument prevails. This stable state is reached after a single computation step from  $[1, 1, 1]$  to  $[0, 0, 0]$ .

According to Definition 3.1, an argument prevails if its associated neuron has activation in the interval  $(A_{min}, 1]$  with  $A_{min} > 0$ . Dually, whenever an argument is defeated, its associated neuron

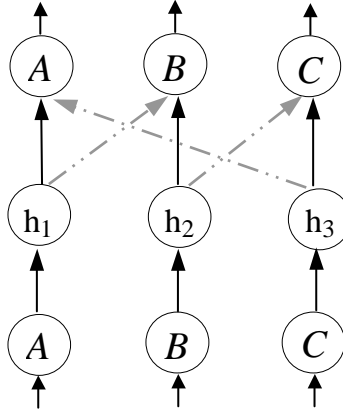


FIGURE 7. A circular argumentation neural network.

should have activation in the interval  $[-1, -A_{min})$ . In the case of circular argumentation networks, however, there is a third possibility when arguments cancel each other and the neurons' activations lie in the interval  $[-A_{min}, A_{min}]$ , typically converging to zero, as illustrated above. In this case, we know that arguments do not prevail, and it might be useful in some situations to make a distinction between a clear defeat and a failure to prevail. We will return to this issue when we consider argument learning. First, we need to study the more involved situation where the network oscillates between states.

Unfortunately, not all argumentation neural networks are as well-behaved as the ones considered so far. Take the case in which an argument  $A$  attacks two arguments  $B$  and  $C$ ;  $B$  and  $C$  in turn both attack an argument  $D$ , and  $D$  attacks  $A$  in a cycle, as depicted in Figure 8. Assume that all the attacks have the same strength. Figure 9 shows the neural network for the argumentation network of Figure 8. Assume, as before, that solid arrows have weight  $W$ , and dotted arrows have weight  $-W$ . Given input  $(A, B, C, D) = [1, 1, 1, 1]$ , from Figure 9, it is clear that the values of output neurons  $A$ ,  $B$ , and  $C$  will be zero, as the weights  $W$  and  $-W$  cancel each other out. The input potential of output neuron  $D$ , however, will be  $-W \cdot h(W)$ , and so the value of output neuron  $D$  will be  $O_D = h(-W \cdot h(W))$ . The next time round, the input potential of output neuron  $A$  will be  $-W \cdot h(O_D \cdot W)$ , and  $A$  will have a positive activation  $h(-W \cdot h(O_D \cdot W))$ . As a result, if  $A$  successfully defeats  $B$  and  $C$  then  $D$  prevails, and thus defeats  $A$ . In this case,  $B$  and  $C$  prevail, and defeat  $D$ , in a cycle.

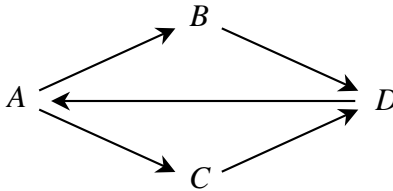


FIGURE 8. Semantic circularity.

Let us look at this cycle in more detail in the neural network. Let us assume, for convenience, that

$W = 1$ ,  $h(x) = 1$  if  $x \geq 1$ ,  $h(x) = -1$  if  $x \leq -1$ , and  $h(x) = x$  for  $-1 < x < 1$ .<sup>9</sup> We start with input vector  $[1, 1, 1, 1]$ , and obtain output vector  $[0, 0, 0, -1]$ . We then use  $[0, 0, 0, -1]$  as input to obtain output  $[1, 0, 0, -1]$ . Let us use  $\mapsto$  to denote the above mapping from input to output vectors, so that we have:  $[1, 1, 1, 1] \mapsto [0, 0, 0, -1] \mapsto [1, 0, 0, -1] \mapsto [1, -1, -1, -1] \mapsto [1, -1, -1, 1] \mapsto [0, -1, -1, 1] \mapsto [-1, -1, -1, 1] \mapsto [-1, 0, 0, 1] \mapsto [-1, 1, 1, 1] \mapsto [-1, 1, 1, -1] \mapsto [0, 1, 1, -1] \mapsto [1, 1, 1, -1] \mapsto [1, 0, 0, -1] \mapsto \dots \mapsto [1, 0, 0, -1] \dots$ , which shows that we have reached an infinite loop.

Can we learn anything from the sequence of arguments computed by the network? Initially, there is a situation in which  $A$  seems to prevail. Then,  $A$  and  $D$  together prevail. Then,  $D$  alone; then  $B$ ,  $C$ , and  $D$  together; then  $B$  and  $C$  only; and finally  $A$ ,  $B$ , and  $C$ , before we go back to the situation in which  $A$  alone seems to prevail. One way to deal with this problem would be to simply assume that the loop itself indicates that no argument should prevail at the end. One may argue, however, that this does not really solve the problem. Alternatively, one could try and use the information obtained by the computation of arguments itself as an indication of how one should go about trying to solve the loop. In the example above, for instance, it seems that either  $\{A, D\}$  or  $\{B, C\}$  could serve as a basis for a stable set of arguments. Nevertheless, more information would be needed, and the loop itself could serve as the trigger for a search for new information. In the case of the network of Figure 9, one could start by searching for information in support for  $\{A, D\}$  and in support for  $\{B, C\}$ , and only then in support for the other combinations. It seems that the only real solution to the problem of semantic loops is to have new information in the form of new evidence about the relative strength of the arguments and to learn from it, as we discuss in the following section.

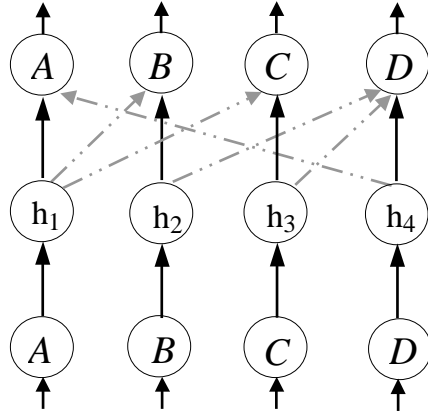


FIGURE 9. A neural network encoding semantic circularity.

### 4.2 Argument Learning

Consider again the neural network of Figure 9. Suppose that new evidence becomes available in favour of arguments  $A$  and  $C$  so that we would like both arguments to prevail. We do not know how this should affect arguments  $B$  and  $D$ , but we know that, now, given input vector  $[1, 1, 1, 1]$ , we would like the network of Figure 9 to produce output  $[1, ?, 1, ?]$ , instead of  $[0, 0, 0, -1]$ . Since we do not have any information about  $B$  or  $D$ , the natural candidates for  $?$  are the original values (so that

<sup>9</sup>This gives an approximation of the standard sigmoid activation function.

$[1, ?, 1, ?]$  becomes  $[1, 0, 1, -1]$ ). This will produce an *error* of zero for output neurons  $B$  and  $D$  during learning, which is the best way of reflecting the lack of new information about such concepts. An error of zero will produce no change on the weights directly associated with  $B$  and  $D$ , but of course the changes of other weights may affect the overall result of the network. Let us exemplify this in the case of the network of Figure 9 for our training example  $[1, 1, 1, 1] \mapsto [1, 0, 1, -1]$ .

We use the standard backpropagation learning algorithm<sup>10</sup> [29]. The use of backpropagation is made possible because the network is recurrently connected only when it comes to the computation of the arguments, not during learning. The recurrent connections are important for the reasoning process, having weights always fixed at 1. During learning, we are interested in establishing a new mapping from the input to the output, and thus a learning algorithm that applies to feedforward networks, such as backpropagation, suffices. We use  $W = 4.0$  (solid arrows in Figure 9), and  $W' = -4.0$  (dotted arrows in Figure 9).<sup>11</sup> Recall that  $\theta = 0$  and that any other connection not shown in Figure 9 is given weight zero initially.

After training, the thresholds of output neurons  $A$  ( $\theta_A$ ) and  $C$  ( $\theta_C$ ) have been changed to  $-0.8$ , the weights from  $h_1$  to  $A$  ( $W_{A,h_1}$ ) and from  $h_3$  to  $C$  ( $W_{C,h_3}$ ) have been changed to 4.8, and the weights from  $h_4$  to  $A$  ( $W_{A,h_4}$ ) and from  $h_1$  to  $C$  ( $W_{C,h_1}$ ) have been changed to  $-3.2$ . In addition, some very minor changes have occurred in the weights linking the input to the hidden layer of the network and, as expected, no changes have occurred in the weights leading to output neurons  $B$  or  $D$ , namely  $W_{B,h_i}$  and  $W_{D,h_i}$ ,  $1 \leq i \leq 4$  (recall that, for the purpose of learning, the network is fully-connected).

The computation of the arguments in the trained network is as follows. In addition to the transition from  $[1, 1, 1, 1]$  to  $[1, 0, 1, -1]$  learned as expected, the network then maps  $[1, 0, 1, -1]$  into  $[1, -1, 1, -1]$ , which is a stable state. The newly learned sequence  $[1, 1, 1, 1] \mapsto [1, 0, 1, -1] \mapsto [1, -1, 1, -1] \mapsto [1, -1, 1, -1]$  is now loop free; stable state  $[1, -1, 1, -1]$  corresponds to the acceptance of  $\{A, C\}$  as prevailing arguments.

As another example, let us consider the Nixon diamond problem. In the traditional Nixon diamond problem, Nixon is a *quaker* ( $Q$ ) and a *republican* ( $R$ ). Quakers are generally *pacifists* ( $P$ ), while republicans are generally *non-pacifists* ( $\neg P$ ). This produces an inconsistency in a number of formalisations of the problem [1]. Briefly, if the strength of the support for Nixon’s pacifism is the same as the strength of the support for his non-pacifism, the neural network will conclude that both  $P$  and  $\neg P$  prevail. If, in addition, we assume that  $P$  attacks  $\neg P$ , and vice-versa,  $\neg P$  attacks  $P$ , both with the same strength, then the stable state of the neural network will contain neither  $P$  nor  $\neg P$ . Finally, if we are faced with a situation in which we need to choose between  $P$  and  $\neg P$ , we could learn to enforce a stable state in the network in which one but not the other argument prevails.

Suppose we need to make a decision about Nixon’s pacifism. We need to seek new information. We find out that Nixon is a *football fan* ( $F$ ), and that football fans are normally non-pacifists. We then use this information to convince ourselves that Nixon is indeed a non-pacifist. We need to add an input, a hidden, and an output neuron to our original network to represent  $F$ . Then, to make sure that  $F$  attacks and defeats  $P$ , we simply train the following example  $[1, 1, 1, 1, 1] \mapsto [1, 1, -1, 1, 1]$ , given arguments  $Q, R, P, \neg P, F$  in this order.

There are also situations in which the number of times that an example occurs should be relevant to the decision about the strength of an argument. In such cases, since the backpropagation algorithm does not really emphasise this aspect, alternative forms of learning would need to be investigated

<sup>10</sup>We use  $\tanh$  as activation function, a learning rate of 0.1, and a term of momentum of 0.4. We train the network on the single training example ( $[1, 1, 1, 1], [1, 0, 1, -1]$ ) until a mean square error of 0.01 is reached.

<sup>11</sup>This is because  $\tanh(4.0) = 0.999$ , while  $\tanh(1.0) = 0.761$ . As a result,  $W = 4.0$  gives a good approximation for  $h(W) = 1$ , as in our previous assumption.

and compared with backpropagation. We believe this is an interesting research area with many open research issues to be addressed in the near future as we combine neural networks-based learning, case-based reasoning, and argumentation theory.

### 4.3 Cumulative (accrual) Argumentation

We have mentioned that neural networks deal with cumulative argumentation in the sense that a number of arguments, neither being individually stronger than a given argument, may defeat this argument collectively. There is some controversy on whether arguments accrue. While Pollock denies the existence of cumulative argumentation [27], Verheij defends that arguments can be combined either by subordination or by coordination, and may accrue in stages [32]. In this section, we give an example of accrual by coordination, which happens to be a natural property of neural networks.

Consider the following scenario. *Suppose you are the head of state of a country who needs to decide whether or not to go to war in order to remove a violent dictator from power. First, you consider the prospects of the loss of lives of fellow countrymen and women in action, and the killing of innocent civilians, which form a strong argument against going to war. Let us call this argument  $A_1$ . In support for the war, according to documented evidence from your intelligence services, the dictator possesses chemical and biological weapons, having made use of such weapons in the past. We call this argument  $A_2$ . You also happen to possess what you believe is credible information about the fact that the dictator has recently acquired uranium from another country, most probably in order to continue with an unauthorised nuclear weapons capability programme. Let this be argument  $A_3$ . In addition, recent information sources indicate that the dictator has the capability and the will - having done so in the past - to attack neighbouring countries. Let us name this argument  $A_4$ . Finally, you receive evidence that the dictator has provided safe haven for well-known members of an active international terrorist network. This is argument  $A_5$ . The task at hand is to decide whether or not it is right to remove the dictator.*

Let  $B$  denote the proposition *it is right to remove the Dictator*, and consider the situation in which  $A_1$  attacks  $B$  while  $A_2, \dots, A_5$  support  $B$ . Assume, further, that  $A_1$  is stronger than  $B$ , i.e.  $v(A_1, B) = 1$ . We apply the *neural argumentation algorithm* and obtain the neural network of Figure 10. Taking  $A_{min} = 0.5$ , we calculate  $W > 2.2$ . Taking, e.g.,  $W = 3$ , we calculate  $W' < -4.5$ . Let us take  $W' = -5$ .

According to the algorithm,  $W = 3$  and  $W' = -5$  form an acceptable set of weights. Although  $A_1$  is supposed to defeat  $B$  if contrasted with any of  $A_2, \dots, A_5$ , the cumulative support of  $A_2, \dots, A_5$  for  $B$  actually allows it to prevail. This can be seen in the network by inspecting the input potential of neuron  $B$ , which is approximately  $(4 \cdot W) + W' = 7$ , i.e. a relatively large positive value, which activates neuron  $B$ . Of course, a different outcome could be obtained by ensuring that  $W'$  is large enough (in absolute terms) to counteract the influence of all the other arguments together. A value of  $W' = -16$ , for example, would produce an input potential of  $-4$ , which would be sufficient to deactivate neuron  $B$ . Of course, these values should depend on your degree of belief on the arguments to go to war and against it, or on the influence of a number of examples (e.g., previous cases) from which you could learn to make an informed decision.

## 5 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

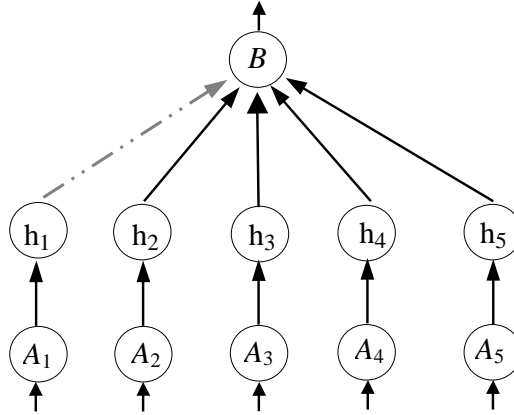


FIGURE 10. Cumulative support Argumentation.

neural networks. A theorem then shows that such a translation is correct. We have shown that the model works not only for acyclic argumentation networks, but also for circular networks, and it enables cumulative argumentation through learning.

Larger-scale experiments on learning argumentation 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 the massive parallelism of neural networks.

As future work, we shall consider fibring neural networks [12] and their relation to the general argumentation framework of Williamsom and Gabbay [35]. This framework incorporates, in addition to the notions of attack and support, the idea of recursive causality, according to which causal relations may take causal relations as input values; for example, the fact that smoking causes cancer may cause the government to restrict smoking advertising [35]. This corresponds to fibring neural networks, allowing nodes to behave as networks in a recursive way, and weights to be defined as functions of the values of other networks. Finally, the model presented here could be extended so as to consider probabilistic weights in argumentation frameworks, in the style of [19]. This would allow for a quantitative approach to argumentation in an integrated model of reasoning under uncertainty and inductive learning.

## Acknowledgments

Artur d'Avila Garcez has been partly supported by The Nuffield Foundation. Luis C. Lamb has been partly supported by the Brazilian Research Council CNPq, CAPES, and by the FAPERGS Foundation. We would like to thank the anonymous referees for several supportive comments that led to the improvement of the presentation of this paper.

## References

- [1] G. Antoniou. *Nonmonotonic Reasoning*. MIT Press, Cambridge, MA, 1997.
- [2] K. R. Apt and D. Pedreschi. Reasoning about termination of pure prolog programs. *Information and Computation*, 106:109–157, 1993.

- [3] H. Barringer, D. M. Gabbay, and J. Woods. Temporal dynamics of support and attack networks: From argumentation to zoology. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Joerg H. Siekmann*, pages 59–98. Springer-Verlag, 2005.
- [4] T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13:429–448, 2003.
- [5] P. Besnard and A. Hunter. Towards a logic-based theory of argumentation. In *Proceedings of 17th National Conference on Artificial Intelligence AAAI'00*, pages 411–416. AAAI Press, 2000.
- [6] A. Bondarenko, P. Dung, R. Kowalski, and F. Toni. An abstract, argumentation theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.
- [7] N. K. Bose and P. Liang. *Neural Networks Fundamentals with Graphs, Algorithms, and Applications*. McGraw-Hill, 1996.
- [8] G. Brewka. Dynamic argument systems: A formal model of argumentation processes based on situation calculus. *Journal of Logic and Computation*, 11(2):257–282, 2001.
- [9] C. I. Chesñevar, A. G. Maguitman, and R. P. Loui. Logical Models of Argument. *ACM Computing Surveys*, 32(4):337–383, December 2000.
- [10] 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.
- [11] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer-Verlag, 2002.
- [12] A. S. d'Avila Garcez and D. M. Gabbay. Fibring neural networks. In *Proceedings of 19th National Conference on Artificial Intelligence AAAI'04*, pages 342–347, San Jose, California, USA, July 2004. AAAI Press.
- [13] 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, pages 921–928. MIT Press, Cambridge, MA, 2004.
- [14] A. S. d'Avila Garcez, L. C. Lamb, K. Broda, and D. M. Gabbay. Applying connectionist modal logics to distributed knowledge representation problems. *Intl. Jnl. on Artificial Intelligence Tools*, 13(1):115–139, 2004.
- [15] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.
- [16] D. M. Gabbay and J. Woods. The law of evidence and labelled deduction: A position paper. *Phi News*, 4:5–46, October 2003.
- [17] A. J. García and G. R. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.
- [18] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th Logic Programming Symposium*, pages 1070–1080, 1988.
- [19] R. Haenni, J. Kohlas, and N. Lehmann. Probabilistic argumentation systems. In J. Kohlas and S. Moral, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5. Kluwer, 2000.
- [20] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.
- [21] 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.
- [22] R. Kowalski and F. Toni. Abstract argumentation. *Artificial Intelligence and Law*, 4(3-4):275–296, 1996.
- [23] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
- [24] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [25] D. Nute. Defeasible logic. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 3*, pages 355–395. Oxford University Press, 1994.
- [26] J. Pollock. Defeasible Reasoning. *Cognitive Science*, 11:481–518, 1987.
- [27] J. Pollock. Self-defeating arguments. *Minds and Machines*, 1(4):367–392, 1991.
- [28] H. Prakken and G. A. W. Vreeswijk. Logical systems for defeasible argumentation. In D. Gabbay and F. Guentner, editors, *Handbook of Philosophical Logic*. Kluwer, 2nd edition, 2000.
- [29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.
- [30] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, 1994.
- [31] L. G. Valiant. Three problems in computer science. *Journal of the ACM*, 50(1):96–99, 2003.
- [32] B. Verheij. Accrual of arguments in defeasible argumentation. In *Proceedings of 2nd Dutch/German Workshop on Nonmonotonic Reasoning*, pages 217–224. Utrecht, 1995.

18 *Value-based Argumentation Frameworks as Neural-Symbolic Learning Systems*

- [33] B. Verheij. *Rules, Reasons, Arguments: formal studies of argumentation and defeat*. PhD thesis, Maastricht University, Holland, 1996.
- [34] G. A. W. Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90(1–2):225–279, 1997.
- [35] J. Williamson and D. Gabbay. Recursive causality in bayesian networks and self-fibring networks. In D. Gillies, editor, *Laws and Models in Science*. King's College Publications, London, 2004.

Received 6 October 2004