A neural network-based approach for the computation of the answer sets of logic programs

Marcello Balduccini

February 1, 2002
Revised on February 2, 2002

Goal

Define a mapping from A-Prolog programs to neural networks

- powerful engine for the computation of the answer sets of logic programs
- declarative programming/specification language for neural networks

A-Prolog

- *Signature*: $\Sigma = \langle C, F, P \rangle$

  C: set of constant symbols
  F: set of function symbols
  P: set of predicate symbols

- *Term*: a constant symbol or $f(t_1, t_2, \ldots, t_n)$.

- *Atom*: $p(t_1, t_2, \ldots, t_n)$. n is the *arity* of predicate p.

- *Literal*: $p(t_1, t_2, \ldots, t_n)$ and $\neg p(t_1, t_2, \ldots, t_n)$

- *Rule*:

  $$l_0 \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n.$$

  "if $l_1, \ldots, l_m$ are believed to be true, and there is no reason to believe $l_{m+1}, \ldots, l_n$, then l_0 must be believed to be true."

- "not" is called *negation as failure*.

- Some useful definitions:

  $$\begin{array}{lcl} head(r) & = & l_0 \\ body(r) & = & \{l_1, \ldots, l_n\} \\ pos(r) & = & \{l_1, \ldots, l_m\} \\ neg(r) & = & \{l_{m+1}, \ldots, l_n\} \end{array}$$

A-Prolog Programs

- An A-Prolog program is a set of rules.
- Rules with *variables* are schemas.
- A *ground* program is a program not containing variables.
- A *propositional* program is a program where all predicates have arity 0.
- A program is *stratified* if there exists a mapping, λ, such that, for every rule of the program,

$$\lambda(l_0) \geq \lambda(l_i) \quad \text{for all } 1 \leq i \leq m$$
$$\lambda(l_0) > \lambda(l_i) \quad \text{for all } m+1 \leq i \leq n$$

Answer sets of A-Prolog programs

A *basic* program is a program not containing negation as failure.

Answer set of a basic program Π: a consistent set of literals S is an answer set of Π if:

1. S is closed under of rules of Π (i.e. for every rule $r \in \Pi$, if $body(r) \subseteq S$, then $head(r) \in S$),

2. S is the minimal set, under set-theoretic inclusion, satisfying the previous property.

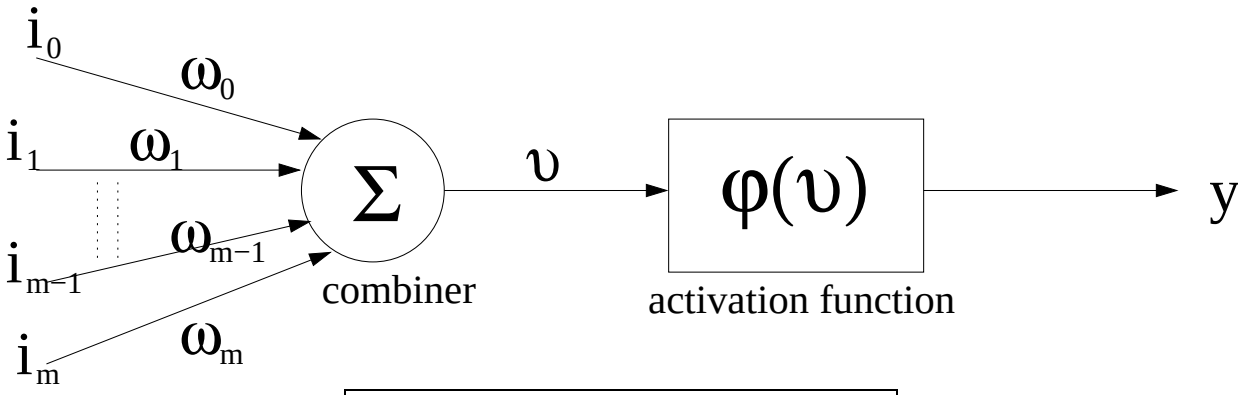
The *reduct* of a (general) program Π w.r.t. a consistent set of literals, S, is denoted by Π^S and is computed as follows:

1. For any rule $r \in \Pi$, if $l \in neg(r)$ and $l \notin S$, remove "not l" from the body of r.

2. Remove from Π^S any remaining rule, r, such that $neg(r) \neq \emptyset$.

A consistent set of literals, S, is an answer set of a program Π if it is an answer set of the reduct Π^S.

Model of a Neuron



$$
\begin{aligned}
y &= \varphi(v) \\
v &= \sum_k \omega_k i_k
\end{aligned}
$$

Commonly used activation functions:

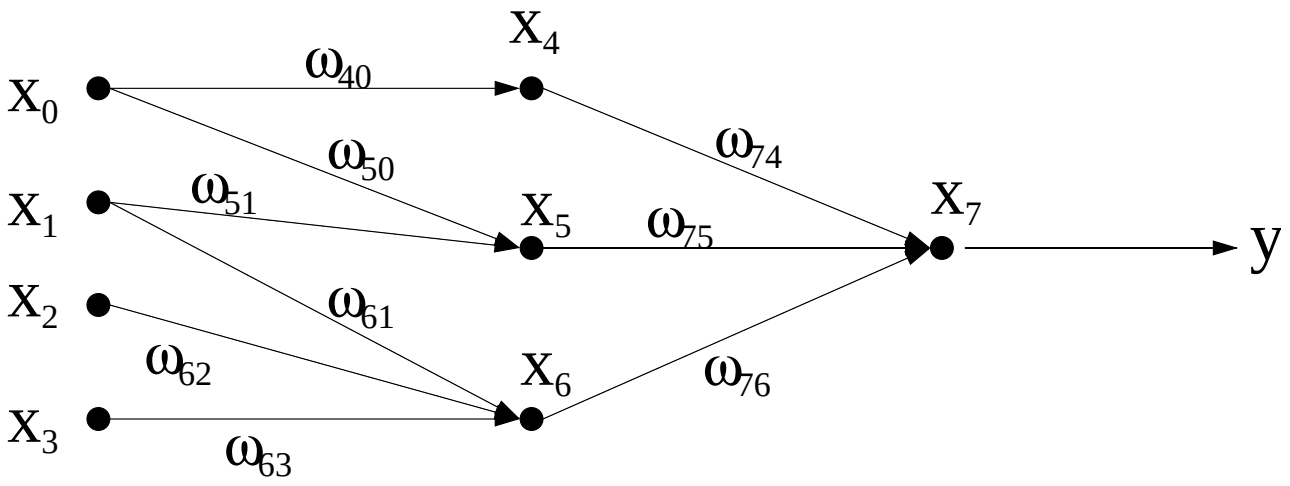
- Threshold Function

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

- Sigmoid Function

$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

Feed-forward Neural Networks



- directed acyclic graph
- computes the composition of the functions calculated by its neurons
- all (non-input) neurons usually compute the same activation function

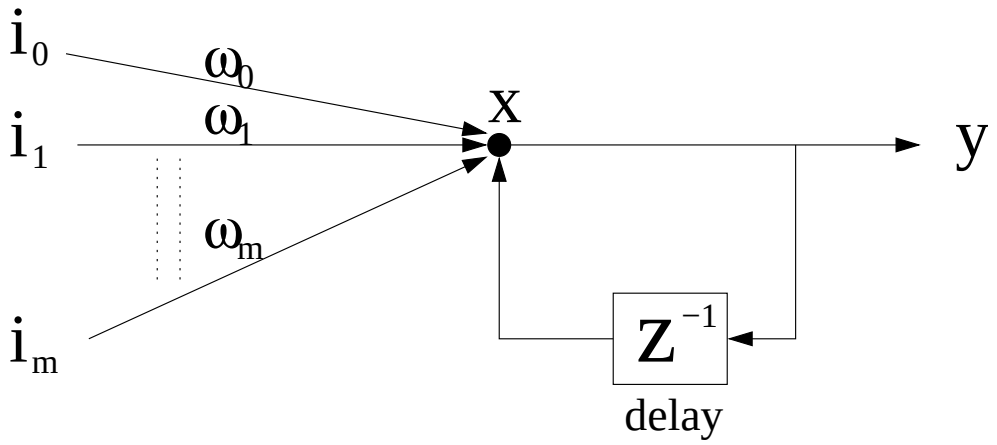
Feed-forward Neural Networks (2)

Theorem: a 3 layer (backpropagation) network can represent any continuous function if we allow an infinite number of hidden nodes.

Theorem: a 4 layer (backpropagation) network can represent any *almost continuous* function (i.e. any function with a finite number of jump discontinuities) if we allow an infinite number of hidden nodes.

Recurrent Neural Networks

the self-feedback case



(Typical) eqns. for self-feedback neurons:

$$\begin{aligned} y(t+1) &= \varphi(v(t+1)) \\ v(t+1) &= y(t) + \beta \sum_k \omega_k i_k(t) \end{aligned}$$

- can be used to solve systems of differential equations

$$\begin{cases} \dot{x} = f(x,y) \\ \dot{y} = g(x,y) \end{cases}$$

- the computation is considered to be terminated at time T if

$$\forall t \geq T \ |y(T) - y(t)| < \varepsilon$$

(i.e. the network has converged)

NNEngine

In the following discussion, we will restrict our attention to the class of A-Prolog programs satisfying the following property, that we will call property (*):

- rules are of the form

  $$a_0 \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n$$

  where $a_0, \ldots, a_n$ are *propositional* atoms

- rules cannot have an empty head

- each atom is defined by at most one rule

Notice that propositional atoms have been chosen in order to keep this presentation simple. Our results extend immediately to programs with ground atoms.

NNEngine
Building the network

Definition. Given a program, Π, such that $pred(\Pi) = \{a_0, \ldots, a_n\}$, the network associated with Π, net(Π), is defined as follows.

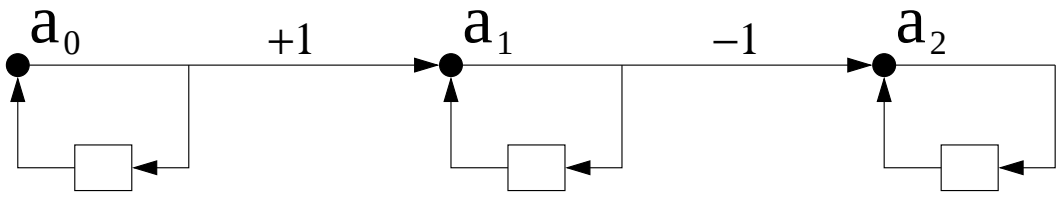
Nodes. For each atom $a_i \in \Pi$, net(Π) contains a node, x_i, with label a_i.

Arcs. net(Π) contains:

- *self-feedback* arcs for all nodes;
- an arc $x_i \stackrel{\omega_{ji}}{\rightarrow} x_j$ for every rule $r \in \Pi$ such that $a_j = head(r)$ and $a_i \in body(r)$;

$$w_{ji} = \begin{cases} +1 & \text{if } a_i \in pos(r) \\ -1 & \text{otherwise} \end{cases}$$

Example. Program $\{ a_2 \leftarrow \text{not } a_1;\ a_1 \leftarrow a_0 \}$.



The network is fully defined by the $n \times n$ weight matrix $W = (\omega_{ij})_{ij}$. Let $\mathcal{T}$ be a function that, given a program Π, returns the weight matrix of net(Π).

NNEngine

Building the network (2)

Activation function: $\varphi(v) = \begin{cases} 1 & \text{if } v > 1 \\ v & \text{if } -1 \leq v \leq 1 \\ -1 & \text{if } v < -1 \end{cases}$

Output (activation value) of a neuron:

$$\begin{aligned} y_k(t+1) &= \varphi(v_k(t+1)) \\ v_k(t+1) &= y_k(t) + \beta \min_{j \neq k} \omega_{kj} y_j(t) \end{aligned}$$

Value of y_k at time 0 (*initial activation value*):

$$y_k(0) = \begin{cases} 1 & \text{if } a_k \text{ is a fact} \\ -1 & \text{if } a_k \text{ has no definition} \\ \varepsilon_k & \text{otherwise } (-1 < \alpha_1 \leq \varepsilon_k \leq \alpha_2 < 0) \end{cases}$$

An $(n+1)$-element vector $\vec{I}$ is a *legal initial activation vector* w.r.t. Π if each of its components satisfies the above condition.

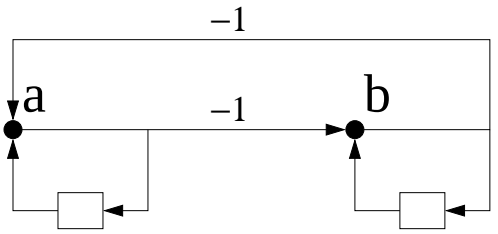
NNEngine

Dynamics of the network

Let $\vec{\varphi}$ be the extension of φ to vectors.

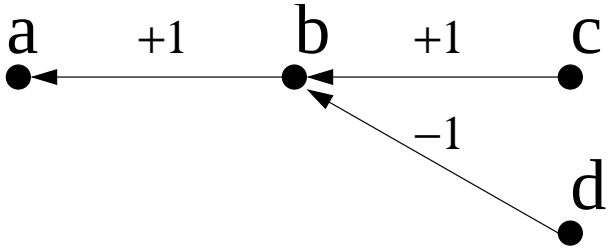
function $NN(W, \vec{I})$;
Input: a weight matrix $W = \mathcal{T}(\Pi)$ and an initial state vector $\vec{I}$.
Output: an activation vector.
$\vec{y}(0) = \vec{I}$;
$t = 0$;
repeat
$\vec{v}(t+1) = \vec{y}(t) + \beta(\ \min_j \omega_{kj} y_j(t)\ )_k$;
$\vec{y}(t+1) = \vec{\varphi}(\vec{v}(t+1))$;
$t = t + 1$;
until network has converged;
return $\vec{y}(t)$.

NNEngine
Examples

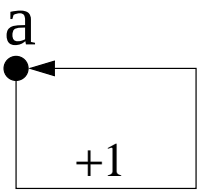


$$\begin{cases} a & \leftarrow & \text{not } b. \\ b & \leftarrow & \text{not } a. \end{cases}$$

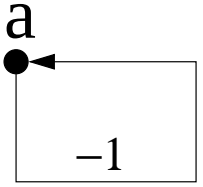
(simplified representation)



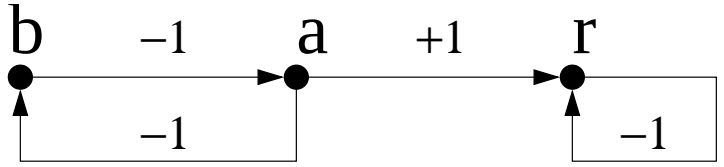
$$\begin{cases} a & \leftarrow & b. \\ b & \leftarrow & c, \text{not } d. \\ c & \leftarrow & . \end{cases}$$



$$\{ \; a \leftarrow a.$$



$$\{ \; a \leftarrow \text{not } a.$$



$$\begin{cases} a & \leftarrow & \text{not } b. \\ b & \leftarrow & \text{not } a. \\ r & \leftarrow & a, \text{not } r. \end{cases}$$

NNEngine
Properties

Definition. Let Π be a program such that $pred(\Pi) = \{a_0, \ldots, a_n\}$, and $\{x_0, \ldots, x_n\}$ be the set of neurons of net(Π).

A vector, $\vec{y}$, *determines* a set of atoms, A, if

$$A = \{a_i \mid y_i = 1\}.$$

Theorem. Let Π be a stratified program, and $W = \mathcal{T}(\Pi)$. For any legal initial activation vector, $\vec{I}$:

- $NN(W, \vec{I})$ terminates, and
- the vector returned by $NN(W, \vec{I})$ determines the answer set of Π.

Comparison with the MCN approach

Rushton, J.N. (2001) "*Compositional semantics in a localist neural network.*" Proceedings of the 2001 International Conference on Artificial Intelligence.

- ground vs non-ground programs
- negation as failure
- multiple vs single models
- one-rule definitions of atoms

Conclusions

- potentially powerful engine for the computation of answer sets of A-Prolog programs:
  - massively parallel
  - neurons compute a comparatively simple function
- integration of neural networks and logic:
  - neural network learning techniques possibly applicable to the synthesis of logic programs
  - adding prior knowledge to neural networks
  - declarative specification of neural networks

Future work:

- extending to all non-stratified programs satisfying property (*)
- extending to arbitrary A-Prolog programs.