Report Number: WUCS-91-31

1991-07-01

Converting Binary Thresholds Networks into Equivalent Symmetric Networks

Gadi Pinkas

We give algorithms to convert any network of binary threshold units (that does not oscillate) into an equivalent network with symmetric weight matrix (like Hopfield networks [Hopfield 82] or Boltzmann machines [Hinton, Sejnowski 88]). The motivation for the transformation is dual: a) to demonstrate the expressive power of symmetric networks; i.e. binary threshold networks (that do not oscillate) are subsumed in the energy minimization paradigm; 2) to use network modules (developed for the spreading activation paradigm for example), within the energy minimization paradigm. Thus optimization [Tank, Hopfield 88] and approximation of hard problems can be combined with efficient modules,... **Read complete abstract on page 2.**

Converting Binary Thresholds Networks into Equivalent Symmetric Networks

Gadi Pinkas

Complete Abstract:

We give algorithms to convert any network of binary threshold units (that does not oscillate) into an equivalent network with symmetric weight matrix (like Hopfield networks [Hopfield 82] or Boltzmann machines [Hinton, Sejnowski 88]). The motivation for the transformation is dual: a) to demonstrate the expressive power of symmetric networks; i.e. binary threshold networks (that do not oscillate) are subsumed in the energy minimization paradigm; 2) to use network modules (developed for the spreading activation paradigm for example), within the energy minimization paradigm. Thus optimization [Tank, Hopfield 88] and approximation of hard problems can be combined with efficient modules, that solve tractable sub-problems; 3)to unify a large class of networks under one paradigm. For acyclic networks we give an algorithm that generates an equivalent symmetric network that is of the same size and performs as efficiently as the original network. For the conversion of recurrent networks, we introduce several techniques; however, the generated networks may be larger (in size) then the original.

Converting Binary Threshold Networks into Equivalent Symmetric Networks

Gadi Pinkas

WUCS-91-31

July 1991

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Converting Binary Threshold Networks into Equivalent Symmetric Networks

Gadi Pinkas[1]
TR:91-31
Department of Computer Science
Washington University
Campus Box 1045
St. Louis, MO 63130
pinkas@cics.wustl.edu

Abstract

We give algorithms to convert any network of binary threshold units (that does not oscillate) into an equivalent network with symmetric weight matrix (like Hopfield networks [Hopfield 82] or Boltzmann machines [Hinton, Sejnowski 86]). The motivation for the transformation is dual: 1) to demonstrate the expressive power of symmetric networks; i.e., binary threshold networks (that do not oscillate) are subsumed in the energy minimization paradigm; 2) to use network modules (developed for the spreading activation paradigm for example), within the energy minimization paradigm. Thus, optimization [Tank, Hopfield 86] and approximation of hard problems can be combined with efficient modules, that solve tractable sub-problems; 3) to unify a large class of networks under one paradigm.

For acyclic networks we give an algorithm that generates an equivalent symmetric network that is of the same size and performs as efficiently as the the original network. For the conversion of recurrent networks, we introduce several techniques; however, the generated networks may be larger (in size) then the original.

1 Introduction

We would like to show that symmetric networks are expressive enough to contain any network that is based on binary threshold units if the network is guaranteed to converge. We start by describing a unified model for all the topologies of binary threshold units and then give algorithms to convert acyclic networks to symmetric ones without increasing the size and without loosing efficiency. Later, we discuss several approaches toward the conversion of arbitrary recurrent networks (still based on binary threshold units) into symmetric. We succeed to convert any recurrent network that converges in finite time, into a symmetric network whose global minima are exactly the stable states of the recurrent network. In the general case however, we cannot guarantee that the size of the network remains the same. Some of the techniques may generate local minima,

[1]Supported by NSF grant number: 22-1321-57136

thus, performance may degrade. However, for some recurrent networks we find efficient symmetric representation (in both size and performance), and if we are ready to pay in size, we can always find a symmetric network that performs with the same efficiency as the original network.

2 A unified model

A network with binary threshold units is a collection of zero/one units that are connected using weighted directed arcs. The network is characterized by a matrix of weights (W) that is associated with its connections; i.e., w_{ij} means that there is a connection from unit u_i to unit u_j; u_i is an "input" of u_j and u_j is an "output" of u_i. The set of inputs of u_i is denoted by I_i.

Let s be a state of the network (W); i.e., s is a vector of zero/one instantiations of the units in the network.

A binary threshold unit is a unit that has zero/one activation and computes

$$net_i(W,s) = \sum_{j \in I_i} w_{ij} s_j + w_i.$$

It updates its activation value by:

$$a_i = \begin{cases} 1 & \text{if } net_i(W,s) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

We can extend the model by adding simulated annealing [Kirkpatrick83], so that the probability of a unit to become one is:

$$P(a_i = 1) = \frac{1}{1 + e^{-net_i(W,s)/T}}$$

DEFINITION **2.1** *A network is a collection of binary threshold units, each asynchronously performs the update operation. The network is in a stable state if no more updates occur.*

DEFINITION **2.2** *A unit u_i is dependent on another unit u_j iff there exists a directed path from u_j to u_i.*

DEFINITION **2.3** *An acyclic (also called feed-forward) network is a network that has no dependency cycles; i.e., no unit depends on itself.*

DEFINITION **2.4** *A symmetric network is one that for every connection w_{ij}, the connection $w_{ji} = w_{ij}$ also exists.*

DEFINITION **2.5** *A network that is not acyclic and not symmetric is called recurrent.*

Recurrent networks (not like acyclic or symmetric) may oscillate (convergence is not guaranteed); however, in this report we are interested only in networks that converge.

3 Converting Acyclic Networks

3.1 Key Idea

Every unit in an acyclic feed forward network has weighted inputs and weighted outputs. Only the inputs affect the activation of the unit. We would like to add the symmetric output weights; i.e., for every output weight o_{ij} we would like to add an input weight o_{ji} such that $o_{ij} = o_{ji}$. The behavior of the unit should not be affected by adding this new input arcs, and we may need to scale up the original input weights to achieve this property. We'll show that in acyclic networks, it is always possible to scale the input weights, so that adding more input weights (from other units) does not change the behavior of the unit.

EXAMPLE **3.1** Assume unit u_1, with two inputs u_2, u_3 and two outputs u_4, u_5. The weights are $w_{21} = 1$ and $w_{31} = -2$ for the inputs, $w_1 = 1$ the bias, and $o_{14} = -3$ and $o_{15} = 2$ for the outputs. We would like that the output units u_4, u_5, will be used also as inputs, using the weights $o_{41} = -3$ and $o_{51} = 2$ respectfully. The behavior of u_1 should remain the same as it was before the addition; i.e., the activation of u_1 should be insensitive to units u_4 and u_5, and should react the same way as it used to do, for all the possible states of u_2 and u_3.

By scaling up the bias and the weights that are arriving from u_1 and u_2, we get $w_1 = -3.5, w_{21} = 7, W_{31} = -14$, which is insensitive to the new weights arriving from the "output" units (o_{41} and o_{51}).

We can achieve such desired property, for every unit in the acyclic network using the following idea:

An acyclic network can be looked as an acyclic dependency graph using the dependency relation defined earlier. Consider a topological order based on the dependency relation. We start with a unit on which no other unit is dependent. At least one such unit exists in every acyclic network (an output unit which is not fed to any other unit). We get a sequence of units, with the property that a unit always appears before the units it is depended on. We can now apply a scaling algorithm converting a unit's output connections (adding the symmetric connections) by scaling its inputs, in the this order. Any scale of weight will influence only units that have not yet been converted. Thus the algorithm will halt when there are no more units to convert. The following section contains the details of the algorithm and proofs of correctness.

3.2 Converting acyclic networks - Details

3.2.1 Definitions

Let u_i be a unit. We assume it has N inputs denoted by the weights $I_i = \{w_{ji} \mid j = 1...N\}$, and M outputs denoted by $O_i = \{o_{il} \mid l = 1...M\}$. We want to add

the symmetric input weights $\{o_{li} \mid o_{li} = o_{il}, l = 1...M\}$ without changing the behavior of the units.

DEFINITION **3.1** *Let $maxneg_i$ be the maximal value of negative input that the unit u_i can get:*

$$maxneg_i = max_s\{net_i(W,s) \mid net_i(W,s) < 0\}$$

DEFINITION **3.2** *Let $minpos_i$ be the minimal value of positive input that the unit u_i can get:*

$$minpos_i = min_s\{net_i(W,s) \mid net_i(W,s) \geq 0\}$$

DEFINITION **3.3** *Let $minneg_i$ be the maximum negative input that unit u_i can get from the output units (if we add the symmetric weights):*

$$minneg_i = \sum_{o_{il}<0} o_{il},$$

DEFINITION **3.4** *Let $maxpos_i$ the maximum positive input that unit u_i can get from the output units (if we add the symmetric weights):*

$$maxpos_i = \sum_{o_{il}>0} o_{il},$$

DEFINITION **3.5** *A network with weights W is equivalent to a network with weights W' (with the same units) iff for every unit u_i and for every state s of the other units (not including u_i) $net_i(W,s) < 0$ iff $net_i(W',s) < 0$.*

Intuitively, equivalence means that the corresponding units (in both networks) behave the same way to an instantiation of the other units.

3.2.2 Intuition

Given a network with weights W and a unit u_i with $\{w_{ji}\}$ inputs and $\{o_{il}\}$ outputs, we want to generate an equivalent set of weights W' that is the result of updating W and adding the symmetric output weights of u_i; i.e., we add the symmetric connections $\{o_{li}\}$ and possibly update the bias (w_i) and the input weights $\{w_{ji}\}$.

If $net_i(W,s) < 0$, then $net_{w_i}(W,s) \leq maxneg$. We would like to scale all the input weights of u_i by a positive factor of α (generating W'), so that any positive input arriving from the output units is not enough to make $net_i(W',s)$ positive; i.e., we want that for all s such that $net_i(W,s) < 0$ also $\alpha(net_i(W,s)$+(any input from the output units)< 0. But, $\alpha(net_i(W,s))$+(any input) $\leq \alpha(maxneg_i) + maxpos_i$. So, we need $\alpha(maxneg_i) + maxpos_i < 0$. Therefore,

$$\alpha > \frac{-maxpos_i}{maxneg}.$$

In the same fashion, we would like (when $net_i(W,s) \geq 0$)

$$net_i(W',s) = \beta(net_i(W,s)) \geq \beta(minpos_i) + minneg_i \geq 0.$$

Thus,

$$\beta \geq \frac{-minneg_i}{minpos_i}.$$

If we want to get both properties, we need to take the maximum of the two scales

$$\gamma = max\{\alpha, \beta\}.$$

Multiplying the bias and the input weights of unit u_i guarantees that the network created is insensitive to inputs arriving from the output units. The new network is equivalent to the original one. If we want $net_i(W',s)$ to be *strictly* less (or greater) than zero, we might compute:

$$\alpha \geq \frac{-\epsilon - maxpos_i}{maxneg_i}$$

$$\beta \geq \frac{\epsilon - minneg_i}{minpos_i}$$

where ϵ is a small positive value.

EXAMPLE **3.2** $w_{21} = 1; w_{31} = -2; w_1 = -1/2; o_{41} = -3; o_{51} = 2; \epsilon = 1/2$

$$\alpha = \frac{-1/2 - 2}{-1/2} = 5$$

$$\beta = \frac{1/2 + 3}{1/2} = 7$$

$$\gamma = max\{5, 7\} = 7$$

The scaled network contains $w_{21} = 7 \times 1; w_{31} = -7 \times 2; w_1 = -7 \times 3.5; o_{14} = o_{41} = -3; o_{51} = o_{15} = 2;$.

The only problem left is to compute β when $minpos_i = 0$. When this case occurs, we can add the following constant to the bias of unit u_i:

$$b = \frac{\mid maxneg_i \mid}{2}$$

Updating the bias ($w_i = w_i + b_i$) generates an equivalent set of weights. The change in the bias is too small to affect when $net_i(W,s) \leq maxneg_i < 0$ (since $net_i(W,s) + b$ is still less then zero). Also, the change in the bias, causes no problems when $net_i(W,s)$ is positive, since $net_i(W,s) + b$ is also positive. The result is an equivalent network with $minpos_i = b$ instead of zero, and $maxneg_i = -b$ instead of $-2b$.

3.2.3 Algorithm

Arrange the units of the acyclic network in a sequence according to a topological order; i.e., if u_i is dependent on u_j, then u_i appears before u_j.
For all the units u_i in this sequence, perform the procedure: CONVERT(u_i, W).

CONVERT(u_i, W):
Compute $minpos_i, maxneg_i, maxpos_i, minneg_i$ and $b = -maxneg_i/2$.
If $minpos_i = 0$ then $w_i = w_i + b$; $minpos_i = b$; $maxneg_i = -b$;
$\alpha = \frac{\epsilon - minneg_i}{minpos_i}$
$\beta = \frac{\epsilon + maxpos_i}{-maxneg_i}$
$\gamma = max\{\alpha, \beta\}$
for every output unit u_l of u_i, introduce the new arc $o_{li} = o_{il}$
for every input unit u_j of u_i, update $w_{ji} = \gamma w_{ji}$ and the bias $w_i = \gamma w_i$.

LEMMA **3.1** *The network that is generated by updating the bias $w_i = w_i + b$ (when $minpos = 0$) is equivalent to the original network (before the update).*

Proof:
$net_i(W, s) < 0 \Rightarrow net_i(W, s) + b < 0$
$net_i(W, s) \geq 0 \Rightarrow net_i(W, s) + b \geq 0$
□

LEMMA **3.2** *The network generated by scaling the input weights and add the symmetric connections of the output weights, is equivalent to the original network.*

Proof:
$net_i(W, s) < 0 \Rightarrow net_i(W', s) = \gamma(net_i(W, s) + \sum_j o_{ji} < \beta(maxneg_i) + maxpos_i < 0$
$net_i(W, s) \geq 0 \Rightarrow net_i(W', s) = \gamma(net_i(W, s) + \sum_j o_{ji} \geq \alpha(minpos_i) + minneg_i < 0.$
□

LEMMA **3.3** *If the network is acyclic, then the algorithm halts and the network generated is symmetric and equivalent to the original one.*

Proof:
All units are converted. All output weights become symmetric. In acyclic nets all input weights are also output weights, therefore all weights are converted into symmetric weights. Every update keeps the network equivalent to the original, therefore the result is an equivalent symmetric network.
□

Thus, we can convert an acyclic network into an equivalent symmetric one with the same size and topology. The new network performs with the same efficiency as the original one.

3.3 Computing *maxneg* and *minpos* is hard

The problem of computing $maxneg_i$ and $minpos_i$ is NP-hard.

Let u_i a unit with N inputs. Given the set of weights $W = \{w_{ji}\}$, a bias w_i and a constant c, the decision whether there exists an assignment s for the input units, such that $net_i(W, s)$ is negative and $net_i(W, s) \geq c)$, is NP-complete. The proof is by reduction from the knapsack problem. We can conclude therefore, that the corresponding search problem (finding the maximum of all negative values) is NP-hard.

Our algorithm to convert an acyclic network is therefore inefficient, when the connectivity of the network is in the order of the number of units. However, if the connectivity is a constant (every unit receives input from at most k other units, where k is a small constant), then the algorithm performs efficiently in linear time.

4 Recurrent networks

In general, recurrent networks cannot be converted into symmetric ones. Examples for recurrent nets that cannot be represented as symmetric are networks that oscillate. A symmetric network can not oscillate since it always converges into a local (global) minimum of its energy function. However, we may want to convert recurrent networks that converge (never oscillate). We'll see that it is always possible to do so, but we might pay in size.

We introduce four techniques: One that may generate large networks but efficiency is guaranteed. The second technique may not always halt but if it halts, it generates an efficient network with the same size and performance as the original network. The third approach generates networks whose energy function is guaranteed to have global minima that is exactly the set of stable states of the original recurrent network; but there may be spurious local minima (that are not global). The fourth approach is a combination of the second and the third techniques. It has better chances of generating compact networks with only few spurious local minima.

4.1 Duplicating units

Assume that a recurrent network converges within l steps. We can duplicate the network k times and avoid cycles by generating l layers. We get an acyclic network that can then be converted into symmetric. Thus, we may generate a large network but its performance is the same as the performance of the original network. Theoretically, if the recurrent network converges in polynomial time (computation is tractable) the size and performance of the equivalent symmetric network is also polynomial. The problem is that we may not know the worse case l, and even if we do we may have only an upper bound that is too large to be practical.

4.2 An optimistic use of the algorithm for acyclic networks

Another approach is to try and use a variation on the algorithm developed for acyclic networks:

Repeat until all units are symmetric: Select a unit u_i with asymmetric weights and transform it using CONVERT2(u_i, W) procedure.

CONVERT2(u_i, W):
Perform CONVERT(u_i, W).
If the input weights of u_i were scaled then, recursively perform CONVERT2(u_j, W) for every input u_j of u_i (even if u_j was previously converted).

The algorithm may not always halt. However, if it does halt, then we get a symmetric network with the same size and performance as the original one. A sufficient condition for the algorithm to stop is that the network is acyclic. It is not a necessary condition. To find necessary conditions for the algorithm to halt is an open problem.

4.3 Constraints on stable states

If the recurrent network converges into a stable state, it means that eventually units are not updated any more. If we could find what are the constraints on these stable states, we would be able to express them as energy function, thus, obtaining a network whose global minima are exactly the stable states.

when a unit is in a stable state it means that either its activation is one and $net_i(W, s) \geq 0$ or its activation is zero and $net_i(W, s) < 0$. There exists, therefore, a boolean function φ_i of the k inputs of u_i, such that $\varphi_i \leftrightarrow u_i$ holds. We can find φ_i by exhaustively search for all the positive values of net_i. Once found, the formula $\varphi_i \leftrightarrow u_i$ can be treated as a hard constraint that every stable state must satisfy. We can find such formula for each of the units in the network . This set of formulas is a set of hard constraints that must hold iff the network is in stable state. Using algorithms developed in [Pinkas 91a], we can convert these logical constraints into symmetric networks, whose global minima are exactly the stable states. We may generate a k-order ([Sejnowski 86]) symmetric network of the same size of the original network, or a standard quadratic network (second order connections) with $O(2^k N)$ hidden units. If k is a small constant, the size of the network that is generated is linear in N. However, there may be a loss of efficiency as a result from spurious local minima that may be introduced by the constraints.

Note that when we use the technique on a network that oscillates, we generate a network that always converge (thus not equivalent). The oscillations are caused by constrains that cannot be satisfied, but the symmetric network will

converge to a state that minimizes the violation; i.e., minimizes the weighted sum of the constraints that are violated (see [Pinkas 91b]).

4.4 Combination of the second and third methods

We can decompose any recurrent network into two parts: 1) a set of disjoint acyclic sub-networks that cover all the units and 2) a set of arcs (that do not appear in any of the acyclic networks of the first part) that cause cycles when added to one of the acyclic sub-networks. The idea is to use the acyclic-to-symmetric algorithm for the acyclic sub-networks and then add stable-state-constraints for those units that could not been converted because they contain an output arc that causes a cycle. In this way, we may efficiently convert a large portion of the network and minimize the use of the third technique.

We first decompose our network to a subset of disjoint acyclic sub-networks by disconnecting units from the output arcs that cause cycles, and making nodes that are output of such arcs, leaves of the sub-networks[2]. One way to do it is to look at the graph (of the network) where all the arcs are reversed and find a spanning tree (the back-arcs cause cycles). The next step is to disconnect the units with back-arcs from their outputs (from their inputs in the original network), thus, transforming these units into leaves of an acyclic graph, and decomposing the spanning tree into further acyclic sub-graphs.

We perform the algorithm to convert acyclic to symmetric on every acyclic sub-network that has been generated. We then re-connect the sub-networks using the (possibly scaled) asymmetric connections removed earlier, and we use the constraint satisfaction technique to express the conditions that such units (ex-leaves with asymmetric inputs) are in a stable state. When we add a constraint, we have to be careful to scale its weights; this is done by scaling the weights generated by the constraint so the net input will be greater (or less) then the maximal positive (or minimal negative) input arriving from the now symmetric output connections of the ex-leave.

References

[Hinton, Sejnowski 86] G.E Hinton and T.J. Sejnowski "Learning and Relearning in Boltzman Machines" in J. L. McClelland, D. E. Rumelhart *"Parallel Distributed Processing:Explorations in the Microstructure of Cognition"* Vol I pp. 282 - 317 MIT Press 1986

[Hopfield 82] J.J. Hopfield "Neural networks and physical system with emergent collective computational abilities," *Proceedings of the National Academy of Sciences* USA, 1982,79,2554-2558.

[2] As heuristics, try and make the number of such leaves to be as small as possible and also minimize the number of input connections arriving into these nodes.

[Hopfield 84] J.J. Hopfield "Neurons with graded response have collective computational properties like those of two-state neurons". *Proceedings of the National Academy of Sciences* USA, 1984,Vol 81, pp. 3088-3092.

[Tank, Hopfield 86] J.J. Hopfield, D.W. Tank "Neural Computation of Decisions in Optimization Problems" *Biological Cybernetics*, Vol 52, pp. 144-152.

[Kirkpatrick83] S. Kirkpatrick, C.D. Gelatt, M. P. Vecchi, "Optimization by simulated annealing," *Science, 1983, 220, 671-680*

[Pinkas 91a] G. Pinkas, "Symmetric neural networks and propositional logic satisfiability", to appear in *Neural Computation* vol 3-2, 1991.

[Pinkas 91b] G. Pinkas, " NM reasoning and inconsistency using symmetric connectionist networks", to appear in the *proceedings of IJCAI-91*,Sydney, August 1991.

[Sejnowski 86] T.J. Sejnowski, "Higher-order Boltzman machines", Neural Networks for Computing", *Proc. of the American Institute of Physics", 151*, Snowbird (Utah) pp 3984, 1986.