Complex Systems **2** (1988) 191–196

Steepest Descent Can Take Exponential Time for Symmetric Connection Networks*

Armin Haken
Michael Luby
Department of Computer Science, University of Toronto, 10 King's College Road, Toronto, M5S 1A4, Canada

Abstract. We construct a family of symmetric weight connection networks that take exponential time to reach a stable configuration when the sequential steepest descent update rule is used.

1. Introduction

A connection network is a graph together with assigned edge and node weights. If the edges in the graph are undirected, then it is called a symmetric connection network. The nodes of a connection network can be in one of two states called 1 and -1.[1] At each node N, the influence function is defined to be the sum over all edges to N, of the quantity "edge weight times the state of the node at the other end of the edge", plus the node weight of N. A node is called happy if it is in either state and the influence at the node is zero, if it is in state 1 and the influence is positive, or if it is in state -1 and the influence is negative. Otherwise, the node is called unhappy. A configuration of the network is a specification of a state for each node in the network. A stable configuration of the network is a configuration where every node is happy.

One of the goals in connectionist computing is to design good update rules for changing the states of the nodes which reach a stable configuration starting from an initial configuration. Two essential properties of the update rule are that it should be simple and that it should reach a stable configuration very quickly from an input configuration. One natural candidate for an update rule is the greedy or steepest descent rule, which is to always change the state of a node in the network which is under the greatest magnitude of influence and also unhappy. As discussed in [3], any update rule which only

*Supported by grants from the Natural Sciences and Engineering Research Council of Canada, A-8092 and G-1366, and the University of Toronto.

[1]Other authors let the two states be 0 and 1. There is an easy conversion from a network using -1 and 1 to one using 0 and 1 for the states such that the stable states in the two networks are exactly the same [5].

changes the state of unhappy nodes ensures that a stable configuration will be eventually reached starting from any input configuration.

In this paper, we construct a family of networks for which there is an initial state such that the steepest descent update rule has to be applied a number of times that is exponential in the size of the network.

2. Related work

Connectionist networks have recently been widely studied as a possible parallel model of computation. For example, [3] suggests using symmetric connection networks as a storage device for associative memory. There are two foci in this work; the first is to fix a "learning update" rule which changes the weights on the edges and nodes so that stable configurations correspond to information that is stored in the network, and the second is to fix a "retrieval update" rule for recalling stored information, which is a rule for going from an arbitrary input configuration to a stable configuration. To put our result in the proper light, it is known that when the node and edge weights are small (i.e. the absolute values of the edge and node weights are polynomial in the number of nodes in the graph) the greedy update rule always reaches a stable configuration in polynomial time [3]. Thus, our examples necessarily use large node and edge weights, whose values can nevertheless be expressed using a polynomial number of bits. It is known that the problem of finding a stable configuration in a connection network with directed edges is NP-hard [1,4].

Our result does not show that there is no polynomial time algorithm to find a stable configuration in a symmetric connection network, it only shows that the "obvious" algorithm of using steepest descent can take exponential time. In fact, although no polynomial time algorithm is known in general to find a stable configuration, there is evidence that this symmetric problem is not NP-hard, suggesting that perhaps there is a polynomial time algorithm [2,4].

3. The construction

We construct a family of networks together with an initial configuration such that the steepest descent update rule takes exponential time to reach a stable configuration. The intuitive idea behind the construction is "flows of unhappiness": consider the network shown in figure 1. In all figures, edge weights are written near the edge and node weights are written in parentheses near the node. All missing node weights are zero. If all nodes are in state -1, then they are all happy except for the leftmost. If the state of the leftmost node is switched, the next node to the right becomes unhappy. When that node is switched, the unhappy node is the next one to the right. Note that the unhappiness moves left to right but not the other way. We call such a situation a "flow of unhappiness". A larger network that is being subjected to the update rule may have flows of unhappiness in various parts.

Which flow moves when the steepest descent update rule is applied depends on the magnitude of the influence involved, which in the above example is the difference in successive connection weights. Two connection weights in sequence that are almost the same will serve to delay the flow of unhappiness compared to flows in parts of the network where there is a greater difference between the weights.

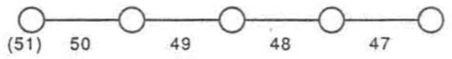

Figure 1: A network in which unhappiness flows from left to right.

A path for a flow of unhappiness can split into two as shown in figure 2. All nodes are initially in state -1. In this example, a flow moving from left to right becomes two flows. It is not predetermined which branch will flow faster, since the edge weights are unbalanced by 1 at all of the nodes.

The final bit of intuition that motivates the construction is the "two into one flow valve." This valve is a piece of the network with two paths leading in and one path out. A sequence of state changes along either of the input paths propagates through to cause a sequence of state changes in the outgoing path. A piece of network that computes the XOR of two "input" nodes and sets an "output" node meets the specifications.

Figure 3 shows a piece of connection network which performs this function which we call an XOR module. Suppose all nodes shown are happy. If one of the nodes on the left changes state due to state changes further left, then after a sequence of update rule applications the rightmost node changes state and all nodes in the module are once again happy. If a node on the left changes state again then after a few update rule applications the rightmost node changes state once again and all nodes within the module are happy. Thus, using the steepest descent update rule, changing the state of a node on the left eventually causes the rightmost node to change state in a XOR module.

The networks that exhibit the exponential settling behavior consist of a series of XOR modules with the input paths of each module split from the output path of the module to the left. The leftmost module simply has its input paths split from one node. That leftmost node is in a sense the source of the flow of unhappiness through the network. Figure 4 shows a sequence of two XOR modules put together.

The weights on the edges and nodes of the rightmost module are as in figure 3. For the other modules, the weights are all 20 times the weights of the module to the right. The leftmost node has weight $220 * 20^n$, where n

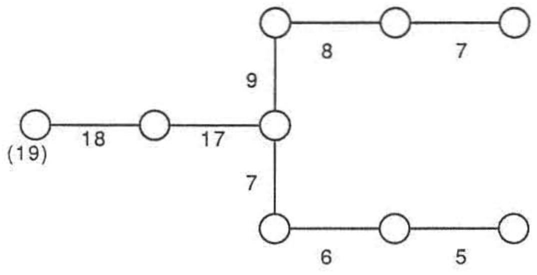

Figure 2: A flow of unhappiness splits into two.

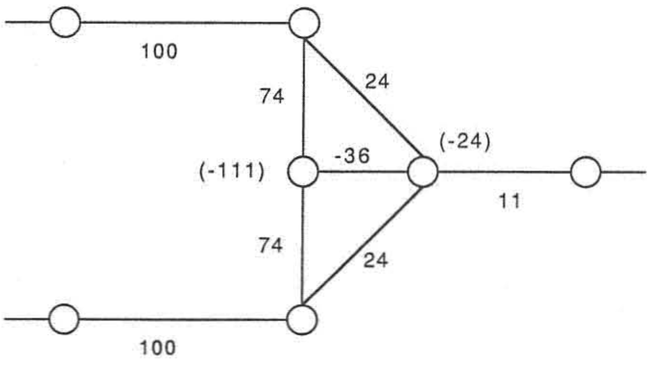

Figure 3: The XOR module that channels unhappiness from two inputs into one output.

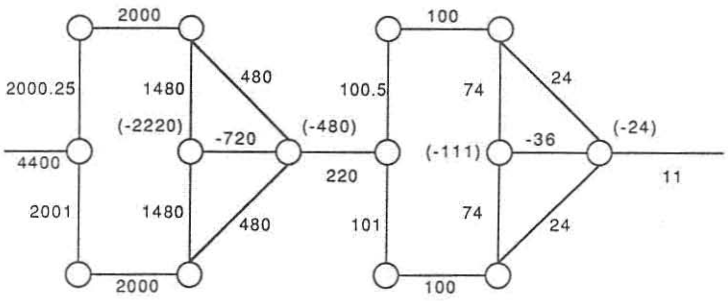

Figure 4: Relationship between the weights of two successive XOR modules with delay edges shown.

is the number of XOR modules. For each XOR module, there are delays (in the form of two edge weights that are almost equal) on the input paths. The delay on the lower input of each module is an imbalance of 1 between weights. The upper delay is caused by an imbalance of 1/2 at the rightmost module. For the other modules, the weights are chosen to give an imbalance at the upper delay node that is half of the corresponding imbalance of the module to the right. The network is stable except for the leftmost node when all nodes are in state -1.

For such a network consisting of n XOR modules, if all nodes except for the leftmost are happy, the rightmost node changes state 2^n times during the settling process. This claim can be verified using induction on n: In case $n = 0$ the rightmost node and the leftmost node are the same and the claim is trivially true. Assume the claim for $k-1$ modules. Now consider the network consisting of k XOR modules. When the leftmost node switches, the two delay nodes in front of the leftmost XOR are unhappy. The top delay node will not switch state until all other nodes in the network are happy, since no other node in the network is capable of being unbalanced by as little as 2^{-k}. The bottom delay node changes state, and the output node at the leftmost XOR changes state, as can be checked by considering the possible stable states of the XOR and the way they change when one of the inputs changes. The next node to the right of that output node is then unhappy, so it changes state. The situation to the right of that first XOR module is just the case for $k-1$ modules, so by inductive hypothesis the rightmost node changes state 2^{k-1} times as that part of the network stabilizes. Note that the upper left delay node is still unhappy throughout these state changes to the right, but only gets switched after all other nodes are happy. When that delay node eventually switches state, the output to the leftmost XOR again switches state and as before the inductive hypothesis applies to the portion of the network to the right. Therefore the rightmost node switches state 2^{k-1} times again, making a total of 2^k switches. Thus the network takes more than 2^k applications of the greedy update rule to stabilize when started with all nodes in state -1.

4. Conclusions and open problems

We have exhibited a family of symmetric connection networks together with input configurations which take exponential time to reach a stable configuration when the steepest descent update rule is used. For this same family, it can be easily shown that the probabilistic update rule which randomly selects an unhappy node with probability directly proportional to the magnitude of influence on the node runs in expected exponential time. On the other hand, consider the following probabilistic update rule which was suggested in [3]: randomly choose an unhappy node (independently of the magnitude of the influence on the node) and change its state. For the connection networks and the input configurations we give in our example, this update rule has expected running time which is polynomial. We do not know if this is true for all symmetric connection networks and all input configurations.

5. Acknowledgements

We thank both Gail Godbeer and John Lipscomb for many helpful discussions concerning this work.

References

[1] N. Alon, "Asynchronous threshold networks", *Graphs and Combinatorics*, 1 (1985) 305–310.

[2] G. H. Godbeer, "The Computational Complexity of the Stable Configuration Problem for Connectionist Models", M.Sc. Thesis, Department of Computer Science, University of Toronto, 1987.

[3] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proceeding National Academy of Sciences*, **79** (1982) 2554–2558.

[4] J. Lipscomb, "On the computational complexity of finding a connectionist model's stable state vectors", M.Sc. Thesis, Department of Computer Science, University of Toronto, 1987.

[5] B. Selman, "Rule-Based Processing in a Connectionist System for Natural Language Understanding", Technical report CSRI-168, Computer Systems Research Institute, University of Toronto, 1985.