In Proceedings of the *1988 IEEE Systems Man and Cybernetics Conference*, pp. 681-684, 1988.

Digital Neural Networks

Tony R. Martinez
Computer Science Dept., 230 TMCB
Brigham Young University, Provo, Utah 84602

To Appear in the Proceedings of the 1988
IEEE International Conference on System Man and Cybernetics

Abstract: Demands for applications requiring massive parallelism in symbolic environments have given rebirth to research in models labeled as *neural networks*. These models are made up of many simple nodes which are highly interconnected such that computation takes place as data flows amongst the nodes of the network. To present, most models have proposed nodes based on simple analog functions, where inputs are multiplied by weights and summed, the total then optionally being transformed by an arbitrary function at the node. Learning in these systems is accomplished by adjusting the weights on the input lines. This paper discusses the use of digital (boolean) nodes as a primitive building block in *connectionist* systems. Digital nodes naturally engender new paradigms and mechanisms for learning and processing in connectionist networks. The digital nodes are used as the basic building block of a class of models called *ASOCS* (Adaptive Self-Organizing Concurrent Systems). These models combine massive parallelism with the ability to adapt in a self-organizing fashion. Basic features of standard neural network learning algorithms and those proposed using digital nodes are compared and contrasted. The latter mechanisms can lead to vastly improved efficiency for many applications.

Introduction

Higher demands for both computing power and utility, coupled with the onset of new technologies, has caused a resurgence of research in the area of architectures inspired by nervous systems. This field of *neural networks,* or *connectionist computing*, comprises highly interconnected architectures of relatively simple computing nodes, which function in a parallel fashion. These models are also set apart by their ability to *learn* a given functionality through training, rather than the explicit *programming* required by the traditional von Neumann machine. The type of basic node used in a connectionist network has a strong influence on the mechanisms of learning and processing of the overall model. The majority of current neural network models use a standard atomic mechanism for the connections between and the functioning of nodes. This mechanism is comprised of boolean signals between nodes which are multiplied by real valued *weights*, and then summed at the nodes [2,3,8,9]. This value is then treated by higher level nonlinear functions and mechanisms which vary between models and which give them their unique characteristics. This paper discusses the differences incurred when the basic atomic mechanism described above is replaced by a purely digital technique. This leads to new ways of both processing and learning in connectionist networks which provide significant improvements for many applications.

Neural Networks with *Standard* Nodes

As mentioned above, the majority of current neural network models use an atomic mechanism entailing a linear multiplication of the output of a node and a weighting factor. Figure 1 shows a generic representation of this atomic mechanism.

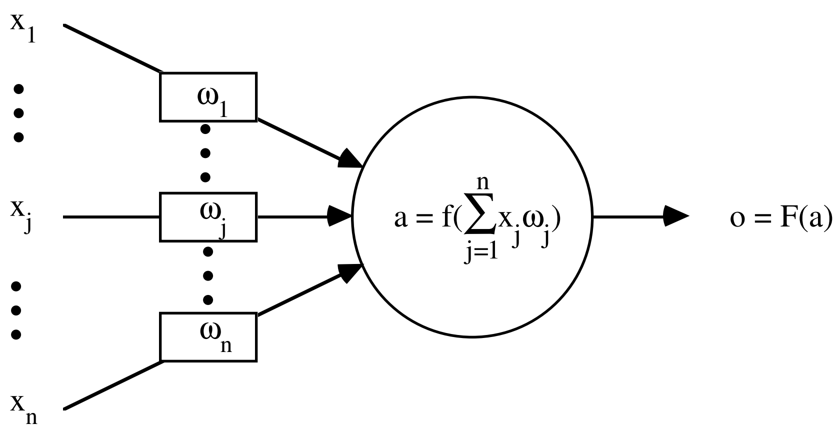


Figure 1 - Representation of analog mechanism

The model of a single node consists of x_n inputs which are typically boolean or real values. The ω_n weights are typically real numbers and each x_j is multiplied by its corresponding ω_j before entering the actual node. At the node these values are summed together giving a real valued total. The above is the linear aspect of most nodes. The summed total may then be the parameter of some function *f* whose result is the *activation (a)* of the unit. For a simple linear unit this *f* is just the identity function. Common nonlinear functions used to calculate the node activation include the threshold function, where the output is 1 if the summed total is greater than some threshold (otherwise it is 0), sigmoid function (also called the squashing function), and stochastic sigmoid functions. The *output (o)* of the unit is typically the same as the activation. However, the output may be some function *F(a)* of the current activation. The topologies into which many nodes combine to make a neural network include feedforward (where all communication is in one direction), feedback (a feedforward net where some outputs are connected back to previous nodes), and bidirectional, where a connection between nodes carries input and output in both directions. Most current bidirectional networks are symmetric, in that the weight on a line is the same for inputs going in either direction.

The basic function of a single node, and typically of the entire network, is to classify a set of input patterns into a set of output states. A network is trained to perform a set of classifications by use of a *training set*. A training set is composed of a list of input vectors, together with the desired output vector for each input vector, which the network should *learn*. An input vector is applied at the input of a network, and the consequent output of the network is compared with the goal output. If they are the same, no change is made to the weights of the network. However, if the output is incorrect the weights of the network are adjusted in a fashion which will decrease the magnitude of the previous error. Another input vector from the training set is then tested with possible weight adjustments, and the process of cycling through the training set continues until all patterns in the set can be classified without an error. (Alternatively, presentation of training patterns could continue until the error rate is within some set value.) This method of closing in on a desired goal through iterative changing of parameters is called *convergence.* One aspect of convergence algorithms with training sets is that after weights are adjusted to fit the current pattern, the network may then no longer correctly classify previous patterns already presented in the set. We call this phenomenon *unlearning*. This is partially why it is necessary to iterate many times through the training set before convergence to a correct network is attained.

The adjustment of weights is typically controlled by an equation of the form

$$\Delta\omega_{ij} = cE_p o_i ,$$

where $\Delta\omega_{ij}$ is the change in the weight on the line from node i to node j, c is a learning constant, o_i is the output from node i, and E_p is a measure of the error for the pattern p. For a network with a single layer of adaptable weights, such as Rosenblatt's simple perceptron or Widrow's Adaline [8,11], E_p is simply ($t_{pj} - o_{pj}$), where t_{pj} is the target output of node j and o_{pj} is the actual output after presentation of a training pattern. Convergence theorems have been proven for this class of networks stating that if there is a solution for the training set, then this learning rule will converge to a correct network in finite time, if each pattern in the training set is repeated in finite time. The class of patterns which this class of network can classify are exactly the *linearly separable* classifications. For boolean inputs, the number of total possible boolean classifications grows as 2^{2n}, where n is the number of inputs (or features). However, the number of linearly separable (*LS*) functions grows as

$$LS(P,n) = 2\sum_{i=0}^{n} \frac{(P-1)!}{(P-1-i)!i!} \text{ for } P > n \quad \text{and } 2^P \quad \text{for } P \le n ,$$

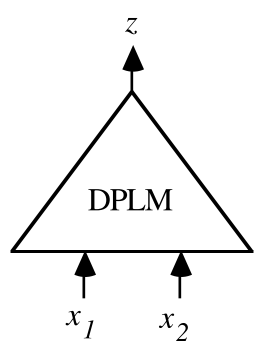
where n is the number of inputs and P is the number of patterns to be classified [7]. Note that for a general classifier P is the total possible patterns, which equals 2^n for the boolean case. In this case the exponential growth of the total possible functions far exceeds the growth for the LS functions, and the ratio of LS over possible functions quickly approaches 0 as n grows.

Models to perform more or arbitrary mappings have been sought through use of multi-layer networks. Multiple layers of linear summing units do not help since any number of linear weight matrices can be combined into a single weight matrix. Thus, no additional functionality is gained through multi-layer linear units. A multi-layer network of units as described above, with a nonlinear f function can do any arbitrary classification. However, learning algorithms with proven convergence theorems have not yet been found for this class of network. Recently, multi-layer learning algorithms have been put forth [2,9] where empirical results have shown promising potential. Back-Propagation, for example, uses a gradient descent algorithm on the squared sum error of the outputs. A differentiable sigmoid activation function is used at the nodes allowing derivation of a gradient descent algorithm where *hidden* nodes receive an error signal recursively from the nodes to which they output. It follows that this is a hillclimbing algorithm which can and does get stuck in incorrect local minima. These models have the disadvantage that after training there is no guarantee that they have converged to a correct function. Even with a learning scheme with proven finite time convergence, one cannot know when correct convergence is attained without cycling through the complete training set without errors. I call this the *stochastic* nature of the current models.

Connectionist models typically have two modes of operation. The *processing* mode, where data transformation and pattern classifications are taking place, and the *adaptation* mode, where learning takes place. One important aspect of proposed learning algorithms is their *local* processing capability. Each node (or weight) uses only local information and values received directly from the nodes to which they are connected to compute required changes. Thus, highly parallel implementations of these networks are possible, where high-speed parallel execution can take place in both the processing and adaptation phases.

Digital Nodes and Connectionist Networks

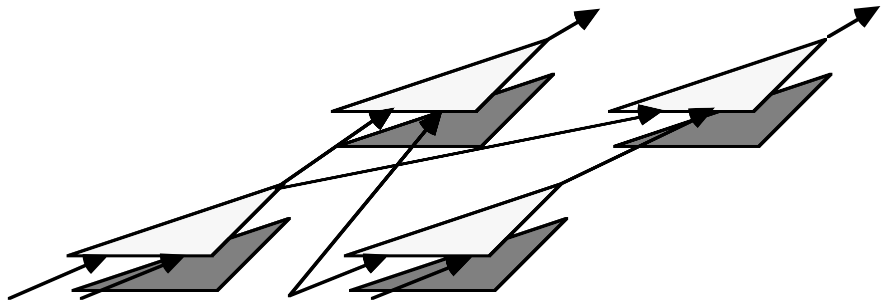
As an alternative to the weight summing units, programmable gates which do direct boolean functions on their inputs have been proposed [10]. The basic gate in this family of dynamic programmable logic modules (DPLM) is the two input single output implementation.



The inputs and outputs are all boolean and the node can be set to any one of the 16 boolean functions of 2 inputs. A threshold gate can compute the 14 linearly separable boolean functions of two inputs, but not the *exclusive-or* and the *equivalence* function. A network of threshold units, however, can solve any arbitrary boolean function. Therefore, the uniqueness of the DPLM is not found in its potential functionality, since a network of threshold units is equivalent, but in the new styles of learning it engenders. Adaptation is accomplished by changing the function of the node, rather than weights on the input lines. Because of this, gradient seeking iterative algorithms which make small changes on weights are not natural.

A class of models which have the DPLM as the atomic unit has been proposed, with the name of *Adaptive Self-Organizing Concurrent Systems (ASOCS)* [5]. Very briefly, an ASOCS model consists of an arbitrary network of

DPLM's, each with two inputs and one output which can connect to multiple nodes allowing both feedforward and feedback mechanisms. During execution the DPLM's process boolean input into boolean outputs. Each DPLM is shadowed by a control unit which is active during the adaptation phase, and which can change the function of its corresponding DPLM. Thus the total network is made up of the DPLM plane (shown on top and in lighter shade) and the control plane.



The input to the system is incremental in time, and differs slightly from the training set mechanism. The input is actually a rule, called an *instance*, which states that if a given conjunction of boolean input variables is true, then a specified output variable should be set. Examples of instances are shown below.

$$x_1 x_2 \rightarrow z_1 \qquad x_2 \overline{x_4}\,\overline{x_5} \rightarrow \overline{z_2} \qquad x_4 z_2 \rightarrow \overline{z_3}$$

Note that the antecedent of the instance need only contain those input variables that are deemed critical, rather than having to specify all possible inputs of an environmental state. Thus the instance $x_1 x_2 \rightarrow z_1$ states that if x_1 and x_2 are on then z_1 should come on regardless of all other input variables.

Instances are input incrementally and the totality of instances is called the *instance set*. A new instance may contradict all or a portion of a previous instance, in which case the new overrides the contradicted portion. The new instance is broadcast to the control plane of an ASOCS model, and each control unit is able to make local decisions on how to change the function of its corresponding DPLM and how it connects to other nodes. After this adaptation phase the system will correctly fulfill the modified instance set and a new instance can be entered. There is no time at which learning can no longer take place, since new instances can arbitrarily be input at any time in the life cycle of the model. Features of the learning mechanism are discussed in the next section as they are compared with standard gradient following schemes. Detailed presentation of ASOCS models is found elsewhere [4,6].

Comparison of Standard and Digital Models

It cannot be stressed too highly that the real differences being discussed in this paper are not so much the potential functional variances of summed weight nodes versus programmable boolean gates, but rather the differences in the types of learning mechanisms they naturally engender. In this section, critical features of the learning mechanisms of the two models are compared and contrasted.

A first question is what kind of mappings, or classifications, can a model potentially accomplish. (For simplicity we assume boolean inputs and outputs). We saw that the purely linear model and the models with only one layer of weights can only solve the linearly separable classifications, regardless of the learning mechanism used. Multi-layer networks with nonlinearities can solve all arbitrary mappings. However, they are still constrained by their learning algorithms. This is partially due to the gradient descent mechanism which for certain functions can always end up in local minima. As opposed to the single layer case, this is not a deficiency of the network itself, but of the mechanism of learning. The same topology may or may not solve a given function depending on the initial setting of weights, order of training, or other parameters. Some seek to escape the local minima through simulated annealing or other stochastic mechanisms [2], but convergence to a correct network remains a probabilistic endeavor.

In contrast, the ASOCS model is not only capable of arbitrary mappings, but the learning mechanism, which does not depend on a gradient following scheme, guarantees an immediate and correct mapping at all times after the input of a new instance.

A second critical feature is the time necessary to converge to a solution. In the weight changing algorithms the training set is continuously cycled through until a solution is found. Many cycles are typically required before convergence. This occurs for two basic reasons. First, each change in weights typically only modifies the network to come closer to the solution of the current pattern. It might take many cycles of just one pattern to reach its solution. This effect can be mitigated in large degree through more complex weight adjustment schemes. However, the second problem is not so easily remedied. This, as stated above, is unlearning, which is the tendency of weight change for one pattern to cause the model to no longer discriminate earlier patterns. Unlearning worsens if one attempts larger weight changes to remedy the first problem of gradual convergence. This all leads to extreme learning times. As an example, training of the simple 2 input XOR function is reported in [9] as taking 558 sweeps through the 4 input patterns.

In ASOCS, an instance need only be presented to the system one time. Adaptation takes place and convergence is guaranteed. Also, the network continues to correctly discriminate all instances previously presented. Only the portion of the instance set directly contradicted by the new instance is no longer fulfilled. Since adaptation does not cause the *unlearning* situation discussed above, there is no need to specify a training set a priori. Instance are entered at any time during the execute-adapt life cycle of the model. Due to its distributed and self-organizing learning scheme, the ASOCS model is also able to accomplish the adaptation necessary after an instance input in time linear with the depth of the network. Thus, the relative speed of learning actually increases as the network gets larger.

One of the important claims we hope to fulfill in connectionist computing is that of *generalization*. This is the phenomenon that after training of a network, it will with high probability respond correctly to input patterns which it has not been trained with. In weight summing algorithms, similar input vectors typically generate similar outputs. The generalization is then in terms of the similarity of inputs; (an example measure is that of hamming distance). However, this is but one type of generalization. Correct generalization depends on the specific application. In the ASOCS scheme, generalization is attained through the use of instances, which are rules that only specify the critical input variables, rather than the entire input vector. One of the apparent powers of natural nervous systems is the ability to discriminate the current important inputs from the massive barrage of total inputs. If one puts his hand to a hot stove, immediate retraction should take place regardless of the many other impinging input variables. Instances allow a natural mechanism to specify critical variables for specific situations, and generalization takes place in that many possible input vectors impinging during processing can match the single rule which has specified the critical variables.

Although, the scope of this paper does not allow a comprehensive study, other criteria worthy of comparison and consideration include number of required nodes, fault tolerance, sequential mechanisms, and ease of implementation. In fact, the availability of a maturing field of digital VLSI was one of the initial motivations for starting this work. Fabrication of initial ASOCS test chips is currently underway [1].

Conclusion

This paper has sought to compare and contrast some of the critical features of learning mechanisms engendered for connectionist architectures by the use of weight summing nodes versus programmable boolean gates. The compared features include limits on learning, speed of learning, and generalization. This paper is not meant to criticize any type of models or learning mechanisms. Indeed, it is becoming increasingly apparent that different types of connectionist models and learning schemes are needed depending on the target application. A tremendous amount of research and effort is put into study of models with weight summing nodes. Although this is positive, it is the hope of this author that we can continue to stretch our creative energies and search more of the vast space of potential computing mechanisms in the goal to achieve revolutionary technologies.

Bibliography

1. Chang, J. and J. J. Vidal, "Inferencing in Hardware," *Proceedings of the MCC-University Research Symposium*, Austin, TX, (July 1987).

2. Hinton, G., Sejnowski, T, and D. Ackley, "*Boltzmann Machines: Constraint Satisfaction Networks that Learn,"* Tech. Rep CMU-CS-84-119, CMU, Pittsburgh, PA, (1984).

3. Hopfield. J. J., and D. W. Tank, "'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics*, (52), pp. 141-152, (1985).

4. Martinez, T. R., "Adaptive Self-Organizing Logic Networks," Ph.D. Dissertation, Technical Report - CSD 860093, University of California, Los Angeles, CA (May 1986).

5. Martinez T. R., "Models of Parallel Adaptive Logic," *Proceedings of the 1987 IEEE Systems Man and Cybernetics Conference*, pp. 290-296, (October, 1987).

6. Martinez, T. R. and J. J. Vidal, "Adaptive Parallel Logic Networks," *Journal of Parallel and Distributed Computing*, Vol. 5, (1988).

7. Nilsson, N., *Learning Machines*, McGraw-Hill, (1965).

8. Rosenblatt, F. *Principles of Neurodynamics*, Spartan Books, Washington, D.C., (1962).

9. Rumelhart, D. and McClelland, J., *Parallel Distributed Processing:*, Vol. I, pp. 318-362, MIT Press, (1986).

10. Verstraete, R. A., "Assignment of Functional Responsibility in Perceptrons," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, CA, (June 1986).

11. Widrow, B., "Generalization and Information Storage in Networks of Adaline 'Neurons'," *Proceedings of the Conference on Self-Organizing Systems,* pp. 435-462, (1962).