Learning by Gradient Descent in Function Space

Ganesh Mani
Computer Sciences Department
University of Wisconsin—Madison
Madison, WI 53706.
ganesh@cs.wisc.edu

Abstract: Traditional connectionist networks have homogeneous nodes wherein each node executes the same function. Networks where each node executes a different function can be used to achieve efficient supervised learning. A modified back-propagation algorithm for such networks, which performs gradient descent in "function space," is presented and its advantages are discussed. The benefits of the suggested paradigm include faster learning and ease of interpretation of the trained network.

1 Introduction

Connectionist networks (Rosenblatt, 1962; Grossberg, 1981; Rumelhart, McClelland & the PDP Research Group, 1986) are usually thought of to be graph-like interconnections of processing elements. Each processing element is capable of a simple computation such as summation or summation coupled with thresholding. In such networks, the delta rule has been used for learning linear concepts and the generalized delta rule (Rumelhart, McClelland & the PDP Research Group, 1986) has been employed for learning non-linear concepts. Recently, newer algorithms for learning in connectionist networks have been proposed. These include learning by generation of links and nodes as required (Fahlman & Lebiere, 1990; Honavar & Uhr, 1989) and learning by embedding a priori knowledge into the network (Shavlik & Towell, 1989). An important aspect of learning in connectionist networks is to make the network exhibit a particular global behavior, but to implement the learning locally (Williams, 1986).

In this paper, we propose a new learning mechanism for learning in connectionist or artificial neural networks. One of the assumptions underlying this algorithm is that the different nodes in the network can implement different functions. This is being proposed as an efficient computational paradigm and is not based on neurobiological evidence in this regard. However, it is plausible that different neurons, microcircuits or local circuits (Shepherd, 1988) in the brain are computing different functions at different points in time. For example, the temperature term in some of the activation functions can be considered to be a tunable variable in the process of modeling neurons as adaptive elements (Tawel, 1989); tuning the temperature term turns out to be a special case of our approach.

Even though gradient descent in function space can be used for all the generic categories of tasks described by Rumelhart and Zipser (see p. 161 of Rumelhart, McClelland & the PDP Research Group, 1986), it is best suited for classification and regularity detection tasks. This is because the network makes less fine-grained transitions in function space when compared with those in weight space. In the presence of regularities, the network will be able to arrive at the correct generalization in spite of the coarse transitions. The generalization involves choosing a function (from a set of pre-defined functions) to be implemented at each node. By casting the generalization or regularity detection problem as a search problem, we are allowing the generalization language (Mitchell, 1982), denoted by the set of functions each node can execute, to bias the concept learned.

In the next section, we will describe the learning algorithm for updating the function implemented at each node of a connectionist network. The subsequent sections will describe the solution to an example task, discuss advantages of this approach and explore the potential for combining this approach with other related approaches, including traditional back-propagation or re-weighting.

2 The learning algorithm

In this section, we will describe the update rule for changing the function to be executed by each node following the presentation of a training example. The update rule is derived using assumptions similar to those used in the derivation of the generalized delta rule for updating link weights using back-propagation of the error signal (Rumelhart, McClelland & the PDP Research Group, 1986).

Assume that a sequence $\mathcal{F}$ of totally-ordered (according to a restrictive relation) functions $f^1, f^2 \ldots f^m$ is available at each node. Each f^i is a function from $\mathcal{R}^n$ to $\mathcal{R}$ for real-valued activations and $[0,1]^n$ to $[0,1]$ for boolean-valued activations (n is the number of input links to the node; note, however, that an input value of 0 can be assumed if a link is absent). The values of n and m may vary from one node to another and the update rule is unaffected by this. g is a generating function that generates the family of functions at each node. Then we define the derivative of g, for our purposes, to be the partial derivative of the output of the node with respect to a change in the function the node executes. For an input/output pair p, let i_{pj} refer to the value of the jth component in the input pattern, o_{pj} refer to the jth element of the actual output pattern, and $\Delta_p f_j$ refer to the change in the function (say, from f^i to f^{i+1}).

The update rule for changing the node functions is the following (E_p is the measure of error for pattern p):

$$\Delta_p f_j = -\eta \frac{\partial E_p}{\partial f_j}$$

where

$$\frac{\partial E_p}{\partial f_j} = -\delta_{pj} (g_j^{'})_{f_j}$$

where $(g_j^{'})_{f_j}$ is the derivative of the generating function g evaluated at the point f_j, i.e., the current function associated with node j, and the error signal

$\delta_{pj} = (t_{pj} - o_{pj})$ if j is an output node
$\quad = \sum_k \delta_{pk} f_k'(i_{pk})$ otherwise.

If weights are being used on the links, then the error signal for the non-output nodes is given by
$\delta_{pj} = \sum_k \delta_{pk} f_k'(i_{pk}) w_{kj}$
where w_{kj} is the weight on the link connecting node j with node k. For a derivation of this update rule, see Appendix I.

Implementing the above algorithm is analogous to performing gradient descent in E, the sum of squared error, as a function of all possible functions for the nodes in the network. This learning procedure is more efficient than standard back-propagation since the number of function updates following each presentation of the pattern is $O(n)$, while the corresponding number for updating link weights is $O(n^2)$. These estimates are for a network with n nodes, assuming near-complete connectivity.

3 An example task

Let us examine the task of learning the boolean "Exclusive-OR" (XOR) function using the paradigm outlined above. Although the XOR learning task is not an example of a substantial real-world task, it finds a prominent place in connectionist literature as a benchmark task because of its non-linear nature. This simple, but instructive, task is used here for ease of illustration.

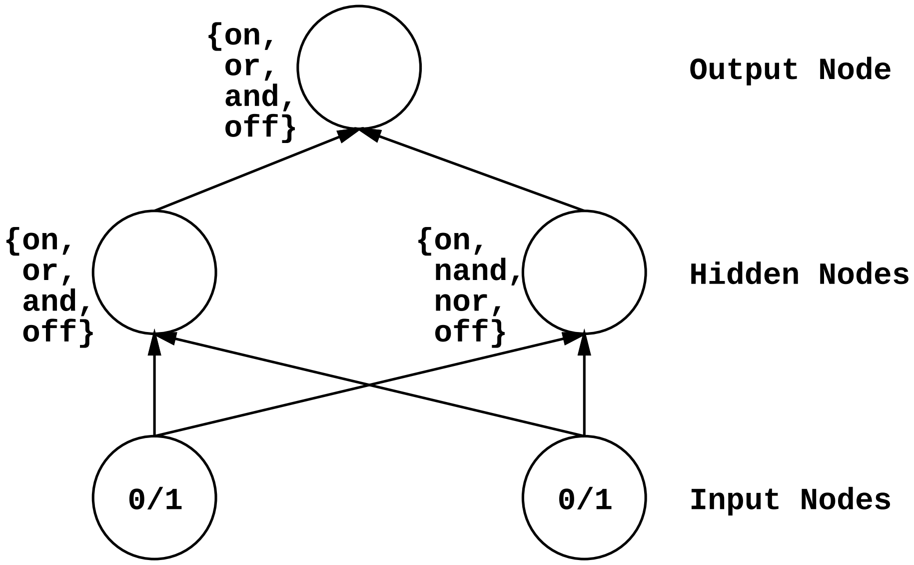

Fig. 1: Network for learning XOR

A feed-forward network consisting of two hidden nodes and one output node, connected as shown in Fig. 1, is employed for learning the XOR concept. Each node is initialized to a random function from the function set shown adjacent to it in Fig. 1. Each function set satisfies a *total ordering* which is exploited for defining a gradient across the set and for updating the function values. Note that one of the hidden nodes (the one on the right in Fig. 1) uses a function set that is different from the one used by the other hidden node and the output node. This is because it is impossible to learn the XOR concept (or any general boolean concept) using "and" and "or" functions alone without using negated inputs. On the other hand, the "nand" function is a *complete* function which means that it can be used to learn any boolean concept[1].

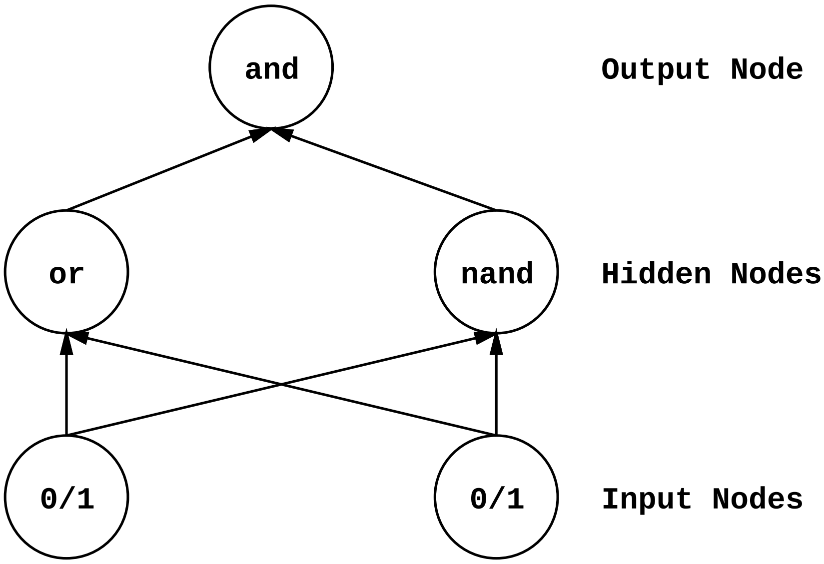

Fig. 2: Network after learning XOR

The input patterns are presented to the network in the usual fashion, and digital gradients are used in propagating the error back to the hidden nodes. In the particular version of the algorithm under discussion, link weights are not used in the learning process and are all set to 1. The network, after the XOR concept has been learned, is shown in Fig. 2. Boolean functions can be learned using such an approach in a few epochs (typically 1 to 5), as opposed to traditional re-weighting approaches which take hundreds of epochs (Rumelhart, McClelland & the PDP Research Group 1986). The efficacy of learning in this paradigm stems from two factors: the richer representation employed by the network and the transition of each node from one meaningful function to another. An analogous effect could have been achieved by using weights and thresholds, but learning would be slowed down because the network would very gradually try to find the optimal values of these variables to settle down to. Simply increasing the learning rate in the generalized delta rule (Rumelhart, McClelland & the PDP Research Group, 1986) would not be of much help since the network will often end up missing the optimal values by taking jumps that are too large. Apart from speeding up training,

[1]in fact, a general-purpose computer can be built out of "nand" gates alone.

gradient descent in function space has other advantages. These will be elaborated upon in the next section.

4 Discussion

The example discussed in the last section as well as our current research emphasis is motivated by the goal of achieving symbolic processing in connectionist networks. However, if re-weighting is retained as an adjunct learning mechanism, or if real-valued function sets are used, the connectionist or subsymbolic flavor is retained. Thus, our approach has the potential to capture the strengths of both symbolic and subsymbolic approaches.

The question, obviously arises, as to what are the ramifications of combining this paradigm with other approaches such as re-weighting and generation (Fahlman & Lebiere, 1990; Honavar & Uhr, 1989). Combining function-change with re-weighting can be thought of as gradient descent in E, the sum of squared error, as a function of all possible functions for the nodes and all possible weights for the links (function-weight space). The rule for updating link weights can be derived on the same lines as the generalized delta rule or the derivation in Appendix I. By calculating $\partial E/\partial f$ and $\partial E/\partial w$ independently and moving in the gradient direction implied by them, it is possible to perform steepest descent in this hybrid space.

Another way of combining re-weighting with function change is to permit re-weighting to proceed until a local minimum in weight space or $E(w)$ (which is actually a "cross section" of the hybrid space, $E(f,w)$) is encountered. From this point, a transition is made to a different point in $E(f,w)$ by changing the functions associated with a few of the nodes. This can be repeated until the point is not a local minimum in $E(f,w)$. The function change can be performed using the update rule described in Section 2.

If generation is used, the generation heuristic can also decide the function (and the corresponding function set) the node should be initialized to, when it is generated. This function may end up getting changed with time as a result of applying the update rule above. Thus, this paradigm lessens the burden on the generation heuristic; weak generation heuristics can be combined with function change to arrive at an appropriate network for the problem at hand.

The rich representation employed by the network and the transition of each node from one meaningful (as defined by our a priori choice of functions that go into a function set) function to another make learning efficient. It is also possible to embed inductive biases into the network; for example, to learn a conjunctive sub-concept, we can constrain the node corresponding to the sub-concept to execute the "AND" function. Thus, the choice of function sets at each node provides a way to bias the concept learned; a priori knowledge can be encoded by choosing these function sets appropriately. Knowledge can also be encoded in weights if weights are being used in the network; the viability of this approach has been demonstrated by Shavlik and Towell (1989).

One of the major criticisms against traditional connectionist networks has been their black-box nature; the final output, often, minimizes the error and forms the desired concept but the intermediate layers do not give any further clues as to why the particular concept was chosen. Our framework provides a way to get around this criticism by explicitly providing the ability to compose results of computations performed by sub-networks. We can express the function computed by a network as a composite function in terms of the functions computed at the different layers and the inputs. For example, for the network in Fig. 1, we can a priori postulate the output of the network to be $f_o(f_{h1}(i1, i2), f_{h2}(i1, i2))$ where f_o is the function associated with the output node, f_{h1} and f_{h2} are the functions associated with the hidden nodes, and $i1$ and $i2$ are the inputs. Note that as learning proceeds, the actual functions that are executed at the nodes change, but the compositionality property is satisfied at all times.

Thus, the benefits of the suggested method include faster learning and ease of interpretation of the trained network. Properties such as productivity, systematicity and compositionality, which Fodor and Pylyshyn (1988) argue as being crucial for a symbol-manipulating system, are satisfied by this paradigm. Detailed arguments in support of this are outside of the scope of the paper and will be presented elsewhere. As mentioned earlier, the subsymbolic flavor can be retained by using weights and by considering re-weighting as an adjunct learning mechanism. Thus, our approach suggests a way to choose an appropriate point on the symbolic–subsymbolic continuum depending on the problem and domain characteristics.

This research also opens up the possibility of generalized gradient descent; ideally, gradient descent should be performed in LTFW space, where L refers to the set of learning algorithms under consideration, T refers to the set of candidate topologies for the network, F refers to the set of function configurations for the network, and W refers to the set of weight configurations for the network. Standard back-propagation performs gradient descent in W space and in this paper, we have discussed gradient descent in F space and FW space. We are currently exploring techniques to define appropriate gradients for discrete functions representing topologies, learning algorithms and node functions.

5 References

1. Fahlman, S.E. and Lebiere, C. (1990). The Cascade Correlation Learning Architecture. *In:* Advances in Neural Information Processing Systems 2, (ed.) D.S. Touretzky. San Mateo, CA: Morgan Kaufmann.

2. Fodor, J.A. & Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: A critical appraisal. *Cognition 28.*

3. Grossberg, S. (1981). Adaptive Resonance in Development, Perception and Cognition. *SIAM-AMS Proceedings 13.*

4. Honavar, V. and Uhr, L. (1989). Generation, Local Receptive Fields and Global Convergence Improve Perceptual Learning in Connectionist Networks. *Proc. of the Eleventh IJCAI,* San Mateo, CA: Morgan Kaufmann.

5. Mitchell, T.M. (1982). Generalization as Search. *Artificial Intelligence 18(2).*

6. Rosenblatt, F. (1962). *Principles of Neurodynamics.* New York: Spartan Books.

7. Rumelhart, D.E., McClelland, J.L. and the PDP Research Group. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* Cambridge, MA: MIT Press/Bradford Books.

8. Shavlik, J.W. and Towell, G.G. (1989). An Approach to Combining Explanation-based and Neural Learning Algorithms. *Connection Science 1(3).*

9. Shepherd, G.M. (1988). *Neurobiology, 2nd ed.* New York: Oxford University Press.

10. Tawel, R. (1989). Does the Neuron "Learn" Like the Synapse. *In:* Advances in Neural Information Processing Systems, (ed.) D.S. Touretzky. San Mateo, CA: Morgan Kaufmann.

11. Williams, R.J. (1986). The Logic of Activation Functions. *In:* Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, (eds.) D.E. Rumelhart, J.L. McClelland, and the PDP research group. Cambridge, MA: MIT Press/Bradford Books.

6 Appendix I

The following is an informal derivation of the gradient descent algorithm for connectionist networks with adaptive nodes. The derivation is analogous to the derivation of the generalized delta rule as described in Chapter 8 of (Rumelhart, McClelland & the PDP Research Group, 1986).

To perform gradient descent in the function space, we must set

$$\Delta_p f_j \propto -\frac{\partial E_p}{\partial f_j}$$

Using the chain rule, we have

$$\frac{\partial E_p}{\partial f_j} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial f_j}$$

where (we have dropped the subscript p on g and f for clarity)

$\frac{\partial o_{pj}}{\partial f_j}$ is $g_j'(f_j)$ and

$-\frac{\partial E_p}{\partial o_{pj}}$ is the error signal δ_{pj}.

Now, it is straightforward to compute δ_{pj} for the output nodes.

$$\delta_{pj} = (t_{pj} - o_{pj})$$

To compute the error signal for the non-output nodes, apply the chain rule on $\frac{\partial E_p}{\partial o_{pj}}$.

$$
\begin{aligned}
\frac{\partial E_p}{\partial o_{pj}} &= \sum_k \frac{\partial E_p}{\partial i_{pk}} \frac{\partial i_{pk}}{\partial o_{pj}} \\
&= \sum_k \frac{\partial E_p}{\partial o_{pk}} \frac{\partial o_{pk}}{\partial i_{pk}} \frac{\partial i_{pk}}{\partial o_{pj}} \\
&= -\sum_k \delta_{pk} f_k^{'}(i_{pk}) w_{kj}
\end{aligned}
$$

Thus, we have a recursive formulation for the error signals and the function updates at the various nodes.

If link weights are all set to 1 ($w_{kj} = 1\ \forall k, j$), the update rule can be specified as

$$\Delta_p f_j = -\eta \frac{\partial E_p}{\partial f_j}$$

where
η is the learning rate and
$\frac{\partial E_p}{\partial f_j} = -\delta_{pj} g_j^{'}(f_j)$
where the error signal

$$
\begin{aligned}
\delta_{pj} &= (t_{pj} - o_{pj}) \text{ if } j \text{ is an output node} \\
&= \sum_k \delta_{pk} f_k^{'}(i_{pk}) \text{ otherwise.}
\end{aligned}
$$