To appear in *Working Notes of the ML-COLT'94 Workshop on Constructive Induction and Change in the Representation, New Brunswick, July 1994.*

A Hypothesis-driven Constructive Induction Approach to Expanding Neural Networks

Vladimir N. Sazonov
Computer Sciences Corporation
3160 Fairview Park Dr.
Falls Church, VA 22042
vsazonov@itchy.starlab.csc.com

Janusz Wnek
George Mason University
4400 University Dr.
Fairfax, VA 22030
wnek@aic.gmu.edu

Abstract

With most machine learning methods, if the given knowledge representation space is inadequate then the learning process will fail. This is also true with methods using neural networks as the form of the representation space. To overcome this limitation, an automatic construction method for a neural network is proposed. This paper describes the BP-HCI method for a hypothesis-driven constructive induction in a neural network trained by the backpropagation algorithm. The method searches for a better representation space by analyzing the hypotheses generated in each step of an iterative learning process. The method was applied to ten problems, which include, in particular, exclusive-or, MONK2, parity-6BIT and inverse parity-6BIT problems. All problems were successfully solved with the same initial set of parameters; the extension of representation space was no more than necessary extension for each problem.

1 INTRODUCTION

Most research on inductive learning from examples has been concerned with learning concept descriptions from examples in a predefined representation space. In such methods, the hypotheses learned are expressed using attributes or terms selected from among those initially provided. In neural networks, this problem was first analyzed by Minsky & Papert (1969). They show shortcomings of neural networks that do not have hidden layers. This motivated research on multilayer perceptrons and algorithms for training them. The backpropagation learning algorithm for multilayer networks was discovered by Werbos (1974) and then independently rediscovered by various researchers (Le Cun, 1985; Parker, 1985; Rumelhart & McClelland, 1986). The major limitation of that approach was the assumption of a fixed internal structure (topology) of a neural network, i.e. number of hidden layers and number of nodes in hidden layers was determined by an expert.

There is a new line of research involving determination of the topology of neural networks. These attempts include knowledge-based neural networks to set up topology and connection weights of an initial neural network based on domain-specific inference rules (Towell et al., 1991), techniques for trimming neural networks via relevance assessment (Mozer & Smolensky, 1988; Weigand et al., 1990), and genetic algorithm-based evolution of the topology and weight distribution of neural networks (Maniezzo, 1993).

This research concerns automatic determination of neural network topology, and is viewed as a constructive induction method. Constructive induction systems perform a double search, one for the most suitable representation space, and second for the most suitable concept description in this space (Wnek & Michalski, 1994). They include mechanisms for generating new, more relevant descriptors, as well as modifying or removing less relevant ones from those initially provided.

This paper describes an approach to constructive induction in which the desirable changes in the representation space are determined by analyzing the hypotheses generated in each iteration of the learning process. For this reason, this approach is a hypothesis-driven constructive induction (HCI), as opposed to a knowledge-driven approach (KCI) or data-driven approach (DCI) (Wnek & Michalski, 1994). In the context of neural network knowledge representation, the KCI approach can be identified, for example, with the KBANN system (Towell et al., 1991). The DCI approach in neural networks is yet to be identified.

2 HCI APPROACH

The HCI approach is based on repetitively detecting strong *patterns* in the learned concept descriptions and then using them in transforming the representation space. A *pattern* is meant to be a component of a generated concept description that is characteristic of a relatively large number of concept examples (Wnek & Michalski, 1994).

In the HCI approach, transformations of the representation space may involve both contraction and expansion operations. Contraction decreases the number of possible concepts that can be represented in the space, and expansion increases that number. Contraction is done by removing descriptors (attributes, predicates, functions, relations, hidden nodes, etc.) from the representation space. Expansion is done by adding new descriptors to the representation space.

The proposed hypothesis-driven constructive induction method, BP-HCI, combines an inductive, backpropagation learning method with a procedure for iteratively transforming the representation space. Another example of the application of the HCI method with the rule learning algorithm AQ is given by Wnek & Michalski (1994).

2.1 BP-HCI METHOD

In general, it is not known what size network works best for a given problem. Further, it is difficult to derive a general solution since each problem demands different capabilities from the network. Choosing proper network size is important. If the network is too small, it will not be capable of solving given problem. On the other hand, if the network is too big then it may be over capable as well as too complex (Baum & Haussler, 1989).

To properly address these issues, it is required to answer two fundamental questions regarding the neural network topology:

- What are the criteria for deciding the number of nodes in a hidden layer?
- What are the criteria for deciding the number of hidden layers?

In this paper we address the first question by detecting certain patterns in the behavior of a neural network that allow for controlling growth of the neural network. These patterns are described using formulas ACCORD and ANXIETY (section 2.2).

2.2 BP-HCI: GENERAL DESCRIPTION

We consider a neural network (NN) with binary neurons and standard feed forward architecture. The NN has one output neuron; an amount of input neurons (I) is determined by the problem under consideration (we considered problems with I = 2 and I = 6). The NN has one hidden layer that initially contains one hidden neuron; the NN creates new hidden neuron(s) if necessary; each hidden neuron has I connections to each input neuron and one connection to the output neuron. There are no connections from input neurons to the output neuron.

The NN solves a problem represented by a training set which consists of E training examples (our problems had E = 4 or E = 64). An example is an ordered set of Boolean numbers (0 and 1). Each example is associated with example error $e^\alpha = y^\alpha - d^\alpha$, where y^α and d^α are actual and desirable outputs of the output neuron respectively; α is an example number, $\alpha \in [1, E]$.

The goal of training process is to determine weights of connections and thresholds of neurons such that for any example $|e^\alpha| \le adm\_err$, where adm_err is a given value treated as admissible error (we used adm_err = 0.1).

To achieve this goal we introduce usual cost function (CF) as a sum of $(e^\alpha)^2$ across all E examples. The training process is divided into epochs (iterations); each epoch comprises presentation of all examples from the training set. After the presentation we apply back propagation method to find the gradient of CF in weights_thresholds space. Then we adjust weights and thresholds such that representation point makes a step in weights_thresholds space in the direction that is opposite to the gradient. Unlike usual back propagation learning rule, the length of the step (LOS) is the same for all epochs (we obtained good results for LOS = 0.0005).

To determine how successful the training process is we consider a measure ACCORD meaning "accord in direction finding between two subsequent epochs." By definition

$$ACCORD_n = \frac{(\hat{P} * \vec{S}_n)^2}{(\hat{P} * \vec{S}_n - \hat{P} * \vec{S}_{n-1})^2 + (\hat{P} * \vec{S}_n)^2 / 100} \qquad (1)$$

where n is epoch number, vector $\vec{S}_n$ is a step in the weights_thresholds space, $\hat{P}$ is a projection operator onto a subspace (about the subspace see below). If the directions found by two subsequent epoch coincide, i.e. if $\vec{S}_n = \vec{S}_{n-1}$, then ACCORD achieves maximal possible value 100. If the directions are opposite ones, then $\vec{S}_n = -\vec{S}_{n-1}$, and ACCORD achieves minimal possible value $1/4.01 \cong 0.249$. We interpret low values of ACCORD as an indication that the NN is in the vicinity of a (local) minimum.

To determine when a new neuron must be created (with its connections, weights and the threshold) we assign to each connection an ANXIETY measure. Consider a connection that links neuron #i to neuron #j. Let us assume the training process comprises N completed epochs at the present moment; then by definition

$$ANXIETY_{ij} = \sum_{n=1}^{N} \exp\left(-\frac{ACCORD_n}{G\_ACC}\right) * \sum_{\alpha=1}^{E} \left|\frac{\partial (e_n^\alpha)^2}{\partial w_{ij}}\right| \qquad (2)$$

where, G_ACC (good accord) is constant (we used G_ACC = 2.5), n is epoch number, w_{ij} is a weight of the connection ij. Recall that $(e_n^\alpha)^2 = (y_n^\alpha - d_n^\alpha)^2$ is a

contribution of example #α into CF at epoch #n. The derivative of $(e_n^\alpha)^2$ is calculated by the back propagation method.

Suppose the NN "knows" very well where to go in weights_thresholds space at epoch #n (i.e. $ACCORD_n$ is much greater than G_ACC), then no significant contribution to ANXIETY occurs due to little exponent factor in (2). At the opposite case NN "jumps" around a point of minimum in weights_thresholds space and $ACCORD_n$ < G_ACC. In this case $ANXIETY_{ij}$ can increase significantly provided some examples "want" significant changes for the weight ij (i.e. the derivative of $(e_n^\alpha)^2$ is far from zero for some α).

We need to set a bound value for ANXIETY; we denote it as BOUND_ANX and set to 200. (Currently, this value is established experimentally). During the training process we also need to watch maximum values of inputs for all neurons across examples, which have $|e_n^\alpha| > adm\_err$. In a sense these examples are not "satisfied" yet. Let MAX_INPUT_i be maximum value of inputs for neuron #i across "unsatisfied" examples.

We obtained good results if determined ACCORD in a two dimensional subspace with the base consisting of (1) the threshold of the last created hidden neuron or the threshold of the initial hidden neuron, if no new hidden neurons are created yet; (2) the weight of the connection from the last created hidden neuron (or the initial hidden neuron) to the output neuron.

The BP-HCI algorithm works as follows:

0. Given NN architecture and initial values for weights and thresholds.

1. Train the NN for one epoch (using all training examples).

2. Evaluate the NN. If for all examples $|e_n^\alpha| \le adm\_err$, then exit.

3. If for all connections $ANXIETY \le BOUND\_ANX$, then go to 1.

4. For a connection with highest ANXIETY, such as $ANXIETY_{ij} > BOUND\_ANX$
   - Create a new neuron #n with one output connection to neuron #j. Let the weight $w_{nj} = 0$. Let the threshold of the new neuron $t_n = MAX\_INPUT_i$. Let the new neuron have the same input connections and weights as neuron #i.
   - For all connections let ANXIETY = 0.
   - Go to 1.

We set w_{nj} to 0 because any other choice could increase the value of CF just after the new neuron is constructed. The choice $t_n = MAX\_INPUT_i$ is not a mandatory one. Let us call a "twin situation" a case in which each example produces approximately the same output for two neurons that have the same input and output connections. If a twin situation arises, one neuron can be deleted. The value of t_n must be chosen to avoid twin situation for the new neuron and neuron #i just after the new neuron is created. Such a choice of t_n allows us to achieve this goal, if t_i is significantly smaller than MAX_INPUT_i.

2.3 BP-HCI: DETAILS

Let x, y and τ stand for an input, an output and a threshold of a neuron respectively. We have the relation y=S(x - τ). The sigmoidal function S was chosen as

$$S(a) = \frac{1 + 2 * exten}{1 + \exp(-a / SCALE)} - exten \qquad (3)$$

where

$$exten = \frac{1}{\exp(0.5 / SCALE) - 1}$$

We used SCALE = 0.2 and consequently exten = 0.0894. The function (3) has the following important properties: S(a=0) = 0 and S(a=1) = 1. Note however that S(a<0) < 0 and S(a>1) > 1.

Initially we set all weights to 1 and all thresholds to 0.5. These settings together with function (3) give us the following useful features of the NN: if we feed zeros to all input neurons, then the output value of the output neuron is equal to zero too; if we feed 1 to an input number, the output value is equal to 1 too. Further, the case when output value is equal to 1 is referred to as positive response.

3 EXPERIMENTS AND DISCUSSION

Table 1 contains some data pertaining to our experiments. We considered ten problems. To describe a problem we use a notation shown in row "schema" of Table 1. For instance, 6:0,2,4,6 means that NN has 6 input neurons and must give positive response if and only if no input neuron fires or any 2 input neurons fire or any 4 input neurons fire or all 6 input neurons fire; it is the well known "parity" problem. Where applicable, the commonly used name of the problem is given in row "name." The next row contains E - total number of examples in the training set. The next 5 rows show epochs when new hidden neurons were created. For instance, solving inverse parity problem a NN creates the fifth hidden neuron at epoch #10565 (recall that NN initially has 1 hidden neuron). The row "tr.comp." shows epochs when training was completed. The last row "cr.h.n." contains a number of created hidden neurons.

Table 1: Summary of Experiments with BP-HCI

Schema	2:1,2	2:2	2:1	6:0	6:2	6:1,6	6:1,3,4,5	6:1,3,5,6	6:1,3,5	6:0,2,4,6
Name	OR	AND	XOR	---	MONK2	---	---	---	inverse parity	parity
E	4	4	4	64	64	64	64	64	64	64
2nd	---	---	2588	---	2383	2721	1429	1831	1860	4895
3rd	---	---	---	---	---	5472	3755	4499	3364	6319
4th	---	---	---	---	---	---	8181	8853	6838	8796
5th	---	---	---	---	---	---	---	11826	10565	11539
6th	---	---	---	---	---	---	---	---	13258	16183
tr.comp	1	160	4046	5148	7900	12942	10350	13885	15327	18295
cr.h.n.	0	0	1	0	1	2	3	4	5	5

To the best of our knowledge for all the problems this number coincides with the number of hidden neurons that one must add to the hidden layer to solve the problem. Thus our method creates minimal necessary number of additional hidden neurons. Recall that we consider NN that has 1 hidden layer and 1 output neuron which is linked to hidden neuron(s) only.

Any NN performs numerical calculations; the result always depends on parameters. It is worth noting that our method is able to solve 10 problems with one and the same set of parameters. This supports ideas put in the method.

Currently all parameters were established experimentally. A theoretical search for parameters may be a subject of further research.

4 CONCLUSIONS

The automatic determination of the NN topology is relatively new area of interest and is still in an early stage of development. However, during the past few years some algorithms to develop NN architecture have been proposed. For example, a special issue of *IEEE Transactions on Neural Networks* (Vol.5, No 1, 1994) was devoted to evolutionary computation. Some other recent papers explore similar ideas (Hanson, 1990; Fahlman & Lebiere, 1990; Wynne-Jones, 1992; Cios & Liu, 1992; Odri et al, 1993; Redding et al, 1993; Sperduti & Starita, 1993; Nabhan & Zomaya, 1994).

In a sense the topology of our NN is polar to the topology given by cascade-correlation method (Fahlman & Lebiere, 1990) where there may be more than 1 hidden layer and each hidden layer contains only 1 neuron. The cascade-correlation method has an excellent rate of training. But rates of operating of trained NN may be better for NN with 1 hidden layer provided all neurons in all layers act simultaneously and independently. The large amount of hidden layers may make it difficult to use the advantages of parallel processing.

Sperduti & Starita, 1993 found that parity-4BIT problem can be solved by a NN that has the only hidden layer with 3 neurons. This is possible because their neurons can have different values of parameter SCALE, that makes computation much more complex. If we applied our method to parity-4BIT we would obtain a hidden layer with 4 neurons.

Note that we do not split neurons (compare with Hanson, 1990 or Wynne-Jones, 1992). When we create a new neuron all previous neurons remain intact. We create new neurons to help overloaded connections.

Another important feature of the BP-HCI method is that connections begin to feel ANXIETY and create new neurons only when the NN is in the vicinity of a local minimum in the subspace. There are many ways to determine local minimums. We used the concept of ACCORD.

The current BP-HCI method does not consider deleting nodes from a layer, and adding/deleting layers to/from the network. Such transformations are needed for flexible adaptation of any network. This problem will be considered in future work.