From: AAAI-94 Proceedings. Copyright © 1994, AAAI (www.aaai.org). All rights reserved.

Learning To Learn : Automatic Adaptation of Learning Bias

Steve G. Romaniuk
Department of Information Systems and Computer Science
National University of Singapore
10 Kent Ridge Crescent
Singapore 0511
e-mail: stever@iscs.nus.sg

Abstract

Traditionally, large areas of research in machine learning have concentrated on pattern recognition and its application to many diversified problems both within the realm of AI as well as outside of it. Over several decades of intensified research, an array of learning methodologies have been proposed, accompanied by attempts to evaluate these methods, with respect to one another on small sets of real world problems. Unfortunately, little emphasis was placed on the problem of *learning bias* - common to all learning algorithms - and a major culprit in preventing the construction of a *universal* pattern recognizer. State of the art learning algorithms exploit some inherent bias when performing pattern recognition on yet unseen patterns. Automatically adapting this learning bias - dependent on the type of pattern classification problems seen over time - is largely lacking. In this paper, weaknesses of the traditional *one-shot* learning environments are pointed out and the move towards a learning method displaying the ability to *learn about learning* is undertaken. Trans-dimensional learning is introduced as a means to automatically adjust *learning bias* and empirical evidence is provided showing that in some instances *learning the whole can be simpler than learning a part of it.*

Introduction

It is a well known fact that if we consider the universe of all possible pattern recognition problems, learning algorithms [1] tend to exploit some bias, when generalizing from a finite set of examples. Customarily, we refer to this generalization as *induction.* An abundance of learning algorithms (pattern classifiers) have been proposed over the last few decades (Quinlan 1979; Fahlman & Lebiere 1990; Frean 1991; Romaniuk 1993b; Cheng et al. 1988). In all instances, the inherent bias in learning algorithms remains static, thereby restricting the universal application of these algorithms to a multitude of conceivable pattern classification problems. In other words, every single one of these algorithms can only be applied with some degree of success to a subclass of pattern recognition tasks. No universal application is possible, due to the algorithms' inability to automatically modify its learning bias, based on the types of pattern recognition problems it may be confronted with over a period of time. This lack of any form of *meta-learning* capability in traditional pattern recognition algorithms, has received little attention from the machine learning (pattern recognition) and neural network communities. Instead, increasingly *limited* pattern recognition algorithms are proposed, void of this most fundamental property commonly attributed to any intelligent agent. A few notable exceptions are (Chalmers 1990; Bengio et al. 1992).

The purpose of this article is to outline a learning system capable of automatically adjusting learning bias based on prior performance. The proposed system is based on a simple feedforward neural network, in which hidden units are automatically recruited and locally trained using the perceptron learning rule. Learning is seen as a bottom-up feature construction process in which previously learned knowledge (stored as feature units within the network) represents the primary mechanism for performing *meta-learning.*

Automatic Neural Network Construction

Overview

During the past years neural networks have been applied to a wide variety of pattern recognition tasks with varying degrees of success. In many cases, a major drawback has been, finding the right number of hidden units, layers and connectivity of the network - at least for networks trained by backpropagation like algorithms. Often users rely on rules of thumb (heuristics) to decide how to choose these parameters. The outcome is at best challenging, many times it is simply frustrating. The problem is further compounded by varying (initially random) weights, an array of different tuning parameters (e.g. momentum

[1] Throughout this paper we will use the terms *learning* and *pattern recognition* interchangeably.

term, learning rate, etc.), type of transfer function (e.g. sigmoid, threshold, gaussian, etc.) and finally the choice of learning algorithm itself (backprop, quickprop, etc.). Given this less than rosy situation, it should come as no surprise that researchers have looked for ways to improve this situation. Within the last couple years several different approaches to automatically construct neural networks have been proposed (Baffes & Zelle 1992; Fahlman & Lebiere 1990; Frean 1991; Romaniuk 1993b). These algorithms differ in how they construct networks (purely horizontal or vertical growth, or hybrids in between) and the kind of local learning rules they utilize. They also contrast with regard to the types of problems they can solve, the quality of the final networks, network complexity, speed, and their ability to converge on certain types of data. For more information the interested reader may want to consult (Baffes & Zelle 1992; Fahlman & Lebiere 1990; Frean 1991; Romaniuk 1993b).

Evolutionary Growth Perceptron

Evolutionary Growth Perceptron (Romaniuk 1993b) (EGP) represents an approach to automatically construct neural networks by exploiting natural selection. Networks are assembled in a bottom-up manner, a feature at a time. Every newly recruited unit (feature) added to the network, receives its input connections from all previously installed hidden units and input units. Once an element is set up, it is trained employing the perceptron learning rule for a fixed number of epochs. Evolutionary processes are invoked to decide which patterns of the training set should be made use of when training the current hidden feature. An individual chromosome encodes the subset of patterns administered for training the present unit. A population of these chromosomes are then evaluated on the task of overall error reduction. After either a fixed number of generations has passed, or there is no improvement in error reduction, the element trained to the lowest error rate is permanently installed in the network. If there are still patterns left, which are miss-classified by the newly added unit, then a new element is created and the above process repeated. Otherwise, training halts and the last unit in the network is designated the output of the network.

EGP learns the difficult N-parity function with about $\lfloor \frac{N}{2} \rfloor$ hidden units (Romaniuk 1993b).

Learning to Learn

Traditional One-shot Learning Approach

Over the last few decades a multitude of ideas have entered the realm of machine learning, resulting in countless interesting approaches and advances to this important area. The oldest learning systems date back to the 1940s (Hebbian learning) and the introduction of the perceptron. Later years would see the

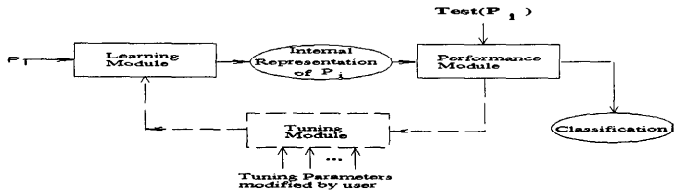

Figure 1: Traditional One-shot Learning Approach

advances of inductive learning systems such as ID3 (Qiunlan 1979) (based on decision trees). The eighties saw a revival of the neural network community with the introduction of the backpropagation learning rule. More recently approaches have been formalized that attempt to integrate seemingly different paradigms such as connectionism and symbolism. Other advances have been made to automate the network building process (Fahlman & Lebiere 1990; Frean 1991; Baffes & Zelle 1992; Romaniuk 1993b). With the discovery of ever new methodologies to tackle learning problems, the question of evaluating these diverse algorithms has remained mostly unchanged. Commonly, we can identify 2 approaches: The first approach considers an empirical study which includes testing a new learning algorithm $L \in \mathcal{L}$ (element of the class of all learning algorithms $\mathcal{L}$) on a set of either artificial or natural (real world) problems $P \in \mathcal{P}$. This set P is for almost all empirical studies small (about 1 to a dozen problems). In general, several well accepted problems - also known as benchmarks - are selected for an empirical study. Besides including a set of benchmarks, a few well known learning algorithms (that is well publicized) are picked in order to perform a comparison. Evaluation of learning algorithms proceeds by dividing a set of patterns, that describe the problem under consideration into 2 partitions: a train (T_{Train}) and test set (T_{Test}). The degree of generalization (performance accuracy) of algorithm L is measured by determining how many patterns in T_{Test} are correctly recognized by L after being trained on patterns in T_{Train}. After training and testing (classification) have been completed for a given problem P, results obtained thus far are removed and a new problem is tackled. We refer to this procedure as *one-shot* learning. A graphical representation of this approach is depicted in Figure 1. Note, that the tuning module is optional and may even be lacking in some learning systems (e.g. ID3 (Qiunlan 1979)). Other approaches - predominantly gradient decent-based learning algorithms (e. g. backpropagation) - are equipped with a multitude of tuning parameters, which when appropriately set by the user, can improve the learning modules overall performance on a given pattern recognition task. We can think of the tuning module as being a primitive mechanism to adapt the bias of the learning module. This adaptation is under the direct control of the human user.

It is a well known fact, that any such empirical com-

parison cannot yield significant results due to the sheer size of potential problems that can exist in the real world. As a matter-of-fact, it is straight forward to verify that for every problem $P = T_{Train} \cup T_{Test}$, for which some algorithm $L \in \mathcal{L}$ can achieve accuracy α (denoted by $A_L(T_{Test}) = \alpha$), there always exists a problem $P' = T_{Train} \cup T'_{Test}$ [2] such that $A_L(T'_{Test}) = 1 - \alpha$. Hence, the average accuracy is always $\frac{1}{2}$. [3] This is actually true regardless of how we partition the examples describing problem P into train and test sets, as long as $T_{Test} \neq \emptyset$.

The above essentially states, there cannot exist a universal learning algorithm (neither biological or artificial), which can perform equally well on all pattern classification problems. Instead, every learning algorithm must exploit some *learning bias* which favor it in some domains, but handicap it in others. The following may serve as an example: Both decision-tree based, as well as, backpropagation-like learning algorithms have a more difficult time representing highly non-separable functions (e.g. parity) than a truly linear function (e.g. logical or). When we say *more difficult to learn*, we refer to training time, as well as, how the function is represented and the performance accuracy we can expect, after training is completed. A decision-tree based algorithm, for example, grows an increasingly deeper tree structure, whereas a backprop network requires additional hidden units (and connections) to learn even/odd parity. Due to these observations, it should become apparent why simple empirical studies are of little value in deciding the quality of a new learning approach or allow for comparing different approaches.

A second technique for evaluating learning systems, is based on developing a theoretical framework. A good example of this approach is the *Probably Approximately Correct* learning model (PAC) (Valiant 1984). Here, one attempts to identify sub-classes of problems which can be learned in polynominal time and for which confidence factors can be stated for achieving a given performance accuracy. Unfortunately, these methods are too rigid and constrained to be of practical use. For one, sub-classes may be very specific, such that results about them seem to say little. A more substantial shortcoming is: sub-classes of problems tend to be highly abstract. For example, a restricted class of problems might be $CNF_n(k)$, that is, all propositional formulae over n boolean variables in conjunctive normal form with at most k literals. Results like this provide no indication on how abstract results can be placed in relation with their real world counterparts. Having determined that algorithm L tends to fare well with problems belonging to some sub-class of tasks, still does not help us decide how to solve a specific real world task, unless we show that our model is in some correspondence with the real world problem. But to determine this membership may require utilizing algorithm L in the first place. In short, we are forced to apply trial-and-error to decide which pattern recognition algorithm is best suited for solving the task at hand. Only, if substantial amounts of a priori domain knowledge are available, can we be sure to select the right learning algorithm, before actually trying it out. For many important tasks an appropriate pattern recognition algorithm may not even be available.

[2] Outputs of patterns in T'_{Test} are inverted from those in T_{Test}.

[3] Assuming binary encoded problems.

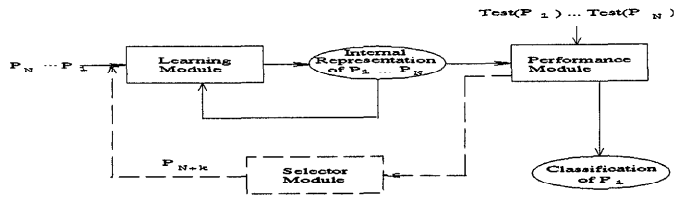

Figure 2: Multi-shot Learning Approach

Multi-shot Learning Approach

The problem we are facing - to develop learning methods that can be employed to solve various real world problems - can not be overcome with simple *one-shot* learning algorithms. For this matter, we propose to look at algorithms which have the capability to *learn about learning*. These new algorithms must be capable of shifting their built-in bias (to favor certain subclasses of tasks) deliberately, by utilizing past performance, to give them wider applicability than simple one-shot learning systems. Figure 2 outlines the multi-shot method. In this approach, previous knowledge (in form of an internal representation of earlier learned problems) is fed back into the learning module, whenever a new task is presented. An optional selector module may provide supplementary pattern classification problems (similar), depending on the results obtained from the performance module and automatically supply them to the learning module.

This ability, - to learn knowledge on top of already existing knowledge, without every time having to restart learning from ground zero - can have an impact on learning time, representational complexity, as well as generalization on yet unseen patterns. To move beyond simple *one-shot* learning we examine the subsequently listed task:

Given: A set of problems $\mathcal{P} = \{P_1^{n_1}, P_2^{n_2}, \ldots, P_k^{n_k}\}$, where $P_i^{n_i} \in \mathcal{C}^{n_i}$.
The $P_i^{n_i}$ belong to the class of classificatory problems $\mathcal{C}$ (continuous inputs/binary output) of variable input dimension n_i.
Goal: Develop a representation R_k that can correctly classify all $P_i^{n_i} \in \mathcal{P}$. No specific order is imposed on the problems in $\mathcal{P}$ when presented to the meta-learning system.

We refer to an algorithm which can solve the above task as a *trans-dimensional learner*. We should observe at this point, that a *network constructing* trans-dimensional learner is one of numerous alternatives. For example, we could have chosen a decision-tree like representation, instead of a connectionist network when designing TDL. A decision in favor of a connectionist representation was cast, due to earlier obtained positive results involving EGP and considering the algorithms simplicity.

Before we outline the individual modules comprising TDL we draw attention to 2 approaches regarding automatic learning bias adjustment:

First, the *explicit* approach. Here, a standard pattern recognition algorithm has its learning behavior regulated by modifying one or more of its tuning parameters. If a learning algorithm is not equipped with tuning parameters, they may be added retrospectively. For example, a decision-tree based algorithm like ID3 has no tuning parameters. An extension of ID3 (GID3 (Cheng et al. 1988)) uses a tolerance parameter which controls the amount of generalization by guiding attribute selection. Explicit learning bias adjustment proceeds by presenting either similar types of problems (e.g. even/odd parity) or identical problems (e.g. only even-parity) to the algorithm. Performance on prior tasks (degree of generalization and effectiveness of knowledge representation) are measured and tuning parameters are appropriately modified. It is anticipated that for similar or identical problems, the same setting (or related) of tuning parameters, can result in improved performance. For example, a backpropagation-based algorithm may learn to adapt its learning rate and momentum term in such a way, that it can more readily learn a parity function after having been trained on a few instances of this type of function.

The second approach is *implicit*. In this case, there are no explicit tuning parameters, instead learning bias is shifted by relying on earlier stored knowledge. Hence, this method is identical to *multi-shot* learning. The internal knowledge representation of previous learned tasks is exploited to guide future learning. The herein proposed meta-learning algorithm is based on the *implicit* approach.

TDL : Trans-Dimensional Learner

In this section, the basic modules deemed important in the construction of TDL are discussed. Earlier, it was pointed out that an automatic neural network construction algorithm (EGP) would serve as the primary means for building high level network features. It was also indicated that the EGP algorithm assembles new features by fully connecting previously created features to the one currently under construction. This can very quickly lead to a high fan-in of connections to the hidden units. Consequently, storage requirements and simulation time increase rapidly, as fresh hidden features are installed. Apart from this surface problem, a more harmful side-effect can be identified: dilution in quality of high-level features. Recall, that the perceptron rule is utilized to locally train newly recruited features. Now, the perceptron learning rule attempts to discover an appropriate linear combination of the inputs to the trained feature, such that overall error is reduced. If the number of features forming the input to the current unit is large, many more weight assignments are possible, of which many may lead to shallow local optimums. To prevent this from occurring, it is essential to restrict the number of hidden features selected for constructing new features. To accomplish this feat, a quality measure is invoked to determine the potential of previously constructed units to act as good hidden feature detectors when solving the current goal.

The quality measure is defined as,

$$Q_{k,i}(F_i, P_k) = \frac{A_k(F_i, P_k)}{|T_k|}\left(1 - \frac{\log(L(F_i)+1)}{\log(\max_{\forall j}(L(F_j))+1)}\right) \quad (1)$$

,where $L(F_i)$ returns feature F_i's layer location. The first factor of Equation 1 measures feature F_i's accuracy on problem P_k, whereas the cost factor incorporates information about feature F_i's layer location relative to the current networks height.

To further reduce the complexity of the resulting connectionist network, it can prove extremely beneficial, to prune an already trained feature unit, before resorting to train a new one. It has bee pointed out that besides further reducing the fan-in to individual units, pruning can also improve the generalization capability of a network.

It may be necessary to provide a mechanism which allows the network to forget some of its learned knowledge, especially if over time this knowledge is deemed unimportant. Forgetting information is paramount to conserving network resources such as units and connections. Finally, if we think of biologically inspired learning systems then, due to changes in the environment, some of the accrued knowledge may become obsolete. It would indeed be wasteful to retain this information.

A High-level description of the proposed trans-dimensional learner is presented below. In order to handle training patterns of various input dimensions, a pool of input features is initially set aside. The pool is left-adjusted, which implies the first input value of a training pattern is stored in the foremost input unit of the pool and so on, until all input values have been assigned to corresponding input units. Remaining pool units are assigned a default value of 0, to guarantee that no activations are propagated from any potential connections that emanate from these units to other network units (Effect identical to being disconnected).

Nomenclature for TDL:

$$
\begin{aligned}
P_k &: \textit{kth problem to be learned.} \\
\mathcal{P} &: \textit{set of all problems.} \\
N &: \textit{network constructed by TDL.} \\
F_C &: \textit{current unit in network } N. \\
I_l &: \textit{lth input of problem } P_k. \\
In(P_k) &: \textit{number inputs for problem } P_k. \\
Q_{k,i}(F_i, P_k) &: \textit{quality measure for ith unit.} \\
A(F_C, P_k) &: \textit{accuracy of feature } F_C \textit{ for } P_k. \\
\alpha &: \textit{threshold for quality measure } (0.1).
\end{aligned} \tag{2}
$$

(1) For all $P_k \in \mathcal{P}$, $k \in [1, N]$ do
  (1.1) While $A(F_C, P_k) < 1$ do
    (1.1.1) Create new feature F_{new}
    (1.1.2) Connect all I_l, $l \in [1, In(P_k)]$ to F_{new}
    (1.1.3) For all $F_i \in N$ do
      (1.1.3.1) If $Q_{k,i}(F_i, P_k) > \alpha$ then
        (1.1.3.1.1) Connect F_i to F_{new}
    (1.1.4) $N = N \cup \{F_{new}\}$
    (1.1.5) $F_C = F_{New}$
    (1.1.6) EGP(F_C)
    (1.1.7) Prune feature F_C

To close out this section, Figure 3 displays an example of a TDL generated network that correctly recognizes the or-function of input dimensions 2 - 7.

Empirical Results

To furnish empirical evidence supporting the effectiveness of TDL, a collection of well-known neural network benchmark problems have been selected in conjunction with some simpler functions to allow for a meaningful study. These problems are: **and**, **or**, **even-parity**, **odd-parity** of input dimension 2 - 7, 4-, 8-, 16-bit **encoder**, and 2-, 3-bit **adder**.

Combining the above tasks yields a total of 29 functions (29-fkt problem). As a subset we also consider the even/odd-parity functions (parity problem). For all experiments 10 trial runs are conducted and their average is reported. Also, every feature is trained for at most 60 epochs.

Learning The Whole Can Be Simpler Than Learning A Part Of It

In Figure 4a we present results relating the number of generations (amount of time required to evolve a network) and the number of combined units (hidden and output units in final network). The first observation we make is, the response curve for the 29-fkt problem displays a sudden steep increase in the final stages of training, as opposed to the almost linear response obtained during learning the parity problem.

Secondly, around 20 functions (of the original 29) are learned in the final stages of training, that is in the last

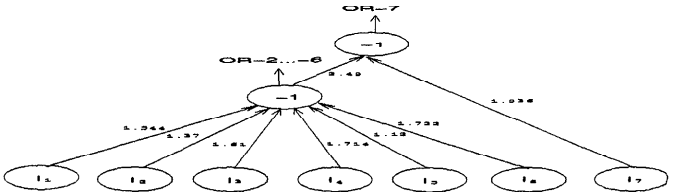

Figure 3: OR-2 through OR-7 as learned by TDL

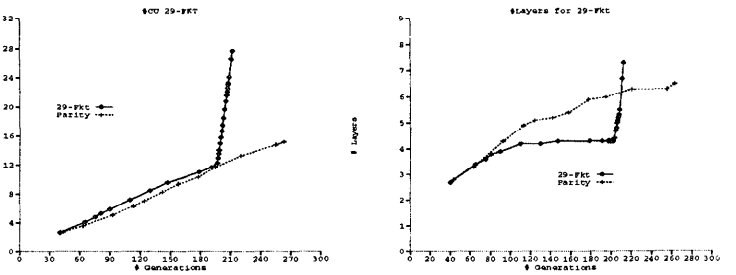

Figure 4: # Generations vs. (a) # Combined Units (b) # Layers for learning 29-Fkt and Parity

20 generations. In other words, more than two-thirds of the problems are acquired in the last one-tenth of the training phase. This result underlines how TDL benefits from previously learned information and indicates how the ability to learn to learn has a profound impact on total training time and final network configuration.

The most impressive finding can be gleaned, when comparing the absolute magnitude in number of generations required to learn either parity or 29-fkt task. The parity problem is learned after nearly 270 generations have passed, whereas the 29-fkt problem is acquired after less than 210 generations. Since the parity problem is a subset of the original 29-fkt problem, this result is clear evidence of the phenomena: *learning the whole can be simpler than learning a part of it.*

Furthermore, this finding substantiates the fact, the time needed to learn a subset of functions can be significantly reduced by assimilating additional functions, even when they have little in common with the original set of functions. Recall, the composition of the 29-fkt problem. Besides containing the simpler **and**- and **or**-functions, it also consists of **encoder** and **adder** functions. The task of identifying **even** and **odd** parity has little in common with the task of performing binary encoding or binary addition.

Another interesting observation is that the number of combined units is about the same as the number of functions learned. Both for parity as well as the 29-fkt task, the increase in number of combined units with respect to the number of functions acquired (regardless of the problems input dimension) is about *linear.*

Figure 4b depicts the relationship between the number of generations and the number of layers. The steep increase in layers for the 29-fkt problem coincides with

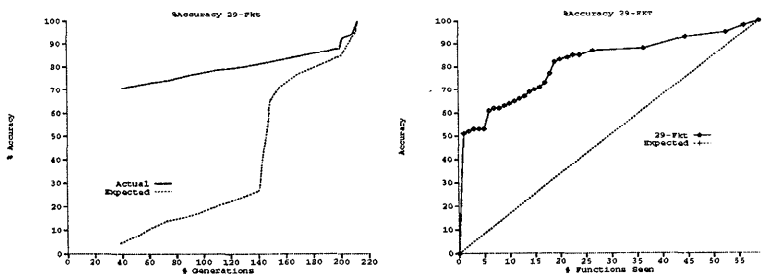

Figure 5: (a) Pattern (b) Function Recognition Accuracy for 29-Fkt Problem

the increase in combined units. This indicates that newly formed feature detectors in the network are constructed from already existing low level feature detectors. In other words, overall savings in combined units and training time are realized by creating high-level features, resulting in a rapid overall increase in the number of layers.

In Figure 5a the accuracy (degree of generalization on yet unseen patterns) obtained by TDL for the 29-fkt problem is displayed.

Figure 5b furnishes insight into the function recognition capability of TDL for the 29-fkt task. The graph depicts the relation between the average number of problems seen and the percentage of correctly identified functions. [4]

Summary

Taking the step from one-shot learning to develop an automatic network construction algorithm capable of trans-dimensional learning can be viewed as a step towards a new, and more powerful learning environment. Perceiving units within a network solely as features and learning as a process of bottom-up feature construction were necessary notions to develop a feasible implementation of a trans-dimensional learner. Basing local feature training on the simple perceptron rule and combining evolutionary methods to effectively create training partitions, have substantially contributed towards the construction of a more flexible learning system. Contrary to intuition, it was noted that learning a large set of diverse problems can be either equal to or even simpler than attempting to learn a subset of the very same problems. Even across a highly diversified set of domains can powerful hidden features be constructed and help decrease learning time and network complexity as more problems are encountered. These powerful hidden features support adjusting *learning bias* depending on the type of problems presented by simplifying the learning process itself and help elevate TDL above simple *one-shot* learning systems. The findings of this study are of importance, since they suggest that learning new features can be substantially improved by learning on top of pre-existing knowledge, even if there appears little in common between the two. Even though this is an initial study in the applicability of trans-dimensional learning to solve the general problem of *learning about learning*, it is one that has given rise to some interesting results and it is hoped that future work will prove as fertile.

[4] The 29-fkt problem actually consists of 59 functions. This number is obtained by adding up the number of outputs for all 29 problems.

References

Baffes, P.T. and Zelle, J.M (1992). Growing Layers of Perceptrons: Introducing the Extentron Algorithm, *Proceedings of the 1992 International Joint Conference on Neural Networks* (pp. II-392- II-397), Baltimore, MD., June.

Bengio, Y., Bengio, S., Cloutier, J., Gecsei, J. (1992) On the optimization of a synaptic learning rule, *Conference of Optimality in Biological and Artificial Neural Networks*, Dallas, USA.

Chalmers, D.J. (1990) The Evolution of Learning: An experiment in Genetic Connectionism, In D.S. Touretsky, J.L. Elman, T.J. Sejnowski, and G.E. Hinton (Eds.) *Proceedings of the 1990 Connectionists Models Summer School.*

Cheng, J., Fayyad, U.M., Irani, K.B., Qian, Z. (1988) Improved Decision Trees: A Generalized Version of ID3, *Proceedings of the 5th Inetrnational Conference on Machine Learning*, Ann Arbor, Michigan, June.

Fahlman, S.E. and Lebiere, C. (1990). The Cascade-Correlation Learning Architecture, In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2* (pp. 524-532). San Mateo, CA.: Morgan Kaufmann.

Frean, M. (1991). The Upstart Algorithm: A Method for Constructing and Training FeedForward Neural Networks, *Neural Computation*, 2, 198-209.

Holland, J.D. (1975) Adaption in Natural and Artificial Systems. University of Michigan Press, AnnArbor, MI.

Quinlan, J.R., (1979) Discovering rules by induction from large collections of examples. In D. Michie (Ed.), *Expert systems in the micro electronic age.* Edinburgh University Press.

Romaniuk, S.G., Hall, L.O. (1993) Divide and Conquer Networks. *Neural Networks*, Vol. 6, pp. 1105-1116.

Romaniuk, S.G. (1993) Evolutionary Growth Perceptrons. In S. Forrest *Genetic Algorithms : Proceedings of the 5th International Conference*, Morgan Kaufmann.

Valiant, L.G. (1984). A Theory of the learnable. *Comm. Ass. Comput. Mach.* 27(11), 1134-1142.