Comparison of Performance of Variants of Single-layer Perceptron Algorithms on Non-separable Datasets

Rajesh Parekh, Jihoon Yang and Vasant Honavar *
Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, IA 50011. U.S.A.
{parekh|yang|honavar}@cs.iastate.edu

Abstract

We present a detailed experimental comparison of the *pocket algorithm*, *thermal perceptron*, and *barycentric correction procedure* algorithms that most commonly used algorithms for training *threshold logic units* (TLUs). Each of these algorithms represent stable variants of the standard *perceptron learning rule* in that they guarantee convergence to zero classification errors on datasets that are *linearly separable* and attempt to classify as large a subset of the training patterns as possible for datasets that are not linearly separable. For datasets involving patterns distributed among M different categories ($M > 2$) a group of M TLUs is trained, one for each of the output classes. These TLU's can be trained either *independently* or as a *winner-take-all* (WTA) group. The latter mechanism accounts for the interactions among the different output classes and exploits the fact that a pattern can ideally belong to only one of the M output classes. The extension of the pocket algorithm to the WTA output strategy is direct. In this paper we present heuristic extensions of the thermal perceptron and the barycentric correction procedure to WTA groups and empirically verify their performance. The performance of these algorithms was measured in a collection of carefully chosen benchmarks datasets. We report the training and generalization accuracies of these algorithms on the different datasets along with the learning time in seconds. In addition, a comparison of the learning speeds of the algorithms is indicated by means of learning curve plots on two datasets. We identify and report some distinguishing traits of these algorithms which could possibly enable making an informed choice of the training algorithm (combined with constructive learning algorithms) when certain characteristics of the dataset are known.

1 Introduction

Multi-layer networks of threshold logic units (TLU) [Gallant, 1993; Chen *et al.*, 1995; Parekh, 1998] offer an attractive framework for the design of trainable pattern classification systems for a number of reasons including: potential for parallelism and fault tolerance; significant representational and computational efficiency over disjunctive normal form (DNF) expressions and decision trees [Gallant, 1993]; and simpler digital hardware implementations than their continuous counterparts such as sigmoid neurons used in back-propagation networks [Rumelhart *et al.*, 1986].

*This research was partially supported by the National Science Foundation Grant IRI-9409580 and the John Deere Foundation Grant to Vasant Honavar.

Consider a TLU j with a weight vector $\mathbf{W}_j = [w_{j0}\ w_{j1}\ \cdots\ w_{jN}]$ (where $w_{j0} = \theta_j$ is the threshold). The output o_j^p, of the TLU, in response to an input pattern $\mathbf{X}^p = [x_0^p\ x_1^p\ \cdots\ x_N^p]$ ($\forall p\ x_0^p = 1$) is given by: $o_j^p = f(n_j^p) = 1$ if $n_j^p = \sum_{i=0}^{N} w_{ji} x_i^p \geq 0$ and $o_j^p = f(n_j^p) = -1$ otherwise. Such a neuron computes the bipolar hardlimiter function f of its net input $n_j^p = \mathbf{W}_j \cdot \mathbf{X}^p$.

The pattern classification properties of a TLU (and networks of TLUs) are better understood in geometrical terms [Nilsson, 1965; Chen *et al.*, 1995]. A TLU or neuron j with weight vector $\mathbf{W}_j$ implements an $(N-1)$-dimensional hyperplane given by $\mathbf{W}_j \cdot \mathbf{X}^p = 0$ which partitions the N-dimensional Euclidean pattern space defined by the coordinates $x_1, \cdots, x_N$ into two regions (or two classes). A given set of *examples* $\mathcal{S} = \mathcal{S}_+ \cup \mathcal{S}_-$ where $\mathcal{S}_+ = \{(\mathbf{X}^p, d_j^p) \mid d_j^p = 1\}$ and $\mathcal{S}_- = \{(\mathbf{X}^p, d_j^p) \mid d_j^p = -1\}$ (where d_j^p is the desired output of the TLU j for the input pattern $\mathbf{X}^p$), is said to be *linearly separable* if and only if $\exists \hat{\mathbf{W}}_j$ such that $\forall \mathbf{X}^p \in \mathcal{S}_+$, $\hat{\mathbf{W}}_j \cdot \mathbf{X}^p \geq 0$ and $\forall \mathbf{X}^p \in \mathcal{S}_-$, $\hat{\mathbf{W}}_j \cdot \mathbf{X}^p < 0$. A number of iterative algorithms are available for finding such a $\hat{\mathbf{W}}_j$ (if one exists — i.e., when $\mathcal{S}$ is linearly separable). Most of them use some variant of the perceptron weight update rule: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta(d_j^p - o_j^p)\mathbf{X}^p$ (where η is the learning rate, $\eta > 0$) [Rosenblatt, 1958; Nilsson, 1965].

An extension of the simple perceptron model to multiple output categories is rather straightforward with one TLU being allocated per output category. Assuming that the training patterns can belong to M output categories, the M TLUs can be trained either *independently* or as a *winner-take-all (WTA)* group. The former is similar to 2-category classification which involves just a single TLU. For M categories, the TLU i is trained with $\mathcal{S}_+$ being the set of patterns belonging to class i and $\mathcal{S}_-$ being the set of patterns belonging to all the other classes. The WTA output strategy takes into account the fact that each pattern can belong to only one output class. Here the weight updates are geared toward pushing the target TLU to have the highest net input among the group of M TLUs. Let $\mathbf{D}^p$ and $\mathbf{O}^p$ represent the desired and obtained output vectors in response to input pattern $\mathbf{X}^p$ respectively. The weight vectors of the M TLUs are $\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_M$ respectively and $\mathbf{O}^p = [O_1^p, O_2^p, \cdots, O_M^p]$. $\mathbf{O}^p$ is computed as follows: If $\exists j \in 1, \cdots, M$ such that $\mathbf{W}_j \cdot \mathbf{X}^p > \mathbf{W}_i \cdot \mathbf{X}^p\ \forall i \neq j, i = 1, \cdots, M$ then $O_j^p = 1$ and $O_i^p = -1, i \neq j$. If $\exists j_1, j_2, \cdots, j_k \in 1, \cdots, M$ such that $\mathbf{W}_{j_1} \cdot \mathbf{X}^p = \mathbf{W}_{j_2} \cdot \mathbf{X}^p = \cdots = \mathbf{W}_{j_k} \cdot \mathbf{X}^p$ and $\mathbf{W}_{j_1} \cdot \mathbf{X}^p > \mathbf{W}_i \cdot \mathbf{X}^p \forall i \notin \{j_1, j_2, \cdots, j_k\}$ then $O_j^p = -1, \forall j \in 1, \cdots, M$. The weights are then modified according to the perceptron weight update rule as independent training: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta(D_j^p - O_j^p)\mathbf{X}^p\ \forall j \in 1, \cdot, M$. The WTA training offers a significant advantage over independent training in that pattern classes that are only pairwise separable from each other can be correctly classified using WTA while in independent training only pattern classes that are independently separable from each other can be correctly classified.

Some advantages of the perceptron family of learning algorithms include the existence of well-known convergence results [Rosenblatt, 1958; Nilsson, 1965; Minsky & Papert, 1969], and substantially faster learning as compared to typical gradient-based error minimization strategies. However, they also have the following limitations:

- They behave poorly on datasets that are not linearly separable - i.e., the classification accuracy on the training set can fluctuate considerably from iteration to iteration [Gallant, 1993].
- They alone are not sufficient to implement arbitrarily complex decision regions that may be necessary to deal with training sets that are not linearly separable.

The focus of this paper is on variants of perceptron algorithms that address these two limitations. These variants, while preserving the convergence properties of the perceptron algorithm on linearly separable data, attempt to find *near-optimal* weights (so as to correctly classify as large a fraction of the training set as possible) when the training dataset is not linearly separable. One approach to overcome the second limitation is to use *generative* or *constructive* learning algorithms [Honavar

& Uhr, 1993; Gallant, 1993; Parekh, 1998; Honavar, 1998b]. Such constructive algorithms rely on the addition of typically one (but in some cases, a few) neurons at a time to build a multi-layer perceptron that correctly classifies a given training set. Each added neuron needs to be trained using an appropriate weight modification algorithm. Since constructive learning algorithms are designed to deal with non-linearly separable datasets, the behavior of the weight modification routine on such data is critical to their performance. A perceptron learning algorithm has its own *inductive bias*. In other words, the process of determining the weights is different in each algorithm. The bias makes a particular algorithm more suitable to a particular problem and yield better performance than other algorithms. It is against this background that we approach our study of the performance of variants of perceptron algorithms on non-linearly separable datasets.

This paper considers the following three algorithms: *pocket algorithm* [Gallant, 1993], *thermal perceptron* [Frean, 1992] and *barycentric correction procedure* [Poulard, 1995]. The performance of the algorithms is compared using a variety of both real-world and toy datasets. A majority of these datasets are linearly non-separable. The three algorithms give 100% training accuracy on datasets that are linearly separable and attempt to classify as large a subset of the training set as possible in the case of non-linearly separable datasets.

The rest of this paper is organized as follows: Section 2 introduces the three single layer learning algorithms, provides the pseudo code and analyzes the time and space complexity of each algorithm. Section 3 describes the datasets used in experiments. Section 4 depicts the results of the comparative experiments of three algorithms. Section 5 concludes with a summary and discussion of future research.

2 Description of the Training Algorithms

2.1 Pocket Algorithm

The perceptron algorithm updates weights iteratively by adding (or subtracting) a fraction of the misclassified pattern to the current weight vector in a bid to correctly classify as many patterns as possible as learning proceeds.

The key idea behind the *pocket algorithm* [Gallant, 1993] which is explicitly designed to improve the behavior of the perceptron algorithm on non-linearly separable data is to maintain an additional weight vector $\mathbf{W}_{pocket}$ in addition to current $\mathbf{W}$. $\mathbf{W}_{pocket}$ stores the best weight setting encountered during training. A further refinement on this idea, called the *ratchet* modification (RP) [Gallant, 1993], is to ensure that replacement of $\mathbf{W}_{pocket}$ by $\mathbf{W}$ is performed only if $\mathbf{W}$ correctly classifies a greater fraction of the training set than $\mathbf{W}_{pocket}$. The pocket convergence theorem [Gallant, 1993] guarantees that the RP will find an optimal weight setting given enough training time.

In the following discussion on multi-category algorithms, $\mathbf{W}$ and $\mathbf{W}_{pocket}$ denote the entire set of weight vectors of M output neurons in the network. $\mathbf{W}_j$ represents a specific weight vector of neuron j.

2.1.1 Independent and WTA RP

1. Initialize $\mathbf{W}$ (can be initialized to 0 or small random values);
2. for k := 1 to (# of Epoch) do
3. Select a training example $(\mathbf{X}^p, \mathbf{D}^p)$ at random;
4. Compute the output vector $(\mathbf{O}^p)$;

5. if ($\mathbf{O}^p = \mathbf{D}^p$) then // correct classification

6. if (run of correct classification with $\mathbf{W}$ is longer than that with $\mathbf{W}_{pocket}$) then

7. if ($\mathbf{W}$ correctly classifies more training examples than $\mathbf{W}_{pocket}$) then

8. Replace $\mathbf{W}_{pocket}$ by $\mathbf{W}$ and adjust the length of correct run;
else

9. Update weight vectors: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta \cdot (D_j^p - O_j^p)\mathbf{X}^p, \forall j \in 1, \cdots, M$

2.1.2 Time Complexity

Let N_{epoch} be the total number of epochs for which the algorithm is trained. Let N_{in} and N_{out} be the number of input and output neurons respectively. Let $N_{pattern}$ be the number of training patterns.

Step 1,4 and 8 take $\mathcal{O}(N_{in} \cdot N_{out})$. Step 3 and 6 take $\mathcal{O}(1)$. Step 5 takes $\mathcal{O}(N_{out})$. Step 7 takes $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$. Step 9 takes $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the total time complexity (at step 2) is $\mathcal{O}(N_{epoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$.

2.1.3 Space Complexity

The space requirement for input patterns and their targets is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$; and for $\mathbf{W}_{pocket}$ and $\mathbf{W}$ is $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the overall space complexity is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$.

2.2 Thermal Perceptron Algorithm

The rationale behind the thermal perceptron algorithm (TP) [Frean, 1992] is to control the weight updates to avoid drastic changes for outliers as learning progresses. The fact that the weight update rule of the standard perceptron algorithm for misclassifications is the same irrespective of the magnitude of the error can cause severe fluctuations in the classification rate for non-separable datasets. A damping factor is introduced in the weight update equation to stabilize learning: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta \frac{1}{T}(D_j^p - O_j^p)\mathbf{X}^p e^{-|\phi|/T}$ where ϕ is the net input for the output neuron and T is the temperature. The temperature T is set to an initial value T_0 at the start of learning and gradually annealed to 0 as the training progresses. Since the exponent effectively decays the learning rate, the probability of undoing previous work is decreased as training progresses. In effect, the algorithm behaves like the perceptron algorithm at the start and avoids any large weight changes at the end of training. Note that the performance of this algorithm is heavily dependent on the initial temperature. This difficulty can be overcome to a significant extent if at the end of each epoch the initial temperature T_0 is set to the average net input over that particular epoch [Burgess, 1994]. The TP can be directly applied to multi-category classification problems using WTA computation as in RP. However, it is reasonable to account for the interactions between output neurons in the computation of ϕ. In other words, the difference of net inputs between target output neuron and the neuron with highest net input (step 8 of the pseudo-code in Section 2.2.1) is used as ϕ in the weight update formula. (In fact, direct extension of TP to WTA groups was found to perform poorly and thus the above method was implemented).

2.2.1 Independent and WTA TP

1. Initialize $\mathbf{W}$ (can be initialized to 0 or small random values);
2. Set initial temperature $T_0 = 1$; $\gamma = 1$; $T = \gamma T_0$;
3. for k := 1 to (# of Epoch) do
4. for l := 1 to (# of Pattern) do
5. Select a training example $(\mathbf{X}^p, \mathbf{D}^p)$ at random;
6. Compute the output vector $(\mathbf{O}^p)$;
7. if $(\mathbf{O}^p \neq \mathbf{D}^p)$ then // incorrect classification
8. Update weight vectors:
   - Independent: $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta \frac{1}{T} \cdot (D_j^p - O_j^p)\mathbf{X}^p e^{-|\mathbf{W}_j \cdot \mathbf{X}^p|/T}, \forall j \in 1, \cdots, M$
   - WTA (neuron i has the highest net input):
     $\mathbf{W}_j \leftarrow \mathbf{W}_j + \eta \frac{1}{T} \cdot (D_j^p - O_j^p)\mathbf{X}^p e^{-|\mathbf{W}_i \cdot \mathbf{X}^p - \mathbf{W}_j \cdot \mathbf{X}^p|/T}, \forall j \in 1, \cdots, M$
9. Compute the average net input (ϕ_{avg}) over all output neurons;
10. $\gamma = \gamma - (1/\#\text{ of Epoch})$; $T_0 = (2T_0 + 2\phi_{avg})/3$; $T = \gamma T_0$;

2.2.2 Time Complexity

We use the same notation as described in RP. Step 1 and 6 take $\mathcal{O}(N_{in} \cdot N_{out})$. Step 2, 5, 9 and 10 take $\mathcal{O}(1)$. Step 7 takes $\mathcal{O}(N_{out})$. Step 8 takes $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, step 4 takes $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$. Therefore, the total time complexity (at step 3) is $\mathcal{O}(N_{epoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$.

Note that TP has a merit of not requiring the ratchet test (i.e., computing the overall training accuracy), but it depends on expensive exponent calculations and floating point arithmetic.

2.2.3 Space Complexity

As in RP, input patterns and their targets need $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$, and $\mathbf{W}$ requires $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the overall space complexity is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$.

2.3 Barycentric Correction Procedure

The barycentric correction procedure (BCP) [Poulard, 1995] is an efficient algorithm for training single layer neurons. It is based on the geometric properties of the training patterns and provides a framework for rapidly determining a stable weight setting that correctly classifies as large a subset of the training patterns as possible.

The BCP algorithm for two-category classification involves iteratively computing the barycenters for each of the two classes and the threshold in a bid to minimize the number of misclassifications. The BCP features separate methods for computing the weights and the threshold of the TLU being trained. Let N_1 and N_0 be the number of patterns in $\mathcal{S}_+$ and $\mathcal{S}_-$ respectively. The barycenters $\mathbf{b}_1$ and $\mathbf{b}_0$ represent the weighted averages of the patterns in $\mathcal{S}_+$ and $\mathcal{S}_-$ respectively with $\boldsymbol{\alpha}$= $(\alpha_1, \cdots, \alpha_{N_1})$ and $\boldsymbol{\mu}$= $(\mu_1, \cdots, \mu_{N_0})$ representing the weighting coefficients for patterns belonging

to $\mathcal{S}_+$ and $\mathcal{S}_-$ respectively. The weight vector $\mathbf{W} = (w_1, \cdots, w_N)$ is determined as $\mathbf{W} = \mathbf{b}_1 - \mathbf{b}_0$. The threshold θ is then chosen to optimize classification accuracy. The sets $\nu_1 = \{-\mathbf{W} \cdot \mathbf{X}^p \mid \mathbf{X}^p \in \mathcal{S}_+\}$ and $\nu_0 = \{-\mathbf{W} \cdot \mathbf{X}^p \mid \mathbf{X}^p \in \mathcal{S}_-\}$ representing the projections of the individual patterns on the weight vector $\mathbf{W}$ are first computed. If $\max(\nu_1) < \min(\nu_0)$ it is clear that the projections of the patterns belonging to the two classes do not overlap and hence the patterns $\mathcal{S}_+$ and $\mathcal{S}_-$ are linearly separable. θ is set to $\frac{\max(\nu_1)+\min(\nu_0)}{2}$ and the separating hyperplane is given by $\mathcal{H} = (\theta, \mathbf{W})$. If however, $\max(\nu_1) \geq \min(\nu_0)$ then the pattern set is not linearly separable and θ is selected randomly from the interval $[-\mathbf{W} \cdot \mathbf{b}_1 \ .. \ -\mathbf{W} \cdot \mathbf{b}_0]$. Like RP, BCP maintains a pocket hyperplane $\mathcal{H}_{pocket} = (\theta_{pocket}, \mathbf{W}_{pocket})$ capturing the threshold and weights encountered during training that together give minimum classification error. For each training epoch i the candidate pocket hyperplane is denoted as $\mathcal{H}^i_{pocket} = (\theta^i_{pocket}, \mathbf{W})$. θ^i_{pocket} is selected from a pool of values representing the overlapping region of patterns belonging to both classes in the set of projections $\nu = \nu_1 \cup \nu_0$. The separation (or *gap*) of the patterns from the hyperplane $\mathcal{H}^i_{pocket}$ is computed as the sum of the distances from $\mathcal{H}^i_{pocket}$ of the closest patterns on either side. Finally, $\mathcal{H}^i_{pocket}$ replaces the current pocket hyperplane ($\mathcal{H}_{pocket}$) if the number of misclassifications is less than the number of misclassifications of $\mathcal{H}_{pocket}$ or if the number of misclassifications is the same and the gap is greater than the gap of $\mathcal{H}_{pocket}$. To end one epoch, the weighting coefficients of the patterns that are still misclassified are boosted up by a positive weighting modification. Intuitively, this causes the misclassified patterns of the two classes to be weighted more heavily in the computation of the barycenters. Training is performed for a prespecified number of epochs at the end of which the best weights represented by the pocket hyperplane are returned.

The multicategory extension of the BCP is implemented as a sequence of M calls to the two-category BCP procedure once for each of the M pattern classes. The training set for output neuron j is constructed by assigning target output 1 to patterns of class j and output -1 to all other patterns.

The extension of the BCP to WTA groups involves treating the barycenters for each class of patterns as the weights for the corresponding neuron. The thresholds for the neurons are then determined by minimizing the *loss* due to misclassification. The loss-minimization algorithm [Hrycej, 1992] can be adapted for this purpose. Of course, the loss minimization algorithm can be used by itself as a weight training rule for the weights of the TLU. However, the convergence speed of this process which is based on loss minimization by gradient decent is excruciatingly slow for the complex loss surface when it is used to train the weights and the threshold. Thus we use the loss minimization routine to compute just the thresholds in this case. Given the set of weights, the loss can be defined as the sum squared error incurred in classifying each pattern. Suppose the TLU numbered i with weight vector $\mathbf{W}_i$ (remember that in the case of the BCP the threshold θ_i is computed separately from the weight vector) produces the highest activation among all the neurons for pattern $\mathbf{X}_p$. Suppose j is the correct classification for pattern $\mathbf{X}^p$. Then the cumulative loss for the training patterns is given by $Q = \sum_p (n^p_i - n^p_j)^2$ where $n^p_i = \mathbf{X}^p \cdot \mathbf{W}_i$ is the activation of neuron i in response to pattern $\mathbf{X}^p$. It can be shown that the cumulative loss function is a convex differentiable function of the modifiable thresholds, and consequently, has a unique minimum [Shynk, 1990; Hrycej, 1992]. The thresholds that correspond to the minimum value of Q are found by gradient descent. It is easy to prove that such a learning rule is guaranteed to find a set of separating thresholds if the training set is linearly separable. Even if the training set is not linearly separable, this method guarantees to find the thresholds that minimize the cumulative loss and hence maximize the number of correctly classified patterns. However, the quality of the solution is a function of the distribution of patterns in the pattern space. Because the purpose of introducing loss minimization algorithm here is to find the optimal thresholds to minimize the misclassification

and use the routine as an inner loop of WTA BCP, we provide a limited number of iterations to perform the gradient descent instead of allowing indefinite training time. Finally, to obviate the oscillation of cumulative loss due to large learning rate, we dynamically decrease the learning rate η if the cumulative loss diverges during the gradient descent.

2.3.1 Independent BCP

Following is the BCP for two class problems. For multi-category problems, this procedure is run several times (once for each category) as described before.

1. Initialize $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$ to values in the range $[1, a]$;
2. for k := 1 to (# of Epoch) do
3. Compute $\mathbf{b_1}$ and $\mathbf{b_0}$;
4. Set $\mathbf{W} = \mathbf{b_1} - \mathbf{b_0}$;
5. Compute ν_1 and ν_0;
6. if $(\max \nu_1 < \min \nu_0)$ then begin;
7. Set $\theta = \frac{\max \nu_1 + \min \nu_0}{2}$;
8. Return $\mathcal{H} = [\theta, \mathbf{W}]$ and stop;
   else
9. Pick θ randomly from $[-\mathbf{W} \cdot \mathbf{b}_1 \; .. \; -\mathbf{W} \cdot \mathbf{b}_0]$;
10. Compute the candidate θ^i_{pocket} and the gap of $\mathcal{H}^i_{pocket}$;
11. Replace the pocket hyperplane $\mathcal{H}_{pocket}$ by $\mathcal{H}^i_{pocket}$ if $\mathcal{H}^i_{pocket}$ correctly classifies more training examples than $\mathcal{H}_{pocket}$ or has same classification accuracy and a larger gap;
12. Update $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$;

2.3.2 WTA BCP

Let $\boldsymbol{\alpha}$ denote a collection of weighting coefficients $\boldsymbol{\alpha}_j$. Let $\mathbf{W}$ (and $\mathbf{W}_{pocket}$) denote a collection of weight vectors $\mathbf{W}_j$ (and pocket vectors $\mathbf{W}_{pocket}$) for each class j. Let $\boldsymbol{\Theta} = [\theta_1, \cdots, \theta_M]$ be the thresholds of the M neurons and $\boldsymbol{\Theta}_{pocket}$ be the pocket thresholds.

1. Initialize $\boldsymbol{\alpha}$;
2. for k := 1 to (# of Epoch) do
3. for j := 1 to M do
4. Compute $\mathbf{b}_j$;
5. Set $\mathbf{W}_j = \mathbf{b}_j$;

6. Determine $\boldsymbol{\Theta}$ by loss minimization;
7. if (all patterns are correctly classified by $\mathcal{H} = [\boldsymbol{\Theta}, \mathbf{W}]$) then
8. return $\mathcal{H}$ and stop;
9. else if ($\mathcal{H}$ results in fewer errors compared to $\mathcal{H}_{pocket} = [\boldsymbol{\Theta}_{pocket}, \mathbf{W}_{pocket}]$) then
10. $\mathcal{H} \leftarrow \mathcal{H}_{pocket}$;
11. Update $\boldsymbol{\alpha}$;

2.3.3 Time Complexity

Using the same notation as in RP we analyze the time complexity as:

1. Independent BCP
   Step 1, 3, 6 and 12 take $\mathcal{O}(N_{pattern})$. Step 4 takes $\mathcal{O}(N_{in})$. Step 5 and 11 take $\mathcal{O}(N_{pattern} \cdot N_{in})$. Step 7, 8 and 9 take $\mathcal{O}(1)$. Step 10 takes $\mathcal{O}(N_{pattern} \lg N_{pattern})$. Thus, the total time complexity (at step 2 with multicategory) is $\mathcal{O}(\max[N_{out} \cdot N_{epoch} \cdot N_{pattern} \lg N_{pattern}, N_{out} \cdot N_{epoch} \cdot N_{pattern} \cdot N_{in}])$. (For multicategory problems, the algorithm should be run for each output.)
2. WTA BCP
   Step 1, 4 and 11 take $\mathcal{O}(N_{pattern})$. Step 5 takes $\mathcal{O}(N_{in})$, and therefore step 3 takes $\mathcal{O}(\max[N_{out} \cdot N_{in}, N_{out} \cdot N_{pattern}])$. Step 6 takes $\mathcal{O}(N_{innerepoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$ (*innerepoch* is needed for loss minimization). Step 7 takes $\mathcal{O}(N_{pattern} \cdot N_{in})$. Step 8 and 9 take $\mathcal{O}(1)$. Step 10 takes $\mathcal{O}(N_{in} \cdot N_{out})$. Thus, the total time complexity (at step 2) is $\mathcal{O}(N_{epoch} \cdot N_{pattern} \cdot N_{in} \cdot N_{out})$.

2.3.4 Space Complexity

As in RP, the space requirement for the input patterns and their targets is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$. $\mathbf{W}$, $\mathbf{W}_{pocket}$ and $\mathbf{b}$ require $\mathcal{O}(N_{in} \cdot N_{out})$, and $\boldsymbol{\Theta}$ and $\boldsymbol{\Theta}_{pocket}$ require $\mathcal{O}(N_{out})$. Thus, the overall space complexity is $\mathcal{O}(N_{pattern} \cdot N_{in} \cdot N_{out})$.

3 Datasets

In order to conduct a thorough and systematic comparison between the three algorithms, a wide range of datasets was chosen based on a set of carefully chosen criteria which involved:

- *Attribute Type*: binary/bipolar, integer and real valued attributes.
- *Number of Output Categories*: two classes or multiple output classes.
- *Linear separability*: separable and non-separable sets of training patterns.

The real-world datasets used are available at UC Irvine's Machine Learning repository [Murphy & Aha, 1994]. Table 1 summarizes the characteristics of the datasets selected for our experiments.

Table 1: Datasets used in the experiments. *Train* and *Test* are the umber of patterns in the training and test sets, respectively. *Attribute* is the number of input attributes. *Class* is the number of output classes.

Dataset	*Train*	*Test*	*Attribute*	*Attribute Type*	*Class*
balance	416	209	4	real	3
concentric (two concentric circles)	1666	834	2	real	2
glass	142	72	9	real	6
ionosphere	234	117	34	real	2
liver	230	115	6	real	2
p7 (7-bit parity)	128	0	7	bipolar	2
pima	512	256	8	real	2
r5 (5-bit random)	32	0	5	bipolar	3
sep (separable data)	200	100	4	real	2
soybean	33	14	35	integer	4
wdbc	380	189	30	real	2
wine	120	58	13	real	3
WTA-sep (separable data in WTA sense)	44	21	2	real	5

4 Experiments and Results

Several experiments were practiced to make a fair comparison between the three perceptron algorithms (BCP, RP and TP) in terms of classification accuracy, training time, and learning curve. Also, a constructive learning algorithm was chosen to study the inductive bias of the perceptron algorithms in the constructive learning algorithm.

4.1 Classification Accuracy

To compare the classification accuracy, sufficient learning time (in terms of the number of *epochs*) was allowed to each algorithm. An epoch indicates a single random pattern presentation in RP, l randomly drawn patterns from the training set in TP (where l is the size of the training set), and a presentation of the entire set of training patterns in BCP. A run of RP was terminated upon attaining 100% accuracy on the training data or when the pocket weights did not undergo update for a stretch of 50,000 epochs (pattern presentations). Training was conducted until either all patterns were correctly classified or 500 training epochs were reached for TP and BCP. In the case of TP, a heuristic alteration of the initial temperature was performed after each epoch as suggested in [Burgess, 1994]. (See the pseudo-code given in Section 2.2.1).

Table 2 and 3 report the average accuracy and the standard deviation over 25 runs of the three algorithms with independent and WTA training strategy, respectively. Datasets without a test set have '$-$'s in the column for testing accuracy. Dynamic reduction of the learning rate was performed in the loss minimization routine for the WTA BCP. If the cumulative loss diverged over 5 consecutive epochs of the loss minimization routine the learning rate was decreased to 0.95 times its current value (with the initial learning rate set to 1.0). $\boldsymbol{\Theta}$ is randomly initialized and $\boldsymbol{\alpha}$ are randomly initialized to integer values between 1 and 4.

As we can see from Table 2 and 3, given enough training time the three algorithms are comparable in general. However, each algorithm outperforms the others in some datasets. For almost all multi-category datasets, WTA strategy gave higher accuracies (except the **wine** dataset for BCP).

Table 2: Classification Accuracy (independent)

	RP		TP		BCP	
Dataset	*train*	*test*	*train*	*test*	*train*	*test*
balance	88.5 ± 0.5	85.6 ± 0.8	84.3 ± 2.8	82.2 ± 1.4	87.7 ± 0.5	82.1 ± 1.1
concentric	62.7 ± 0.0	64.0 ± 0.0	62.7 ± 0.0	64.0 ± 0.0	55.9 ± 0.0	53.1 ± 0.1
glass	46.8 ± 1.6	41.8 ± 4.5	39.1 ± 2.1	28.9 ± 2.9	54.4 ± 2.0	40.7 ± 4.1
ionosphere	91.9 ± 0.9	95.0 ± 1.7	95.4 ± 0.5	91.6 ± 2.3	91.9 ± 0.2	95.1 ± 0.9
liver	70.7 ± 1.0	70.8 ± 3.0	71.7 ± 0.9	72.1 ± 2.6	72.0 ± 0.3	70.2 ± 1.7
p7	61.0 ± 4.5	–	65.2 ± 1.6	–	63.6 ± 3.8	–
pima	68.9 ± 1.0	68.7 ± 2.0	72.0 ± 0.8	71.0 ± 1.4	74.6 ± 0.4	77.8 ± 0.7
r5	56.9 ± 3.4	–	58.6 ± 4.9	–	57.6 ± 3.8	–
sep	100 ± 0.0	99.8 ± 0.5	100 ± 0.0	99.8 ± 0.5	100 ± 0.0	99.8 ± 0.4
soybean	100 ± 0.0	92.3 ± 7.4	100 ± 0.0	94.3 ± 4.6	100 ± 0.0	80.9 ± 4.0
wdbc	92.3 ± 0.3	91.4 ± 4.1	92.7 ± 0.2	90.8 ± 1.4	90.3 ± 0.0	91.6 ± 0.2
wine	71.8 ± 6.3	75.9 ± 10.3	77.1 ± 1.5	85.4 ± 2.2	73.9 ± 3.9	83.2 ± 3.3
WTA-sep	71.1 ± 1.4	78.7 ± 3.1	59.9 ± 12.2	62.5 ± 11.7	58.6 ± 1.1	57.0 ± 1.0

Table 3: Classification Accuracy (WTA)

	RP		TP		BCP	
Dataset	*train*	*test*	*train*	*test*	*train*	*test*
balance	92.1 ± 0.9	89.2 ± 1.4	91.8 ± 1.1	89.5 ± 0.8	92.3 ± 0.3	89.9 ± 0.2
glass	56.5 ± 4.5	48.2 ± 6.9	58.8 ± 3.9	43.2 ± 3.6	49.0 ± 0.5	51.2 ± 1.9
r5	67.9 ± 1.9	–	76.3 ± 1.8	–	66.6 ± 1.5	–
soybean	100 ± 0.0	98.9 ± 2.7	100 ± 0.0	96.9 ± 4.2	100 ± 0.0	100 ± 0.0
wine	91.7 ± 2.4	95.7 ± 1.2	91.5 ± 0.4	93.5 ± 1.1	70.2 ± 0.4	81.3 ± 0.6
WTA-sep	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0

4.2 Training Time

In the previous set of experiments the total training time was selected in an ad hoc manner and more training time does not necessarily improve the performance of the algorithms. We have compared the relative speeds of the three algorithms by measuring the total CPU time in seconds taken by each algorithm to reach 50%, 60%, 70%, 80% and 90% classification accuracy on the training set. Table 4∼9 show the average training times and the standard deviations needed to achieve the above accuracy milestones for 25 runs of each dataset. The total time represents the CPU time taken to achieve either 100% classification accuracy (on separable datasets) or the total time to complete the maximum epochs allowed for training. A '-' in a column indicates that the corresponding level of training was not achieved (i.e., either the training jumped to a higher level of accuracy or the training could achieve only a lower accuracy level). Since the independent BCP training algorithms for datasets with multiple output classes involves independently training each class of patterns, it is not possible to measure the total time to achieve the various accuracy levels.

From the results we observe that in general RP takes the least total time for training. This can be attributed to the simplicity of the algorithm. BCP on the other hand has the merit of achieving high accuracy very rapidly. The high total training time in the case of BCP is indicative of the time spent in training without any substantial improvement in training accuracy. The quick convergence to high accuracy levels in the BCP can be exploited to rapidly train constructive networks.

Table 4: Training Time for RP (independent)

Dataset	*50%*	*60%*	*70%*	*80%*	*90%*	*total time*
balance	0.04 ± 0.01	0.05 ± 0.01	0.06 ± 0.02	0.09 ± 0.03	–	4.03 ± 1.00
concentric	0.05 ± 0.00	0.07 ± 0.02	–	–	–	65.63 ± 60.62
glass	4.94 ± 0.00	–	–	–	–	12.77 ± 3.40
ionosphere	0.03 ± 0.01	0.05 ± 0.01	0.07 ± 0.02	0.13 ± 0.05	1.06 ± 0.62	6.70 ± 1.23
liver	0.01 ± 0.01	0.06 ± 0.04	0.82 ± 0.83	–	–	3.78 ± 1.14
p7	0.02 ± 0.02	1.16 ± 1.34	–	–	–	4.12 ± 0.97
pima	0.03 ± 0.01	0.04 ± 0.02	3.77 ± 2.79	–	–	4.92 ± 1.65
r5	0.33 ± 0.50	–	–	–	–	5.83 ± 1.48
sep	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.01	0.02 ± 0.01	0.35 ± 0.17
soybean	0.02 ± 0.01	0.03 ± 0.01	0.03 ± 0.01	0.05 ± 0.02	0.06 ± 0.02	0.08 ± 0.02
wdbc	0.06 ± 0.02	0.10 ± 0.01	0.13 ± 0.03	0.11 ± 0.03	0.21 ± 0.07	8.82 ± 4.91
wine	0.23 ± 0.25	0.59 ± 0.87	10.14 ± 1.91	20.62 ± 0.00	–	15.71 ± 5.38
WTA-sep	0.13 ± 0.06	0.40 ± 0.22	1.45 ± 0.51	–	–	4.61 ± 0.85

Table 5: Training Time for TP (independent)

Dataset	*50%*	*60%*	*70%*	*80%*	*90%*	*total time*
balance	0.26 ± 0.31	0.25 ± 0.23	0.17 ± 0.18	0.26 ± 0.22	–	48.48 ± 33.69
concentric	0.08 ± 0.02	0.08 ± 0.03	–	–	–	905.99 ± 1001.31
glass	5.04 ± 2.35	–	–	–	–	59.29 ± 30.02
ionosphere	0.04 ± 0.02	0.06 ± 0.02	0.11 ± 0.03	0.20 ± 0.06	1.00 ± 0.52	148.13 ± 27.32
liver	0.05 ± 0.06	0.23 ± 0.12	0.81 ± 1.13	–	–	30.50 ± 12.68
p7	0.01 ± 0.00	2.32 ± 1.24	–	–	–	22.70 ± 2.80
pima	0.17 ± 0.30	0.23 ± 0.30	8.71 ± 3.84	–	–	79.87 ± 28.33
r5	0.23 ± 0.21	1.27 ± 0.07	–	–	–	9.15 ± 3.52
sep	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.01	0.03 ± 0.01	0.39 ± 0.13
soybean	0.04 ± 0.01	0.05 ± 0.01	0.05 ± 0.01	0.07 ± 0.01	0.08 ± 0.02	0.10 ± 0.02
wdbc	0.30 ± 0.37	2.11 ± 2.08	1.17 ± 0.81	1.57 ± 1.00	1.72 ± 1.18	175.08 ± 36.51
wine	0.29 ± 0.13	0.51 ± 0.49	5.73 ± 0.36	–	–	88.34 ± 4.30
WTA-sep	0.14 ± 0.06	0.38 ± 0.20	0.96 ± 0.35	–	–	7.26 ± 1.68

Table 6: Training Time for BCP (independent)

Dataset	*50%*	*60%*	*70%*	*80%*	*90%*	*total time*
concentric	0.43 ± 0.01	–	–	–	–	206.97 ± 2.21
ionosphere	–	–	0.04 ± 0.01	0.14 ± 0.08	1.29 ± 0.68	14.08 ± 0.28
liver	–	0.02 ± 0.01	0.28 ± 0.21	–	–	11.54 ± 0.32
p7	0.01 ± 0.00	0.46 ± 0.44	–	–	–	4.16 ± 0.22
pima	–	0.08 ± 0.00	0.42 ± 0.25	–	–	39.04 ± 0.64
sep	–	–	–	–	0.02 ± 0.00	0.18 ± 0.04
wdbc	–	–	–	0.06 ± 0.00	1.65 ± 0.37	25.30 ± 0.52

4.3 Learning Curve

There can be various possibilities in the process of perceptron training. For example, an algorithm can reach a *near-optimal* solution very fast but approach to the optimal solution slowly from there.

Table 7: Training Time for RP (WTA)

Dataset	*50%*	*60%*	*70%*	*80%*	*90%*	*total time*
balance	0.12 ± 0.03	0.15 ± 0.04	0.20 ± 0.07	0.23 ± 0.07	1.84 ± 1.50	12.54 ± 3.81
glass	3.59 ± 1.92	19.48 ± 4.23	–	–	–	23.94 ± 8.17
r5	0.03 ± 0.02	0.25 ± 0.28	5.09 ± 1.32	–	–	10.80 ± 2.36
soybean	0.03 ± 0.01	0.02 ± 0.01	0.03 ± 0.01	0.04 ± 0.02	0.06 ± 0.02	0.09 ± 0.03
wine	0.14 ± 0.05	0.21 ± 0.12	2.33 ± 0.82	6.14 ± 1.26	13.13 ± 2.71	24.26 ± 3.49
WTA-sep	0.08 ± 0.03	0.12 ± 0.06	0.22 ± 0.07	0.31 ± 0.09	0.57 ± 0.17	1.14 ± 0.28

Table 8: Training Time for TP (WTA)

Dataset	*50%*	*60%*	*70%*	*80%*	*90%*	*total time*
balance	0.43 ± 0.54	1.92 ± 1.04	1.46 ± 1.08	1.50 ± 1.08	2.96 ± 1.63	112.16 ± 51.20
glass	4.51 ± 3.19	17.09 ± 0.87	–	–	–	137.21 ± 22.79
r5	0.03 ± 0.02	0.22 ± 0.43	1.60 ± 0.68	–	–	75.75 ± 1.63
soybean	0.06 ± 0.02	0.08 ± 0.03	0.08 ± 0.02	0.12 ± 0.01	0.12 ± 0.02	0.15 ± 0.04
wine	0.55 ± 0.30	0.61 ± 0.24	3.90 ± 1.39	7.11 ± 1.07	9.70 ± 1.25	97.32 ± 10.33
WTA-sep	0.12 ± 0.04	0.18 ± 0.06	0.31 ± 0.07	0.49 ± 0.11	0.75 ± 0.18	1.19 ± 0.20

Table 9: Training Time for BCP (WTA)

Dataset	*50%*	*60%*	*70%*	*80%*	*90%*	*total time*
balance	–	–	–	7.92 ± 0.37	15.44 ± 3.75	358.20 ± 11.19
glass	–	–	–	–	–	174.25 ± 2.31
r5	0.40 ± 0.01	0.84 ± 0.33	–	–	–	32.88 ± 0.39
soybean	–	–	–	–	0.13 ± 0.13	0.20 ± 0.34
wine	–	2.43 ± 0.01	3.07 ± 0.19	–	–	130.61 ± 0.51
WTA-sep	–	–	0.62 ± 0.00	0.62 ± 0.01	0.64 ± 0.04	4.00 ± 5.46

On the other hand, another algorithm can reach the optimal solution with a constant speed. (For example, we can choose the former for problems that need a *reasonable* solution within a time constraint, and choose the latter for problems that need a *good* solution within a finite time limit). Therefore, studying the bias of perceptron algorithms with respect to the learning speed is of interest. We compared the learning speeds of the algorithms by plotting the learning curves of the algorithms on **ionosphere** and the **pima** datasets. The choice of these two datasets was arbitrary keeping in mind that both datasets are real-world and substantially large and fairly good training accuracies are possible with each of the three algorithms on these datasets. Ten runs were performed with the same parameter settings as described earlier. The training and generalization accuracies were measured at the end of each epoch for each of the three algorithms. Note that for the purpose of this experiment in the case of RP and TP, one epoch was measured as a presentation of l randomly chosen training patterns (where l is the total number of training patterns for the dataset) while in the case of the BCP each epoch involved seeing all the training patterns once. Training was performed for 500 epochs.

Figure 1 and 2 show the learning curves for the three algorithms. In the case of **ionosphere** both BCP and RP climb to a high level of training accuracy very rapidly. TP does poorly at the start but does stabilize to a good training and test accuracy toward the end of the training epochs. The **pima** dataset clearly shows the demarcation between the three training algorithms. Here the

BCP outperforms both TP and RP in both the training and generalization accuracies. TP starts of poorly but eventually stabilizes to an accuracy level comparable to RP.

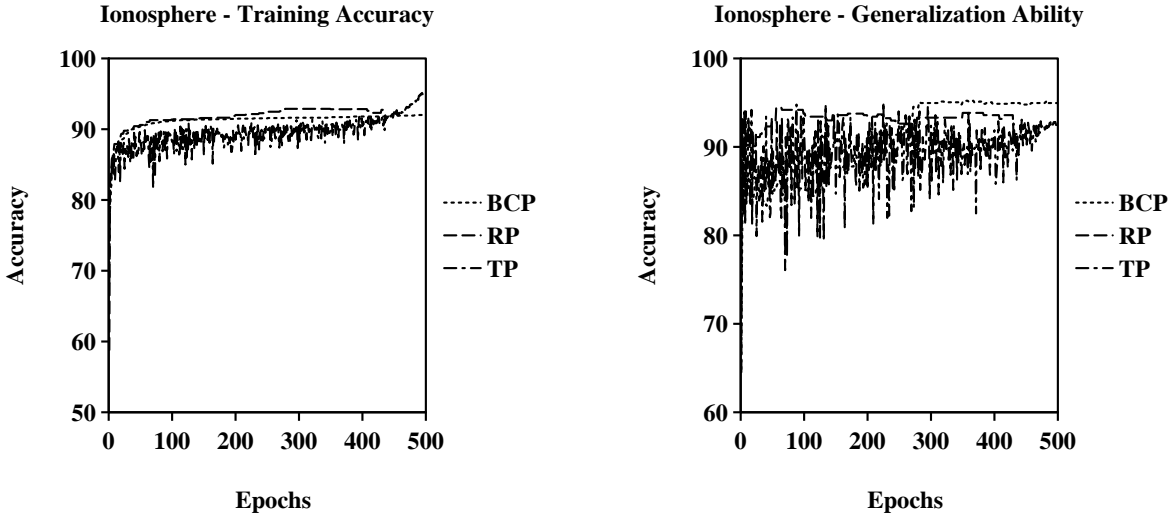


Figure 1: Learning Curve for **ionosphere**

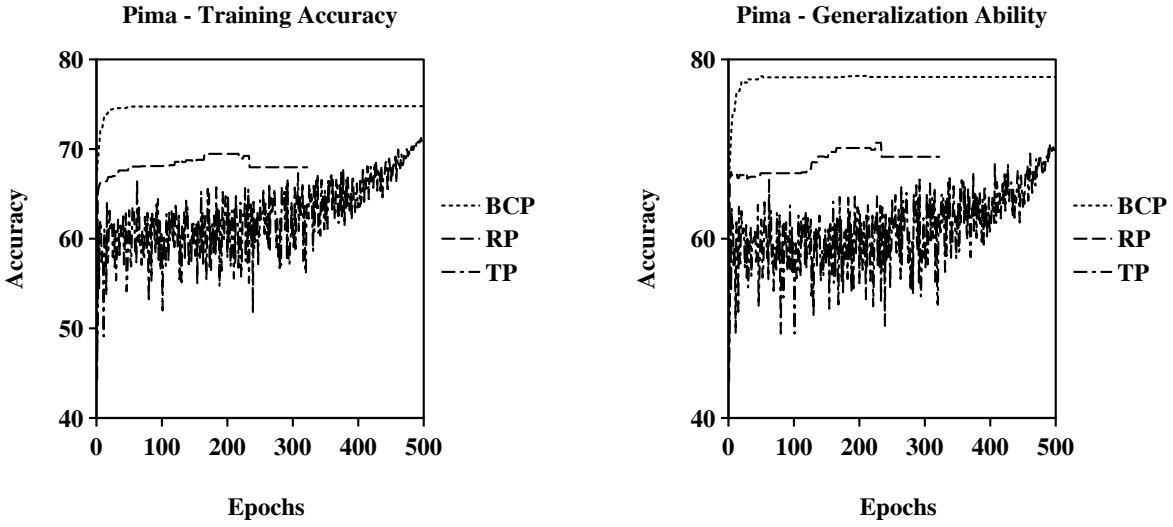


Figure 2: Learning Curve for **pima**

4.4 Perceptron Algorithms in a Constructive Neural Network Algorithm

As explained in Section 1, a perceptron learning algorithm can not classify a linearly non-separable data [Minsky & Papert, 1969]. *Constructive neural network learning algorithms* [Gallant, 1993; Honavar & Uhr, 1993; Honavar, 1998a] provide a way around this problem. They keep recruiting a set of hidden neurons and setting the connections (i.e., weights) between neurons in a systematic

way until the stopping criteria are satisfied (e.g., 100% training accuracy is reached or the number of hidden neurons recruited exceed some limit). Several constructive learning algorithms appeared in the literature and shown to guarantee 100% training accuracy theoretically [Mézard & Nadal, 1989; Nadal, 1989; Frean, 1990; Gallant, 1990; Marchand *et al.*, 1990; Burgess, 1994; Yang *et al.*, 1996; Parekh *et al.*, 1997; Parekh, 1998; Yang *et al.*, 1998]. Most of them employ a perceptron style weight update procedure (e.g., RP) to determine the weight setting between neurons. The performance of the perceptron algorithm used in the constructive learning algorithm determines the overall performance of classification. In other words, if the perceptron learning algorithm fail to find a proper weight setting for newly recruited neurons, it would not decrease the classification error and thus not converge to 100% training accuracy. Therefore, exploring the bias of a perceptron learning algorithm in the context of constructive learning algorithm is clearly of interest.

The Tiling algorithm [Mézard & Nadal, 1989; Parekh, 1998] is chosen in our experiments. The Tiling algorithm constructs a strictly layered network of TLUs. The bottom-most layer receives inputs from each of the input neurons. The neurons in each subsequent layer receive inputs from those in the layer immediately below itself. Each layer maintains M *master neurons* for M output classes. The network construction procedure ensures that the master neurons in a given layer correctly classify more patterns than the master neurons of the previous layer. Each layer maintains a set of ancillary neurons that are added and trained to ensure a *faithful representation* of the training patterns. The *faithfulness* criterion states that no two training examples belonging to different classes should produce identical output at any given layer. A sample Tiling network is shown in Figure 3.

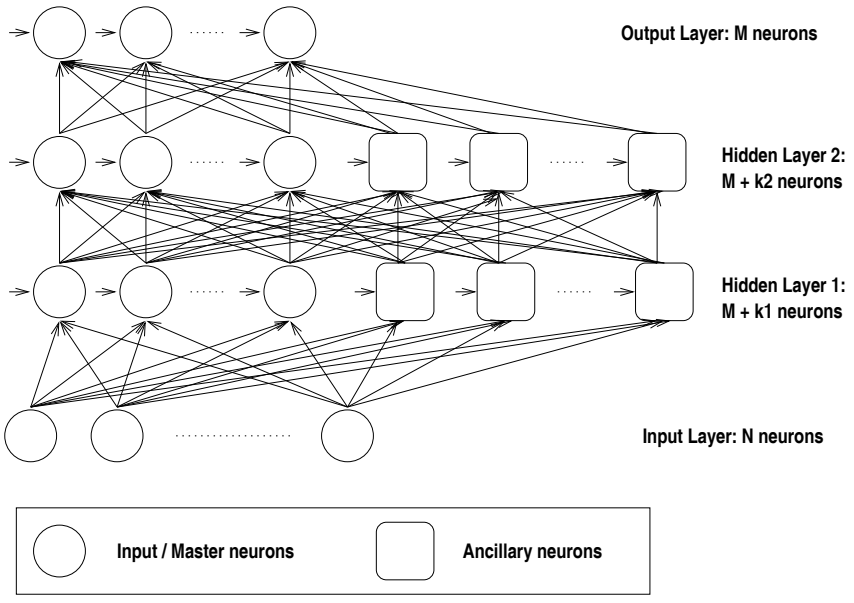


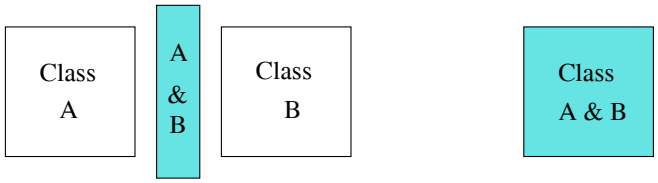
Figure 3: Tiling Network

The Tiling algorithm is shown to outperform other constructive learning algorithms in various datasets [Parekh, 1998]. The reason is most likely due to the fact that the Tiling algorithm trains neurons on progressively smaller subset of the entire training set. Against this, we chose Tiling algorithm in our experiments.

Three artificial datasets are used to study the internal bias of RP, TP and BCP:

- **concentric**: concentric circles dataset in Table 1.
- **D1**: two well-formed clusters of two classes with a region of mixed patterns belonging to different classes (See Figure 4(a)). There are 250 patterns in both training and test sets.
- **D2**: totally randomly generated patterns (See Figure 4(b)). There are 250 patterns in both training and test sets.

Table 10 shows the average performance of 10 runs of the perceptron algorithms on the three artificial datasets. Table 11 shows the average performance of 10 runs of the Tiling algorithm combined with the perceptron algorithms for the three artificial datasets (Tiling-RP, Tiling-TP, and Tiling-BCP).



(a) D1: Marginally non-separable data (b) D2: Random data

Figure 4: Two artificial datasets used in the study of inductive bias. White boxes include patterns of a classification. Shaded boxes include patterns of different classification (A and B).

Table 10: Comparison of performance of RP, TP and BCP in a single layer perceptron. *train* and *test* are training and test accuracies, respectively. *time* is the training time.

	RP		TP		BCP	
Dataset	*train*	*test*	*train*	*test*	*train*	*test*
concentric	62.7	64.0	62.7	64.0	55.9	53.1
D1	91.9	90.7	91.0	90.8	92.1	90.2
D2	55.7	53.3	53.3	47.6	56.8	56.8

Table 11: Comparison of performance of RP, TP and BCP in Tiling algorithm. *success* is the number of runs that succeeded to converge.

	Tiling-RP			Tiling-TP			Tiling-BCP		
Dataset	*success*	*train*	*test*	*success*	*train*	*test*	*success*	*train*	*test*
concentric	0	-	-	0	-	-	10	100	99.1
D1	3	100	89.2	10	100	90.0	10	100	89.3
D2	9	100	50.5	9	100	50.0	10	100	51.7

Table 10 shows comparable accuracy between the three algorithms in **D1** and **D2**. BCP gave lower accuracy for **concentric** since the barycenters of patterns in both classes are very close. However, it gave higher generalization than RP and TP in **D2**.

The performance of Tiling algorithm with different perceptron algorithms is different from that of perceptron algorithms used alone. In particular, Tiling-BCP always converged to networks with 100% training accuracy while Tiling-RP and Tiling-TP did not converge at all in **concentric**. In **D1** and **D2**, Tiling-BCP always converged to networks with 100% training accuracy while Tiling-RP and

Tiling-TP failed to converge in several runs. All three algorithms gave comparable generalization accuracy. This shows that Tiling-BCP outperforms Tiling-RP and Tiling-TP though there was not much difference in single layer perceptron learning.

5 Summary and Discussion

Three most commonly used variants of perceptron learning algorithms (pocket algorithm [Gallant, 1993], thermal perceptron [Frean, 1992], and barycentric correction procedure [Poulard, 1995]) are compared experimentally in a collection of benchmark datasets. Both independent and winner-take-all strategies were designed for the algorithms and evaluated.

Using the datasets chosen in this paper, the three algorithms were comparable in training accuracy though some algorithm performed better than others in some datasets. The difference of accuracy was large for some datasets due to the inductive bias of each algorithm. For example, **BCP** performed poor on **concentric** dataset which produces similar barycenters for each class. On the contrary, BCP performed even better than RP and TP on the difficult **pima** dataset. As expected, WTA training gave higher accuracies than independent training.

BCP approached the best accuracy very fast while RP and TP required sufficient time. Because of its simplicity, RP was the fastest given the stopping criteria set up for each algorithm.

BCP's fast convergence was also verified by the learning curves on **ionosphere** and **pima** datasets. Both BCP and TP showed nice performance of continuously increasing generalization accuracies during learning.

Constructive learning algorithms are necessary to get 100% training accuracy for linearly non-separable datasets. Most of the constructive learning algorithms rely on a perceptron style weight update procedure to find the weight setting for generated neurons. The performance of a constructive learning algorithm heavily depends on the perceptron learning algorithm it employs. Each perceptron learning algorithm has its own inductive bias. Exploiting the bias and choosing the right algorithm for a dataset will be of significant importance. Preliminary experiments on three artificial datasets show that BCP outperforms RP and TP in a constructive learning algorithm. Further experiments with various datasets combined with several constructive learning algorithms are necessary for a thorough study of the inductive bias of each perceptron learning algorithm.

References

Burgess, N. (1994). A Constructive Algorithm That Converges for Real-Valued Input Patterns. *International Journal of Neural Systems*, **5**(1), 59–66.

Chen, C-H., Parekh, R., Yang, J., Balakrishnan, K., & Honavar, V. (1995). Analysis of Decision Boundaries generated by Constructive Neural Network Learning Algorithms. *Pages 628–635 of: Proceedings of WCNN'95, July 17-21, Washington D.C.*, vol. 1.

Frean, M. (1990). The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation*, **2**, 198–209.

Frean, M. (1992). A Thermal Perceptron Learning Rule. *Neural Computation*, **4**, 946–957.

Gallant, S. (1990). Perceptron Based Learning Algorithms. *IEEE Transactions on Neural Networks*, **1**(2), 179–191.

Gallant, S. (1993). *Neural Network Learning and Expert Systems.* Cambridge, MA: MIT Press.

Honavar, V. (1998a). Machine Learning: Principles and Applications. *In:* Webster, J. (ed), *Encyclopedia of Electrical and Electronics Engineering*. New York: Wiley. To appear.

Honavar, V. (1998b). Structural Learning. *In:* Webster, J. (ed), *Encyclopedia of Electrical and Electronics Engineering*. New York: Wiley. To appear.

Honavar, V., & Uhr, L. (1993). Generative Learning Structures for Generalized Connectionist Networks. *Information Sciences*, **70**(1-2), 75–108.

Hrycej, T. (1992). *Modular Learning in Neural Networks*. New York: Wiley.

Marchand, M., Golea, M., & Rujan, P. (1990). A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons. *Europhysics Letters*, **11**(6), 487–492.

Mézard, M., & Nadal, J. (1989). Learning Feed-forward Networks: The Tiling Algorithm. *J. Phys. A: Math. Gen.*, **22**, 2191–2203.

Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press.

Murphy, P., & Aha, D. (1994). *Repository of Machine Learning Databases*. Department of Information and Computer Science, University of California, Irvine, CA.

Nadal, J. (1989). Study of a Growth Algorithm for a Feedforward Network. *International Journal of Neural Systems*, **1**(1), 55–59.

Nilsson, N. (1965). *The Mathematical Foundations of Learning Machines*. New York: McGraw-Hill.

Parekh, R. (1998). *Constructive learning: Inducing grammars and neural networks*. Ph.D. thesis, Department of Computer Science, Iowa State University, Ames, IA.

Parekh, R., Yang, J., & Honavar, V. (1997). MUpstart - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification. *Pages 1924–1929 of: Proceedings of the IEEE/INNS International Conference on Neural Networks, ICNN'97.*

Poulard, H. (1995). Barycentric Correction Procedure: A Fast Method of Learning Threshold Units. *Pages 710–713 of: Proceedings of WCNN'95, July 17-21, Washington D.C.*, vol. 1.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, **65**, 386–408.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning Internal Representations by Error Propagation. *In: Parallel Distributed Processing: Explorations into the Microstructure of Cognition*, vol. 1 (Foundations). Cambridge, Massachusetts: MIT Press.

Shynk, J. J. (1990). Performance Surfaces of a Single-layer Pereptron. *IEEE Transactions on Neural Networks*, **1**, 268–274.

Yang, J., Parekh, R., & Honavar, V. (1996). MTiling - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification. *Pages 182–187 of: Proceedings of the World Congress on Neural Networks'96.*

Yang, J., Parekh, R., & Honavar, V. (1998). DistAl: An Inter-pattern Distance-based Constructive Learning Algorithm. *Pages 2208–2213 of: Proceedings of the International Joint Conference on Neural Networks (IJCNN'98).*