Learning of higher-order perceptrons with tunable complexities

J. Phys. A: Math. Gen. **31** (1998) 7771–7784. Printed in the UK PII: S0305-4470(98)90280-3

Learning of higher-order perceptrons with tunable complexities

Hyoungsoo Yoon† and Jong-Hoon Oh

Physics Department and Basic Science Research Institute, Pohang University of Science and Technology, Pohang, South Korea 790–784

Received 5 January 1998, in final form 10 July 1998

Abstract. We study learning from examples by higher-order perceptrons, which realize polynomially separable rules. The model complexities of the networks are made 'tunable' by varying the relative orders of different monomial terms. We analyse the learning curves of higher-order perceptrons when the Gibbs algorithm is used for training. It is found that learning occurs in a stepwise manner. This is because the number of examples needed to constrain the corresponding phase-space component scales differently.

1. Introduction

The multilayer perceptron [1–3] consists of a number of processing units called neurons, which by themselves are only capable of learning linearly separable rules. While multilayer perceptrons with a large number of neurons have been proven to be universal function approximators, due to practical reasons there have been growing interests in utilizing different types of processing units. Radial basis function networks, for example, use a processing element which activates according to the distance from its centre to the pattern vector. For the McCulloch–Pitts neuron, the decision surface is a hyperplane orthogonal to the 'weight vector' of the neuron.

In this paper we consider the so-called *higher-order neurons* [4–6], which are a generalization of the conventional threshold units. Unlike the case of first-order neurons, the decision boundaries are not restricted to hyperplanes, and hence higher-order neurons are capable of learning rules more complex than linearly separable ones. Processing elements, which are capable of learning up to mth-order polynomially separable rules, are called *mth-order neurons*.

The *higher-order multilayer perceptron*, which is a layered feedforward network consisting of higher-order neurons, is known to have various interesting and useful properties. Among others, the *invariance constraints*, such as the requirement that the same patterns should be recognized as such regardless of their locations or orientations, can be handled directly in higher-order networks [7]. Since this invariance constraint is such an important rule, and yet hard to incorporate in practical pattern recognition problems [3], the higher-order neural networks have been increasingly studied [4–13] despite the fact that the higher-order units suffer from the so-called *curse of dimensionality*.

† Present address: Racah Institute of Physics and Center for Neural Computation, Hebrew University, Jerusalem 91904, Israel.

The increased cost of using higher-order neurons is also expected to be compensated by the fact that the increased computational power at the neuron level can reduce the overall complexity of the network as a whole. The classic example is the XOR problem, which cannot be implemented in a single first-order neuron ('simple perceptron') and was believed for a long time to be an indication that neural networks had very limited capability [14]. This XOR rule can be learnt by introducing one hidden layer. A second-order neuron with two input nodes is, however, capable of learning all of the 16 Boolean functions of two binary input variables including the XOR.

In general, a single mth-order neuron can *memorize* any Boolean function with up to m input variables. This is another advantage of using higher-order units. In the conventional multilayer approach, one is never certain of the complexity of the rules which the given architecture of a multilayer perceptron can handle. In using higher-order neural networks, however, one can start with a low-order neuron and increase its order gradually [12] instead of adding more 'hidden' nodes. In this way one can always deal with a well-defined class of rules, that is, polynomially separable rules of specified orders.

Even though this is by no means the only possible use of higher-order networks, we limit ourselves mainly to such uses of the networks consisting of single neurons, which we simply call *higher-order perceptrons* in this work. This is essentially a neural-network version of the familiar paradigm of curve fitting with polynomial functions.

This paper is organized as follows. We will first present the definitions of the higher-order perceptron and the polynomially separable rule in section 2. There are two possible choices in combining different order monomial terms in a single processing element: one giving equal importance to each *weight* and the other giving equal importance to each *monomial class* of weights. In this work we investigate the latter choice, which appears to be more natural. The storage property of higher-order perceptrons with the first choice was studied by Kohring [6] and it was found that they exhibited no essential novelties over first-order networks.

The storage capacity is computed in section 3 using the statistical mechanics formalism developed first by Gardner [15] and later extended by many authors [16–22]. Since we are interested in the novel features by going beyond the first order, we will restrict our calculation to second-order perceptrons. The results for general cases will be given as its straightforward extrapolation without repeating the calculations. It is shown that the complexity of higher-order networks is essentially *tunable* by adjusting the relative weight parameters. In section 4, the learning property of the higher-order perceptron will be investigated for both fixed and tunable complexity networks. In particular, the generalization errors will be calculated, also in the framework of the replica method. Finally, some open questions and directions for future research will be given in section 5.

2. Higher-order perceptrons

The mth-order perceptron is defined as the following function for an N-dimensional input vector, $\boldsymbol{x} = \{x_1, \ldots, x_N\}$,

$$
\begin{aligned}
y &= g\left(\sum_{0 \leqslant i_1 \leqslant \cdots \leqslant i_m \leqslant N} w_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m} \right) \\
&= g\left(w^{(0)} + \sum_{i=1}^{N} w_i^{(1)} x_i + \cdots + \sum_{1 \leqslant i_1 < \cdots < i_m \leqslant N} w^{(m)}_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m} \right)
\end{aligned}
$$

where x_0 is defined as 1 and $g(\cdot)$ is a transfer function, which in this work is taken to be a threshold function, $g(x) = \mathrm{sgn}(x)$. We will only consider binary patterns in this paper. That is, the input variables x_i's are restricted to ± 1 for $1 \leqslant i \leqslant N$. Note that the 'weight matrix', $w_{i_1 \ldots i_m}$, consists of submatrices of different symmetries, $w^{(k)}_{i_1 \ldots i_k}$ with $k = 0, \ldots, m$. The single-element matrix, $w^{(0)}$, is what is usually called the bias and $w^{(1)}$ is the weight vector. The diagonal parts of the submatrices, those elements which have the same indices for more than one position, are all zero, since these parts are moved to the lower-order monomial contributions due to the fact that $(x_i)^2 = 1$.

A submatrix of kth order has

$$d_k = \binom{N}{k} \tag{1}$$

terms and there are

$$D_m = \sum_{k=0}^{m} d_k$$

adjustable weights in total. D_m is called the degree of freedom of the mth-order perceptron.

One can easily see that the perceptron function given above can realize a class of rules whose decision boundaries form mth-order polynomial hypersurfaces in the N-dimensional pattern space

$$\sum_{0 \leqslant i_1 \leqslant \cdots \leqslant i_m \leqslant N} w_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m} = 0.$$

Such class of rules are called the *polynomially separable rules*. The separating capacities of these rules have been derived using combinatorial geometry in a very general context by Cover [23]. The results show that the maximum storage capacity of a neural network with D_m degrees of freedom is $2D_m$. In the next section we shall see, however, that we can make this number *tunable* by imposing separate normalization for each monomial class of weights, which will be explained later.

It might initially appear that the highest-order monomial term dominates the summation inside the transfer function $g(\cdot)$ due to the combinatorics. However, this is not always the case, especially for sufficiently well-trained networks. For example, when the complexity of the target rule warrants only up to the m'th-order term with $m' < m$, the weights of the terms higher than m'th order will vanish. We can also control the relative contributions of each monomial term to the final output by using separate normalization for each weight submatrix, which we do in this work. This can be viewed as a straightforward extension of the conventional paradigm of using first-order neurons, since the bias term and each weight are normally treated differently. By introducing a set of new parameters, γ_i, controlling the relative weight for each monomial term, we rewrite the perceptron function as follows

$$y = g\left(\sum_{k=0}^{m} \frac{\gamma_k}{\sqrt{d_k}} \sum_{1 \leqslant i_1 < \cdots < i_k \leqslant N} w^{(k)}_{i_1 \ldots i_k} x_{i_1} \ldots x_{i_k} \right).$$

We impose the following spherical constraint for each monomial class of weights $w^{(k)}_{i_1 \ldots i_k}$,

$$\sum_{1 \leqslant i_1 < \cdots < i_k \leqslant N} (w^{(k)}_{i_1 \ldots i_k})^2 = d_k \tag{2}$$

for all $0 \leqslant k \leqslant m$. Note that the normalization is done over the number of independent weights, so that the typical value of each weight is of order unity before learning takes place. The relative weight parameters are in turn normalized as follows

$$\sum_{k=0}^{m}(\gamma_k)^2 = 1.$$

In this paper we shall call γ_k's *tuning parameters*.

Within this set-up of the problem, one can easily see that all the monomial terms decouple from each other in the sense that the average of the product of two different monomial terms over all the possible example patterns is equal to the product of their averages, which vanishes due to the Z_2 symmetry of each input variable. Hence, it is sufficient to retain only two monomial classes of weights in the argument of the transfer function in order to illustrate general characteristics of the higher-order perceptron. In this paper we consider a second-order perceptron without a bias term, whose output y is given by

$$y = g\left(\frac{\gamma_1}{\sqrt{N}}\sum_{i=1}^{N} W_i x_i + \frac{\gamma_2}{\sqrt{\frac{N(N-1)}{2}}}\sum_{i<j}^{N} Y_{ij} x_i x_j\right) \tag{3}$$

where we have introduced two new symbols W_i and Y_{ij} for $w_i^{(1)}$ and $w_{ij}^{(2)}$, respectively. This equation is shown graphically in figure 1 for the case of $N = 3$. As is apparent from the figure, the higher-order perceptron can be viewed as a three-layer network with product units in the first layer, linear units with non-overlapping receptive fields in the second layer, and threshold units in the third layer.

3. Storage capacity

Let us suppose that a data set of p random patterns, which are pairs of N input and 1 output binary numbers, are given to a network. If p is small enough, the network can learn, or become able to classify, these patterns by adjusting the decision boundaries appropriately. However, as p increases, some stored patterns lie closer and closer to the boundaries, and beyond a certain critical value the patterns will no longer be completely separable by this polynomial hypersurface. We define the number of patterns, p_c, at which this happens as the separating capacity or *storage capacity* of the network. If we think of these random patterns as input–output pairs from a certain Boolean function, the storage capacity can also be viewed as the upper bound of the complexity of the Boolean functions realizable by the given neural network, and hence it represents the *model complexity*.

In this section we calculate the storage capacity of the second-order perceptron á la Gardner [15]. The output of the network is given by (3), which is rewritten here as

$$y(\boldsymbol{x}) = \mathrm{sgn}(\gamma_1 u(\boldsymbol{x}) + \gamma_2 v(\boldsymbol{x}))$$

where the argument of the transfer function has been divided into two terms, defined as

$$u(\boldsymbol{x}) = \sqrt{\frac{1}{N}}\sum_{i=1}^{N} W_i x_i$$

$$v(\boldsymbol{x}) = \sqrt{\frac{2}{N(N-1)}}\sum_{i<j}^{N} Y_{ij} x_i x_j.$$

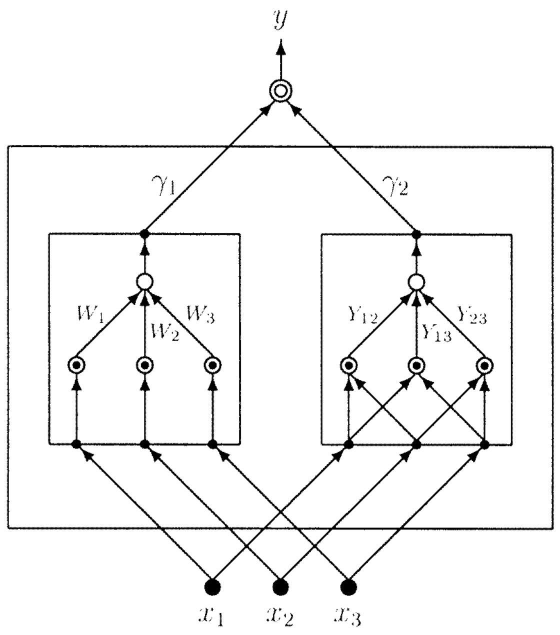

Figure 1. An 'anatomical' view of a second-order perceptron with three input nodes ($N = 3$). The outermost box represents the usual 'blackbox' view of the perceptron, or the neuron, while the two inner boxes represent the first- and second-order monomial terms, respectively. The open circles perform summation of the incoming arguments while the dotted circles perform product. There are, in general, d_k dotted circle elements for the kth-order monomial class of weights. The double circle can be viewed as the conventional first-order threshold unit. All the arrows without labels simply indicate the flow of the computation.

The weight vectors, W_i and Y_{ij}, are separately normalized to satisfy the spherical constraints, (2)

$$\sum_{i=1}^{N} W_i^2 = N \tag{4a}$$

$$\sum_{i<j} Y_{ij}^2 = \frac{N(N-1)}{2}. \tag{4b}$$

The relative weights of the two monomial terms are assumed to be constant, with the following constraint

$$(\gamma_1)^2 + (\gamma_2)^2 = 1. \tag{5}$$

Hence the weight space is the direct product of two hyperspheres with fixed radii. The relative volumes of these two 'component' spaces are controlled by (4*a*)–(5). It should be noted that this choice of phase space reflects our prior knowledge, or hypothesis, concerning the class of rules to which the target rule belongs.

Now we begin our calculation of the storage capacity by evaluating the *version space*, which is the volume of the weight space in which the following equations,

$$y^{\mu} = y(\boldsymbol{x}^{\mu})$$

are satisfied for all input–output pairs of examples, $\{\boldsymbol{x}^\mu, y^\mu; \mu = 1, \ldots, p\}$. It is given by

$$V = \int \mathrm{d}\mu(\boldsymbol{W})\, \mathrm{d}\mu(\boldsymbol{Y}) \prod_\mu \theta(y^\mu(\gamma_1 u^\mu + \gamma_2 v^\mu))$$

where

$$u^\mu = u(\boldsymbol{x}^\mu)$$
$$v^\mu = v(\boldsymbol{x}^\mu)$$

and the weight space measure is

$$\mathrm{d}\mu(\boldsymbol{W}) = \prod_i \mathrm{d}W_i\, \delta\left(\sum_i W_i^2 - N\right)$$
$$\mathrm{d}\mu(\boldsymbol{Y}) = \prod_{i<j} \mathrm{d}Y_{ij}\, \delta\left(\sum Y_{ij}^2 - \frac{N(N-1)}{2}\right).$$

The *typical* size of this volume is then obtained by averaging $\ln V$ over all realizations of the input–output pairs at constant p. This is done through the *replica* method,

$$\langle\langle \ln V \rangle\rangle = \lim_{n\to 0} \frac{1}{n} \ln\langle\langle V^n \rangle\rangle$$

in which one needs to calculate the total permissible volume of n replicated systems,

$$\langle\langle V^n \rangle\rangle = \prod_{\sigma=1}^{n} \int \mathrm{d}\mu(\boldsymbol{W}^\sigma)\, \mathrm{d}\mu(\boldsymbol{Y}^\sigma) \left\langle\!\!\left\langle \prod_\mu \theta(y^\mu(\gamma_1 u^\mu + \gamma_2 v^\mu)) \right\rangle\!\!\right\rangle. \qquad (6)$$

Hereafter, the quenched average $\langle\langle\ \rangle\rangle$ is taken over the example pairs, that is, over $\boldsymbol{x}^\mu$'s, whose distribution is assumed to be completely random,

$$P(x_i) = \tfrac{1}{2}\delta(x_i - 1) + \tfrac{1}{2}\delta(x_i + 1) \qquad i = 1, \ldots, N \qquad (7)$$

and over y^μ's, whose distribution is determined by the presumed rule which is, in this section, assumed to be selected at random for the purpose of calculating storage capacity. Hence

$$P(y|\boldsymbol{x}) = \tfrac{1}{2}\delta(y - 1) + \tfrac{1}{2}\delta(y + 1)$$

where $\delta(x) \equiv \delta_{x0}$ is the Kronecker delta.

The quenched average of (6) is taken through the standard method [15], giving

$$\langle\langle \ln V \rangle\rangle = p \int \mathrm{D}t\, \ln\left[\int_0^\infty \frac{\mathrm{d}r}{\sqrt{2\pi(1-q)}} \mathrm{e}^{-\frac{1}{2}\frac{(r+t\sqrt{q})^2}{1-q}}\right] + \frac{N}{2}\left[\ln(1-q_1) + \frac{q_1}{1-q_1}\right]$$
$$+ \frac{N(N-1)}{4}\left[\ln(1-q_2) + \frac{q_2}{1-q_2}\right] \qquad (8)$$

where the overlap order parameter

$$q = \gamma_1^2 q_1 + \gamma_2^2 q_2 \qquad (9)$$

is introduced under the replica symmetric assumption

$$q_1 = \frac{1}{N}\left\langle\!\!\left\langle \sum_i W_i^\sigma W_i^\nu \right\rangle\!\!\right\rangle \qquad (10a)$$

$$q_2 = \frac{2}{N(N-1)}\left\langle\!\!\left\langle \sum_{i<j} Y_{ij}^\sigma Y_{ij}^\nu \right\rangle\!\!\right\rangle \qquad (10b)$$

where $\sigma \neq \nu$. In (8), t is a Gaussian random variable with unit variance, and the integration measure Dt is defined as

$$\mathrm{D}t = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}t^2}\, \mathrm{d}t.$$

By extremizing $\langle\langle \ln V \rangle\rangle$ with respect to q_1 and q_2, we obtain the following saddle-point equations,

$$\frac{p\gamma_1^2}{\sqrt{2\pi}} \int \mathrm{D}t \left[H\left(\frac{t\sqrt{q}}{\sqrt{1-q}} \right) \right]^{-1} t\mathrm{e}^{-\frac{1}{2}\frac{q}{1-q}t^2} = N \frac{q_1\sqrt{q}(1-q)^{3/2}}{(1-q_1)^2}$$
$$\frac{p\gamma_2^2}{\sqrt{2\pi}} \int \mathrm{D}t \left[H\left(\frac{t\sqrt{q}}{\sqrt{1-q}} \right) \right]^{-1} t\mathrm{e}^{-\frac{1}{2}\frac{q}{1-q}t^2} = \frac{N(N-1)}{2} \frac{q_2\sqrt{q}(1-q)^{3/2}}{(1-q_2)^2}$$

where

$$H(x) = \int_x^\infty \mathrm{D}y = \frac{1}{2}\,\mathrm{erfc}\left(\frac{x}{\sqrt{2}} \right). \tag{11}$$

At the maximum storage, the volume of the version space becomes 0 by definition, and the order parameter q becomes 1. Hence, in order to obtain the storage capacity, we take the limit $q \to 1$, which is somewhat problematic due to the presence of two overlap order parameters q_1 and q_2. As it turns out, q_1 first approaches 1 at around $p \sim d_1 = N$, but the volume of the version space remains finite. Only when a number of examples of order $d_2 = N(N-1)/2$ are used does q_2, and hence the total q approach 1, at which point the total volume of the version space shrinks to zero. We obtain the following storage capacity for the second-order perceptron,

$$p_c = 2\left[\gamma_1\sqrt{N} + \gamma_2\sqrt{\frac{N(N-1)}{2}} \right]^2. \tag{12}$$

When either $\gamma_1 = 0$ or $\gamma_2 = 0$, this agrees with Cover's finding [23], namely, that the storage capacity is twice the number of the free weights in the network. Note that p_c varies from $2N$ to $N(N-1)$ as γ_1 and γ_2 change. When both γ_1 and γ_2 are of order unity, the storage capacity is of order $N(N-1)/2$ and is continuously tunable depending on γ_2.

The reason why our result, (12), predicts a capacity less than the optimal, $2 \times \frac{N(N+1)}{2}$, is that we have imposed the separate normalization condition for each monomial term in the argument of the transfer function. Hence, the decision boundary is hyperquadric surfaces with strong hyperplanar shape depending on the magnitude of γ_2. This tunability of model complexity can be very useful in many situations despite the fact that it is achieved in the expense of its storage capability. One such example in the context of learning will be the next theme of this paper.

This result can be easily extended to an arbitrary order higher-order perceptron with multiple monomial terms as long as its order is much smaller than the number of input nodes. In general, for an mth-order perceptron with $m \ll N$,

$$y = \mathrm{sgn}\left(\sum_{k=0}^{m} \frac{\gamma_k}{\sqrt{d_k}} \sum_{i_1<\cdots<i_k} w^{(k)}_{i_1\ldots i_k} x_{i_1}\ldots x_{i_k} \right) \tag{13}$$

the storage capacity is given by

$$p_c = 2\left[\sum_{k=0}^{m} \gamma_k\sqrt{d_k} \right]^2$$

where d_k was defined in (1).

4. Gibbs learning of polynomially separable rules

We shall now investigate the generalization ability of a second-order perceptron. We assume that the target rule is learnable in principle. That is, we train the network with example sets generated by a presumed teacher network which has the same architecture as that of the student. This idea has been used by many authors [16–21].

Multilayer perceptrons are usually trained by so-called *back-propagation* [4], which is a gradient descent algorithm in the error landscape in the weight space. The error, or more precisely the *training error*, is defined as the squared difference between the student's and teacher's outputs for a given set of inputs. The goal of learning in neural networks is, however, not to minimize the error for a given example set but to minimize the total error for all possible inputs. This is called the *generalization error* and is defined as the expected error over the distribution of all possible inputs.

Unlike the training error, the generalization error is not 'observable'. This is one of the essential difficulties in neural-network learning; while our goal is to minimize the generalization error, the only available information is the training error for a subset of all possible inputs. Although it is known that the training error is uniformly convergent to the generalization error in the limit of an infinite number of examples [2], this is usually a very important issue in practical situations, in which only a finite number of examples are normally available.

In this section we investigate the generalization ability of a second-order perceptron when it is trained through *Gibbs learning*, which is a stochastic version of the back-propagation with a finite level of noise. It is known that Gibbs learning relaxes the system into an equilibrium, in which the weights follow the Gibbs distribution according to the training error. Hence, by studying the equilibrium properties of the weight space, one can gain a great deal of information concerning various aspects of learning.

The student and the teacher networks are supposed to perform the following functions:

$$y = \mathrm{sgn}(\gamma_1 u + \gamma_2 v)$$
$$y^t = \mathrm{sgn}(\gamma_1^t u^t + \gamma_2^t v^t)$$

where the superscript t indicates that they are teacher variables. The u's and v's, defined earlier, are given here for convenience:

$$u = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} W_i x_i$$
$$u^t = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} W_i^t x_i$$
$$v = \sqrt{\frac{2}{N(N-1)}} \sum_{i<j}^{N} Y_{ij} x_i x_j$$
$$v^t = \sqrt{\frac{2}{N(N-1)}} \sum_{i<j}^{N} Y_{ij}^t x_i x_j .$$

W_i's and Y_{ij}'s for both student and teacher are continuous weights with spherical constraints as before,

$$\sum_{i=1}^{N} W_i^2 = N$$

$$\sum_{i=1}^{N}(W_i^t)^2 = N$$
$$\sum_{i<j} Y_{ij}^2 = \frac{N(N-1)}{2}$$
$$\sum_{i<j}(Y_{ij}^t)^2 = \frac{N(N-1)}{2}$$

and again we impose the following constraints on the γ's:

$$(\gamma_1)^2 + (\gamma_2)^2 = 1$$
$$(\gamma_1^t)^2 + (\gamma_2^t)^2 = 1.$$

Since the output is binary, we define the error function as

$$\epsilon(\gamma_2, \boldsymbol{W}, \boldsymbol{Y}; \boldsymbol{x}) = \theta(-(\gamma_1 u + \gamma_2 v)(\gamma_1^t u^t + \gamma_2^t v^t)).$$

Then the training error is given by

$$e_t(\gamma_2, \boldsymbol{W}, \boldsymbol{Y}) = \frac{1}{p}\sum_{\mu=1}^{p}\epsilon(\gamma_2, \boldsymbol{W}, \boldsymbol{Y}; \boldsymbol{x}^\mu)$$

and the generalization error by

$$e_g(\gamma_2, \boldsymbol{W}, \boldsymbol{Y}) = \langle \epsilon(\gamma_2, \boldsymbol{W}, \boldsymbol{Y}; \boldsymbol{x})\rangle_{P(\boldsymbol{x})}$$

where $P(\boldsymbol{x})$ is the distribution of inputs, which is normally assumed to be known *a priori* and is set to be isotropic, (7), in this work.

We shall discuss in this paper two different learning schemes of higher-order perceptrons. First, we study the scheme where tuning parameters, γ's, of the student network are predetermined or 'quenched' and hence its model complexity is fixed. In this scheme, we assume that the model complexity is the same as the rule complexity ($\gamma_2 = \gamma_2^t$) and hence the teacher's rule is exactly learnable. Second, we investigate a different learning scheme in which the model complexity is adjustable through adaptation of the tuning parameters.

The partition function takes slightly different forms for these two learning schemes. First, when the model complexity is fixed, the partition function is given as usual:

$$Z = \int \mathrm{d}\mu(\boldsymbol{W})\,\mathrm{d}\mu(\boldsymbol{Y})\,\mathrm{e}^{-p\beta e_t(\gamma_2, \boldsymbol{W}, \boldsymbol{Y})}$$

where β^{-1} represents the training noise. Second, when the model complexity is tunable, we also need to integrate over the distributions of the tuning parameters:

$$Z = \int \mathrm{d}\mu(\gamma_2)\,\mathrm{d}\mu(\boldsymbol{W})\,\mathrm{d}\mu(\boldsymbol{Y})\,\mathrm{e}^{-p\beta e_t(\gamma_2, \boldsymbol{W}, \boldsymbol{Y})}$$

where γ_2 is integrated from 0 to 1, that is,

$$\mathrm{d}\mu(\gamma_2) = \mathrm{d}\gamma_2\,\theta(\gamma_2)\theta(1-\gamma_2).$$

The *typical* distributions are then obtained by averaging $\ln Z$ over all possible example sets, which is done through the replica method as in the calculation of the storage capacity

$$\langle\langle \ln Z\rangle\rangle = \frac{N}{2}\left[\frac{q_1 - R_1^2}{1-q_1} + \ln(1-q_1)\right] + \frac{N(N-1)}{4}\left[\frac{q_2 - R_2^2}{1-q_2} + \ln(1-q_2)\right]$$
$$+2p\int_0^\infty \mathrm{D}z \int \mathrm{D}t\, \ln\left[\mathrm{e}^{-\beta} + (1-\mathrm{e}^{-\beta})H\left(\frac{\sqrt{q-R^2}t - Rz}{\sqrt{1-q}}\right)\right] \qquad (14)$$

where the mutual overlap order parameter between replicas, q, is defined in the same way as in the previous section, (9)–(10*b*), and the overlap between a replica and the teacher is defined as

$$R = \gamma_1\gamma_1^t R_1 + \gamma_2\gamma_2^t R_2$$

where, again under the replica symmetric ansatz,

$$R_1 = \frac{1}{N}\left\langle\!\!\left\langle \sum_{i=1}^{N} W_i^t W_i^\sigma \right\rangle\!\!\right\rangle$$

$$R_2 = \frac{2}{N(N-1)}\left\langle\!\!\left\langle \sum_{i<j} Y_{ij}^t Y_{ij}^\sigma \right\rangle\!\!\right\rangle.$$

$H(\cdot)$ was defined in (11).

Now we discuss the results of this calculation for the two learning schemes.

4.1. Fixed complexity

The most probable state of the system is obtained by extremizing the free energy, (14), with respect to the order parameters, q and R.

$$\gamma_1\gamma_1^t pT = N\frac{(1-q)^{1/2}}{(1-q_1)^2}(q-R^2)^{1/2}[(q_1-R_1^2)(1-q)+RR_1(1-q_1)] \tag{15}$$

$$\gamma_1\gamma_1^t pZ = N\frac{(1-q)^{1/2}}{(1-q_1)^2}[(1-R^2)R_1(1-q_1)-R(q_1-R_1^2)(1-q)] \tag{16}$$

$$\gamma_2\gamma_2^t pT = \frac{N(N-1)}{2}\frac{(1-q)^{1/2}}{(1-q_2)^2}(q-R^2)^{1/2}[(q_2-R_2^2)(1-q)+RR_2(1-q_2)] \tag{17}$$

$$\gamma_2\gamma_2^t pZ = \frac{N(N-1)}{2}\frac{(1-q)^{1/2}}{(1-q_2)^2}[(1-R^2)R_2(1-q_2)-R(q_2-R_2^2)(1-q)] \tag{18}$$

where

$$T = \sqrt{\frac{2}{\pi}}\int_0^\infty \mathrm{D}z \int_{-\infty}^{\infty} \mathrm{D}t\, \frac{\mathrm{e}^{-\frac{1}{2}s^2}t}{(\mathrm{e}^\beta-1)^{-1}+H(s)}$$

$$Z = \sqrt{\frac{2}{\pi}}\int_0^\infty \mathrm{D}z \int_{-\infty}^{\infty} \mathrm{D}t\, \frac{\mathrm{e}^{-\frac{1}{2}s^2}z}{(\mathrm{e}^\beta-1)^{-1}+H(s)}$$

and

$$s = \frac{\sqrt{q-R^2}t - Rz}{\sqrt{1-q}}.$$

These equations are solved numerically for $\gamma_2 = \gamma_2^t$. In this case, the teacher's rule is completely realizable in principle and the generalization error will approach zero as the number of examples increases. The generalization curve is obtained through the following identity [17] in the thermodynamic limit:

$$e_g(p) = \frac{1}{\pi}\cos^{-1} R^*$$

where R^* is the solution of the saddle-point equations, (15)–(18), at given p.

The results show that the learning occurs in a stepwise manner. The first-order part of the phase space is completely constrained at $p \sim d_1 = N$ and complete or near-complete learning takes places only after $p \sim d_2 = N(N-1)/2$ examples are used for training. This

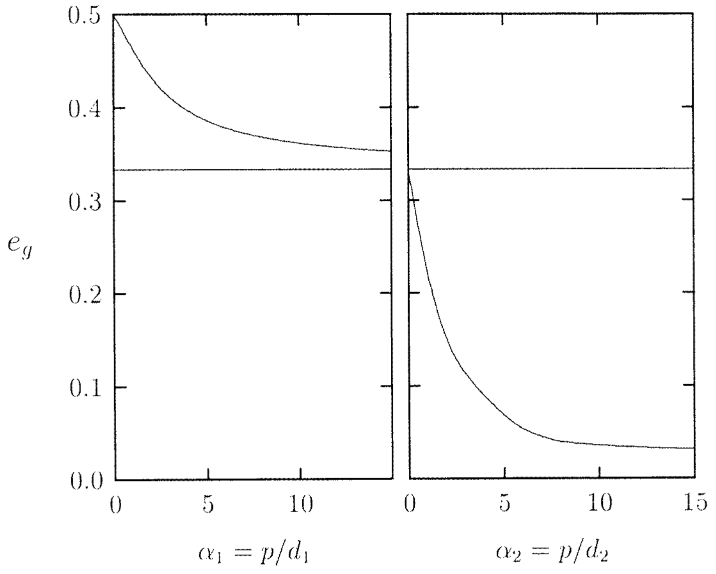

Figure 2. Generalization error as a function of the number of training examples. The number of examples, p, is normalized by $d_1 = N$ for the left curve and by $d_2 = N(N-1)/2$ for the right one. $1/\sqrt{2}$ is used for both γ_1 and γ_2 and the training temperature is set to $\beta = 10$. The asymptotic value of the first order regime is $\frac{1}{\pi}\cos^{-1}(\frac{1}{2}) \approx 0.33$.

is shown in figure 2 for $\gamma_1 = \gamma_2 = \frac{1}{\sqrt{2}}$. The numbers of examples are normalized by N for the curve on the left-hand side and by $N(N-1)/2$ for the other one. The left curve asymptotically approaches $\frac{1}{\pi}\cos^{-1}(\frac{1}{2}) \approx 0.33$, indicating $q_1 = R_1 = 1$ as $\frac{p}{N} \to \infty$.

4.2. Adaptable complexity

We now consider the learning scheme in which one trains the tuning parameters as well as the weights. Since we would like to pick a *single* best student, we also extremize the free energy over γ_2. Then, in addition to (15)–(18), we have one more stationary condition:

$$\left[(\gamma_1^t\gamma_2 q_1 - \gamma_1\gamma_2^t q_2)\frac{1}{(1-q)}\frac{1-R^2}{(q-R^2)^{1/2}} - 2(\gamma_1^t\gamma_2 R_1 - \gamma_1\gamma_2^t R_2)\frac{R}{(q-R^2)^{1/2}}\right]T$$
$$-\left[(\gamma_1^t\gamma_2 q_1 - \gamma_1\gamma_2^t q_2)\frac{1}{(1-q)}R + 2(\gamma_1^t\gamma_2 R_1 - \gamma_1\gamma_2^t R_2)\right]Z = 0. \tag{19}$$

Note that its solution should lie between 0 and 1, by definition.

One can easily see that the following equations

$$\frac{\gamma_1^t}{\gamma_1}q_1 = \frac{\gamma_2^t}{\gamma_2}q_2$$
$$\frac{\gamma_1^t}{\gamma_1}R_1 = \frac{\gamma_2^t}{\gamma_2}R_2$$

constitute a solution. These equations impose strong constraints on the solutions of q's and R's. First, one can easily see that

$$\frac{q_1}{q_2} = \frac{R_1}{R_2}.$$

Furthermore, since $q_2 = R_2 = 0$ in the regime $p \sim N$ (it follows from (15)–(18), as in the previous case), one concludes that $\gamma_1 = 1$ and $\gamma_2 = 0$. Therefore the network behaves like a first-order perceptron when there are only sufficient number of examples which can be used for the training of the first-order monomial term. It should be noted that this minimizes the generalization error,

$$e_g = \frac{1}{\pi}\cos^{-1}(\gamma_1\gamma_1^t R_1 + \gamma_2\gamma_2^t R_2).$$

As the number of examples grows, the network complexity increases accordingly. This is an interesting result because one might have expected $\gamma_1 = \gamma_1^t$ and $\gamma_2 = \gamma_2^t$ for all p since it 'matches' their complexities. As p increases to infinity, all order parameters become 1, and hence $\gamma_1 = \gamma_1^t$ and $\gamma_2 = \gamma_2^t$ as expected.

In order to show that these general features of the second-order perceptron do not depend on the particular algorithm, we implemented the (higher-order) perceptron algorithm. In this algorithm, the student vector changes infinitesimally, at each time step, according to the following rule

$$W_i{}' = W_i + \eta^{(1)} y^t x_i \theta(-yy^t)$$
$$Y_{ij}{}' = Y_{ij} + \eta^{(2)} x_i x_j \theta(-yy^t)$$

where $\eta^{(1)}$ and $\eta^{(2)}$ are called the learning rate parameters. For the sake of simplicity we do not normalize the weights and let $\eta^{(1)} = \eta^{(2)} = 1$.

This algorithm indeed exhibits this cascade-like learning behaviour. We present the learning curve of a fully realizable case in figure 3, where $N = 20$ and $\gamma_2 = \gamma_2^t = \frac{1}{\sqrt{2}}$. As predicted by theory, R_1 rises first in the 'first-order regime' and R_2 approaches 1 only after order of N^2 examples available. This crossover in behaviours is less prominent in e_g due to the fact that N is only 20.

Before we close this section, we present the results for generic higher-order perceptrons. For an mth-order perceptron learning from a teacher, (13), the free energy as a function of order parameters becomes

$$\langle\langle \ln Z \rangle\rangle = \sum_{k=0}^{m} \frac{d_k}{2}\left[\frac{q_k - R_k^2}{1-q_k} + \ln(1-q_k)\right]$$
$$+2p\int_0^\infty \mathrm{D}z \int \mathrm{D}t\ \ln\left[\mathrm{e}^{-\beta} + (1-\mathrm{e}^{-\beta})H\left(\frac{\sqrt{q-R^2}t - Rz}{\sqrt{1-q}}\right)\right]$$

where

$$q = \sum_{k=0}^{m} \gamma_k^2 q_k$$
$$R = \sum_{k=0}^{m} \gamma_k \gamma_k^t R_k$$

and

$$q_k = \frac{1}{d_k}\left\langle\!\!\left\langle \sum w^{(k)}_{i_1,\ldots,i_k} w^{(k)}_{i_1,\ldots,i_k} \right\rangle\!\!\right\rangle$$
$$R_k = \frac{1}{d_k}\left\langle\!\!\left\langle \sum w^{t,(k)}_{i_1,\ldots,i_k} w^{(k)}_{i_1,\ldots,i_k} \right\rangle\!\!\right\rangle.$$

Again $m \ll N$ has been assumed. By extremizing the free energy over the order parameters, one obtains a cascade-like learning curve, as in the second-order perceptron, in which each

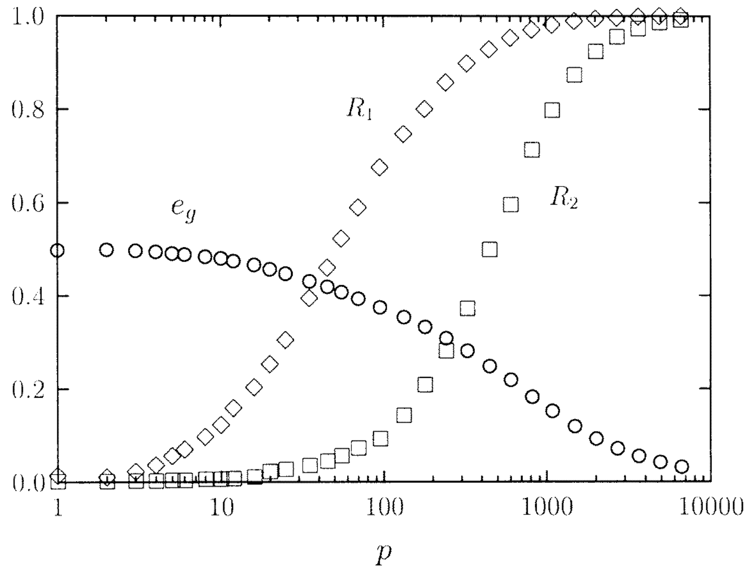

Figure 3. Learning curve of a second-order perceptron with $N = 20$. The second-order perceptron algorithm is used with $\eta^{(1)} = \eta^{(2)} = 1$. The overlap order parameters, R_1 (◇) and R_2 (□), grow significantly in different scales of p, as explained in the text. The generalization error e_g (○) approaches zero in the limit of infinite training examples, $p \to \infty$. The weights are not normalized in this simulation unlike in the rest of the paper. Each data point is an average over 1000 runs.

weight space component is constrained sequentially as the number of examples increases. Near-perfect learning occurs only after order of d_m examples are used for training, at which point the residual error decreases as $1/p$ [17].

5. Conclusion

We have calculated the storage capacity and the learning curve of the higher-order perceptron using Gardner's statistical mechanics approach [15]. One of the most interesting properties of the higher-order perceptron is that its effective order can be made tunable, that is, the storage capacity and the complexity of the learnable rules can be varied with the control parameters, γ's. This particular model of higher-order perceptrons is different from those considered by other authors, for instance, [6].

We have found that the storage capacity is a continuously varying function of the tuning parameters of different order monomial terms. What these parameters do is essentially change the *relative* volume of the phase-space component corresponding to each monomial class of weights.

We have also shown that, when the finite-temperature Gibbs algorithm is used for training, the higher-order perceptron exhibits stepwise or cascade-like learning as a function of the number of training examples. This is due to the fact that the number of examples required to constrain each monomial component of the phase space scales differently with the system size. As we mentioned earlier, this mode of using higher-order perceptrons fits naturally with the conventional paradigm in statistics, namely, polynomial curve fitting [12].

One of the major obstacles in using higher-order networks in practical applications, is their combinatorial explosion in terms of the number of free parameters. This problem can be overcome in various ways. One obvious possibility is diluting weights [5, 24, 25] either randomly or systematically until the optimal complexity is obtained. Diluted networks can still learn higher-order correlations in patterns, which is not possible for the lower-order networks. The so-called *support vector machines* are based on similar ideas, but do not suffer from the curse of dimensionality [13]. The invariance constraint can also be directly programmed into higher-order networks, thereby reducing the number of independent weights and the computational burden in terms of the required number of training examples. Further research along this line may eventually make the higher-order unit a more attractive alternative to the conventional first-order neuron by simultaneously improving performance and reducing computational cost.

Acknowledgments

This work was partially supported by the Basic Science Special Program of POSTECH and the Korea Ministry of Education through the Basic Science Research Institute, POSTECH. It was also supported by the 'Non-Directed Fund' of the Korea Research Foundation, 1995. HY was supported in part by postdoctoral fellowship from the Korea Research Foundation.

References

[1] Rumelhart D E, McClelland J L and the PDP Research Group (ed) 1986 *Parallel Distributed Processing* vol 1 (Cambridge, MA: MIT)
[2] Haykin S 1994 *Neural Networks: A Comprehensive Foundation* (Englewood Cliffs, NJ: MacMillan)
[3] Bishop C M 1995 *Neural Networks for Pattern Recognition* (Oxford: Clarendon)
[4] Rumelhart D E, Hinton G E and Williams R J 1986 *Parallel Distributed Processing* vol 1, ed D E Rumelhart, J L McClelland and the PDP Research Group (Cambridge, MA: MIT) ch 8, pp 318–62
[5] Kanter I 1988 *Phys. Rev.* A **38** 5972
[6] Kohring G A 1990 *J. Physique* **51** 145
[7] Giles C L and Maxwell T 1987 *Appl. Opt.* **26** 4972
[8] Kanter I 1987 *J. Phys. C: Solid State Phys.* **20** L257
[9] Lee Y C *et al* 1986 *Physica* **22D** 276
[10] Chang C and Cheung J Y 1992 *Int. Joint Conf. on Neural Networks* vol III (New York: IEEE) pp 511–16
[11] Linhart G and Dorffner G 1992 *Int. Joint Conf. on Neural Networks* vol III (New York: IEEE) pp 705–10
[12] Ring M 1994 *Advances in Neural Information Processing Systems* vol 6 (Denver, CO: Morgan Kaufmann) pp 115–22
[13] Vapnik V N 1995 *The Nature of Statistical Learning Theory* (New York: Springer)
[14] Minsky M L and Papert S 1969 *Perceptrons* (Cambridge, MA: MIT)
[15] Gardner E 1988 *J. Phys. A: Math. Gen.* **21** 257
[16] Sompolinsky H, Tishby N and Seung H S 1990 *Phys. Rev. Lett.* **65** 1683
[17] Seung H S, Sompolinsky H and Tishby N 1992 *Phys. Rev.* A **45** 6056
[18] Opper M, Kinszel W, Kleinz J and Nehl R 1990 *J. Phys. A: Math. Gen.* **23** L581
[19] Opper M and Kinzel W 1996 *Physics of Neural Networks* ed J L V Hemmen, E Domany and K Schulten (Berlin: Springer)
[20] Barkai E, Hansel D and Kanter I 1990 *Phys. Rev. Lett.* **65** 2312
[21] Kang K, Oh J-H, Kwon C and Park Y 1993 *Phys. Rev.* E **48** 4805
[22] Oh J-H, Kwon C and Cho S (ed) 1995 *Neural Networks: The Statistical Mechanics Perspective (Progress in Neural Processing 1)* (Singapore: World Scientific)
[23] Cover T M 1965 *IEEE Trans. Electron. Comput.* **14** 326
[24] Hassibi B, Stork D G and Wolff G 1993 *Advances in Neural Information Processing Systems* vol 5 (Denver, CO: Morgan Kaufmann) pp 263–70
[25] Canning A and Gardner E 1988 *J. Phys. A: Math. Gen.* **21** 3275