Artificial Neural Networks with Nonlinear Synapses and Nonlinear Synaptic Contacts

Ping Liang and Nadeem Jamali
School of Computer Science
Technical University of Nova Scotia
Halifax, NS B3J 2X4, Canada

Abstract

A neural network model with polynomial synapses and product contacts is investigated. The model further generalize the sigma-pi and product units models. All the coefficients and exponents of the polynomial terms, the degrees of the polynomials (the number of polynomial terms) are learned, not predetermined. This can be useful when there is no a priori knowledge available to determine what higher order terms to be included. The polynomial synapses together with product contacts can produce any polynomial terms. Since the number of learnable parameters are learned, in this aspect, our network is much like the growth networks. The growth networks in general have poor generalization. Several mechanisms in our network contribute to a better generalization performance than the growth networks. Gradient descent algorithms for training feedforward networks with polynomial synapses and product contacts are developed. Experimental results are presented.

1 Introduction

The McCulloch-Pitts neural network model [1] has the following three simplifying assumptions:

1. The interaction of all postsynaptic signals is a simple summation in a latent period.
2. A synapse only *linearly* amplifies or attenuates the presynaptic signal. This is modeled by a connection weight that multiplies the presynaptic signal to produce the postsynaptic signal.
3. There is only one synaptic contact between two neurons.

Because of the above simplifications, all the nonlinearity necessary to achieve a desired transformation is restricted to the neurons, i.e., the threshold or the sigmoidal transfer functions of the neurons. Physiological data show that the above three assumptions are not true in a real neural network. It is known that a neuron may have more than one synaptic contacts with another neuron. However, because of the first two assumptions, such multiple synaptic connections are assumed to be included by a single connection weight.

In real neural networks, the interaction of all postsynaptic signals is more complicated than a simple summation. Neurophysiological data show that the transformation between the presynaptic and postsynaptic signals is not linear. The transfer function of a synapse may be better modeled by a nonlinear transformation than by a simple multiplication of a linear coefficient.

Several models removing some of the above simplifying assumptions have been reported. The sigma-pi units by Rumelhart, Hinton and Williams [2] and the product units by Durbin and Rumelhart [3] allow multiplication between presynaptic or postsynaptic signals. Giles and Maxwell [4] investigate a special case of the sigma-pi units, called high-order neural networks. The high-order networks use sigma-pi uints only at the input layer. Encouraging results are reported in both cases. These models can be further generalized along the following lines:

1. The synapse transfer function in these models is still only a linear amplification. Although the product units and sigma-pi units introduce cross product terms, they do not have the freedom of introducing higher order terms of each presynaptic signal itself.
2. The sigma-pi and product units must be "handcrafted" in the network using a priori knowledge before learning. Although this is not necessarily a disadvantage since embedding a priori knowledge is a way to alleviate the the load of the learning algorithm, it is still desired to have a network that is able to learn nonlinear synapse transformations when there is no a priori knowledge available.
3. The number of learnable parameters are fixed in the sigma-pi, product and most other network models (except the growth networks discussed later). Again, this is not a disadvantage if a priori knowledge is available to choose the right number of learnable parameters. The number of learnable parameters should be kept to a minimum to ensure good generalization [5]. However, when there is no a priori knowledge available, it is desired to have a ntework that will gradually increase its number of learnable parameters if the training examples cannot be successfully loaded with a given number of learnable parameters. Since the number of learnable parameters is the key in determining the generalization performance of network, the increase of the number of learnable parameters should be curtailed.

This paper reports our effort in developing a neural network model along the directions suggested above. The new models are not restricted by neurophysiological realizability constraints and no claim of adherence to neurophysiological data is made. Our goal at this stage is to investigate how the improvement of model at the synapse and unit level may contribute to improvement of the network performance. We only consider feedforward neural networks in this paper. Recurrent networks and unsupervised learning networks will be considered in the future.

2 Polynomial Model of Nonlinear Synapses and Its Backpropagation Learning Algorithm

From a computational point of view, by restricting the synapses being linear and restricting all the nonlinearity to the neurons, the freedom to introduce and change independent nonlinear terms of the inputs is lost. Functions that involve nonlinear terms of individual inputs and nonlinear cross terms will not be readily learned. This severely limits the functions that can be realized by a given network. Remark that although a three layer network can approximate arbitrarily close any L^1 function in a finite interval [6-8], it requires an increasing number of hidden unit to do so. For fixed networks, there is a difference in which one provide a better approximation.

We propose to introduce a transformation at each synapse, which may be linear or nonlinear. The actual transformation should be learned. To make the learning feasible, the transformation must be of a form that can accommodate both linear and nonlinear transformations by adjusting its parameters. More importantly, the form of the transformation should be amenable to analysis and derivation of the learning algorithm. As a first experiment, we propose to use polynomials to model the nonlinearity of synapses. That is, the postsynaptic signal now becomes a polynomial of the presynaptic signal instead of a simple multiplication of a coefficient. The symbols for nonlinear synapses and nonlinear contacts are shown in Fig. 1. Note that a synapse is now denoted with an endbulb contacting the neuron instead of just a connection. The number of terms, the exponents, and the coefficients of the polynomials at the synapses are moved in the direction of least error, by a learning mechanism that is derived using the gradient descent method.

The differences between this paper and networks using sigma-pi units, product units, or predetermined polynomial terms [2-4] of inputs are threefold.

1. Nonlinearity is introduced at each synapse. Each synapse has a polynomial transfer function. Higher order terms of a presynaptic signal can be added by learning. The polynomial synapses together with product contacts can produce any polynomial terms. The product terms in our model are produced as a result of product of two or more postsynaptic signals after polynomial synapses. As a result, one product contact may produce many product terms of different degrees.

2. All the coefficients and exponents of the polynomial terms, the degrees of the polynomials (the number of polynomial terms) are learned, not predetermined. This can be useful when there is no a priori knowledge available to determine what higher order terms to be included.

3. The number of learnable parameters are learned. In this aspect, our network is much like the growth networks [9-13]. The growth networks in general have poor generalization. Our network include mechanisms preventing the number of learnable parameters from growing too large. In addition, the units used are continuous functions instead of hard limit as in the growth networks; every learnable parameter is affected by all the training examples. These factors add together contributing to a better generalization performance than the growth networks.

The polynomial synapse model is described below. The transformation at a synapse from neuron i to neuron j (neurons at the lowest level perform a sigmoidal transformation to scale all inputs to within [0, 1], as explained below), is of the form:

$$f_{ij} = f_{ij}(y_i) = w'_{ij} y_i^{p_{ij}} + \sum_{a=1}^{m_{ij}} w_{aij} y_i^a \tag{1}$$

where y_i is the output of neuron i, and m_{ij} is a positive integer which is updated in learning. p_{ij} is the only exponent that is updated in each iteration during learning. At any time, terms involving each of the integer exponents up to m_{ij} in addition to $y_i^{p_{ij}}$ are present. In the beginning, $m_{ij} = 0$, and $p_{ij} = 1$. During learning, only p_{ij} changes. When it exceeds 2 for the first time, m_{ij} is set to one and a linear term is permanently added. Later on, whenever an integer value larger than $m_{ij} + 2$ is surpassed, a new term of order $m_{ij} + 1$ is introduced. The coefficient of a new term is set to zero when it is first introduced. In our current implementation, during learning and after learning converges, p_{ij} may be equal to or less than m_{ij}, since the gradient descent updating rule may decrement as well as increment p_{ij}. Modification of the algorithm could be made so that when p_{ij} is reduced below m_{ij}, terms with exponents higher than p_{ij} be removed.

Let

$$F_j = F_j(f_{1j}, f_{2j}, \ldots, f_{n_j j}) = \sum_{i=1}^{n_j} f_{ij} + c_j \tag{2}$$

where n_j is the number of inputs coming to j. Then the output of neuron j is given as:

$$y_j = \frac{1}{1 + \exp^{-\lambda F_j}} \tag{3}$$

For every neuron j, the learnable parameters are w_{aij}, $a = 1, 2, ..., m_{ij}$, w'_{ij}, p_{ij}, $i = 1, 2, ..., n_j$, and c_j.

The gradient descent formulation and associated computational arrangements for networks with polynomial synapses are developed. The procedure is similar to that of the standard backpropagation algorithm [2], and is omitted here due to space limit except one remark: The term $\ln(y_j)$ in the derivatives requires that $y_j > 0$. In intermediate layers, this does not cause a problem since $0 < y_j < 1$. At the input layer, this is taken care of by scaling the input to $(0, 1)$.

Discussions of elimination of terms with small coefficient, hardware considerations and scaling of the input are omitted here due to space limit.

3 Networks with Both Summation and Product Contacts

In real neural networks, there are multiple synaptic connections between two neurons. Such multiple connections do not exist in neural network models using linear synapses and summation contacts. This is because all synapses are linear, and if there are multiple connections, their effects are equally modeled by a single synapse with its weight equal to the sum of all connection weights. Neurophysiological data suggest that the interaction between postsynaptic signals is more than just a linear summation. This disagreement between linear synapse neural network models and real neural networks prompted us to investigate nonlinear synaptic contacts and the effects of multiple synaptic connections between two neurons. Multiple synaptic contacts are also present in sigma-pi and product unit networks [2,3]. The product terms in our model are produced as a result of product of two or more postsynaptic signals after polynomial synapses. As a result, one product contact may produce many product terms of different degrees.

We modify the linear summation neuron model by allowing both summation and product of postsynaptic signals, that is, Eq. (2) is modified to:

$$F_j = F_j(f_{1j}, f_{2j}, ..., f_{n_j j}) = \sum_{i=1}^{n_j} f_{ij} + \sum_{i=1}^{m_j} \prod_{k=1}^{k_i} f_{kj} + c_j \quad (4)$$

where n_j is the number of postsynaptic signals in the summation, m_j is the total number of product terms, and k_i is the number of postsynaptic signals involved in the i^{th} product term. Note that some of the postsynaptic signals in the summation term and the product term may come from the same neuron. Also, some of the postsynaptic signals may have both a summation contact and one or more product contacts with a neuron, thus appearing in both the summation and product terms. The sigmoidal transfer function of a neuron is then applied to F_j defined in the above equation.

A postsynaptic signal that contributes a summation term in Eq. (4) is said to have a linear or summation synaptic contact, or summation contact for short with neuron j. Similarly, a postsynaptic signal that contributes to a product term in Eq. (4) is said to have a nonlinear or product synaptic contact, or product contact for short, with neuron j.

From a computational point of view, allowing both summation and product contacts provides a network with the freedom of adding independent cross terms in the polynomial F_j. The gradient descent formulation is generalized to include the product contacts. Derivation of the learning algorithm is omitted.

4 Generalization of Learning and Relation between Nonlinear Synapse Networks and Linear Synapse Growth Networks

From the viewpoint of increasing the function representation capacity, increasing the degrees (and terms) of the polynomials at the synapses of a nonlinear synapse network is equivalent to increasing the size of a linear synapse network. There have been several algorithms in the literature that grows a network for a given classification task [9-13]. We refer to this type of networks (algorithms) as the growth networks (algorithms). The idea is to learn the structure and the weights of the network simultaneously. If the training samples cannot be learned, the growth algorithms will add more neurons and connections. Growth algorithms are guaranteed to converge. The problem is that the algorithm may grow a overly large network and may lead to poor generalization. We claim that increasing the nonlinearity of synapses should yield a better generalization than adding neurons and connections as in growth algorithms. Our claim is supported by experimental results. This is because of the following reasons:

1. For a difficult classification problem, the decision surface to be learned is highly nonlinear in the pattern space at the hidden layers (transformed image space). In a linear synapse network, although the overall decision surface realized is curved, the sub-decision surface *at each neuron* is a plane. Only the position and orientation of the plane can be adjusted to approximate the function to be learned, whereas in a nonlinear synapse network, the decision surface at each neuron is a curved surface. Not only the position and orientation, but also the shape of the curved surface can be adjusted to approximate the decision surface. As a result, a better approximation can be achieved requiring a less number of sub-decision surfaces using a nonlinear synapse network. A less number of sub-decision surfaces should yield a better generalization.
2. Growth algorithms use hard-limit threshold unit. The network in this paper uses sigmoidal functions at neurons. Information is not lost as a result of early immature decision from layer to layer as in the growth algorithms [14]. Therefore, less units may be required than using hard-limit unit. (see the analog factor in [15]).
3. Growth algorithms may end up adding one neuron (and the corresponding connections) for one sample. The degree (and the number of terms), and all the added parameters of the polynomial of a synapse are affected by all training samples. The exponents may go up and may go down depending on the direction of the gradient. Since the generalization capability of a network is determined by the number of its learnable parameters [5], a possible way to ensure good generalization is fix the total number of learnable parameters before-

hand. Then, during learning, these synapses requiring more learnable parameters (exponents and coefficients for a higher degree polynomial) will get more. Synapses may eliminate terms with small coefficients to release the learnable parameters they grabbed but no longer need. This is effectively a network which learns both its parameters and the structure (i.e., distribution of learnable parameters), although it may appear to have fixed graph representation of the network. Such a scheme may be implemented in hardware.

4. In our network, the inputs are scaled to (0,1) at the first layer. As a result, higher order terms has less influence because all signal having a magnitude less than one. This effectively prevents the degrees of the polynomials from incrementing to impractically high. Hence, the total number of parameters in the network is prevented from growing too large. This limiting capability is inherent and is not a hard-limit type (i.e., it does not prespecify a fixed degree for the synapses polynomials).

5 Experimental Results

Networks of various structures with polynomial synapses and linear contacts, and networks with polynomial synapses and both summation and product contacts, are trained using the gradient descent algorithms described in this paper. The learning algorithms and the networks are tested on many examples. Due to space limit, only some of the representative test results are presented below and are compared with networks with linear synapses and summation-only contacts. In the following, one iteration means the presentation of one training sample to the learning algorithm. If a network is able to learn the correct classification of the training samples in less than 1,000,000 iterations, we say that it converges. Otherwise, the network is considered unable to learn the function.

Figs. 2(a) to 2(d) are examples of the 2D sample patterns used in training the networks described below. They will be referred to as pattern 1 to 4. Each training points in Fig. 2 is labeled 1 if it belongs to class 1, and labeled 0 otherwise. The first example shows how much more powerful a single neuron can be by using polynomial synapses. A single neuron with polynomial synapses and summation-only contacts as depicted in Fig. 3 (Network 1) is able to correctly classify patterns 1 and 4. For pattern 1, the synaptic transformations learned are
Synapse 1: $-0.849x_1^{0.005}$ Synapse 2: $-6.830x_2^4.728 - 1.807x_2^3 + 0.140x_2^2 + 6.386x_2$. The bias is -1.310.
As is obvious, a single neuron with only linear synapses cannot learn any of the patterns since they are not linearly separable. Fig. 4 shows a single neuron with polynomial synapses and both summation and product contacts (Network 2). The contacts with synapses 1 and 2 are of the summation type, and the contacts with synapses 3 and 4 are of the product type. The network is able to learn the correct classification for patterns 1, 2 and 4. The synaptic transformations learned are omitted due to space limit. The highest order term in the synapse polynomials is 4.896, and the synapse with the highest degree polynomial has five terms. The multilayer networks in Fig. 5 (Network 3) is able to learn the classification of patterns 1, 3 and 4. The number of iterations for convergence for networks 1, 2 and 3 are shown in Table 1 Also shown in Table 1 are the results of training the linear synapse network (Network 4 in Fig. 6) with the same structure as Network 3. The multilayer linear synapse network is also able to learn patterns 1 and 4, but unable to learn pattern 3, i.e., it does not converge in 1,000,000 iterations. Fig. 7 shows examples of the classification functions learned by the nonlinear synapse networks. Each figure in Fig. 7 is produced by using regularly space sampling points (including training points) as testing samples. Points classified to class 1 is labeled as 1, and points classified to 0 is labeled 0.

The next five sample patterns are classification in a 4D space. The examples are generated by a multidimensional polynomial function. The polynomials for patterns 5 to 9 are

$$P_5: \quad x_1^2 + x_2^2 + x_3^2 + x_4^2 - 3 = 0 \tag{5}$$
$$P_6: \quad -2x_1x_2^2 + x_2^3 + 2x_1x_3 + x_4^2 - 2 = 0 \tag{6}$$
$$P_7: \quad x_1^3 + x_2^3 + x_3^3 + x_4^3 - 3 = 0 \tag{7}$$
$$P_8: \quad x_1^2x_2 - x_2^2s_3 + x_3^3 + 2x_2x_3x_4 - 2x_1 + 3 = 0 \tag{8}$$
$$P_9: \quad -2x_1x_3^2x^4 + x_2^3 - x_1x_2 + 2x_3x_4^3 \tag{9}$$
$$-3x_1x_2x_3x_4 - 3 = 0$$

A total of 225 0r so training samples are randomly generated from the integer points in the $[-2,2]^4$ cubic. Samples on the negative side belongs to one class, and samples on the positive side belongs to another class. A two-layer network (not counting the input layer) shown in Fig. 8 is used to learn P_5, P_6, P_7 and P_8. A three-layer network of similar structure is used to learn P_9.

Table 2 shows the number of iterations for convergence for both the multilayer nonlinear and linear synapse networks. The linear synapse network with the same structure is unable to converge for patterns 6,7, and 8 in 1,000,000 iterations. The learned synapse functions are not shown due to space limit. The highest exponents of the polynomials is 4.027 for P_5, 6.014 for P_6, 6.891 for P_7, 6.630 for P_8, and 7.848 for P_9. For P_5 and P_7, most of synapses have only one term with exponent in the interval of (1,2).

Shown in Table 2 are also the results of generalization test, i.e., testing of the learned function using newly generated samples. The testing samples are the integer points in the $[-2,2]^4$ cubic subtracting the 225 or so training samples. Therefore, there are about 400 testing samples in each case. The first number in the generalization column is the number of test samples correctly classified, and the second number is the total number of testing samples (excluding training samples). For examples, the trained nonlinear synapse network correctly classifies over 96% of the testing samples in the best case, and over 83% in the worst case.

Note that when both the linear synapse network and the polynomial synapse network converge, the number of iterations required by the nonlinear synapse network is much smaller, sometimes several order of magnitude smaller. Note that for a polynomial synapse network, each synapse has more parameters to be updated than a linear synapse network. These computations are done serially in the simulation. In hardware implementation, all parameters can be updated at the same time.

Of course, in all the above examples where a linear synapse network fails to converge, a larger linear synapse network may be able to learn the same function. However, since a linear synapse summation-only network does not have the freedom of adding independent higher order and cross terms, it may require a network with much connections to achieve the same performance of a nonlinear synapse network (with or without product contacts).

Acknowledgement This is work was supported in part by a grant from the Natural Science and Engineering Research Council of Canada.

References

[1] McCulloch, W.S. and W. Pitts, "A logical calculus of ideas immanent in nervous activity", *Bull. of Math. Biophysics*, vol. 5, 1943, pp. 115-133.

[2] Rumelhart, D.E., Hinton, G.E., Williams, R.J., "Learning internal representations by error propagation", in *Parallel distributed processing*, eds. D.E. Rumelhart, J.L. McClelland, MIT Press, Cambridge, MA, 1986, pp. 318-362.

[3] R. Durbin and D.E. Rumelhart, "Product units: a computationally powerful and biologically plausible extension to the backpropagation networks", *Neural Computation*, vol.1, No. 1, 1989.

[4] C.L. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks", *Applied Optics*, vol. 26, 1987, pp. 4972-4978.

[5] Baum, E.B. and Haussler, D.: 'What size net gives valid generalization?', *Neural Computation*, vol. 1, 1989, pp. 151-160.

[6] Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2, 303-314.

[7] Carroll, S.M. and B. W. Dickinson, 1989. Construction of neural nets using the random transform. In *Proc. Intl. Joint Conf. Neural Networks*, June 1989, Washington, DC., vol. I, pp. 607-611.

[8] Hornik, K., M. Stinchcombe and H. White, 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2, 1989, pp. 359-366.

[9] Frean, M., The upstart algorithm: A method for constructing and training feedforward networks", *Neural Comput.*, **2**, pp. 198-209, 1990.

[10] Marchand, M., Golea, M. and Rujan, P., "A convergence theorem for sequential learning in two-layer Perceptrons", *Europhys. Lett.*, **11**, 1990, pp. 487-492.

[11] Mezard, M. and Nadal, J., "Learning in feedforward layered networks: the tiling algorithm", *J. Phys. A*, **22**, 1990, pp. 2191.

[12] Liang, P. "Problem decomposition and subgoaling in artificial neural networks". *Proc. 1990 IEEE Conf. Syst. Man Cybernetics*, Nov. 4-7, 1990. Los Angeles, CA. pp. 178-181.

[13] Yin, H.F. and Liang, P., "A Connectionist Expert System Combining Production System and Associative Memory", vol. 5, No. 4, 1991, pp. 523-544.

[14] Sethi, I.K., "Entropy nets: From decision trees to neural networks", *Proc. IEEE*, vol. 78, 1990, pp. 1605-1613.

[15] Abu-Mostafa, Y.: 'Complexity in neural systems', in *Analog VLSI and neural systems*, by C. Mead, Addison Wesley, Reading, MA, 1989.

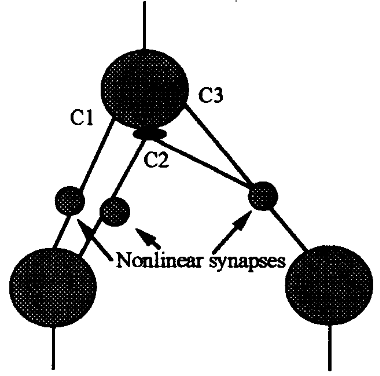

Fig. 1. Symbols for nonlinear synapses, summation and product contacts. C1 and C3 are summation contacts, and C2 is a product contact.

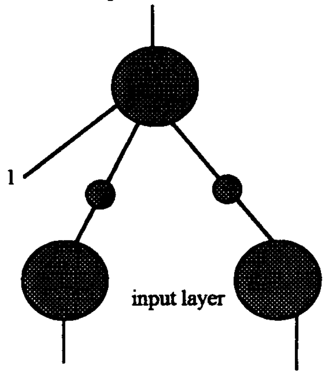

Fig.3: Network 1: a single neuron with nonlinear synapses.

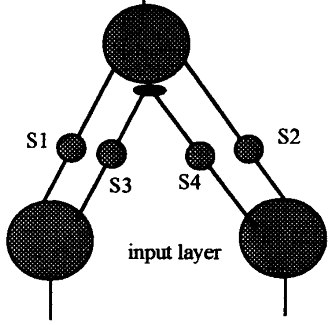

Fig. 4. Network 2: a single neuron with nonlinear synapse and product contact.

Pattern No.	No. of Iteration to convergence			
	Network 1	Network 2	Network 3	Network 4
1	20511	5284	2944	51720
2	not convergent	101465	not convergent	not convergent
3	not convergent	not convergent	10073	not convergent
4	140167	21234	6762	10647

Table 1: 2D experimental results

Pattern No.	Training Sample No.	Nonlinear synapse network		Linear synapse network	
		Iterations	Generalization	Iterations	Generalization
5	228	891	383/397	5296	362/397
6	229	7350	329/396	not	convergent
7	221	1903	367/404	not	convergent
8	226	3745	343/399	not	convergent
9	226	12631	330/399	28723	325/399

Table 2: 4D experimental results.

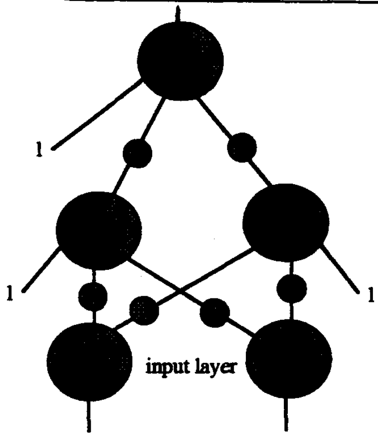

Fig. 5. Network 3: a two-layer nonlinear synapse network.

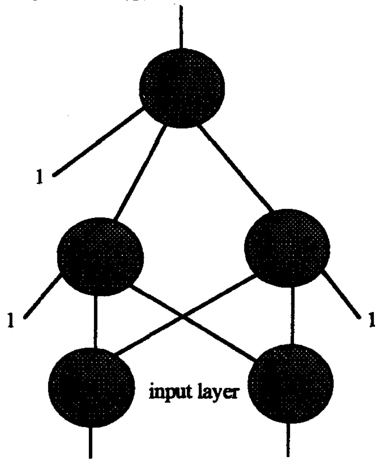

Fig. 6. Network 4: a two-layer linear synapse network having the same graph as Network 3.

```
 0      00        111  1  00           1       0         00 0  00
 0   11 00        011  1 1          1  0   1 0        0  0  0
 0   1            0  1    0         1   0    0          0000 1 1
000  11 00        00     10 0           0 11 0        0   0
    1             0 0  1 0          1  00              0 0       1
0 0               0  0 1  0         1      1            0 1 1 1
0    11   0        0  1  000        1        0        00000 1 1
 00 111 00        000    000            0 1  0        0  00  111
 0  1   00          0 11 1 0         1 00 11 0         00 00   1
 00 1 1 0         00        1       1  0  1              00000
(a) Pattern 1     (b). Pattern 2    (c). Pattern 3    (d). Pattern 4
```

Fig. 2 Training samples of pattern 1 to 4.

```
0000000000        1111111100        1100001100        0000000000
0000111100        0111111100        1100001100        0000000011
0001111100        0011111100        1100001100        0000001111
0001111100        0001111000        1100001100        0000011111
0001111100        0000111000        1100001100        0000011111
0001111100        0000111000        1100001100        0000011111
0001111100        0000111000        1100001100        0000011111
0001111100        0000111100        1100001100        0000001111
0001111100        0000111110        1100001100        0000000111
0001111000        0001111111        1100001100        0000000001
(a) Pattern 1     (b). Pattern 2    (c). Pattern 3    (d). Pattern 4
```

Fig. 7. Classification function learned by nonlinear synapse networks from the training samples in Fig. 2

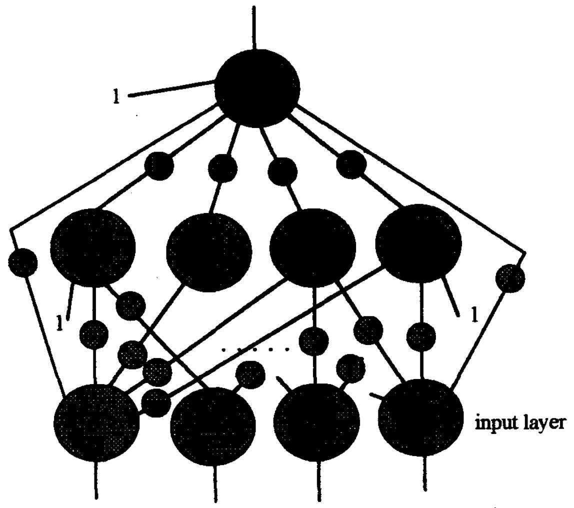

Fig. 8. A two-layer four-input nonlinear synapse network. All input units are directly connected to the output unit.