Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2014) 22: 380 – 399

doi:10.3906/elk-1201-101

A convergent algorithm for a cascade network of multiplexed dual output discrete perceptrons for linearly nonseparable classification

İbrahim GENÇ[1,*], **Cüneyt GÜZELİŞ**[2]

[1]Faculty of Engineering and Architecture, İstanbul Medeniyet University, Göztepe, Kadıköy İstanbul, Turkey
[2]Faculty of Engineering and Computer Sciences, İzmir University of Economics, Balçova İzmir, Turkey

Received: 27.01.2012 • **Accepted:** 21.07.2012 • **Published Online:** 17.01.2014 • **Printed:** 14.02.2014

Abstract: In this paper a new discrete perceptron model is introduced. The model forms a cascade structure and it is capable of realizing an arbitrary classification task designed by a constructive learning algorithm. The main idea is to copy a discrete perceptron neuron's output to have a complementary dual output for the neuron, and then to select, by using a multiplexer, the true output, which might be 0 or 1 depending on the given input. Hence, the problem of realization of the desired output is transformed into the realization of the selector signal of the multiplexer. In the next step, the selector signal is taken as the desired output signal for the remaining part of the network. The repeated applications of the procedure render the problem into a linearly separable one and eliminate the necessity of using the selector signal in the last step of the algorithm. The proposed modification to the discrete perceptron brings universality with the expense of getting just a slight modification in hardware implementation.

Key words: Discrete perceptron, cascade model, learning algorithm, constructive method

1. Introduction

A discrete perceptron, whose output becomes 1 if the weighted sum of inputs exceeds a threshold and 0 otherwise, is known to be capable of realizing any linearly separable threshold function. A set of connection weights achieving the desired linear separation can be found by the perceptron learning rule (PLR), which ensures the convergence in a finite number of steps when providing a proper learning rate. The PLR can be run to find the optimal separating hyperplane that maximizes the margin to the class samples [1], and PLR can also be used to find the largest linearly separable subset of a given linearly nonseparable set of samples[2]. Furthermore, it was proven that the pocket algorithm (PA) [3], which uses PLR, can find weights providing the minimum output error for linearly nonseparable problems.

There have been many attempts to extend the discrete perceptron model to realize any kind of threshold functions. The sequential learning algorithm (SLA) [4], as stated in its name, consists of sequential applications of the PLR; at every step, only one type of inputs (whose desired outputs are the same) are realized, and those input vectors are removed from the learning set. It was proven in [4] that all samples can correctly be classified by the SLA in a 2-layer structure, where the outputs of the first layer become linearly separable and the second layer consists of just 1 neuron. However, the work in [4] dealt with Boolean inputs only. Another algorithm, based on SLA, is the constructive algorithm for real-valued examples (CARVE) [5], which extends the SLA from

*Correspondence: ibrahim.genc@medeniyet.edu.tr

Boolean inputs to real-valued input cases; it uses a convex hull method for the determination of the network weights instead of the PLR. This algorithm gives a near-optimal solution since the task of finding the largest appropriate set is NP-hard and the algorithm only finds good-sized appropriate sets.

Besides multilayer structures, cascade models have also been investigated [6, 7, 8]. These models form a cascade structure where the inputs [6, 7] or the bias-inputs [8] of higher layers' neurons are fed by the outputs of the lower layers' neurons. These models propose structural and algorithmic solutions to linearly nonseparable classification problems against the ones [4, 5] that propose only algorithmic solutions.

The cited works above and some others were investigated in [9], and there is still a demand for developing efficient learning algorithms for multilayer discrete perceptrons to realize arbitrary threshold functions. The novel discrete perceptron model introduced in this paper is a cascade structure accompanied by a kind of SLA and it is capable of realizing any given classification task. It is shown by the experiments that it is also very convenient to run the algorithm in a discrete weight space so that the calculation speed is increased and the implementation complexity is also decreased. In the discrete case, against the real-valued case, the size of the network may increase since a suboptimal solution could be found as a consequence of the limitedness of the number of possible weights. No increment greater than 5% has been observed in most of the experiments presented in Section 4.4 as compared to the real-valued weight space, whereas in one experiment, the increment reaches about 20%. This shows that, according to the nature of the problem, the integer resolution (8bit, 16bit, etc.) should be adjusted accordingly.

The proposed modification of the multiplexed dual output perceptron provides an advantage from the hardware implementation viewpoint as complicating the implementation slightly. As compared to the classical perceptron, the new model adds only a few gate-level logic operations, which are the simplest devices to be used in electronic hardware, e.g., the field-programmable gate array (FPGA). The implementation differences between the classical perceptron neuron and the proposed model are shown for the FPGA implementation case.

As a method for implementation, the FPGA approach, which uses reprogrammable digital ICs, is chosen since the usage of FPGA for neural network implementation provides flexibility of programmable systems along with the power and speed of parallel hardware architectures [10, 11, 12, 13, 14, 15, 16, 17].

The work presented in the paper mainly concentrates on the universality of the proposed network, i.e. it is capable of realizing any threshold class function defined by a finite number of input output samples; and on the convergence of the algorithm developed for designing the proposed network. The proposed cascade network aims to realize a classification task that is defined as the separation of a given data set into 2 different classes based on their labels. FPGA implementation of the cascade network, which is designed by the algorithm to run on a computer, is presented in the paper just for demonstrating its validity when implemented as hardware.

In Section 2, the proposed multiplexed dual output neural network (NN) model is defined. Section 3 describes the learning algorithm developed for that model. Some example problems are shown in Section 4. Section 5 investigates the model in terms of hardware implementation and an implementation example is given. Finally, Section 6 concludes the work presented in the paper.

2. Model

A function, $f : \Re^p \to \{0, 1\}$, separates the elements of a given input set, $X = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\} \subset \Re^p$, into 2 disjoint sets, X_f^+ and X_f^-, as

$$X_f^+ := \left\{ \mathbf{x}^i \in \Re^p \mid f(\mathbf{x}^i) = 1 \text{ for all } i \in \{1, 2, \ldots, N\} \right\}, \tag{1}$$

$$X_f^- := \left\{ \mathbf{x}^i \in \Re^p \mid f(\mathbf{x}^i) = 0 \text{ for all } i \in \{1, 2, \ldots, N\} \right\}. \tag{2}$$

For binary valued functions $f(\mathbf{x})$, there can always associate a function $g(\mathbf{x})$ such that $f(\mathbf{x}) = \text{stp}(g(\mathbf{x}))$ with $\text{stp}(\cdot)$ being the unit step function. Such $g(\cdot)$ functions are called separators. If $g(\cdot)$ is a linear function, it is said to be a linear separator, and so it defines a linear separation in the input space, $\Re^p$.

A supervised classification task for a finite set of samples $X = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$ can be defined by a function $F(\cdot) : \Re^p \to \{0, 1\}$, which is given with its input–desired output pairs, i.e. with the following domain–range values:

$$S_F \;:=\; \left\{ (\mathbf{x}^i, d^i) \mid \mathbf{x}^i \in X, d^i \in \{0, 1\},\ d^i = F(\mathbf{x}^i) \text{ for all } i \in \{1, 2, \ldots, N\} \right\}. \tag{3}$$

S_F can be decomposed into 2 disjoint sets:

$$S_F^+ \;:=\; \left\{ (\mathbf{x}^i, d^i) \mid \mathbf{x}^i \in X_F^+ \text{ with } d^i = F(\mathbf{x}^i) \text{ for all } i \in \{1, 2, \ldots, N\} \right\}, \tag{4}$$

$$S_F^- \;:=\; \left\{ (\mathbf{x}^i, d^i) \mid \mathbf{x}^i \in X_F^- \text{ with } d^i = F(\mathbf{x}^i) \text{ for all } i \in \{1, 2, \ldots, N\} \right\}. \tag{5}$$

Any realized function $f(\cdot) : \Re^p \to \{0, 1\}$ partitions the above given input–desired output pairs into the following 4 sets. It should be mentioned that T^+ and T^- sets represent the correct classified pairs, while the other 2 sets represent the misclassified pairs:

$$T^+ \;:=\; \left\{ (\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^+ \text{ and } \mathbf{x}^i \in X_f^+ \text{ for all } i \in \{1, 2, \ldots, N\} \right\}, \tag{6}$$

$$T^- \;:=\; \left\{ (\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^- \text{ and } \mathbf{x}^i \in X_f^- \text{ for all } i \in \{1, 2, \ldots, N\} \right\}, \tag{7}$$

$$F^+ \;:=\; \left\{ (\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^+ \text{ and } \mathbf{x}^i \in X_f^- \text{ for all } i \in \{1, 2, \ldots, N\} \right\}, \tag{8}$$

$$F^- \;:=\; \left\{ (\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^- \text{ and } \mathbf{x}^i \in X_f^+ \text{ for all } i \in \{1, 2, \ldots, N\} \right\}. \tag{9}$$

Definition 1 (Correct separation) *For a given set S_F, if there can be found a linear separator $f(\cdot) : \Re^p \to \{0, 1\}$ yielding $F^+ \cup F^- = \emptyset$, then the set S_F is said to be linearly separable. Such separators are said to be (completely) correct separators.*

Definition 2 (Semicorrect separation) *A separator realized by $f(\cdot) : \Re^p \to \{0, 1\}$ is said to be a semicorrect separator if it yields either $F^+ = \emptyset$ or $F^- = \emptyset$.*

The proposed model realizes a correct separator for any given classification task of a finite number of real sample vectors. The main idea behind the structure of the model is derived from the fact that the output of a perceptron is either true or false according to a desired output. To realize a given function, the model is designed to have neurons whose outputs are copied so that the copied output is the complement of the other and a selector signal is produced to choose one of them. Now, the problem is transferred from the realization of desired outputs to the realization of the selector signals for given input vectors. In the next step of the design, the selector signal is taken as the desired output of the remaining part of the network.

The proposed network is depicted in Figure 1. The last entry of $\mathbf{w}$ corresponds to the threshold while the last entry of the input vector $\mathbf{x}$ is set to unity.

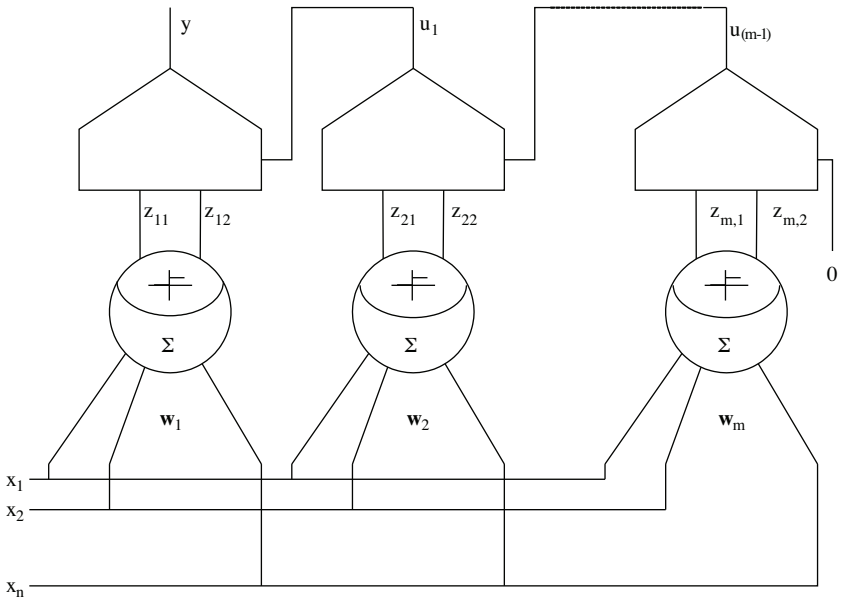


Figure 1. A cascade network of multiplexed dual-output discrete perceptron.

The network of Figure 1 is constructed as starting from the left part of the illustration. The first attempt tries to realize S_F. Set $S_{F_0} = S_F$ and then define, in a recursive way, the input–desired (selector) output pair sets as:

$$S_{F_j} \quad = \quad \left\{(\mathbf{x}^i, u_j^i) \,|\, \mathbf{x}^i \in X,\ u_j^i \in \{0,1\} \text{ for all } i \in \{1,2,\ldots,N\}\right\}. \tag{10}$$

S_{F_j} is the set of input–desired (selector) output pairs that, indeed, defines the function $u_j = F_j(\mathbf{x})$ to be realized in the j^{th} stage. The function $u_j = F_j(\mathbf{x})$ is attempted to be realized by a selector network $y_j = f_j(\mathbf{x})$, which, in fact, partitions the S_{F_j} into the 4 sets again:

$$T_j^+ := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^+ \text{ and } \mathbf{x}^i \in X_{f_j}^+ \text{ for all } i \in \{1,2,\ldots,N\}\}, \tag{11}$$

$$T_j^- := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^- \text{ and } \mathbf{x}^i \in X_{f_j}^- \text{ for all } i \in \{1,2,\ldots,N\}\}, \tag{12}$$

$$F_j^+ := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^+ \text{ and } \mathbf{x}^i \in X_{f_j}^- \text{ for all } i \in \{1,2,\ldots,N\}\}, \tag{13}$$

$$F_j^- := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^- \text{ and } \mathbf{x}^i \in X_{f_j}^+ \text{ for all } i \in \{1,2,\ldots,N\}\}. \tag{14}$$

The realization of the given function $d = F(\mathbf{x})$ by the networks's input–output function $y = f(x)$ follows from the below derivation.

$$y = \bar{u}_1 \cdot z_{11} + u_1 \cdot z_{12} \tag{15}$$

$$z_{11} = \text{stp}\left(\mathbf{w}_1^{\mathrm{T}} \cdot \mathbf{x}\right) \tag{16}$$

$$z_{12} = \text{stp}\left(-\mathbf{w}_1^{\mathrm{T}} \cdot \mathbf{x}\right) \tag{17}$$

$$u_j = \bar{u}_{(j+1)} \cdot z_{j1} + u_{(j+1)} \cdot z_{j2} \tag{18}$$

$$z_{j1} = \text{stp}\left(\mathbf{w}_j^{\mathrm{T}} \cdot \mathbf{x}\right) \tag{19}$$

$$z_{j2} = \mathrm{stp}\left(-\mathbf{w}_j^{\mathrm{T}} \cdot \mathbf{x}\right) \tag{20}$$

$$u_m = 0 \tag{21}$$

Eqs. (15) and (18) are realized by the multiplexers shown in Figure 1. Herein, $\bar{u}$ shows the logical complement of u and m stands for the last stage.

Calculation of the selector signal u_j values is very straightforward. The selector signal is defined as '0' when the output of the neuron, z_{j1}, is equal to the desired output, and it is defined as '1' otherwise. Therefore, the selector signal is just a logical ex-or operation of the actual output of the neuron, z_{j1}, and the desired output, d, as shown in Eqs. (22) and (23).

$$u_1 = d \otimes z_{11} \tag{22}$$

$$u_j = u_{j-1} \otimes z_{j1} \tag{23}$$

Here, the symbol $\otimes$ represents the logical ex-or operator, d could also be notated as u_0, and u_j is the desired value for the $j{+}1^{th}$ stage of the network.

During the design procedure, not only the connection weights but also the structure of the model is learned.

Property 1 *From one stage to the next, the training set changes as follows:*

- *if* $(\mathbf{x}^i, u_j^i) \in S_{Fj}^+$ *and* $\mathbf{x}^i \in T_j^+$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^-$; *we can say that realized '1's are changed to '0'.*
- *if* $(\mathbf{x}^i, u_j^i) \in S_j^-$ *and* $\mathbf{x}^i \in T_j^-$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^-$; *we can say that realized '0's remain '0'.*
- *if* $(\mathbf{x}^i, u_j^i) \in S_j^+$ *and* $\mathbf{x}^i \in F_j^+$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^+$; *we can say that nonrealized '1's remain '1'.*
- *if* $(\mathbf{x}^i, u_j^i) \in S_j^-$ *and* $\mathbf{x}^i \in F_j^-$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^+$; *we can say that nonrealized '0's are changed to '1'.*

The above properties are obvious from Eqs. (3)–(9) and Eq. (18). □

Since the structure of the network is built up by the procedure given above, the weights are the only parameters to be learned. The learning algorithm presented in the next section is employed to calculate weights of each layer of the network.

3. Algorithm

In the model, the number of layers of the network, m, is left undefined because it is not known at the beginning. It is determined along the learning process defined by the algorithm. A pseudocode for the algorithm is given in Table 1.

To explain the basics of the algorithm in a clear way, the simplest linearly nonseparable problem, EXOR, which is illustrated in Figure 2, is considered. The steps of the algorithm when applied to the EXOR problem are as follows: at the beginning, the problem to be solved is defined as a set of input–output pairs: ((0,0), 0), ((0,1), 1), ((1,0), 1), and ((1,1), 0). For the first stage of the algorithm, the PA finds a semicorrect separating hyperplane with minimal constrained error, i.e. only one type of error exists (either false-0 or false-1). The original problem and the separating hyperplane found by the PA are depicted in Figure 2a. The half-space of the hyperplane that gives 1 as output is marked by an arrow. Thus, the only erroneous case for this example is obtained for the input (1,1), whereas although the desired output is 0, the network produces 1 for that input.

Table 1. The learning algorithm for a cascade network of multiplexed dual -output discrete perceptron.

1	a	$j = 1$.
	b	d^is are the desired outputs. Set $u_0^i = d^i \ \forall i$.
2	a	If the set is linearly separable, then the neuron of the j^{th} layer realizes the desired linear separation so **Stop here.**
	b	If the set is linearly nonseparable, a semicorrect separation with a minimal output error could be found by CPA, and $\mathbf{w}_j$ is obtained.
3	Calculate $z_{j1} = f\left(\mathbf{w}_j^{\mathrm{T}} \cdot \mathbf{x}\right)$, $z_{j2} = f\left(-\mathbf{w}_j^{\mathrm{T}} \cdot \mathbf{x}\right)$.	
4	Define u_j^i as $u_j^i = z_{j1}^i \otimes u_{j-1}^i$.	
5	Take u_j^i as the desired output for the remaining part of the network.	
6	Increase j by 1.	
7	Go to step 2 for the realization of u_j^i.	

Hence, the selector signal should be 1 for this input and 0 for the others. At the second stage of the network construction, the problem is transformed into the realization of the selector signal of the first stage, namely, the realization of input–output pairs: ((0,0), 0) ((0,1), 0), ((0,1), 0), and ((1,1), 1). Since this is a linearly separable problem, the PA finds the correct separation as depicted in Figure 2b and the selector signal becomes 0 for all input data, and then the construction stops. The resulting network consists of 2 neurons.

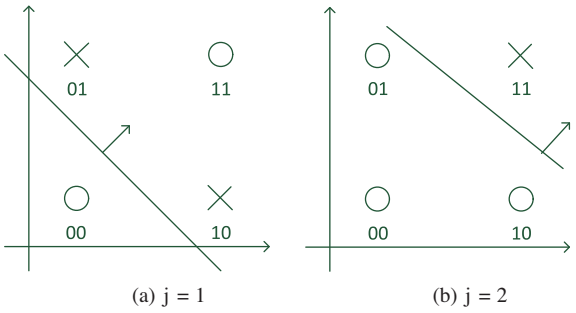


Figure 2. EXOR example illustrating the learning steps of the proposed algorithm, where j represents the step number.

The proposed model and the accompanying algorithm assure that any given classification problem is solved with a finite number of layers. Convergence properties of the algorithm are given and proven in Section 3.1.

The critical stage of the algorithm is Step 2, where the weights of neurons are determined by using a PA. When the problem set is linearly separable, it is well known that the PLR is capable of finding the solution very fast and the algorithm stops at Step 2. For linearly nonseparable sets, it is known that the PLR can provide very valuable information about the given sets. For instance, it can find input vectors violating the linear separability. Thus, it can find the largest linearly separable subset of a given set[2]. On the other hand, the PA [3], which uses the PLR, can find weights providing the minimum output error for linearly nonseparable problems. In the literature, a comparison of different linear separability testing methods on constructive neural networks was studied [18]. The results of the experiments done on 9 different data sets and by using 6 different methods are summarized in Table 2. As can be seen, the PLR is among the faster methods and is much faster than the mean value of all the methods, which is equal to 31.91.

Table 2. Results obtained with the 6 models, using different methods for testing linear separability, and 9 data sets in terms of the convergence time (m represents the average over the data sets, excluding extreme values)[18].

	Simplex	ConvexHull	SVM	PLR	Anderson	Fisher
Iris	0.09	1.35	0.68	0.52	0.19	0.39
Soybean	0.11	1.34	1.81	0.44	18.18	2.11
Monks	1.00	2.80	274.20	2.79	3.62	4.10
Glass	3.47	122.40	63.00	386.40	1.69	6.27
Hepatitis	0.02	2.79	4.33	18.18	4572.00	1.36
Wine	0.02	0.11	9.54	5.34	0.14	0.01
Ionosphere	7.18	2.60	499.80	3.00	25.32	11.64
Wisconsin Breast Cancer	92.40	112.20	7776.00	15.11	3.56	78.00
Sonar	0.13	2.22	10.36	6.89	202.80	6.91
m	1.71	17.90	123.29	7.40	36.48	4.68

A heuristic approach based on consecutive applications of a PA for output error minimization is used in the design of the proposed cascade network of multiplexed dual output discrete perceptrons. The algorithm used to find the minimum output error for each layer might be called a constrained pocket algorithm (CPA) since it is a PA and a constraint is added onto it. The constraint here is added to ensure the convergence of the whole learning process, as explained in Step 2b of Table 1, in order to find not just a minimal error but a semicorrect separation with minimal error. The weights giving the minimum output error under the constraint that only one type of input (whose desired outputs are the same) exists on one side of the hyperplane are saved in the pocket. This constraint is not dependent on domain application; on the contrary, it brings universality to the model and the algorithm.

Moreover, it should be noted that the algorithm is very convenient for adding some other constraints if a specific application needs some special constraints. For example, quantized weight space due to the implementation capabilities limits the learned real-valued weights to a finite set of weights. To get control of possible misclassification due to the quantization of learned weights in an implementation and to make the implementation easier, training of the network would be better done in a discrete weight space, i.e. a constraint that learned weight should be in a set of a finite number of weights, e.g., 16-bit integer-valued weights. It is shown in [19] for perceptrons trained in discrete weight space that if the weights' depth is very large, i.e. there are many possible values for each weight, the learning behavior of the discrete weights will be exactly the same as those of a continuous weight. Furthermore, i) the learning in the case of finite depth is possible by using a continuous precursor; ii) in the case of binary output and on-line learning —and this is exactly the case used in our algorithm— the generalization error decays superexponentially; and iii) perfect learning is obtained when N, the cardinality of the input set, is very large but finite. It seems to be the only disadvantage of the discrete weight case that the size of the network may increase since a suboptimal solution could be found because of limited weight space at some stages.

Some open points of the CPA are that there are no analytical stopping criteria also shared with the original PA and that it may need a long time to find the optimal weights, especially when the training set is large, which may be due to the constraint. However, it should be noted that the proposed algorithm is not strictly bound to the PA and any learning algorithm could be applied at each intermediate step of the construction for determining a suboptimal linear separation. For instance, instead of the PA, there is the thermal perceptron

learning algorithm [20], which is a modified PLR with an annealing factor as exploited in other constructive algorithms[21]. As listed in Table 3, a set of different methods are investigated for linearly nonseparable learning problems and their performances are compared in a constructive neural network algorithm[18]. As summarized in Table 2, although there are no stopping criteria established, the PLR method is found to be among the fast methods in [18].

Table 3. Results obtained with the 6 models, using different methods for testing linear separability, and 9 data sets in terms of the constructed network size (m represents the average over the data sets, and max is the maximum constructed network size)[18].

	Simplex	ConvexHull	SVM	PLR	Anderson	Fisher
Iris	1.50	1.80	1.70	2.70	1.10	2.30
Soybean	2.00	2.40	2.20	3.80	2.70	7.00
Monks	2.00	2.20	3.00	6.50	6.30	2.60
Glass	17.50	17.60	15.40	30.10	1.00	4.40
Hepatitis	1.00	3.20	4.30	12.40	1.60	6.30
Wine	1.00	1.00	2.00	3.00	1.00	1.00
Ionosphere	2.50	3.50	3.00	9.90	8.70	17.10
Wisconsin Breast Cancer	11.40	15.30	12.10	20.70	3.00	3.90
Sonar	1.00	2.30	1.20	24.90	10.00	20.90
m	4.43	5.48	4.99	12.67	3.93	7.28

Comparing the sizes of the constructed networks in Table 3, it can be concluded that the recursive deterministic perceptron (RDP) network[18], which applies a specific construction scheme, produces the worst results for the considered data sets in terms of the network sizes if the PLR is used at each step of the construction. As explained in [18], the lack of a procedure for determining a proper stopping time for the PLR due to the linearly nonseparable nature of the problem may be the reason. However, to the knowledge of the authors of this paper, there is no theoretical result in the literature to allow comparison of the performance of PLR versus the other methods used at each step of constructive networks for obtaining a suboptimal linear separation. As will be made clear in Subsection 4.2, although the CARVE algorithm based on the convex hull method produces the smallest network size, the PA employed by the method proposed in this paper, which uses a modified PLR algorithm at each step with a proper stopping rule, also produces smaller network sizes for random Boolean function data sets compared to some other methods. This means that the performance of the PLR employed in constructive networks can be improved when the PLR is properly used.

On the other hand, comparing the learning algorithms in terms of their generalization performances, it is concluded that the PLR is among the good ones. The algorithms and their generalization results are shown in Table 4. The most remarkable point as regards these values can be described as follows: the PLR gives better generalization results when the problem is more complex. This is seen from Table 4 when md is low, i.e. when it is less than about 75%, the PLR leads to the best or the second best generalization performance. However, there is no possibility of direct comparison of the proposed method to the previous works in terms of the generalization performance, since the most similar methods, those also employing the same data sets[4, 5, 22, 23], do not declare their generalization performances. The generalization performance of the proposed cascade network is discussed in Section 4.5.

Table 4. Results obtained with the 6 models, using different methods for testing linear separability, and 9 data sets in terms of the generalization capacity (here *md* represents the average over methods for each data set) [18].

	Simplex	Conv.Hull	SVM	PLR	Anderson	Fisher	*md*
Iris	92.5	93.25	92.75	93.25	96.75	95.5	94.00
Soybean	71.71	77.71	71.46	73.13	51.67	57.71	67.23
Monks	100	99.48	99.80	84.16	57.63	71.74	85.47
Glass	57.62	67.43	67.29	73.52	57.52	53.57	62.83
Hepatitis	89.93	79.61	84.64	81.08	76.35	67.62	79.87
Wine	98.18	96.36	95.46	95.46	100	100	97.58
Ionosphere	85.72	88.02	88.07	88.58	63.48	36.53	75.07
Wisconsin Breast Cancer	93.73	95.77	95.18	95.32	85.53	76.56	90.35
Sonar	71.20	75.92	70.61	74.40	53.86	72.54	69.76

3.1. Convergence of the algorithm

The convergence of the algorithm will be proven based on a complexity definition such that as the algorithm runs, the complexity decreases or remains the same and in a finite number of steps the complexity becomes zero, which corresponds to a zero misclassification error. The definition of the complexity and a proof for the convergence of the algorithm are given below.

For a given set, S, consisting of input–desired output pairs, a semicorrect separation that minimizes the output error under the constraint can always exist and can be found by the CPA [3]. This is also true for the training sets S_{F_j} constructed for the subnetworks used for the realization of selector signals. The complexity of any such set of inputs–desired outputs, which is defined below, is a measure of divergence away from the linear separability and it gives the minimum number of samples such that the exclusion of them yields the linear separability.

Definition 3 (Complexity) *The complexity $c(S_{F_j})$ of an input–desired output set S_{F_j} is the cardinality of the set $E(S_{F_j})$, which is obtained by using Eqs. (11)–(14) in the following way:*

$$E(S_{F_j}) = \begin{cases} F_j^+, \ \text{if } |F_j^+| < |T_j^-| \ \text{and } F_j^- = \emptyset \\ T_j^-, \ \text{if } |T_j^-| < |F_j^+| \ \text{and } F_j^- = \emptyset \\ F_j^-, \ \text{if } |F_j^-| < |T_j^+| \ \text{and } F_j^+ = \emptyset \\ T_j^+, \ \text{if } |T_j^+| < |F_j^-| \ \text{and } F_j^+ = \emptyset, \end{cases} \tag{24}$$

$$c(S_{F_j}) = |E(S_{F_j})|. \tag{25}$$

□

The complexity defined above relies on the following observations. The separating hyperplane separates the input set into 2 subsets: T_j^- and F_j^+ are on one side of the hyperplane and T^+ and F^- are on the other side. Considering the constraint that either F_j^- or F_j^+ is empty, to obtain a linearly separable set from the original set, there are some possibilities depending on the sets T_j^-, T_j^+, F_j^-, and F_j^+. Assuming that F_j^+ is the empty set, the sets T_j^+ and F_j^- are on one side and T_j^- is on the other side of the hyperplane. If the elements of F_j^- are excluded from the training set, the remaining elements, T_j^+ and T_j^-, are separated by the

hyperplane, and so the problem becomes linearly separable. The other option, which is the exclusion of set T_j^+, results in only F_j^- and T_j^- remaining in the training set. Since the desired outputs for the elements of both sets are all 0, and there is no element with the desired output of 1, the exclusion concludes that there is no need for a separation. These considerations are stated in Theorem 1.

Theorem 1 *For a semicorrect separation that minimizes the output error under the exclusion of the elements of $E(S_{F_j})$, the remaining set is either linearly separable or needs no separation since all samples are of the same kind.*

Proof There are 4 cases: i) $F_j^- = \emptyset$ and the erroneous elements $\mathbf{x}$, which are in F_j^+, are excluded, and then only the elements yielding correct outputs remain, i.e. elements $\mathbf{x}$ are either in T_j^+ or T_j^-. ii) $F_j^- = \emptyset$ and the correct elements $\mathbf{x}$, which are in T_j^-, are excluded, and then only the elements whose desired outputs are all the same remain, i.e. $\mathbf{x} \in T_j^+$ or $\mathbf{x} \in F_j^+$. There is no need for separation. iii) and iv) can be proven by considering $F_j^+ = \emptyset$ and following the same approach as in cases i) and ii). In all of the 4 possible options, it follows that the remaining set is linearly separable. □

Theorem 2 *The algorithm defined in Table 1 converges to a zero output error in a finite number of steps.*

Proof At each stage of the algorithm, which corresponds to the design of a layer of the network, a correct separation or a semicorrect separation occurs. For a semicorrect separation that gives a minimal output error, there are 2 cases:

1. Case 1: $F_j^- = \emptyset$

   (a) If $\forall \mathbf{x}^i \in T_j^+$, i.e. $\mathbf{x}^i \in X_j^+$ and $(\mathbf{x}^j, d^j) \in S_{F_j}^+$, then $\mathbf{x}^i \in S_{F_{j+1}}^-$.

   (b) As the worst case, assume that the same separator is used for the $(j+1)^{th}$ layer as with j^{th} layer but in the opposite direction, so $T_{j+1}^- = T_j^+$, $F_{j+1}^- = T_j^-$, $T_{j+1}^+ = F_j^+$, and $F_{j+1}^+ = \emptyset$. In this case the complexity does not change since the complexities, $c(S_{F_j}) = min\{|F_j^+|, |T_j^-|\}$ and $c(S_{F_{j+1}}) = min\{|T_{j+1}^+|, |F_{j+1}^-|\}$, are equal to each other.

   (c) At each j^{th} step, the algorithm finds either a correct separation (see 2a of Table 1) or semicorrect separation with minimum output error (see 2b of Table 1). In either case, the vectors in T_j^+ define a convex hull whose intersection with the convex hull of the vectors in $F_j^+ \cup T_j^-$ is empty, and the vertices of the convex hull of $F_j^+ \cup T_j^-$ that are the closest vertices to the convex hull of T_j^+ are necessarily in T_j^-. This is because any vertex of F_j^+ with the minimum distance to T_j^+ could be included by T_j^+ without violating the linear separability of T_j^+ and T_j^-. This can be seen as follows: i) the extension of a convex set via including a point from its outside possesses that point as a vertex; ii) the convex hull of a set constructed by excluding a vertex of a considered set does not include that excluded vertex as one of its points; and iii) 2 convex hulls are linearly separable iff their intersection is empty. The vertices of the convex hull of $F_j^+ \cup T_j^-$ that are the closest to

the convex hull of T_j^+ become at the $(j+1)^{th}$ stage the vertices of $T_{j+1}^+ \cup F_{j+1}^-$, which are now in F_{j+1}^-.

(d) From Case (1c), considering $S_{F_{j+1}}$ in the new separation for the (j+1)th layer, $|F_{j+1}^-| < |T_j^-|$ and $|T_{j+1}^+| = |F_j^+|$. Therefore, the complexity from layer j to layer $j+1$ decreases or remains constant: $c(S_{F_{j+1}}) = \min\{|F_{j+1}^-|, |T_{j+1}^+|\} \leq c(S_{F_j}) = \min\{|F_j^+|, |T_j^-|\}$.

(e) Even for the cases of complexity remaining constant, the total cardinality $|T_j^+| + |F_j^-|$ decreases at each step, which can cause a decrease of a certain amount in the complexity after a certain number of steps. When F_{j+1}^+ is empty, which can be analyzed (see Case 2) in a similar way to the analysis of F_j^- being empty, the complexity also decreases or remains constant.

(f) If at each stage F_j^- is empty then it can be concluded that the complexity decreases to zero within a finite number of steps.

2. Case 2: $F_j^+ = \emptyset$

The analysis of this case is the same with the Case 1 but with the following:

(a) The same separator function with the same direction is used from the j^{th} layer to the $j+1^{th}$ layer instead of the opposite direction in point b of Case 1.

(b) From the j^{th} layer to the $(j+1)^{th}$ layer, sets change as follows: $T_{j+1}^+ = F_j^-$, $F_{j+1}^- = T_j^+$, $T_{j+1}^- = T_j^-$, and $F_{j+1}^+ = \emptyset$.

(c) The complexities are as $c(S_{F_j}) = min\{|T_j^+|, |F_j^-|\}$ and $c(S_{F_{j+1}}) = min\{|F_{j+1}^-|, |T_j^+|\}$.

Combining the 2 cases, the complexity is seen to decrease to zero in a finite number of steps. □

To demonstrate how the algorithm works, the next section gives some example applications of the algorithm and experimental results.

4. Experimental results

In this section, experimental results of the work are presented.

The proposed algorithm uses the PA at each step to find the largest separable subset. Because of the stochastic nature of the PA, it finds hyperplanes that give minimal constrained output errors. On the other hand, 2 different hyperplanes giving the same error may lead to networks of different sizes. Thus, multiple trials for each specific classification problem were conducted in the experiments to find the mean size of the constructed NN. The resulting networks for the considered problems are compared to the networks obtained with the other techniques available in the literature.

For the sake of clarity and a better understanding of the algorithm, a synthetic example problem illustrated in Figure 3 is designed as a 4×5 grid, where the desired output values for input samples are assigned randomly.

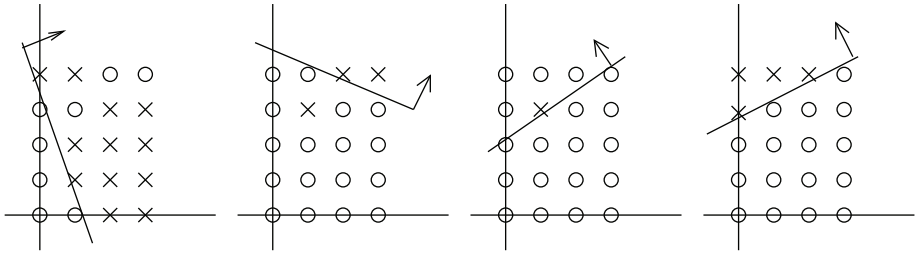


Figure 3. A synthetic example illustrating the learning steps of the proposed algorithm, where j represents the step number.

Table 5. The complexity at each stage of the algorithm for the example shown in Figure 3.

j	$\lvert T_j^- \rvert$	$\lvert F_j^- \rvert$	$\lvert T_j^+ \rvert$	$\lvert F_j^+ \rvert$	Complexity
1	5	3	12	0	3
2	17	0	2	1	1
3	15	4	1	0	1
4	18	0	4	0	0

In each subfigure of Figure 3, x and o represent input vectors whose desired outputs are '1' and '0', respectively. The lines in the figures correspond to the separating hyperplanes constructed by $\mathbf{w}_j$ and arrows show the positive sides of those hyperplanes. Therefore, for all vectors belonging to the positive side of a hyperplane, the output of the network realized at the j^{th} step will be '1'. Please note that at every subfigure, either side of the hyperplane contains only one type of vector, i.e. x or o. The progress of the construction process is as follows: the problem is first defined with input–output pairs as ((0,0), 0), ((1,0), 0), ((2,0), 1), ((3,0), 1), ... ((3,4), 0). By running the PA for neuron 1, an optimal semicorrect separation is found. For this separation, the erroneous outputs are (1,3), (2,4), and (3,4) for which the desired outputs are equal to 0, but the actual output is 1 as depicted in Figure 3a. Thus, a selector is needed, and it should be 1 for these inputs and 0 for the others. Therefore, the problem is transformed into the problem of realization of the selector signal of the first neuron, u_1, which is illustrated in Figure 3b. Training the second neuron to realize u_1 resulted in an optimal separating hyperplane for the problem. For this hyperplane, the only erroneous output is obtained for input vector (1,3). The new problem is the realization of the selector signal of neuron 2, u_2, which is shown in Figure 3c. Applying the same procedure explained above, the problem finally is transformed into a linearly separable one, as seen from Figure 3d. At the last step, a hyperplane not leading to any error is found without a need for a selector signal and then the construction process ends with a cascaded 4-neuron network.

Error logs of the algorithm for the example depicted in Figure 3 are listed in Table 5.

4.1. Parity

The parity problem is a common test to measure the performance of NN classification algorithms. The problem is to classify binary input vectors into 2 sets depending on whether or not the number of binary 1s in the input vectors is even.

The parity problem requires, when using the multilayer perceptron (MLP) with sigmoid activation function, as many hidden neurons in the first layer as the input dimension [24]. Even though sine-type

Table 6. Mean sizes of constructed networks by different methods for random Boolean functions.

n	The proposed model	Tiling algorithm[22]	SLA[4]	CARVE[5]	Regular [23]
4	2.44 ± 0.50	3.9		2.40 ± 0.69	
6	8.40 ± 1.27	16.99	7.28 ± 0.82	5.88 ± 0.67	15.8 ± 2.2
8	35.75 ± 0.95	56.98	18.3 ± 0.69	16.23 ± 0.86	

activation functions or activation functions exploiting the series expansion based on Hermite orthonormal functions overcome the above limits, we selected the sigmoid/sign-type activation function for the sake of comparison. Consequently, in the case of 4-dimensional input space, the network should contain at least 5 neurons, 1 of which is for the output layer.

The experiments with the proposed model and the algorithm show that only n neurons are enough to solve the n-bit parity problem. However, when the input dimension increases, the training set becomes larger and finding the optimal hyperplane for each layer becomes difficult. Therefore, training time may increase.

In the experiments done for $n = 1, \ldots, 5$, network sizes with n neurons are obtained quite quickly for all trials. However, for $n = 6, \ldots, 8$, relatively long training periods are needed. This can be interpreted as a consequence of the stochastic nature of the CPA.

4.2. Random Boolean functions

The realization of a random n-bit Boolean function is a problem of providing desired outputs for all 2^n binary input vectors. Every input vector is randomly assigned with equal probability to a desired binary output representing 1 of the 2 classes.

For each dimension, $n \in \{4, 6, 8\}$, 10 different test sets are produced and every test set is tested for the proposed network several times. The results of the experiments are given in Table 6, and they are parallel to the results of the recent publications summarized in Table 3.

4.3. Two spirals

The 2-spirals problem [25] is a classification task that is a highly nonlinear separation problem. Single hidden layer networks trained by backpropagation generally fail to produce solutions to this problem and constructive algorithms have been demonstrated to be more successful [26].

The model of this paper generates another constructive solution for this 2-spirals problem. Some steps of the solution are given in Figure 4a, Figure 4b, and Figure 4c. The average network size obtained over 8 trials is 57.37 ± 2.56, with a minimum network size of 56 layers and a maximum size of 62 layers.

4.4. Learning in discrete weight space

An implementation of artificial neural networks on digital hardware always involves a quantization of learned connection weights, and so may cause misclassification of learned samples. For better control of such quantization effects, learning of the proposed network weights is also done in discrete weight space.

The discrete weight version of the algorithm is run for random Boolean functions, parity functions, and 2-spirals function. In the experiments, 16-bit signed integers and 16-bit fixed point real values with a 12-bit fraction length are used as discrete weight spaces. As can be seen from the results given below, the produced network sizes are found to be similar with no considerable difference as in accordance to the facts stated in [19]. There are different publications addressing the optimal discretization of weights and inputs; however, our aim

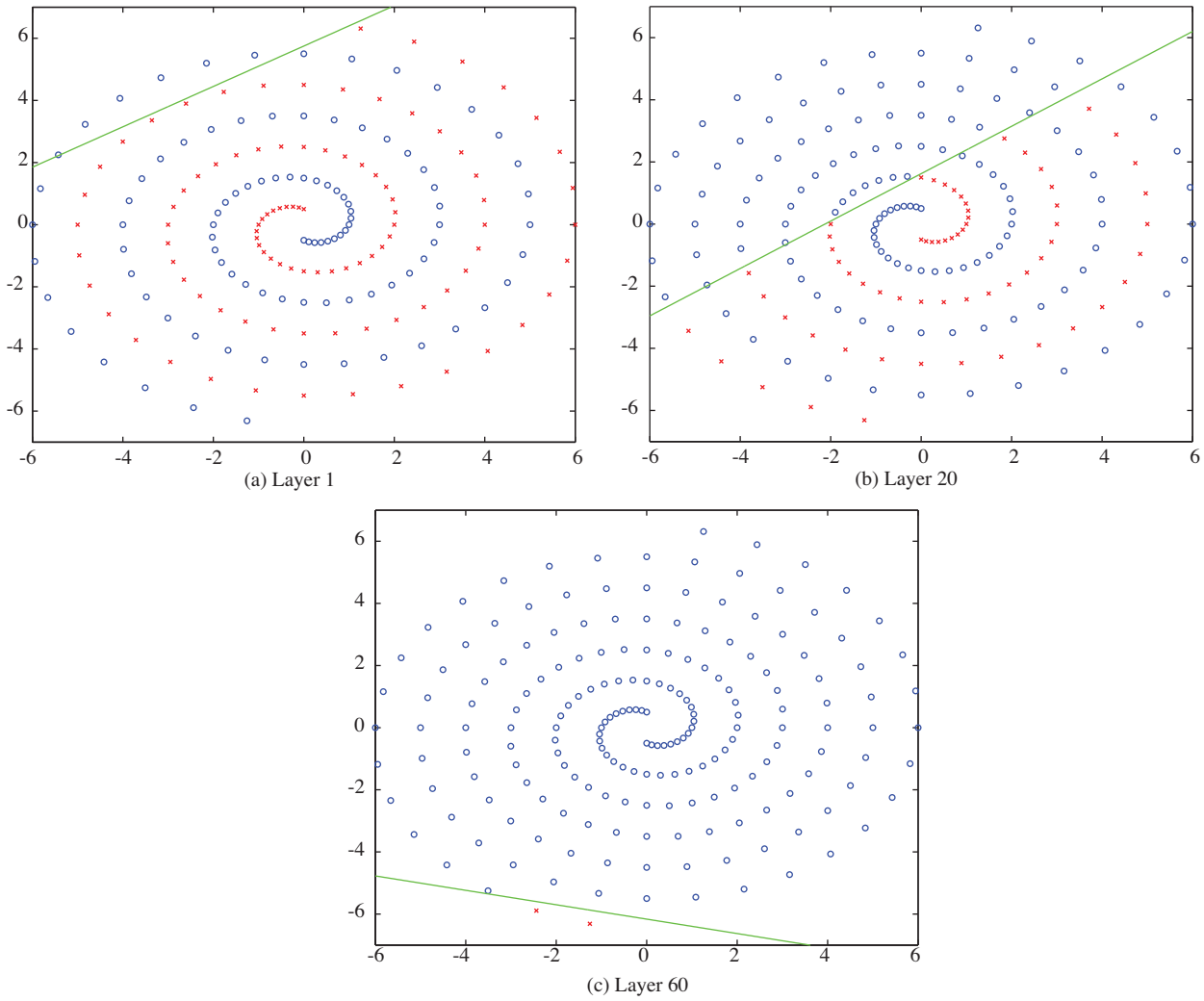


Figure 4. Two-spirals solution epochs obtained by the proposed cascade network of multiplexed dual-output discrete perceptron.

here is not to optimize the discretization but rather to investigate the effect of discretization on our proposed model and algorithm.

4.4.1. Parity

The experiments of the proposed model in discrete weight space are done for parity problems in 4-, 5-, and 6-dimensional input spaces. The results show that only n neurons are enough to solve the problem with discrete weights, similar to the examples mentioned in Section 4.1. The discrete weight space is taken as 16-bit signed integers.

4.4.2. Random Boolean functions

The results of learning in integer weight space for random Boolean functions are listed in Table 7. Functions the same as the ones used in the examples in Section 4.2 are chosen and the weight space is taken as 16-bit signed integers. As can be seen from the results, the produced network sizes are similar to the ones obtained in

Table 7. Random Boolean function network sizes in discrete weight space learning, where n is the input space dimension.

n	Real numbers	16-bit integers
4	2.44 ± 0.50	2.60 ± 0.70
5	4.44 ± 1.00	4.60 ± 1.17
6	8.40 ± 1.27	8.70 ± 1.70
7	19.33 ± 2.08	19.80 ± 2.10
8	35.75 ± 0.95	42.87 ± 2.20

real-valued weight space with no considerable differences up to dimension $n = 7$. When dimension n becomes 8, the 16-bit integer weight space gives worse results compared to fewer dimensions. This trend means that more resolution, which might be provided by a 24-bit or 32-bit integer weight space, is needed. It is difficult to compare the real-valued and integer-valued weight spaces in terms of the computation times required since the PA runs by a predetermined number of epochs instead of stopping criteria based on an error measure. The difference between the computation times is thus obtained as a consequence of the computation time required per epoch in 2 different weight spaces. Based on the observations of the evolution of successive errors in a set of realized simulation experiments, i.e. based on a trial-and-error approach, we set the number of training epochs for the integer weight space as 50% greater than the number used for the real-valued space. Therefore, the computation time needed for the integer weight space can be said to be 50% greater than that for the real-valued weight space case since the computation times needed per epoch are similar for both cases.

4.4.3. Two-spirals function

The proposed algorithm is also run for the 2-spirals function in a discrete weight space. The average network size obtained over 6 trials is 60.17 ± 3.81 with a minimum of 57 and maximum of 67. The weight space is taken as that of 16-bit fixed-point real values with a 12-bit fraction length.

4.5. Generalization performance

While a set of noise-free random Boolean functions, parity functions, and 2-spirals data sets are applied in testing the algorithm developed, i.e. "this is the standard situation for most of the existing constructive algorithms as many of them tries to achieve zero training error[21]", the generalization performance of the proposed model is also investigated. Test sets are constructed from training sets by adding uniformly distributed noise (2-spirals and random Boolean functions) or reserving some data for the test set and training the network with the remaining data (2-spirals).

As seen from the Table 8, generalization performance decreases when the complexity of the problem increases. The main reason for the decrease in the generalization performance is that more complex problems require more neurons in the resulting net. Since the proposed model is a cascaded structure, the errors in the lower levels of cascade network propagate to the output.

5. Hardware implementation

In this section, a hardware implementation of the proposed model is presented for comparison to the classical perceptron model in terms of implementation resource usage. For the implementation, the FPGA, as reprogrammable digital ICs, is selected since the usage of the FPGA for neural network implementation provides flexibility of programmable systems and has an increasing trend in the neural network literature [10, 11, 12, 13].

Table 8. Generalization performance of the proposed model (%).

Problem	Generalization performance
Two-spirals problem	67.01 ± 7.30
8-bit random Boolean functions	72.15 ± 3.82
7-bit random Boolean functions	77.34 ± 3.81
6-bit random Boolean functions	78.13 ± 3.35
5-bit random Boolean functions	80.00 ± 5.63
4-bit random Boolean functions	86.25 ± 6.73

FPGAs are ICs containing programmable logic components and programmable interconnects that includes tens of thousands up to a few millions gates, dedicated memory blocks, multiplier circuits, PLLs, etc. Moreover, new models of FPGA chips are produced with microprocessors embedded in them. The programmability of reconfigurable FPGAs yields the availability of fast, special-purpose hardware for wide applications, and so FPGA-based NNs are becoming a new focus of NN research[16]. The hardware implementation of NNs is superior to software implementations because FPGA supports the advantage of the parallel processing that is the key feature of NN structures.

Many powerful design, programming, synthesis, and simulation tools have been provided by FPGA manufacturers and software development companies. Along with the reprogramming capability of chips, FPGAs bring a short design cycle and reduced design and development phases.

5.1. System architecture

FPGA implementation of NNs could be done in 2 main architectures. The first is fully parallel architecture, such that the number of multipliers and the number of full adders per neuron are equal to the number of inputs of the neuron. The main advantage of this architecture is the capability of the realization of the parallelism of the NNs, thus providing rapidness. However, this approach causes an increase in the usage of the resources in the FPGA and may lead to select bigger and more expensive chip models, especially for huge NNs.

The other architecture saves resources but works a bit slower. In this architecture, only 1 multiplier and 1 accumulator are used per neuron. At each step, 1 input is multiplied by the corresponding weight and added to the accumulator. The calculation of the output is done in n steps, where n is the number of inputs of the neuron. Please note that the second architecture still has the advantages of parallel processing in the network level, i.e. all neurons have their own multipliers/adders and work in parallel in the neural network.

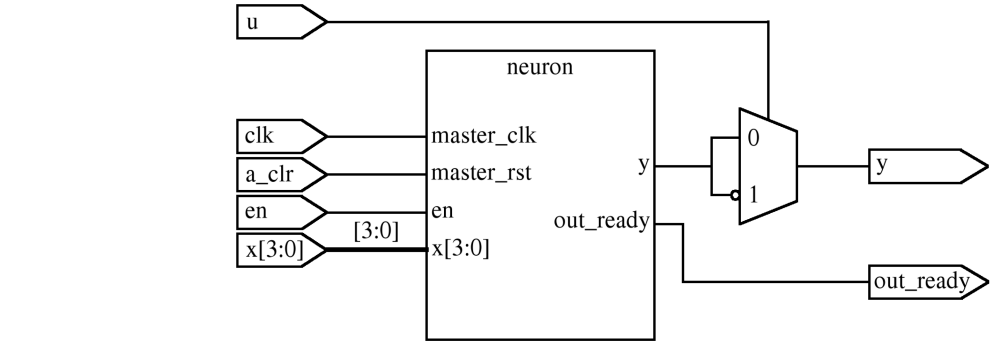
To implement neural networks in the FPGA, the input data and the weights of the neurons should be represented in an efficient way in the hardware. Data should be digital since the FPGA is a digital hardware. A decision should be made in the design phase of the FPGA about the precision (number of bits) and the number format (signed/unsigned integer, floating/fixed-point real value, etc.). Here it is very obvious that higher data precision (the number of bits in the representation) requires large resource usage in the FPGA, providing a smaller quantization error for output.

5.2. Implementation of 4-bit parity problem

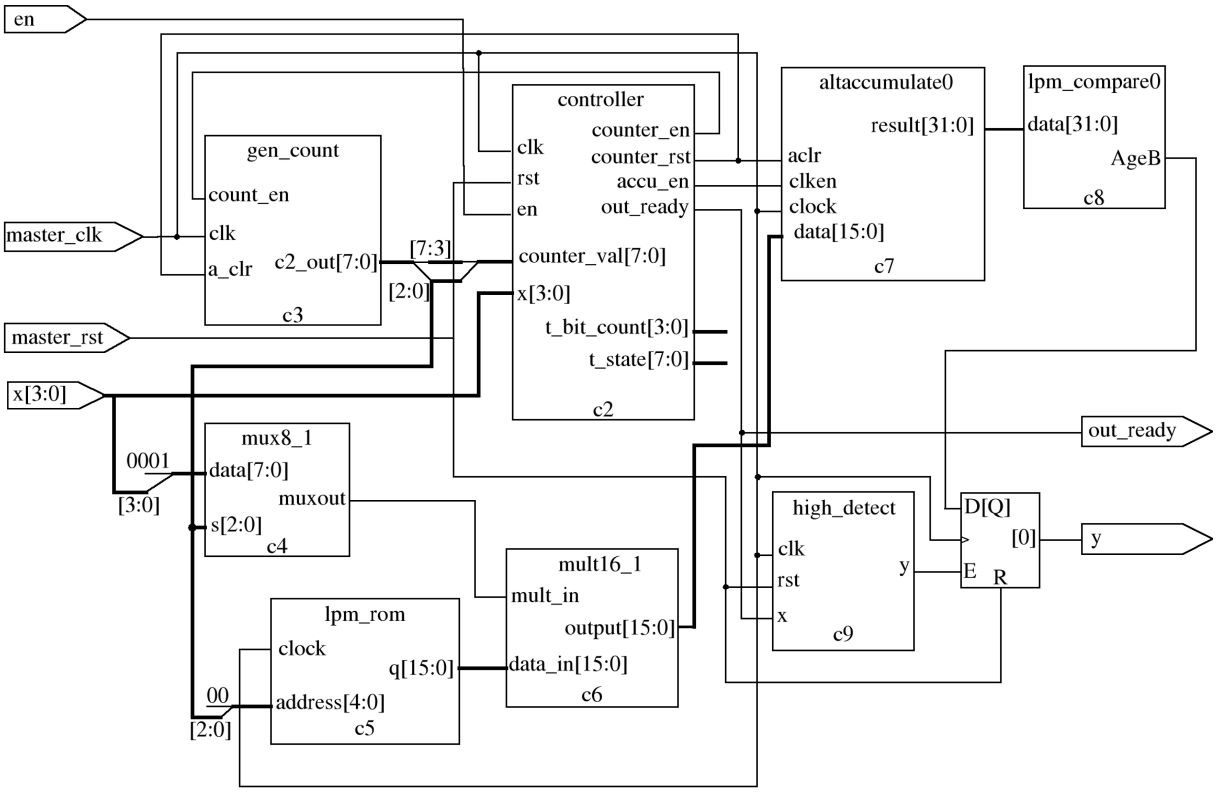
It is shown in Section 4.1 that only n neurons are enough to solve the n-bit parity problem using the proposed model and the algorithm. The algorithm results in a 4-neuron network and provides a set of weights for these neurons.

As explained in Section 5.1, data representation is an important aspect. Although the input values are binary for the parity case, i.e. we can represent them by 1 bit only, the weights obtained by the algorithm have real values with double precision. To prevent misclassification errors for the FPGA implementation of the network, the number format should be determined suitably. Most of the FPGA design tools and chips support IEEE double-precision floating-point number format, but in this format the implementation will be hard to design and realize.

The algorithm for the example in Section 4.4.1 gives a set of weight vectors in 16-bit signed integer number format, where the implementation of the arithmetic operations in integer number format is much easier than in the floating-point format.



(a) The proposed model, where the neuron block stands for the discrete perceptron.



(b) The internal structure of the 'neuron' block in subfigure (a).

Figure 5. FPGA implementation schematics of the proposed model and the internal structure of the discrete perceptron part of it.

In this work, the model is implemented with 1 multiplier and 1 accumulator per neuron. The inputs enter the neuron in parallel and are multiplied in serial by their corresponding weights stored in a ROM. The results of multiplications are saved in an accumulator. There is a comparator at the output, which functions as a hard-limiter. A control unit is added to the neuron design in order to organize the data flow, timing, and serial processes. Digital system architecture is presented using the Very High Speed Integrated Circuits Hardware Description Language (VHDL). The schematic of the implementation is depicted in Figure 5. Figure 5a shows the top-level schematics of the proposed model and Figure 5b depicts the detailed schematics for an implementation of the neuron block from Figure 5a. The design and implementation of the model is verified by using an FPGA simulation tool[27].

5.2.1. Comparison

The FPGA implementation of the proposed NN model for the 4-bit parity problem is seen in Figure 5. The proposed model's neuron is the same as the perceptron's neuron, except for the dual output of the neuron and the multiplexer, thus bringing insignificant increase in the complexity and in the resource usage. A comparison in terms of resource usage between the perceptron neuron and the proposed model neuron is given in Table 9. Here it is seen that resource differences are due to the added multiplexer.

Table 9. The resource usage of neurons in FPGA implementation.

	Perceptron neuron	Proposed neuron
ALUTs*	99	100
Registers	63	63
Total pins	9	10
Memory bits	128	128
Max clock freq**	213.45 MHz	207.21 MHz

* ALUT, the Adaptive Look-Up Table, is the cell in the FPGA chip that is used as the output of logic synthesis. A single ALUT contains a register and a combinational pair.

** The max clock frequency values are obtained without any optimization or any constraint set.

6. Conclusion

A new cascaded NN model and a learning algorithm associated with it were proposed for linearly nonseparable classification problems and it was proven in this paper that the algorithm is convergent. Any given function from the finite subset of R^n to the set of $\{0, 1\}$ can be realized by the model in a finite number of steps, resulting in a network of a finite number of neurons. Because of the nature of the algorithm, the minimum network size is not always guaranteed.

The algorithm can search for weights that give a minimal error under a prespecified constraint, namely semicorrect separation. The algorithm is open to the addition of extra constraints, too, and so providing learning in discrete weight space with a finite number of weights is achieved here by considering the discreteness of the weights as a constraint.

The proposed modification of the discrete perceptron brings universality with the expense of getting just a slight complication in hardware implementation. By comparing FPGA implementations of the model and the classical discrete perceptron, it is shown that the increase in the FPGA resource usage is only about 0.3%.

Acknowledgments

The authors are pleased to acknowledge the helpful and insightful comments of the reviewer and the associate editor.

References

[1] J. Anlauf, M. Biehl, "The AdaTron: An adaptive perceptron algorithm", Europhysics Letters, Vol. 10, pp. 687–692, 1989.

[2] V. P. Roychowdhury, K. Y. Siu, T. Kailath, "Classification of linearly nonseparable patterns by linear threshold elements", IEEE Trans. on Neural Networks, Vol. 6, pp. 318–331, 1995.

[3] M. Muselli, "On convergence properties of pocket algorithm", IEEE Trans. on Neural Networks, Vol. 8, pp. 623–629, 1997.

[4] M. Marchand, M. Golea, P. Ruján, "A convergence theorem for sequential learning in two-layer perceptrons", Europhys. Lett., Vol. 11, pp. 487–492, 1990.

[5] S. Young, T. Downs, "CARVE—a constructive algorithm for real-valued examples", IEEE Trans. on Neural Networks, Vol. 9, pp. 1180–1190, 1998.

[6] G. Martinelli, F. M. Mascioli, G. Bei, "Cascade neural network for binary mapping", IEEE Trans. on Neural Networks, Vol. 4, pp. 148–150, 1993.

[7] G. Martinelli, F. M. Mascioli, "Cascade perceptron", Electronics Letters, Vol. 28, pp. 947–949, 1992.

[8] İ. Genç, C. Güzeliş, "Discrete perceptron with input dependent threshold value", in Conference on Signal Processing and its Appl., Vol. 1, pp. 36–41, Ankara–Turkey, (In Turkish), 1998.

[9] M. do Carmo Nicoletti, J. R. Bertin, L. Franco, J. M. Jerez, "Constructive neural network algorithms for feedforward architectures suitable for classification tasks", in Constructive Neural Networks, editors L. Fanco, D. A. Elizondo, J. M. Jeres, pp. 1–23, Berlin, Springer-Verlag, 2009.

[10] W. Qinruo, Y. Bo, X. Yun, L. Bingru, "The hardware structure design of perceptron with FPGA implementation", in IEEE Int. Conf on Systems, Man and Cybernetics, Vol. 1, pp. 762–767, 2003.

[11] S. Şahin, Y. Becerikli, S. Yazıcı, "Learning internal representations by error propagation", in Neural Information Processing: 13th Int. Conf. ICONIP'06, editors I. King, J. Wang, L.-W. Chan, D. Wang, pp. 1105 – 1112, Springer-Verlag, 2006.

[12] D. Ferrer, R. Gonzàlez, R. Fleitas, J. P. Acle, R. Canetti, "NeuroFPGA – Implementing artificial neural networks on programmable logic devices", in Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Designersğ Forum (DATEğ04), 2004.

[13] Y. Maeda, M. Wakamura, "Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation", IEEE Trans. on Neural Networks, Vol. 16, pp. 1664–1672, 2005.

[14] O. Cadenas, G. Megson, D. Jones, "A new organization for a perceptron-based branch predictor and its FPGA implementation", in Proceedings of the IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design, 2005.

[15] S. Vitabile, V. Conti, F. Gennaro, F. Sorbello, "Efficient MLP digital implementation on FPGA", in Proceedings of the 2005 8th Euromicro conference on Digital System Design (DSDğ05), 2005.

[16] J. Liu, D. Liang, "A survey of FPGA-based hardware implementation of ANNs", in Proceedings of ICNN&B International Conference on Neural Networks and Brain, pp. 915–918, 2005.

[17] İ. C. Göknar, M. Yıldız, S. Minaei, E. Deniz, "Neural CMOS-integrated circuit and its application to data classification", IEEE Trans. on Neural Networks and Learning Systems, Vol. 23, pp. 717–724, 2012.

[18] D. A. Elizondo, J. O. de Lazcano-Lobato, R. Birkenhead, "Choice effect of linear separability testing methods on constructive neural network algorithms: An empirical study", Expert Systems with Applications, Vol. 38, pp. 2330–2346, 2011.

[19] M. Rosen-Zvi, I. Kanter, "Training a perceptron in a discrete weight space", Physical Review E, Vol. 64, pp. 046109–1–9, 2001.

[20] M. Frean, "A "thermal" perceptron learning rule", Neural Computation, Vol. 4, pp. 946–957, 1992.

[21] J. L. Subirats, L. Franco, J. M. Jerez, "C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons", Neural Networks, Vol. 26, pp. 130–140, 2012.

[22] M. Mézard, J.-P. Nadal, "Learning in feedforward layered networks: The tiling algorithm", J. Phys. A: Math. Gen., Vol. 22, pp. 2191–2203, 1989.

[23] P. Rujan, M. Marchand, "Learning by minimizing resources in neural networks", Complex Syst., Vol. 3, pp. 229–241, 1989.

[24] D. Rumelhart, G. Hinton, R. Williams, "Learning internal representations by error propagation", in Parallel Distributer Processing: Exploration in the Microstructure of Cognition, editors D. Rumelhart, J. McClelland, chap. 3, 1986.

[25] A. Wieland, "Two spirals", Posted to 'connectionists' mailing list, http://www.ibiblio.org/pub/academic/computer-science/neural-n etworks/programs/bench/two-spirals", current as of June 2012.

[26] S. Fahlman, C. Lebiere, "The cascade-correlation learning architecture", in Advances in Neural Inform. Processing Syst. (NIPS), editor D. Touretzky, Vol. 2, pp. 524–532, San Mateo, CA, 1989.

[27] Altera Corp., Quartus II Development Software Handbook, Vol. 1–5, California, Altera, 2007.