Paper

A world according to artificial neural networks

Alfons Schuster

Abstract — This paper presents results from a preliminary study in the field of artificial neural networks (ANN). The overall aim of our work relates to the field of cognitive science. In this wider framework we try to investigate, reason about, and model cognitive processes in order to obtain a better understanding of the major processing device involved – the human brain. In terms of content this paper presents a novel ANN learning approach. Note that throughout the paper we assume supervised learning. In contrast to the classical ANN learning approach where an ANN algorithm alters an initial random weight assignment until a reasonable solution to a problem is obtained this approach does not alter the initial random weight assignment at all, but provides a solution to the problem by transforming the actual input data. The approach is applied to perceptrons and adalines and its quality is demonstrated on simple classification problems.

Keywords — artificial neural networks, cognitive science, input space transformation.

1. Introduction

A few examples are chosen in order to provide a quick introduction and also to illustrate the motivation behind this work.

Example 1. Imagine a student looking for material supporting an assignment. The student probably browses several books in a library and maybe selects a few of them for more detailed study. Important in this example is (a) the information required by the student is external, in library books etc., (b) the information is represented in different formats, i.e. different books may cover the same topics but the styles etc. may be different in each book, (c) after a while the student may have forgotten some of the studied material, i.e. the information may no longer be available in the brain of the student. In order to re-acquire the information the student may need to go back to the library. On the other hand some of the information may not be considered (studied) on purpose. For example, in case the information is false or obsolete.

Example 2. Imagine a person having a dog and also having a picture of the same dog. Further imagine a second person pointing to the real dog asking the question: *Who's dog is this?* Then the dog owner will answer: *This is my dog*. Imagine now the second person pointing to the picture of the dog asking the same question: *Who's dog is this?* Then the person will answer again: *This is my dog*. The interesting aspects here are (a) there are two completely different representations of the dog, namely the dog itself and the picture of the dog, (b) for the purpose of answering the question however both representations are sufficient, (c) the two representations are external to the processing device – the brain of the dog owner, and (d) the original data, the physical dog, has been transformed into a different representation, the picture.

Example 3. Philosophers have been thinking about similar problems in the past. An example involving the philosopher Immanuel Kant can be summarised as follows [1]. Imagine a person wearing green glasses since the day of his/her birth. The person then would see the world in a shade of green. This may not necessarily have a major influence on the person's life. If the person ever took of the glasses then the person would see the world differently. The points here are (a) how can we be sure to experience the world with our senses as the world really is, i.e. how can we be sure we are not wearing some sort of glasses? and (b) the data transformation (here a green shift) may not necessarily have an impact on our ability to do things or on aspects of our reasoning. For example, the ability to swim is independent of the color of the water.

Example 4. This example particularly focuses on artificial neural networks. ANN learning usually starts with a random weight assignment followed by a training process using a training set [2]. The training process alters the initial random weight configuration and establishes a final weight configuration. This final configuration remains static and specifies the ANN for a particular application scenario. From the viewpoint of ANNs as being a model of the human brain this approach presents a number of questions. For example, let N be the total number of neurons in the brain. Further, imagine a simple classification scenario that can be solved by the utilisation of an ANN. Let n be the total number of neurons in this particular ANN. Now imagine this ANN presenting a part or region in the brain. Since the weight configuration of this ANN remains fixed or static after the training process this part of the brain (the ANN) can only be applied for the particular task it is designed for. This however could lead to the possible conclusion that the number of neurons in the brain that remains available for other tasks is now reduced to $N-n$. Although the number of neurons in the brain is quite substantial this means that sooner or later the brain runs out of neurons. This example may be a bit naive, but it helps to illustrate a major difference between the brain and ANNs, namely the high dynamic and flexibility of the brain opposed to the static inflexibility of ANNs.

Example 5. This is not really an example but rather a consideration. If there are different representations of an entity then a learning device such as an ANN, for example, needs

to be constructed for every representation. The previous example mentioned that this may require a separate set of neurons for the processing of each representation. From an energy point of view all these processes require energy. Nature usually looks for a maximum in efficiency through a minimum of effort (energy). The processing of each representation individually may contradict this particular drive. A summary of these examples may be:

- The same information can be represented in different formats, shapes, representations.
- The different representations of the information are usually external to a learner or learning device (student, brain, ANN).
- Some information may be false or obsolete and therefore may not be needed to be acquired (studied, learned).
- Some of the information learned may get lost or forgotten.
- In case there are different representations for the same information then it is ineffective for a learning device in terms of energy to construct a model for each representation.

Based on these examples and observations we here consider the following approach:

- Instead of producing a model for the learning of different representations of the same information use a single model that is able to learn the different representations.

Very generally speaking this paper has a focus on this issue and investigates a new approach to this problem. The major difference of this approach lies in the fact that instead of modifying the weight configuration of an ANN until the weight configuration suits the problem the approach proposed here transforms the actual data representing the problem until the data fits the initial configuration of the learning device (ANN). The initial weigh configuration assigned to the ANN at the start of the training process remains unchanged throughout this process. The transformed data can be viewed as a different representation of the problem and the examples given before indicate that it is possible to arrive at meaningful conclusions using different representations of the same information (dog/picture). One of the interesting consequences of this approach is that a single, random weight configuration can be used for many different scenarios.

The reminder of the paper is organised as follows. Section 2 presents the basic idea behind this paper, namely ANN training algorithms that are based on the transformation of input data rather than on the modification of weight values. Section 3 introduces two such algorithms. Section 4 summarises the results obtained from an application of the approach on simple classification tasks. Section 5 ends the paper with a summary.

2. Classification through data input space transformation

This section outlines the basic idea behind this paper. Although the paper deals with two well-known ANN training algorithms, namely the perceptron training algorithm and the Adaline training algorithm this section only refers to the perceptron training algorithm [3, 4]. This is basically due to the fact that the situation for both algorithms is very similar. The paper uses a simple classification task as a run-through example for illustration purposes. The task is illustrated in Fig. 1 and involves the correct classification of a particular number of different objects into one of two classes, *Class 1* or *Class 2*.

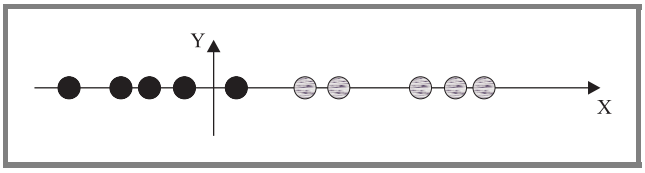

Fig. 1. A simple, one-dimensional, linearly separable classification task.

Imagine, for example, that Fig. 1 illustrates an arbitrary real valued *x-y* co-ordinate system in which the classification scenario takes place. Let the black dots in Fig. 1 represent objects of *Class 1* and the lined circles objects of *Class 2*. From the viewpoint of a classification task Fig. 1 illustrates a simple, one-dimensional, linearly separable task. Such a task can be solved by a perceptron, for example.

2.1. Perceptron classification

At this stage it is not necessary to know the exact details of a perceptron, the details are going to be explained at a later stage. The motivation in this section is to establish an understanding for the basic strategy behind perceptron classification. The perceptron training algorithm starts with a random weight assignment to the perceptron. In the context of the one-dimensional task at hand such a random weight assignment represents an arbitrary division point on the *x*-axes. For example, let the diamond in Fig. 2a be such an initial, random division point.

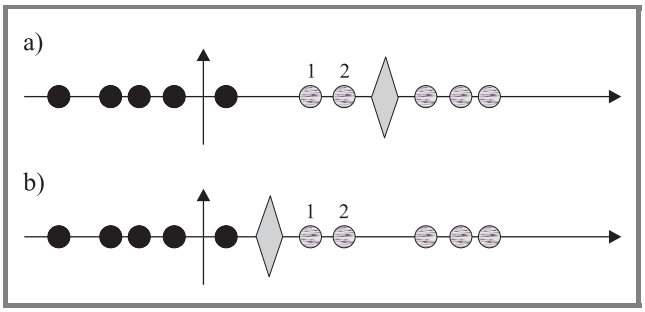

Fig. 2. (a) Possible start of perceptron learning process; (b) possible end of perceptron learning process.

Note that the diamond in Fig. 2a does not separate all objects into their correct classes. The two objects indexed 1 and 2 in Fig. 2a are misclassified. In order to achieve the correct classification of all objects the perceptron training algorithm alters the initial weight assignment in a number of successive, defined steps. This process usually continues until either all objects are classified correctly, or until a predefined number of iterations is reached. In illustrative terms the process of continuously altering the weight configuration of the perceptron is equivalent to an organised movement of the diamond along the *x*-axis in Fig. 2a. For example, after the training process the diamond might end up in the position illustrated in Fig. 2b. This position actually represents a solution to the problem, because all samples of *Class 1* are now on one side of the diamond and all objects of *Class 2* are on the other side. The actual details of the perceptron training algorithm are not so important at the moment, they will be discussed later. At the moment this study is interested in the question whether there is an alternative solution to the classification task given in Fig. 1.

2.2. An alternative solution to the classification problem

Figure 3 indicates an alternative solution to the problem at hand. In this particular case this alternative solution shall be referred to as alternative classification algorithm, or simply ACA.

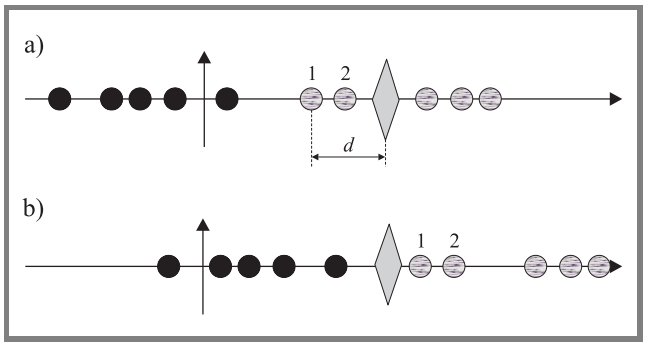

Fig. 3. (a) Original classification scenario; (b) alternative solution to the classification task through a transformation of the actual input data.

Figure 3 illustrates that the ACA is basically an input data transformation process. The ACA also starts with an initial random weight assignment. But then, instead of altering the weight values, which would be equivalent to moving the initial diamond around, the ACA transforms or shifts the actual input data in the *x*-*y* co-ordinate system until the initial location of the diamond presents a solution to the classification task. For example, Fig. 3a represents the original scenario. Note again that the diamond in Fig. 3a does not separate all objects into their correct classes. Figure 3b illustrates a possible scenario after the ACA has finished a training session. Figure 3b indicates that the location of the diamond remains unchanged throughout this training session, but also that the input data has been transformed in such a way that the initial diamond now presents a solution to the classification task. The diamond now separates all objects of *Class 1* from objects of *Class 2*. This transformation of input data is the basic principle behind the algorithms presented in the forthcoming sections. Figure 3 provides one more piece of information. In order to come up with the correct classification of all objects the input data has to be transformed by a certain amount. The index d in the figure indicates that the magnitude (m) of this transformation or shift has to be $m \geq d$.

3. Modified algorithms for perceptron and Adaline

This section presents two new ANN training algorithms. The two algorithms are modifications of the well-known perceptron and Adaline training algorithm and therefore are referred to as modified perceptron algorithm (MPA) and modified Adaline algorithm (MAA). The characteristic feature of the MPA and the MAA is the implementation of the ideas presented throughout the previous sections, that is, classification achieved through the transformation of the data input space as opposed to the update of weight values.

3.1. Perceptron and modified perceptron training algorithm

The MPA algorithm is a derived modification of the perceptron training algorithm. The section therefore starts with a brief introduction to perceptrons. Figure 4 illustrates a typical perceptron.

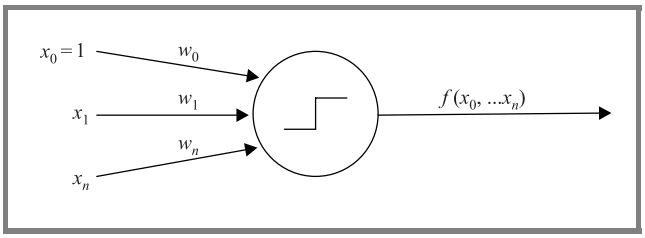

Fig. 4. A typical perceptron.

The perceptron in Fig. 4 has one output, and an undefined number of n inputs $(x_0, \ldots, x_n)$. The dummy input x_0, often called bias, has a constant value of one. Every input has a weight value $(w_0, \ldots, w_n)$ associated with it. The input values and the weight values are usually dealt with as vectors (e.g., $\boldsymbol{w}$ and $\boldsymbol{i}$). The output uses the step function $f(x_0, \ldots, x_n)$ in order to determine into which of two available classes an object belongs. The step function is applied to the weighted sum of the inputs to the perceptron and is often defined as follows:

$$f(x_0, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}, \qquad (1)$$

where x_i and w_i are components of the input vector $\boldsymbol{i}$ and the weight vector $\boldsymbol{w}$, and the weighted sum $\sum_{i=0}^{n} w_i x_i$ is

given by the scalar product between the input vector and the weight vector as: $\boldsymbol{w}\cdot\boldsymbol{i} = w_0x_0 + \ldots + w_nx_n$. Note that solving the one-dimensional, linear separable classification task illustrated in Fig. 1 only requires a single perceptron with two inputs, x_0 and x_1.

In order to explain the perceptron training algorithm we make the following assumptions. For an object of *Class 1* the desired perceptron output shall be 1, and for an object of *Class 2* the desired output shall be -1. If an object is classified correctly then the perceptron remains unchanged. If the desired output is different from the actual output generated by the perceptron then the weight vector needs to be changed such that the error reduces. Theoretically this process is repeated until the desired output and the generated output are the same. In reality however linearly separable problems are not the norm and so the process usually runs for a predefined number of iterations.

Whenever an input vector is presented to a perceptron for classification two types of error can occur.

Case 1. The input vector $\boldsymbol{i}$ belongs to *Class 1* for which the desired perceptron output is 1, but $\boldsymbol{w}\cdot\boldsymbol{i} \leq 0$ (the actual perceptron output is -1).

Case 2. The input vector $\boldsymbol{i}$ belongs to *Class 2* for which the desired perceptron output is -1, but $\boldsymbol{w}\cdot\boldsymbol{i} > 0$ (the actual perceptron output is 1).

A perceptron strives to overcome both types of error through a defined weight vector update. This update establishes a new weight vector $\boldsymbol{w'}$ from a previous weight vector $\boldsymbol{w}$ according to: $\boldsymbol{w'} = \boldsymbol{w} + \Delta\boldsymbol{w}$. In the first case the update has to achieve that $\boldsymbol{w'}\cdot\boldsymbol{i} = (\boldsymbol{w} + \Delta\boldsymbol{w})\boldsymbol{i} > \boldsymbol{w}\cdot\boldsymbol{i}$. In the second case the aim is $\boldsymbol{w'}\cdot\boldsymbol{i} = (\boldsymbol{w} + \Delta\boldsymbol{w})\boldsymbol{i} < \boldsymbol{w}\cdot\boldsymbol{i}$. This behaviour can be achieved by letting $\Delta\boldsymbol{w} = \pm\eta\boldsymbol{i}$, where η is a positive constant called the learning rate. These concepts established define a weight update in the first case as:

$$\boldsymbol{w'}\cdot\boldsymbol{i} = (\boldsymbol{w} + \Delta\boldsymbol{w})\boldsymbol{i} = (\boldsymbol{w} + \eta\boldsymbol{i})\boldsymbol{i} = (\boldsymbol{w}\cdot\boldsymbol{i} + \eta\boldsymbol{i}\cdot\boldsymbol{i}) > \boldsymbol{w}\cdot\boldsymbol{i}. \quad (2)$$

Note in particular that η is a positive constant and that the scalar product $\boldsymbol{i}\cdot\boldsymbol{i} > 0$. In the second case the update looks like:

$$\boldsymbol{w'}\cdot\boldsymbol{i} = (\boldsymbol{w} + \Delta\boldsymbol{w})\boldsymbol{i} = (\boldsymbol{w} - \eta\boldsymbol{i})\boldsymbol{i} = (\boldsymbol{w}\cdot\boldsymbol{i} - \eta\boldsymbol{i}\cdot\boldsymbol{i}) < \boldsymbol{w}\cdot\boldsymbol{i}. \quad (3)$$

Equations 2 and 3 basically represent the core of the perceptron training algorithm. The task now is to design the MPA, the modified perceptron learning algorithm.

Modified perceptron training algorithm. Remember that the initial weight vector remains unchanged in the MPA training process. The MPA updates the input vector $\boldsymbol{i}$ instead. The MPA input vector update is defined as $\boldsymbol{i'} = \boldsymbol{i} + \Delta\boldsymbol{i}$ and so looks quite similar to a weight vector update described before.

The MPA uses this input vector update in order to deal with the two possible error scenarios mentioned before. For example, in the first case the input vector $\boldsymbol{i}$ belongs to *Class 1* for which the desired perceptron output is 1, but $\boldsymbol{w}\cdot\boldsymbol{i} \leq 0$ (the actual perceptron output is -1). In this situation the input vector is updated by the MPA such that $\boldsymbol{w}\cdot\boldsymbol{i'} = \boldsymbol{w}(\boldsymbol{i} + \Delta\boldsymbol{i}) > \boldsymbol{w}\cdot\boldsymbol{i}$. In the second case the input vector $\boldsymbol{i}$ belongs to *Class 2* for which the desired perceptron output is -1, but $\boldsymbol{w}\cdot\boldsymbol{i} > 0$ (the actual perceptron output is 1). Here the input vector is updated such that $\boldsymbol{w}\cdot\boldsymbol{i'} = \boldsymbol{w}(\boldsymbol{i} + \Delta\boldsymbol{i}) < \boldsymbol{w}\cdot\boldsymbol{i}$. With $\Delta\boldsymbol{i} = \pm\eta\boldsymbol{w}$ and the learning rate η it is possible to formulate an MPA update for the first case as follows:

$$\boldsymbol{w}\cdot\boldsymbol{i'} = \boldsymbol{w}(\boldsymbol{i} + \Delta\boldsymbol{i}) = \boldsymbol{w}(\boldsymbol{i} + \eta\boldsymbol{w}) = (\boldsymbol{w}\cdot\boldsymbol{i} + \eta\boldsymbol{w}\cdot\boldsymbol{w}) > \boldsymbol{w}\cdot\boldsymbol{i}. \quad (4)$$

Note, η is a positive constant and the scalar product $\boldsymbol{w}\cdot\boldsymbol{w} > 0$. In the second case the update appears as:

$$\boldsymbol{w}\cdot\boldsymbol{i'} = \boldsymbol{w}(\boldsymbol{i} + \Delta\boldsymbol{i}) = \boldsymbol{w}(\boldsymbol{i} - \eta\boldsymbol{w}) = (\boldsymbol{w}\cdot\boldsymbol{i} - \eta\boldsymbol{w}\cdot\boldsymbol{w}) < \boldsymbol{w}\cdot\boldsymbol{i}. \quad (5)$$

Although Eqs. 4 and 5 capture the essence of the MPA Fig. 5 may provide additional transparency to the whole process.

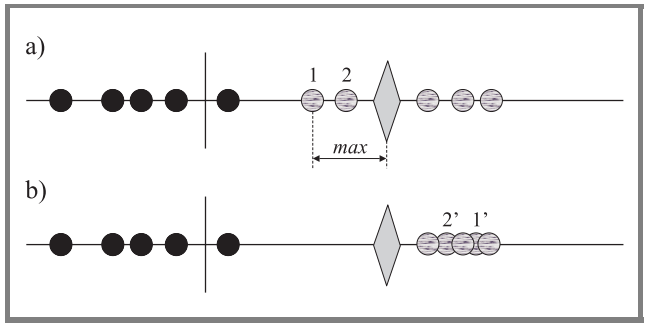

Fig. 5. A possible scenario for the MPA algorithm: (a) at the start; (b) at the end.

Figure 5a illustrates the familiar initial scenario with a random, but now static, weight assignment including the two misclassified objects carrying the labels 1 and 2. The MPA works similar to the perceptron training algorithm. If an object is classified correctly then the MPA does not interact. If however the desired output for an object is different from the actual output generated by the perceptron then the MPA alters the corresponding input vector until (in case of linear separability) the current object is classified correctly. Figure 5b aims to capture this process and illustrates that only the position of the two initially incorrect classified objects has changed. Figure 5b also indicates that the order of the position of such objects may change during the transformation process. For example, the new positions of objects 1 and 2 are now at 1' and 2'.

Further, the individual changes in the input vector of those objects that are actually transformed are recorded by the MPA. The MPA extracts the value *max* from this information (see Fig 5a). The value *max* determines the magnitude by which all objects need to be transformed along the *x*-axis in order to use the perceptron as a meaningful classifier. The value *max* is particularly important for the classification of unknown objects, that is objects that have

not been included in the training session. For such objects it is necessary to add an offset of magnitude *max* to their input vector.

Since results of an application of the MPA are presented and discussed at a later stage we here leave the perceptron training algorithm and its modified alternative and proceed with the investigation of the Adaline training algorithm.

3.2. Adaline and modified Adaline training algorithm

An Adaline is quite similar to a perceptron. For example, an Adaline also uses a step function in order to determine the class membership for different objects. For this reason and because both, perceptrons and Adalines, are well documented in the literature this section only elaborates on those issues that are necessary for the understanding of Adalines and the presented here alternative the modified Adaline training algorithm.

An Adaline is a system that like a perceptron aims for a reduction of the number of misclassifications through a defined weight update. The difference to the perceptron training algorithm is that the Adaline aims to achieve this task by minimising the mean square error $E = (d_j - net_j)^2$ of the system through the application of a gradient descent method, where d_j is the desired output for a particular input vector i_j, and net_j is the weighted sum generated by this input vector and a weight vector. Figure 6 illustrates a typical Adaline.

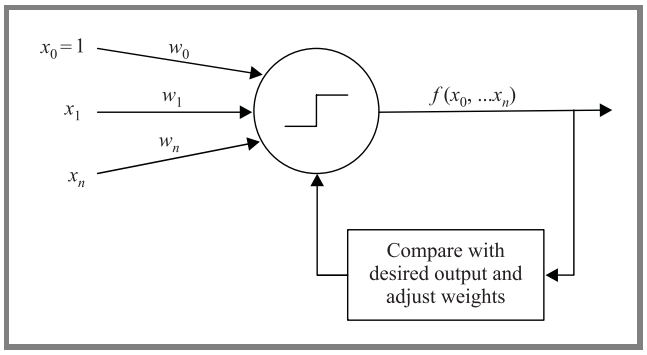

Fig. 6. An typical Adaline at work.

The basic formulas responsible for the learning process in an Adaline are given by Eqs. 6, 7, and 8. These formulas basically indicate how an updated weight value $\boldsymbol{w}$**'** is generated from a previous weight value $\boldsymbol{w}$:

$$\boldsymbol{w}' = \boldsymbol{w} + \Delta\boldsymbol{w}, \tag{6}$$

where

$$\Delta\boldsymbol{w} = \eta(d_j - net_j)i_j, \tag{7}$$

where η is the positive learning rate again, d_j is the desired output for a particular input vector i_j, and net_j is the weighted sum $\sum_{i=0}^{n} w_i x_{i,j}$ generated by the scalar product between the input vector i_j and the current weight vector $\boldsymbol{w}$. From Eqs. 6 and 7 the updated weight vector $\boldsymbol{w}$**'** is produced by:

$$\boldsymbol{w}' = \boldsymbol{w} + \eta\left(d_j - \sum_{i=0}^{n} w_i x_{i,j}\right) i_j. \tag{8}$$

A final difference between the perceptron and the Adaline needs to be mentioned. The perceptron training algorithm as well as its modification the MPA updates a weight vector/input vector only if the output generated by the perceptron is different from the desired output. On the other hand, the Adaline training algorithm and its modified alternative apply an update at every presentation of a sample to the system.

Modified Adaline training algorithm. The task now is similar as before and contains modifying Eq. 8 in such a way that instead of the weights the actual input data is transformed, without loosing the quality of the system as a classifier. The three equations below provide a summary of the mathematical procedures involved. They are very similar to Eqs. 6, 7, and 8 and form the basis for the modified Adaline algorithm MAA:

$$\boldsymbol{i}' = \boldsymbol{i} + \Delta\boldsymbol{i}, \tag{9}$$

$$\Delta\boldsymbol{i} = \eta(d_j - net_j)w_j, \tag{10}$$

$$\boldsymbol{i}' = \boldsymbol{i} + \eta\left(d_j - \sum_{i=0}^{n} w_i x_{i,j}\right) w_j, \tag{11}$$

where $\boldsymbol{i}$**'** is the resulting, updated input vector. A closer look at the different equations reveals again that the main difference is basically a substitution between the weight vector $\boldsymbol{w}$ and the input vector $\boldsymbol{i}$. So much for the mathematical background of the different algorithms. The next section provides a summary of the results obtained from an application of the different algorithms.

4. Results

All algorithms (the original perceptron and Adaline training algorithm as well as their modifications MPA and MAA) were applied and evaluated on simple classification tasks. The problem to solve was always a one-dimensional, linearly separable classification problem, similar to the problem illustrated in Fig. 1. The total number of objects involved in a classification task was variable, but every class contained the same number of objects. The *x*-axis in Fig. 1 was given by the interval [0, 1]. The position of the objects in this interval was generated by the random number procedure included in the Delphi5 programming tool that was utilised for the programming of the algorithms. The position of the initial division point (the diamond in Fig. 1) was variable and so it was possible to generate a particular number of misclassified objects.

Very generally speaking it can be said that the modified algorithms MPA and MAA performed more or less equally well as their original counterparts perceptron and Adaline did. Since the problems posed did not contain any outliers MPA and MAA were always able to solve the classification tasks given to them, provided the number of iterations that was sufficiently high. For a particular task the number of iterations needed by MPA and MAA was about in the same range as the number of iterations needed by the perceptron and Adaline algorithm. In case of outliers MPA and MAA face similar problems as the original perceptron and Adaline algorithm do. The similarity of the approaches may allow the assumption that approaches to tackle the issue of outliers in MPA and MAA are similar to approaches proposed and known for perceptrons and Adalines [2].
The overall conclusion for the undertaken study is that all the different algorithms did perform about equally well on the problems given to them, and so MPA and MAA do not stand back behind the traditional perceptron and Adaline algorithms.

5. Summary and future work

The paper presents two new classification algorithms. To some extent these algorithms are derived from the classical perceptron and Adaline training algorithm. The major difference between the new algorithms and the classical algorithms is in the data manipulation that occurs during the learning process. The classical systems are based on the manipulation of weight vectors, whereas the proposed algorithms manipulate or transform the actual input data entering a system.
The quality of the proposed algorithms was investigated and tested on simple one-dimensional, linearly separable classification tasks. The proposed algorithms performed well on these. Their performance actually did match the quality of the classical approaches.
The approach has a number of interesting aspects. For example, the proposed algorithms allow a single system with an initial, random configuration to learn a variety of similar, but in context completely different, problems without changing the actual system configuration at all. This makes the approach flexible and this is one aspect mentioned at the outset of this study. This aspect might be particularly interesting in a neuroinformatics context. At this stage nobody really knows how the brain really works, and therefore there is a need for new directions and proposals even if they are a bit naive at first sight.

Future work. There are a few directions into which this study can be continued. For example, an interesting route relates to the question whether it is possible to apply the approach to more complex network structures, for instance networks with input, output and hidden layers. It is also possible to invest the approach and potential consequences from a mere cognitive science and maybe philosophical position. For example, how does a system interpret an environment when the information about this environment is transformed in the learning process. We hope to engage into some of these questions in forthcoming studies.

References

[1] P. H. Koesters, *Deutschland Deine Denker*. Gruner+Jahr AG & Co, 1981.

[2] K. Mehrotra, C. K. Monan, and S. Ranka, *Elements of Artificial Neural Networks*. The MIT Press, 1997.

[3] F. Rosenblatt, "The perceptron, a probabilistic model of information storage and organization in the brain", *Psych. Rev.*, vol. 62, pp. 386–408, 1958.

[4] B. Widrow, "Generalization and information storage in networks of Adaline neurons", *Self-org. Syst.*, vol. 10, pp. 435–461, 1962.

Alfons Schuster holds a Ph.D. in computer science from the University of Ulster in Northern Ireland, and a B.Sc. in applied physics from the University of Applied Sciences Munich in Germany. His current research interests include artificial intelligence, robotics, and DNA computing. Dr Schuster has several years working experience in industry and research. He is currently employed as a lecturer at the Faculty of Informatics of the University of Ulster at Jordanstown in Northern Ireland.
e-mail: a.schuster@ulster.ac.uk
Faculty of Informatics
School of Computing and Mathematics
University of Ulster
Shore Road, Newtownabbey, Co. Antrim, BT37 0QB
Northern Ireland