Neural Networks for Two-Group Classification Problems with Monotonicity Hints

P. Lory[1], D. Gietl

Institut für Wirtschaftsinformatik,
Universität Regensburg, D-93040 Regensburg, Germany

Abstract: Neural networks are competitive tools for classification problems. In this context, a hint is any piece of prior side information about the classification. Common examples are monotonicity hints. The present paper focuses on learning vector quantization neural networks and gives a simple, however effective, technique, which guarantees that the predictions of the network obey the required monotonicity properties in a strict fashion. The method is based on a proper modification of the Euclidean distance between input and codebook vectors.

1 Introduction

Monotonicity properties are ubiquitous in classification problems. Examples are given by Archer and Wang (1991) in the field of marketing and by Sill and Abu-Mostafa (1997_2) in the field of medical diagnosis. Probably the most prominent example is credit scoring; see again Sill and Abu-Mostafa (1997_1 and 1997_2). Here the property of monotonicity is very evident: Assume that applicant A has a higher salary than applicant B, all else being equal. Then common sense dictates, that applicant A must be accepted whenever applicant B is accepted.

The following formal definition covers this situation: A class C is called **monotonic in positive i-direction**, if the following holds: If the vector $\mathbf{x}_0 = (x_{10}, x_{20}, \ldots, x_{n0})$ is in the class C, then every vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with the property $x_i > x_{i0}$ and $x_j = x_{j0}$ for $j \neq i$ is in class C, too. Correspondingly, a class C is called **monotonic in negative i-direction**, if the following holds: If the vector $\mathbf{x}_0 = (x_{10}, x_{20}, \ldots, x_{n0})$ is in class C, then every vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with the property $x_i < x_{i0}$ and $x_j = x_{j0}$ for $j \neq i$ is in class C, too.

The present paper studies the application of monotonicity properties to neural networks of the LVQ (learning vector quantization) type (Kohonen (1997)). LVQ networks realize supervised learning and are characterized by an unordered set of *codebook vectors* (reference vectors). These codebook vectors are placed in the input data space with the purpose to define class regions in this space. In standard applications, the class boundaries are piecewise linear.

[1]Correspondence should be addressed to P. L.

The LVQ network architecture consists of an input layer with n input neurons (where n is the dimension of the input data space) and a competitive layer. The competitive layer comprises m neurons, where each competitive neuron is identified with exactly one codebook vector $w_i \in I\!R^n$. Each neuron of the input layer is connected to each neuron of the competitive layer. The n components of a codebook vector can be visualized as weights of the connections between the n input neurons and the competitive neuron that corresponds to the codebook vector. For details, the reader is referred to the books of Kohonen (1997) and Zell (1994).

Let the $I\!R^n$ be partitioned into several classes ('true partition'). The LVQ net tries to to approximate this partition. For this purpose it classifies an input vector $\mathbf{x} \in I\!R^n$ by the following procedure: The neurons of the competitive layer are partitioned into classes. The input vector $\mathbf{x}$ is compared to all the codebook vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m \in I\!R^n$ in a parallel way. The codebook vector $\mathbf{w}_c$ which is closest to the input vector is the winner and causes the corresponding neuron in the competitive layer to 'fire'.

The classification procedure as described above needs a precise definition of the distance between input and codebook vectors. Usually, the *Euclidean distance* $d(\mathbf{x}, \mathbf{w}_j)$ between the input vector $\mathbf{x}$ and the codebook vector $\mathbf{w}_j$ is used. Thus, the winner $\mathbf{w}_c$ among the codebook vectors is given by

$$d(\mathbf{x}, \mathbf{w}_c) = \min_j d(\mathbf{x}, \mathbf{w}_j)\,. \tag{1}$$

During the training phase, codebook vectors have to be found that approximate the true class partition. For that purpose, various iterative learning algorithms have been proposed: LVQ1, LVQ2, LVQ2.1, LVQ3, OLVQ1. In the case of LVQ1 the iteration reads

$$\begin{aligned}
\mathbf{w}_c(t+1) &= \mathbf{w}_c(t) + \alpha(t)\left[\mathbf{x}(t) - \mathbf{w}_c\right], \\
&\qquad \text{if } \mathbf{x} \text{ and } \mathbf{w}_c \text{ of the same class,} && (2)\\
\mathbf{w}_c(t+1) &= \mathbf{w}_c(t) - \alpha(t)\left[\mathbf{x}(t) - \mathbf{w}_c\right], \\
&\qquad \text{if } \mathbf{x} \text{ and } \mathbf{w}_c \text{ of different classes,} && (3)\\
\mathbf{w}_i(t+1) &= \mathbf{w}_i(t) \qquad \text{for } i \neq c, && (4)
\end{aligned}$$

where $\alpha(t)$ is a learning rate. For further details see again Kohonen (1997) and Zell (1994).

2 Monotonicity

In the following, a modification of the learning vector quantization as described in Section 1 is given that reflects the monotonicity properties in a strict fashion. This aim is achieved by a a modification of the distance between input and codebook vectors:

Let the codebook vector $\mathbf{w} = (w_1, \ldots, w_n)$ belong to the class C, which is monotonic in positive i-direction. Then the distance between this codebook

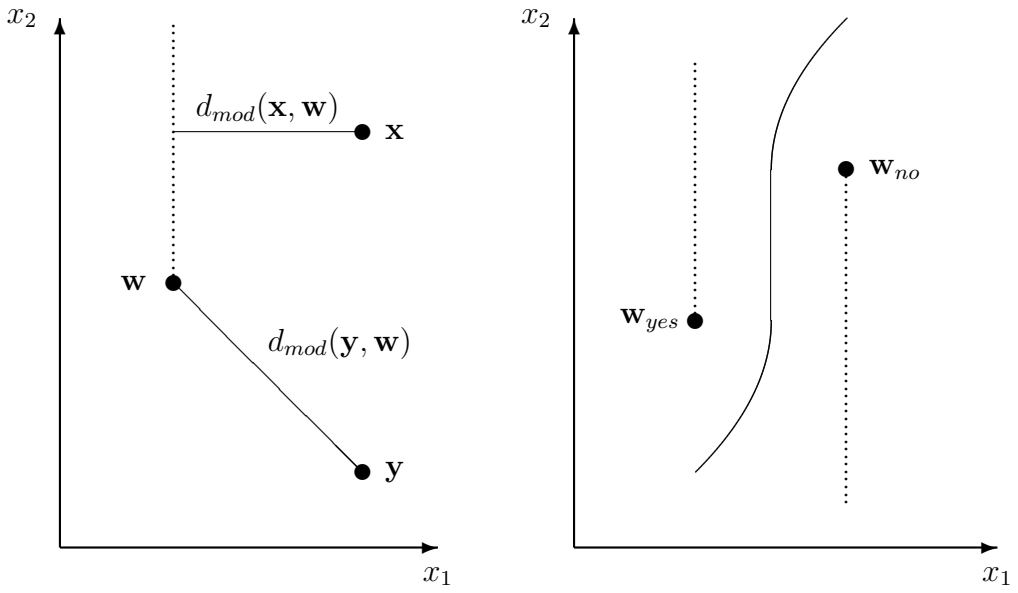

Figure 1: Visualization of the distance (5) in the case of monotonicity in positive 2-direction (left) and class boundary in a simple case (right).

vector and the input vector $\mathbf{x} = (x_1, \ldots, x_n)$ is defined by

$$d_{mod}(\mathbf{x}, \mathbf{w}) := \begin{cases} \sqrt{\sum_{\substack{\nu=1 \\ \nu \neq i}}^{n} (x_\nu - w_\nu)^2} & \text{if } x_i > w_i \ , \\ \\ d(\mathbf{x}, \mathbf{w}) = \sqrt{\sum_{\nu=1}^{n} (x_\nu - w_\nu)^2} & \text{if } x_i \leq w_i \ . \end{cases} \tag{5}$$

If the class C is monotonic in negative i-direction, the following changes have to be made in the definition: $>$ to $<$ and $\leq$ to $\geq$.

The left part of Figure 1 exemplifies the modified distance (5) in the two-dimensional case ($n = 2$). Of course, the class boundary is no longer piecewise linear under this definition. The right part of Figure 1 shows this in a very simple case. Here the class C_{yes} is assumed to be monotonic in positive 2-direction, whereas C_{no} is monotonic in negative 2-direction. Both classes are represented by single codebook vectors $\mathbf{w}_{yes}$ and $\mathbf{w}_{no}$, respectively. The class boundary for this simple example is composed of three elements: 1) the parallel to the x_2-axis, which has the same distance to $\mathbf{w}_{yes}$ and $\mathbf{w}_{no}$; 2) the parabola, which is parallel to the x_1-axis and has $\mathbf{w}_{yes}$ as its focus; 3) the corresponding parabola with the focus $\mathbf{w}_{no}$. In the general case, the class boundary is composed of analogous elements.

The following theorem covers the situation of two-group classification.

Theorem. *Let the input data space be partitioned into two classes C_{yes} and C_{no}, and let these classes be monotonic in positive i-direction and in negative i-direction, respectively. If the distance d_{mod} is used between the input vectors and the codebook vectors, then the class partition $\hat{C}_{yes}$, $\hat{C}_{no}$ given by the neural network of LVQ type satisfies the monotonicity properties strictly.*

Proof: Let the input vector $\mathbf{x}_0 = (x_{10}, x_{20}, \ldots, x_{n0})$ be of the class $\hat{C}_{yes}$. Then it has a minimal distance d_{mod} to one of the codebook vectors of this class (if compared to the distances to the other codebook vectors). Definition (5) implies that this distance cannot increase if x_{i0} is increased (and the other components are fixed). Similarly, the distances between $\mathbf{x}_0$ and the codebook vectors of the class $\hat{C}_{no}$ cannot decrease if x_{i0} is increased. Hence, any vector $\mathbf{x}$ with $x_i > x_{i0}$ and $x_j = x_{j0}$ for $j \neq i$ is in the class $\hat{C}_{yes}$. Consequently, the neural network reflects the monotonicity property of C_{yes} correctly. The classes C_{no} and $\hat{C}_{no}$, respectively, are treated analogously. □

Remark (Several simultaneous directions of monotonicity). For ease of presentation, the above definitions and the theorem have been given for monotonicity properties in only <u>one</u> direction. However, the proposed technique can be generalized immediately in the following way: For each codebook vector, a point set is defined which extends to infinity in all the directions of monotonicity of the corresponding class. The modified distance d_{mod} between an input vector and this codebook vector is defined as the point set distance between the input vector and the point set defined above. This rule can easily be translated into formulas. For example, let the class C be monotonic in positive i-direction and in negative j-direction simultaneously. Then, the adequate definition for the distance between a codebook vector $\mathbf{w} = (w_1, \ldots, w_n)$ representing this class and the input vector $\mathbf{x} = (x_1, \ldots, x_n)$ is

$$d_{mod}(\mathbf{x}, \mathbf{w}) = \begin{cases} \sqrt{\sum_{\substack{\nu \neq i \\ \nu \neq j}} (x_\nu - w_\nu)^2} & \text{if } x_i > w_i \text{ and } x_j < w_j, \\ \sqrt{\sum_{\nu \neq i} (x_\nu - w_\nu)^2} & \text{if } x_i > w_i \text{ and } x_j \geq w_j, \\ \sqrt{\sum_{\nu \neq j} (x_\nu - w_\nu)^2} & \text{if } x_i \leq w_i \text{ and } x_j < w_j, \\ d(\mathbf{x}, \mathbf{w}) = \sqrt{\sum (x_\nu - w_\nu)^2} & \text{if } x_i \leq w_i \text{ and } x_j \geq w_j. \end{cases}$$

The extension of the above theorem to these cases is straightforward.

Three versions of the algorithm are suggested: Version 1 uses d_{mod} only during the application phase of the LVQ network. In more detail: The network is trained using any (e.g. the Euclidean) distance. Then an input vector $\mathbf{x}$ is classified by determining the winning codebook vector $\mathbf{w}_c$ in (1) with $d = d_{mod}$. Thus, the modified distance is used only for class assignment. Note that this is sufficient for the proof of the above theorem. Version 2 uses d_{mod} during the training phase *and* the application phase of the network. In more detail: Each training vector requires the determination of the winning codebook vector $\mathbf{w}_c$ in (1). The distance d_{mod} is used in this process *and* during the application phase (see Version 1). Version 3 is an extension of Version 2 insofar as in (2) and (3) of the LVQ1 learning algorithm only those components of $\mathbf{w}_c$ are updated that have *not* been omitted in the computation of the modified distance between $\mathbf{x}$ and $\mathbf{w}_c$. The extension to LVQ2, LVQ2.1, LVQ3 and OLVQ1 is straightforward.

3 Experimental Results

The techniques of Section 2 have been tested in Gietl (1999) on the basis of real life credit data. These are described and analyzed by classical statistical methods in Fahrmeir et al. (1996). The datebase consists of 1000 applicant case histories (700 good and 300 bad ones). Each customer is characterized by $n = 20$ features. The task is to predict whether or not an applicant will default. The MATLAB Neural Network Toolbox was used in the computations (see Demuth and Beale (1997)). Some functions had to be modified due to the techniques described in Section 2.

The datebase was randomly subdivided into a training set of 400 examples (200/200 good/bad) and a test set of 200 examples (100/100). The *k-hold-out method* (or *modified U-method*) was used in order to guarantee valid results (see e.g. Poddig (1994)). For that purpose the training set was randomly partitioned into five disjunct portions of equal size. The relation good/bad = 1 was maintained in each of these portions. The examples of four of these portions were used for direct training, whereas the fifth portion served as cross validation set ($k = 80$). The portions were exchanged in a rotating manner. So each of the five portions acted as cross validation set exactly once. The parameters m (number of competitive neurons) and N_{ls} (number of learning steps) were determined by maximizing the average of the five classification rates. Here $m \in \{2, 4, 6, \ldots, 20\}$ and $N_{ls} \in \{1000, 2000, \ldots, 5000\}$. The learning rate decreased linearly from 0.1 to 0.0. The following features were assumed to be monotonic: running account, running time of the loan, savings account, number of years at current job. Versions 2 and 3 were inferior and are omitted in the following. The computed maxima were $m = 12$, $N_{ls} = 3000$ in the nonmonotonic case and $m = 6$, $N_{ls} = 1000$ in the monotonic case. In all computations, a ratio of $1 : 1$ for the number of 'good' neurons to the number of 'bad' neurons in the competitive layer was maintained.

The performance of the method was examined on the test set (which was not involved in the training process). Both the classification rates of a combined network and of a master network have been determined. In the combined network, the five networks that arise during the k-hold-out method form a majority decision. The master network is trained with the optimal values for m and N_{ls} found by the k-hold-out method. However, it uses the complete training set (see Poddig (1994)). For the combined network, the classification rate increased from 73.0 % (without monotonicity) to 74.0 % (with monotonicity). In the case of the master network the corresponding rates are 71.5 % and 74.0 %. These improvements are surprisingly low. Possibly, credit screening data reflect the monotonicity already almost perfectly. This is supported by the fact, that comparable studies (Sill and Abu-Mostafa (1997_1, 1997_2)) on the basis of multi-layer networks report a similarly low improvement (1.6 %). Whether for datasets from other fields of application greater improvements can be observed, remains an open question at present.

References

ARCHER, N.P. and WANG, S. (1991): Fuzzy Set Representation of Neural Network Classification Boundaries. *IEEE Transactions on Systems, Man and Cybernetics, 21, 735-742.*

DEMUTH, H. and BEALE, M. (1997): MATLAB Neural Network Toolbox User's Guide. The MathWorks, Inc., Natick, MA.

FAHRMEIR, L., HÄUSSLER, W. and TUTZ, G.: Diskriminanzanalyse, in: FAHRMEIR, L., HAMERLE, A. and TUTZ, G. (Eds.) (1996): Multivariate statistische Verfahren. Walter de Gruyter, Berlin, 357-435.

GIETL, D. (1999): Kreditwürdigkeitsprüfung auf der Basis künstlicher neuronaler Netze vom LVQ-Typ. Diploma thesis, Institut für Wirtschaftsinformatik, Universität Regensburg.

KOHONEN, T. (1997): Self-Organizing Maps. Springer, Berlin.

PODDIG, T.: Mittelfristige Zinsprognosen mittels KNN und ökonometrischen Verfahren, in: REHKUGLER, H. and ZIMMERMANN, H.G. (Eds.) (1994): Neuronale Netze in der Ökonomie. Franz Vahlen, München, 491–545.

SILL, J. and ABU-MOSTAFA, Y.S.: Monotonicity Hints for Credit Screening, in: AMARI, S.-I., XU, L., CHAN, L.-W., KING, I. and LEUNG, K.-S. (Eds.) (1997_1): Progress in Neural Information Processing. Proceedings of the 1996 International Conference on Neural Information Processing ICONIP'96, Hong Kong, 24-27 Sept. 1996, Springer, Singapore, 123-127.

SILL, J. and ABU-MOSTAFA, Y.S.: Monotonicity Hints, in: MOZER, M.C., JORDAN, M.I. and PETSCHE, T. (Eds.) (1997_2): Advances in Neural Information Processing Systems 9. Proceedings of the 1996 Conference, Denver, 2-5 Dec. 1996, MIT Press, London, 634-640.

ZELL, A. (1994): Simulation neuronaler Netze. Addison-Wesley, Bonn.