Relevance determination in learning vector quantization

Thorsten Bojer, Barbara Hammer, Daniel Schunk,
and Katharina Tluk von Toschanowitz

University of Osnabrück, Department of Mathematics/
Computer Science, Albrechtstraße 28, 49069 Osnabrück, Germany,
e-mail: hammer@informatik.uni-osnabrueck.de

Abstract. We propose a method to automatically determine the relevance of the input dimensions of a learning vector quantization (LVQ) architecture during training. The method is based on Hebbian learning and introduces weighting factors of the input dimensions which are automatically adapted to the specific problem. The benefits are twofold: On the one hand, the incorporation of relevance factors in the LVQ architecture increases the overall performance of the classification and adapts the metric to the specific data used for training. On the other hand, the method induces a pruning algorithm, i.e. an automatic detection of the input dimensions which do not contribute to the overall classifier. Hence we obtain a possibly more efficient classification and we gain insight to the role of the data dimensions.

1. Introduction

Kohonen's learning vector quantization (LVQ) provides a very intuitive method of learning a classification based on a finite set of training patterns [8]. So called codebook vectors are adapted to the data such that their receptive fields match the support of the distribution of the respective class as accurately as possible. Various modifications exist which ensure faster convergence of the algorithm (OLVQ), a better adaptation of the borders of the receptive fields to optimum Baysian decision (LVQ2, LVQ3), an adequate initialization of the codebooks according to the data distribution (LVQ+SOM) [7], a dynamic adaptation of the number of codebooks (DLVQ) [3], or an adaptation for complex data structures [10], to name just a few. It combines ideas of unsupervised algorithms, adaptation to the data with Hebbian learning, and supervised learning, adaptation according to the error signal provided by the desired output.

The success of LVQ depends crucially on the fact that the metric which is used for training, usually the Euclidian metric, fits the behavior of the classification which is to be learned. The algorithm is based on the assumption that the dimensions are approximately equally scaled and equally important. Hence, it is necessary to preprocess and scale the data accordingly. This may be difficult to obtain in specific learning tasks since estimating the importance of the input dimensions may require problem specific knowledge. Moreover, data are often high dimensional and data dimensions may be correlated. Some modifications of clustering algorithms exist which adapt the metric *during training* to the specific setting. As an example, fuzzy-k-means clustering is modified by Gustafson and Kessel [5] or Gath and Geva [4] such that a problem specific metric arises. However, these methods are time consuming since they require matrix inversion and they lead to less intuitive training algorithms.

We propose a modification of LVQ which introduces relevance factors to the dimensions: relevance learning vector quantization or RLVQ, for short. The relevance factors or input weights are adapted during training with supervised Hebbian learning based on the error signals. The method can be motivated based on the Hebb rule directly; alternatively, it can be seen as an adaptation of the classical perceptron learning rule [9] for a correlated perceptron learning task. After training, the weights constitute scaling factors for the respective dimensions which are to be added to the Euclidian metric. Based on their values, data dimensions may be identified as not very important and they may be pruned without increasing the error of the overall classifier much. We test the method on various artificial data and real life data.

2. The LVQ algorithm

Assume that a clustering of data into C classes is to be learned and a finite training set $X = \{(x^i, y^i) \subset \mathbb{R}^n \times \{1, \dots, C\} \mid i = 1, \dots, m\}$ of training data is given. We denote the components of a vector $x \in \mathbb{R}^n$ by $(x_1, \dots, x_n)$ in the following. LVQ chooses a fixed number of vectors in $\mathbb{R}^n$ for each class, so called codebooks. Denote the set of codebooks by $\{w^1, \dots, w^K\}$ and assign the label $c^i = c$ to w^i iff w^i belongs to the cth class, $c \in \{1, \dots, C\}$. The receptive field of w^i is defined by $R^i = \{x \in X \mid \forall w^j \, (j \neq i \to |x - w^i| < |x - w^j|)\}$. The training algorithm adapts the codebooks w^i such that for each class $c \in \{1, \dots, C\}$ the corresponding codebooks represent the class as accurately as possible. That means, the difference of the points belonging to the cth class, $\{x^i \in X \mid y^i = c\}$, and the receptive fields of the corresponding codebooks, $\bigcup_{c^i = c} R^i$, should be as small as possible for each class. LVQ initializes the codebooks w^i appropriately, e.g. with random vectors or the center of gravity of the patterns of the respective class, and iteratively updates the codebooks as follows:

repeat: *choose* $(x^i, y^i) \in X$
compute w^j *such that* $x^i \in R^j$

$$w^j := \begin{cases} w^j + \epsilon(x^i - w^j) & \text{if } y^i = c^j \\ w^j - \epsilon(x^i - w^j) & \text{otherwise} \end{cases} \qquad (*)$$

where $\epsilon \in \,]0, 1[$ is the so called learning rate. Obviously, the different data dimensions are ranked equally in the above algorithm and LVQ will fail if the dimensions are not appropriately scaled.

3. The RLVQ algorithm

Assume that X and w^i are chosen as before. In order to make a different scaling of the several input dimensions possible, we introduce input weights λ_1, $\dots$, $\lambda_n \geq 0$. We substitute the Euclidian metric $|x - y|$ by

$$|x - y|^2_\lambda = \sum_{i=1}^{n} \lambda_i (x_i - y_i)^2 \, .$$

Hence the receptive field of codebook w^i becomes $R^i_\lambda = \{x \in X \mid \forall w^j\,(j \neq i \to |x - w^i|_\lambda < |x - w^j|_\lambda)\}$. Substituting R^i by R^i_λ in the above algorithm yields a *different* weighting of the input dimensions. Now the question arises of how to choose the input weights λ_i appropriately. We start with $\lambda_i = 1/n$, i.e. the standard Euclidian metric, and iteratively adapt the weights as follows:

repeat: *choose* $(x^i, y^i) \in X$
compute w^j *such that* $x^i \in R^j_\lambda$
for all l:

$$\lambda_l := \begin{cases} \max\{\lambda_l - \alpha|x^i_l - w^j_l|, 0\} & \text{if } y^i = c^j \\ \lambda_l + \alpha|x^i_l - w^j_l| & \text{otherwise} \end{cases} \qquad (**)$$

for all l:

$$\lambda_l := \lambda_l / |\lambda|$$

where $\alpha \in]0, 1[$ is a learning rate for the input weights. The above rule (**) is either applied *after* regular LVQ training (RLVQi), after regular LVQ training *in common* with additional updates for the codebooks λ according to rule (*) (RLVQii), or in common with rule (*) *from the beginning* of the training process (RLVQiii). Since we are interested in online adaptation of both the codebooks and the input weights, we use RLVQii or RLVQiii in the following.

The update (**) can be motivated by the Hebb paradigm: Assume that the classification of x^i is correct, i.e. $y^i = c^j$. Then those input weights λ_l are decreased most in $(**)$ for which the l th component of the training point differs considerably from the l th component of the codebook. In contrast, input weights λ_l are little decreased for which both components are close together and hence dimension l contributes to the fact that point x^i is classified correct. Together with the normalization of λ this results in an increase of weights λ_l iff dimension l contributes to the correct classification. Assume the classification of x^i is wrong, i.e. $y^i \neq c^j$. Then those input weights λ_l are increased only slightly in $(**)$ where the l th component of the training point is close to the l th component of the codebook. Because of the normalization of λ precisely those λ_l are decreased which contribute most to the wrong classification of x^i.

4. Comparison to perceptron learning

A different motivation of learning rule (**) can be obtained if a correlated perceptron training problem is considered. Assume the codebooks are fixed and we would like to obtain weights λ_l such that $x^i \in \bigcup_{c^j = y^i} R^j_\lambda$ for as many $x \in X$ as possible. We refer to such a problem as $P(\lambda)$ in the following. For simplicity we assume that each class c is represented by only one codebook w^c. Then (x^i, y^i) is classified correct iff $|x^i - w^j|^2_\lambda > |x^i - w^i|^2_\lambda$ for all $j \neq i$. This can be further transformed to $\sum_l \lambda_l z^{ij}_l > 0$ where $z^{ij}_l = (x^i_l - w^j_l)^2 - (x^i_l - w^i_l)^2$. Hence, we obtain a perceptron training problem: Find positive weights of a perceptron (corresponding to λ_l) such that all z^{ij} are classified positive by the perceptron. The standard perceptron algorithm yields the update $\lambda_l = \lambda_l + (x^i_l - w^j_l)^2 - (x^i_l - w^i_l)^2$ if x^i is classified wrong. Splitting this adaptation rule into two parts yields the updates $\lambda_l = \lambda_l + (x^i_l - w^j_l)^2$ if w^j represents a class different from y^i and $\lambda_l = \lambda_l - (x_l^i - w_l^i)^2$ if we consider the class y^i. Since we require $\lambda_l \geq 0$, we have to add a truncation if the above rule yields negative values. The update (**) entirely decouples the two summands and adapts proportional to the absolute distance of the components of the codebook and the training point instead of the quadratic distance.

The comparison to standard perceptron training gives us further insight into the complexity of the task we deal with. The above reasoning shows that we can formulate (part of) our training task, the problem $P(\lambda)$, as a perceptron learning task. The converse is also possible: Assume a perceptron training problem is given, i.e.: given points $p^1, \ldots, p^m \in \mathbb{R}^n$, find a vector $\lambda \in \mathbb{R}^n$ with $\lambda_i \geq 0$ such that $\sum \lambda_i p_i^j > 0$ for the maximum number of points p^j (each perceptron problem can easily be transferred to this special problem with standard argumentation). This can be transferred to the following problem $P(\lambda)$: given codebooks $w^0 = (0, \ldots, 0)$ and $w^1 = (1, \ldots, 1)$ and points $z^1, \ldots, z^m$, where $z_j^i = (1 - p_j^i)/2$, find weights $\lambda_i \geq 0$ such that the maximum number of points z^j belongs to the receptive field R_λ^1. Since the number of points z^j which do not belong to R_λ^1 coincides with the number of points p^j which are not classified correct by the perceptron, this constitutes a cost preserving reduction from perceptron learning to problems $P(\lambda)$. Since perceptron training in the presence of errors is NP-hard (even approximation within any positive factor) [6], $P(\lambda)$ is NP-hard either if data are noisy or the clusters overlap. Appropriate weights can be found in polynomial time assumed data are well behaved and the clusters do separate.

5. Experiments

We split data randomly into a training set and a part which is used for tests only. The reported results are the means of several runs. Each run is performed until convergence with $\epsilon = 0.1$, $\alpha = 0.1$ unless stated otherwise.

Artificial data without overlap: Data set 1 contains 3 classes each comprising 2 clusters with 15 points in each cluster in the first two dimensions (see Fig. 1). In data set 2, we add four dimensions which are copies of dimension 1 and perturbed by Gaussian noise with variances 0.05, 0.1, 0.2, and 0.5, respectively. Furthermore, we add two dimensions with uniform noise with support of diameter 1 or 0.4, respectively, and two dimensions with Gaussian noise with variances 0.5 and 0.2, respectively. Hence the first two dimensions are relevant for the classification, dimensions 3 to 6 may partially substitute dimension 1 with increasing noise added, the remaining dimensions do not contribute to the clustering. We train 6 codebooks, 2 codebooks for each class, with standard LVQ and RLVQiii on data sets 1 and 2. We obtain an accuracy between 0.91 and 0.96 on training set and test set for data set 1 and LVQ or RLVQiii, respectively. RLVQiii provides the same accuracy for data set 2, whereas LVQ obtains an accuracy between 0.81 and 0.89 on data set 2. The input weights determined by RLVQiii are $\lambda_1 \approx \lambda_2 \approx 0.5$ for data set 1, and $\lambda \approx (0.13, 0.12, 0.12, 0.11, 0.1, 0.09, 0.1, 0.08, 0.07, 0.06)$ for data set 2. The result is stable, i.e. the ranking of the input dimensions provided by the weights does not change significantly for several runs. This ranking allows us to prune precisely the dimensions which are irrelevant for this clustering.

Artificial data with overlap: Data set 3 contains data with a considerable overlap of the classes (see Fig. 1). Data set 4 adds the same noise to data set 3 as described for data set 2. LVQ, RLVQiii, and RLVQii yield a training and test accuracy of about 0.8 for 6 codebooks on data set 3. LVQ yields an accuracy of 0.56 to 0.7 on training and test set depending on the separation into training and test set on data set 4. RLVQiii obtains the better accuracy between 0.77 and 0.85, however, it shows instabilities, i.e. very bad accuracy of about 0.3 and a weighting of $\lambda_1 >> \lambda_i$ for $i > 1$ in about a quarter of the runs. The instabilities are avoided by RLVQii, which yields an accuracy between 0.79 and 0.86 depending on the random separation of the data into test and training set. A typical vector λ is $(0.11, 0.12, 0.11, 0.09, 0.09, 0.09, 0.09, 0.09, 0.1, 0.1)$, hence the irrelevant dimensions are detected.

Iris data: The task is to predict three classes of plants based on 4 numerical attributes in 150 instances [1]. We train LVQ, RLVQii, or RLVQiii, respectively with 6 codebooks and $\epsilon = 0.1$, $\alpha = 0.01$. LVQ provides an accuracy of 0.96 for training and test set. Surprisingly, the behavior is not perfectly stable for a constant learning rate. The accuracy changes cyclically between 0.94 and 0.96. RLVQii and RLVQiii show stable behavior and approach an accuracy 0.97 on the training set and 0.95 on the test set. Typical weights λ are $(0.02, 0.01, 0.02, 0.89)$ for RLVQiii and $(0.04, 0.05, 0.03, 0.87)$ for RLVQii. Hence, the last input dimension is ranked as most important and the weights are adapted such that the classification is based on this dimension *only*. Pruning the remaining dimensions shows that the same accuracy is obtained using input dimension 4 only. Note that the better accuracy 100% could be possible with more dimensions, however, neither LVQ nor RLVQ obtained this accuracy.

Mushroom data: The task is to predict as to whether a mushroom is poisonous or edible depending on 22 symbolic attributes [1]. Unary encoding of the attributes leads to 8124 patterns of dimension 117. We train LVQ, RLVQii, and RLVQiii with 6 codebooks and $\epsilon = 0.05$, $\alpha = 0.005$. LVQ obtains an accuracy of 0.96 on the training and the test set. RLVQ yields a slightly unstable behavior where the accuracy changes cyclically in the range from 0.92 to 0.99 on the training and the test set. Pruning the dimensions with small input weights yields the possibility of pruning about 50 dimensions for RLVQiii with an accuracy of more than 0.95 and about 80 dimensions for RLVQii with an accuracy of more than 0.92. The respective dimensions ranked highest (values $\lambda_i > 0.1$) are dimension 83 in all cases and 1 to 5 of dimensions

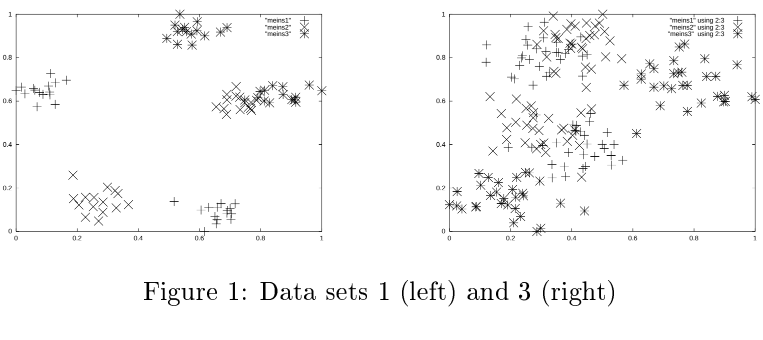

Figure 1: Data sets 1 (left) and 3 (right)

$(1, 6, 10, 18, 19, 20, 47, 73, 87, 94, 101)$. Dimension 83 is *identical* for all patterns, hence it is reinforced in each update corresponding to a correct classification although it does not contribute to the overall classification. Dimension 101 is ranked high in several runs. It corresponds to the attribute 'spore-print-color green'. This attributes allows an accuracy of 0.99 according to [2]. The other dimensions which are ranked as important comprise instances of various attributes, some of them ranked as important in the approach [2] as well.

6. Conclusions

We proposed a method to determine the relevance of the several input dimensions of an LVQ architecture during training automatically. For this purpose, input weights are introduced and adapted to the data with Hebbian learning based on the error signal. The weights adapt the metric to the data. The induced ranking allows input pruning. The results which we obtained with RLVQ are either comparable or better than LVQ in particular if a large number of input dimensions is available. The induced ranking allows us to prune at least half of the input dimensions without a significant increase of the error.

Training shows instabilities if we deal with noisy data and train codebooks and weights in common, hence it is advisable to use RLVQii or RLVQi for general data sets. This may be partially due to the fact that our training problem includes the problem of perceptron training which is NP-hard in the presence of errors. However, even for the noise free case a formal proof of the convergence of RLVQ would be interesting since the method is based on – but not precisely equal to – perceptron training. The update rule (**) motivates alternatives such as a multiplicative update of the input weights. Preliminary results indicate that multiplicative RLVQii (which are not reported in this paper due to space limitations) shows better performance than additive RLVQ and a more distinguished ranking of the different input weights. Further theoretical and experimental analysis of the update rules will be the subject of future work.

References

[1] C.L. Blake and C. J. Merz, UCI Repository of machine learning databases , Irvine, CA: University of California, Department of Information and Computer Science.

[2] W. Duch, R. Adamczak, and K. Grąbcweski, Extraction of crisp logical rules using constrained backpropagation networks, in M. Verleysen (ed.), Proceedings of ESANN'97, D-facto Publications, 1997.

[3] R. Duda and P. Hart, Pattern Classification and Scene Analysis, Wiley, 1973.

[4] I. Gath and A.B. Geva, Unsupervised optimal fuzzy clustering, IEEE Trans. Pattern Analysis and Machine Intelligence 11, 773-791, 1989.

[5] D.E. Gustafson and W.C. Kessel, Fuzzy clustering with a fuzzy covariance matrix, Proceedings of IEEE CDC'79, 761-766, 1979.

[6] K.-U. Höffgen, H.-U. Simon, and K. VanHorn, Robust trainability of single neurons, Journal of Computer and System Sciences, 50, 1995.

[7] T. Kohonen, Self-Organizing Maps, Springer, 1997.

[8] T. Kohonen, Learning vector quantization, in M.A. Arbib (ed.), The Handbook of Brain Theory and Neural Networks, MIT Press, 537-540, 1995.

[9] M. Minsky and S. Papert, Perceptrons, MIT Press, 1988.

[10] P. Somervuo and T. Kohonen, Self-organizing maps and learning vector quantization for feature sequences, Neural Processing Letters 10(2), 151-159, 1999.