6 A Dynamic LVQ Algorithm for Improving the Generalisation of Nearest Neighbour Classifiers

Abstract- A new dynamic strategy for Kohonen's LVQ algorithms using growing and pruning methods is introduced. Once the learning system converges to a solution using a fixed number of prototypes, the growing method incrementally adds new prototypes in those local regions where the misclassification error is greater. The initial locations of the new prototypes are viewed as estimates of the new equilibrium points of the learning algorithm with the augmented number of prototypes. This constructive process is repeated until the classification accuracy measured with a validation set stops decreasing. Then, a pruning algorithm is executed to remove all those prototypes that do not form class borders. Experimental results using NIST hand-written databases are promising since we show that dynamic LVQ algorithms outperform their static counterparts for the same codebook size.

Index Terms- Constructive Learning Algorithms, Growing Algorithms, Incremental Algorithms, Pruning Algorithms, LVQ Algorithms, Nearest Neighbour Classifiers.

1. Introduction

The approximation power of a learning system determines the hypothesis space where the learning algorithm searches a solution. As the hypothesis space augments, the learning system has more changes to solve the classification or regression problem since the approximation error will be typically smaller. However the model induced by the learning algorithm will have presumably worse generalisation capabilities since the estimation error will be (probably) bigger due to computing the parameters of a model of increased complexity using the same number of training samples. Consequently, the learning system must solve efficiently a trade-off between its approximating power and the information about the problem given by the training set. (This well-known problem appears in the literature under several formulations that in spirit are similar: bias-variance trade-off (Geman et. al, 1992), approximation error vs. estimation error (Niyogi & Girosi, 1994), and the relation between the capacity of the learning system and the number of training samples (Vapnik, 1995).)

Constructive (or growing or incremental) and Pruning algorithms are concerned with the problem of choosing the optimal balance between the approximation power and the estimation error of the learning system given a training set of fixed size. (In multi-layer feed-forward neural networks, this problem is choosing the right architecture- e.g. the number of hidden layers and hidden units per layer. In nearest neighbour classification, it is simply the selection of the number of prototypes for each class.) The first group of algorithms starts with a small model and grows it until a satisfactory solution is found. On the other hand, pruning algorithms removes those parameters of a big (and previously trained) model that are not effectively used. Much work on dynamic learning procedures (that is learning algorithms that deal with adaptive architectures) has been devoted to feed-forward neural networks. (See (Kwok & Yeung, 1995) and (Paresh et al., 1995) for surveys on constructive algorithms for feed-forward neural networks in regression and pattern classification respectively. (Bishop, 1995) §9.5 can also serve as an introductory reference for growing algorithm and pruning algorithms using these architectures as well.) However, in this chapter, we focus our attention on adaptive nearest neighbour classifiers that use Kohonen's LVQ algorithms (Kohonen, 1996).

Kohonen's LVQ algorithms design codebooks (or set of prototypes) for 1-NN (nearest neighbour) classifiers that exhibit good generalization. (In this chapter, we will refer to these classifiers as LVQ-based NN classifiers). Kohonen has shown that given certain conditions these local learning algorithms can estimate the Bayesian borders with arbitrary good accuracy, depending on the number of codebook vectors used (p. 206; (Kohonen, 1996)). However, in

practice, we do not know the optimal number of prototypes for each class that achieves a good balance between the approximating power and the estimation error of the NN classifier. Typically, the user of the learning algorithm assigns the number of prototypes before the training process begins. Then several training sessions with different assignations are performed and finally the user choose the better session according to a pre-established criterion (e.g. choose that classifier that gives the best classification accuracy of the validation set). Clearly, a better strategy would be a dynamic assignation of the prototypes according to the errors produced in the classifier. In this way, we could perform several training sessions linked with a growing algorithm that adds new prototypes in those local regions where the misclassification error is greater. This constructive process would be repeated until the classification accuracy measured with a validation set stops decreasing and finally a pruning algorithm could be executed to remove all those prototypes that do not form class borders.

In the next section, we will introduce the dynamic procedure. Section 3 presents some experimental results using NIST hand-written databases (Garris et al., 1997). Finally some discussion and conclusion are given.

2. The dynamic LVQ algorithm

The complexity of Bayes borders is locally defined so we can reduce bias (the difference between the Bayes classifier and our classifier) if we adjust *locally* each border according to the evolution of the classification error during training phase. This means give dynamically more approximating resources (i.e. codevectors) to regions in which training data are poorly classified. However, if we growing process is not conveniently stopped the risk of over-fitting is high (i.e. the excessive tuning to the training set). Consequently, a validation set must supervise the growing process. Besides a pruning process (the removal of useless prototypes) is also convenient to achieve the simplest solution. This also helps avoiding over-fitting. Accordingly, we propose the following dynamic schema based on an earlier proposal (Bermejo et al., 1998):

1. Train using a static LVQ algorithm until the classification error of a validation set stops decreasing
2. *Constructive part*: Add a maximum of MAV new prototypes in those Voronoi regions in which the classification error is greater. The fundamental question of the growing process is to determine the initial location of the new prototypes. Since LVQ algorithm perform gradient descent over a cost function (at least LVQ1 (Bermejo, 2000a) and LVQ2

(Bottou, 1998)), the initial values might be near to an attractor (or an equillibrium) point of the LVQ algorithm. (Bermejo,2000a) shows that the attractor point of LVQ1 is

3.
4. Repeat 1 and 2 until the validation error stops decreasing
5. *Pruning part*: Remove all those prototypes that do not form class borders.

2.1. The Constructive part

The fundamental question of the growing process is to determine the initial location of the new prototypes. Since LVQ algorithm perform gradient descent over a cost function (at least LVQ1 (Bermejo, 2000a) and LVQ2 (Bottou, 1998)), the initial values might be near to an attractor (or an equillibrium) point of the LVQ algorithm. (Bermejo,2000a) shows that the attractor point of LVQ1 is

$$\mathbf{m}_j^{*} = \frac{\sum_{i=1}^{N} 1(\mathbf{x}_i \in R_j)\left(1(cl(\mathbf{x}_i)=cl(\mathbf{m}_j))-1(cl(\mathbf{x}_i)\neq cl(\mathbf{m}_j))\right)\mathbf{x}_i}{\sum_{i=1}^{N} 1(\mathbf{x}_i \in R_j)\left(1(cl(\mathbf{x}_i)=cl(\mathbf{m}_j))-1(cl(\mathbf{x}_i)\neq cl(\mathbf{m}_j))\right)} \quad j=1,...,K \qquad (1)$$

where $\{(\mathbf{x}_i, cl(\mathbf{x}_i)), i=1,...,N\}$ is the training set, $cl(\mathbf{x}_i)$ denotes the class label associated to pattern $\mathbf{x}_i$, $\mathbf{C}=[\mathbf{m_1}^T \mathbf{m_2}^T ... \mathbf{m_K}^T]^T$ is the set of labelled prototypes (or codebook) of the Euclidean Nearest Neighbour (NN) classifier and $R_j = \left\{\mathbf{x} \,\middle|\, \|\mathbf{x}-\mathbf{m}_j\| = \min_{i=1,..,K} \|\mathbf{x}-\mathbf{m}_i\|\right\}$ j=1,...,K are the Voronoi regions where the classifier maps any input pattern that falls in it to the class which its codevector $\mathbf{m_j} \in \Re^p$ belongs.

We will place a new prototype $\mathbf{m}$ belonging to the same class than some training samples $\{\mathbf{x_j}\}$ that fall in a particular Voronoi region R_k induced by the codevector $\mathbf{m_k}$. An initial value near the attractor of this new prototype could be the mean of these samples. This agrees with the heuristic derivation of (Canolli & Valli, 1994). However this mean only takes account on $\{\mathbf{x_j}\}$ and the attractor of the new prototypes will depends on samples previously assigned to the Voronoi region R_k. A simple dependence on the initial value of the new prototype with other samples than $\{\mathbf{x_j}\}$ assigned to R_k could relate them with $\mathbf{m_k}$.

According with the above observation we propose the following constructive (or growing) algorithm:

1. Compute $\mathbf{C}\text{err}_{n+1}$, a subset of $\mathbf{C}_{n+1}$ of size ≤ MAV, composed by those prototypes that cause the greatest number of misclassifications. The formal definition of this set is

$$\begin{aligned} &\{\mathbf{m}_j\} = \mathbf{C}_{n+1} \supset \mathbf{C}\,\text{err}_{n+1} = \{\mathbf{m}_i\},\ |\mathbf{C}\,\text{err}_{n+1}| \le \text{MAV} \\ &E(\mathbf{m}_i) \ge E(\mathbf{m}_j) \quad \forall j = 1 \ldots |\mathbf{C}_{n+1}|,\ \forall i = 1 \ldots |\mathbf{C}\,\text{err}_{n+1}| \end{aligned} \tag{2}$$

where $E(\mathbf{m}_i)$ is the number of classification errors due to codebook vector $\mathbf{m}_i$. In other words, $E(\mathbf{m}_i)$ is the number of input patterns which class is different from vector $\mathbf{m}_i$ that falls into its Voronoi region. $E(\mathbf{m}_i)$ can be expressed as

$$E(\mathbf{m}_i) = \sum_{l=1}^{c} E_{cl}(\mathbf{m}_l) \tag{3}$$

where c is the number of classes and $E_{cl}(\mathbf{m}_i)$ is the number of class cl classification errors due to codebook vector $\mathbf{m}_i$ or the number of class cl vectors that fall in the Voronoi region of $\mathbf{m}_i$ with $\mathbf{m}_i$ belonging to a different class.

2. For each $\mathbf{m}_i$ that belongs to $\mathbf{C}\text{err}_{n+1}$:

2.1. Compute those training samples $\{\mathbf{x}_m, i=1,...,M\}$ assigned to $\mathbf{m}_i$ that belongs to the class that causes the majority of misclassifications in the Voronoi region of $\mathbf{m}_i$.

2.2 Compute mean vector $\boldsymbol{x}_{prod}$ of set $\{\mathbf{x}_m\}$ as:

$$\mathbf{x}_{prod} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{x}_m \tag{4}$$

2.3. Create a new codebook vector with these values:

$$\begin{aligned} &\mathbf{m}'_i = \mathbf{m}_i + \text{BETA}\ (\mathbf{x}_{prod} - \mathbf{m}_i) \\ &\text{class}\ [\mathbf{m}'_i] = \text{class}\ [\mathbf{x}_m] \end{aligned} \tag{5}$$

2.2. The Pruning part

The pruning algorithm must remove those codevectors that do not contribute to form class borders. First, we will delete those prototypes in which no training data are assigned to them (pass 1). Then we erase those codevectors makes hat classification accuracy of the validation set stands or improves when they are not used in the NN classifier (pass 2). This second step is repeated cyclically through the codebook until there is no 'redundant' codevector left.

3. Experiments in Hand-written character Recognition

In this section, we present several simulations using the LVQ_PAK (Kohonen et al., 1995) as a software toolbox to compare our dynamic LVQ algorithm with static LVQ. Since the constructive part of our algorithm can employ any of the LVQx algorithms, we have chosen LVQ3 as a core. We will refer to this part as CLVQ3.

3.1 NIST Database and its preprocessing

This handwritten data set can be found in (Garris et al., 1997) in directories train/hsf_4, train/hsf_6 and train/hsf_7. We have spilt directory train/hsf_7 in two sets (one for validation and one for test) to preserving the original data's class distribution. The other two directories were chosen for training (see Table 1). These images (32x32 pixels) have been pre-processed with a Principal Component Analyser that extracts 64 components from each image (NIST's mis2evt utility). The correlation matrix of the PCA was computed using the training set.

Database	Training Set Size	Validation Set Size	Test Set Size
Upper	24420	1031	10453
Lower	24205	1078	10914

Table 1. Sets Size.

3.2. Codebook Initialization and default parameters in CLVQ3.

Every experiment related to CLVQ3 applies in cascade first *eveninit*, balance, olvq1, lvq1 & lvq2 programs from LVQ_PAK to obtain a CLVQ3's initial codebook. Here it is the parameters values employed in calling these programs: eveninit: noc=<initial codebook size>; olvq1: rlen=90000; lvq1: rlen=90000, alpha=0.01; lvq2: rlen=90000, alpha=0.01, win=0.3. Default parameters values used throughout experiments, those related with LVQ3 algorithm embedded in CLVQ3, are the following: alpha=0.01, win=0.3, epsilon=0.1.

3.3. Experiment #1

We have done three sub-experiments to characterize the parameters of the constructive algorithm and the accuracy of then to compare the accuracy of CLVQ3 with its static counterpart. Besides, we will add new prototypes every EPUP epochs instead of estimating the optimal number of epochs using a validation set to observe if this sub-optimal procedure have a big impact the test error.

3.3.1. Sub-Experiment 1.

CLVQ3 was trained with two MAV values (2 & 10) for several initial codebook sizes (50, 150, 234 & 280), BETA values (0.2, 0.5, 0.7, 0.9 & 1.0) & EPUP values (1, 5 & 20). We apply in cascade 4 times CLVQ3 with RLEN=500000. A total of 480 CLVQ3 simulations were performed.

3.3.2. Sub-Experiment 2.

CLVQ3 was trained with a low initial codebook size (50) for two values of MAV (2 & 5) & BETA (0.2, 0.5, 0.7, 0.9 & 1.0). We apply in cascade 4 times CLVQ3 with RLEN and EPUP having these values: (100000, 1), (500000,5), (500000,10) and (15000000, 20).

3.3.3. Sub-Experiment 3.

We compare best results obtained in experiment 1 with results from applying in cascade LVQ_PAK learning systems (evenint->balance->olvq1->lvq1->lvq2->lvq3) to obtain a codebook of similar sizes. Parameter values are the same of those appeared in the above sub-experiment 1 except RLEN (2500000 in lvq3 and 150000 otherwise).

3.3.4. Experimental Results.

We can observe in table 2 the results of sub-experiments 1 and 2, the learning rate increases smoothly as BETA does and is more stable in time with low values of EPUP. Besides, there is an optimal value of BETA (≤1.0) in terms of generalization (test) error and this value is affected by EPUP. On the other hand, the number of vector pruned basically depends on MAV, although EPUP plays a secondary role (High values of EPUP➔ fewer redundancy). If MAV increases redundant vectors does, even when EPUP is high. If EPUP & RLEN vary dynamically on training, we can obtain reducer codebooks.

Table 3 shows the results of sub-experiment 3. Generalization is better in CLVQ3 than in LVQ's system with the same codebook size, even when CLVQ3 begins on a small codebook. Furthermore, the variance of class training error (training error for each class) is higher in LVQ systems than in CLVQ3 (a relation 15:1 and the variance decreases in CLVQ3 as MAV

augments. Finally, CLVQ3 offers at the end of the growing process a solution more redundant than LVQ's. So after pruning, CLVQ3's codebook is smaller than the pruned codebook of LVQ algorithms.

3.4. Experiment #2

In this section, we apply the CLVQ3 and the pruning algorithms (hereafter, DLVQ3) in upper and lower handwriting characters. In a first phase we have estimated with a validation set the optimal parameters. Ten training runs with DLVQ3 and LVQ algorithms executed in cascade were computed using ten different random sequences of the training set. Table 3 Results can be found on Table 4. We can observe that DLVQ3 generalises better and is more stable (lower test error variance) than LVQ-based system. Also, it has superior performance than other neural approaches reported in (Garris et al., 1997), where test error on upper & lower recognition are 14.7% and 23.1 % with PCA_{64}+PNN & 10.1% and 20.3% with PCA_{128}+MLP respectively.

MAV			2				10				2	5
Initial Codebook Size			50	150	234	280	50	150	234	280	50	50
EPUP			1	1	1	1	1	1	1	1	Dyn	Dyn
BETA			0.7	0.9	0.9	1.0	0.9	0.5	0.9	0.5	0.7	0.7
RLEN= $(5x10^5)x$			11	4	4	4	4	4	3	4	-	-
No Pruning	Final Codebook Size		494	278	361	723	730	830	723	908	723	908
	Training Error		6.12	6.13	4.9	4.51	4.65	4.17	3.18	3.07	7.04	5.85
	Test Error		9.01	8.48	7.92	7.79	8.01	7.81	7.39	7.23	9.28	8.85
Pruning	Final Codebook Size		351	277	351	413	504	484	656	622	208	242
	Training Error		6.22	6.13	4.9	4.51	4.65	4.17	3.18	3.07	7.04	5.85
	Test Error		9.01	8.48	7.92	7.79	8.01	7.71	7.39	7.23	9.28	8.85
	Class training error	Med.	5.86	5.39	4.93	4.47	4.45	4.18	3.01	2.87	6.61	4.96
		Var.	5.17	5.14	3.45	3.23	1.98	1.37	1.47	1.17	4.35	4.76

Table 2. Best Simulation Results of sub-experiment 1-1 (first 8 columns) and sub-experiment 1-2.

Codebook Size		198	245	298	349	412	498	624
Training Error		8.53	8.48	7.55	6.83	6.81	6.25	5.96
Test Error		11.16	11.27	10.66	9.94	10.13	9.67	9.25
Class training error	Med	6.505	6.195	5.83	5.65	5.28	4.69	4.785
	Var	60.64	62.75	56.74	55.7	52.56	53.16	52.49

Table 3. Simulation results of the LVQ_PAK learning strategy for upper handwriting characters (sub-experiment 1-3).

Expert	Upper				Lower			
	Test Error		Final *C* size		Test Error		Final *C* size	
	Med	Var	Med	Var	Med	Var	Med	Var
PCA_{64}+DLVQ3 (without pruning)	7.27	0.016	893	605.5	15.28	0.021	419	31.15
PCA_{64}+DLVQ3 (with pruning)	7.22	0.015	671	870.8	15.27	0.021	322	62.45
PCA_{64}+LVQ_PAK	9.16	0.074	729	0	17.01	0.843	325	0

Table 4. Upper & Lower Handwriting Recognition with DLVQ3 & LVQ algorithms executed in cascade (LVQ_PAK).

4. Discussion

4.1. Bias and Variance reduction

As we pointed out before, the learning system must solve efficiently a trade-off between its approximating power and the information about the problem given by the training set. This can be stated as a bias-variance trade-off (Geman et. al, 1992). *Bias* models the difference between the learner's hypothesis and the optimal solution (the Bayes classifier in our case), while v*ariance* gives account on the dependence of learner's hypothesis on training data and also on optimization algorithms.

In local learning algorithms used for regression, variance are governed largely by the number of training samples that form each local region, while bias is mostly governed by its size (p.3; (Friedman, 1996)). If more data produces a region the hypothesis has a smaller variance; the smaller the region is, the more accurately the approximation is. Bias and variance using LVQ algorithms can also be interpreted in a similar way.

In LVQ algorithms, each local region is the Voronoi region or the region of influence of each codevector. Each codevector is formed due to a certain number of training samples. If there are many codevectors, less training samples contribute to its formation, so they are more dependent on particular training data, although Voronoi regions are smaller and hence we can define classification borders with more accuracy (getting closer to Bayes classifier). Instead, if there are few codevectors, they are less dependent on training sets but Voronoi regions are bigger and the difference between our classifier and Bayes one is greater. Here it appears what we call, loosely speaking, a 'bias-variance' tradeoff since we must achieved a number of codevectors (and an allocation of them) that balance properly between difference of the solution respect to Bayes' ('bias') and its dependence on training data and optimization algorithms ('variance').

The proposed dynamic strategy is concerned with *bias reduction* since it places new algorithms where those regions in which the classification error is greater. Consequently the approximation power of classifier is locally augmented and bias is reduced. However, if do not

stop the constructive process, the algorithm can place too many prototypes to compute them reliably with a finite training set. This leads to over-fitting, that is an excessive tuning to the training set produced by a high estimation error (or high variance) in the computation of prototypes. Consequently, the addition of prototypes is monitored with a validation set to avoid poor generalization due to high variance.

On the other hand, variance could also be reduced using a complementary strategy based on the *local extreme* method (Bermejo, 2000b). This method stabilises locally the class borders of an ensemble of nearest neighbour classifiers. If the generate different classifiers with the dynamic LVQ algorithm using different initial conditions (and ordering of the training samples), we will obtain different solutions due to reaching different local minimum points. Then the application of *local extreme* will reduce the variance of the solutions respect to the optimization algorithm. The synergistic combination of both strategies is displayed in figure 1. This association can be useful as (Bermejo, 2000b) shows.

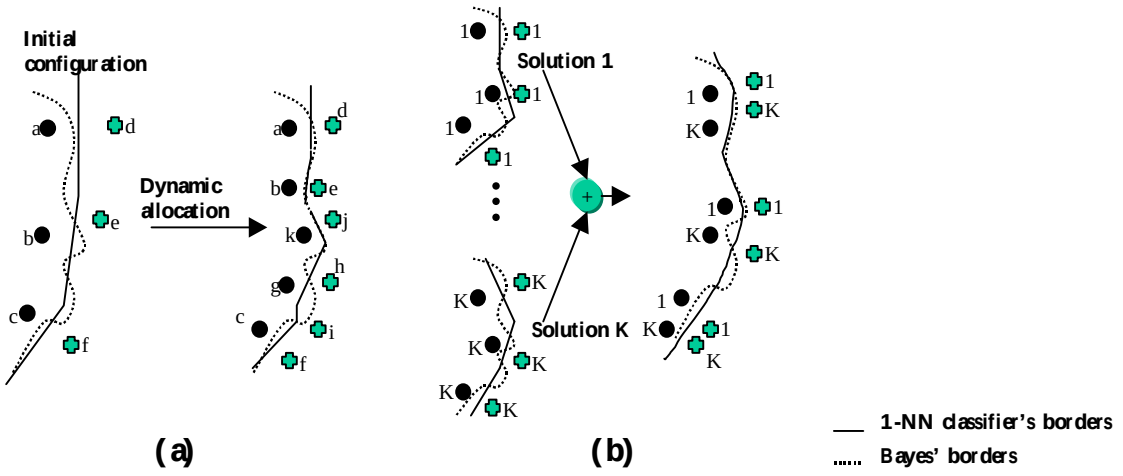

Fig.1. (a) Bias reduction, (b) Variance Reduction.

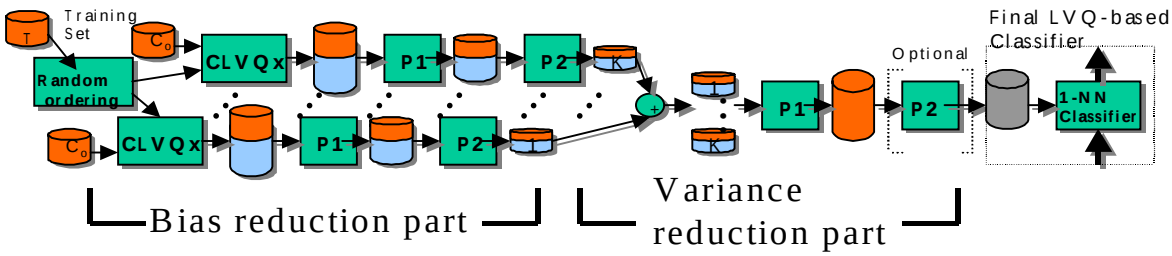

Fig. 2. A possible combination of the dynamic LVQ algorithm and the technique of *local extreme* to achieve a better solution. (Note: P1 and P2 denote pass1 and pass2 of the pruning algorithm.)

4.2. Capacity control

In order to ensure good generalization, the capacity of the learning machine must be controlled (Vapnik, 1995). One possible measure of the capacity of a classifier is its VC dimension h. In nearest neighbour classifiers, h is a function that increases as the input dimension d and the number of protoypes of classifier m augments (Devroye et al., 1996). Vapnik's work ensures a low estimation error if n/h>20 where n is the number of training samples. Unfortunately, this relation cannot be used in real-world sets. Besides, the structural minimization risk (SMR) principle (Vapnik, 1995) (Devroye et al., 1996), which explicitly controls capacity, cannot be applied since the penalised term is too loose for this kind of classifiers. However, since h increases as the number of prototypes, the SMR principle converges to Occam's razor: if we have two NN classifiers with the same training error, choose the simplest (that is the classifier with fewer prototypes). Consequently, the constructive process of the SRM principle that uses a sequence of classifier of increasing capacity, which minimises the training error, could be emulated with our dynamic algorithm. First, we start with a simple solution minimising the validation error. Then we add new prototypes (and hence we increase the capacity of the classifier) and we start again the training phase using the previously computed solution. This growing process ends when the validation error stops decreasing.

4.3. The problem of local minima in LVQ algorithms

The common problem of local minima causes that gradient systems achieve a poor solution. Proper initialisation procedures are a mandatory. However we cannot place the prototypes near a solution if we do not where the solution resides. The dynamic initialisation performed in DLVQ could solve, at least partially, this problem since the new prototypes are placed near their attractors using an estimation based on the training samples that cause its creation and the nearest equilibrium point of the codebook computed in the previous static learning session.

5. Conclusions

We have proposed a dynamic LVQ algorithm concerned with the reduction of bias. DLVQ tries to define more carefully 1-NN classifier's class borders. As a consequence of its dynamic addition of prototypes with the guidance of the training classification error, the generalisation error can be reduced due to the variance of the number of misclassifications of each class is reduced in comparison with LVQ algorithms. DLVQ also uses a validation set to avoid over-fitting and a pruning algorithm that simplifies the final solution.

References

Bermejo, S., Cabestany, J. & Payeras, M. (1998). A New Dynamic LVQ-based Algorithm and Its Application to Hand-written Character Recognition, Proceeding of The European Symposium on Artificial Neural Networks 1998 (ESANN'98), D-Facto Publishers, 203-208.

Bermejo, S. (2000a). Finite-Sample Convergence Properties of the LVQ1 algorithm, the BLVQ1 algorithm and the GLVQ1 algorithm, in Bermejo, S. Learning with Nearest Neighbour Classifiers, Ph.D. Thesis, Barcelona: Universitat Politècnica de Catalunya.

Bermejo, S. (2000b). Local Stabilisation of an Ensemble of LVQ-based Nearest Neighbour Classifiers, in Bermejo, S. Learning with Nearest Neighbour Classifiers, Ph.D. Thesis, Barcelona: Universitat Politècnica de Catalunya.

Bottou, L. & Vapnik, V. (1992). Local learning algorithms, Neural Computation, 4, 888-90.

Bottou, L. (1998). Online Learning and Stochastic Approximations, in David Saal (Ed.) Online Learning and Neural Networks, Cambridge: Cambridge University Press.

Cagnoni, S. & Valli, G. (1994). OSLVQ: a training strategy for optimum-size Learning Vector Quantization classifiers, The International Conference on Neural Networks 1994, 762-765.

Devroye, L., Györfi, L. & Lugosi, G. (1996). A Probabilistic Theory of Pattern Recognition, Berlin: Springer-Verlag

Friedman, J.H. (1996). Local Learning Based on Recursive Covering, Technical Report, Stanford, CA: Stanford University, Department of Statistics.

Garris, M. et al. (1997). NIST Form-Based Handprint Recognition System (Release 2.0), National Institute of Standards and Technology.

Geman, S., Bienenstock, E. & Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma, Neural Computation, 4, 1-58.

Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., & Torkkola, K., LVQ_PAK. The Learning Vector Quantization Program Package. Version 3.1, Helsinki: Helsinki University of Technology, Laboratory of Computer and Information Science.

Kohonen, T. (1996). Self-organizing Maps, 2nd Edition, Berlin: Springer-Verlag.

Niyogi, P. & Girosi, F. (1994). On the Relationship Between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Functions, A.I. Memo No. 1467, Boston, MA: Massachusetts Institute of Technology, Centre for Basis of Computational Learning, Department of Artificial Intelligence.

Vapnik, V. (1995). Learning and Generalization: Theoretical Bounds, in M. Arbib (ed.) The Handbook of Brain Theory and Neural Networks, Boston, MA: MIT Press.