An Online Incremental Learning Vector Quantization*

Ye Xu[1], Shen Furao[1], Osamu Hasegawa[2], and Jinxi Zhao[1]

[1] The State Key Laboratory for Novel Software Technology, Nanjing University
yexu@smail.nju.edu.cn, {frshen, jxzhao}@nju.edu.cn
[2] Imaging Science and Engineering Lab, Tokyo Institute of Technology
hasegawa@isl.titech.ac.jp

Abstract. As described in this paper, we propose online incremental learning vector quantization (ILVQ) for supervised classification tasks. As a prototype-based classifier, ILVQ needs no prior knowledge of the number of prototypes in the network or their initial value, as do most current prototype-based algorithms. It adopts a threshold-based insertion scheme to determine the number of prototypes needed for each class dynamically according to the distribution of training data. In addition, this insertion policy insures the fulfillment of the incremental learning goal, including both between-class incremental learning and within-class incremental learning. A technique for removing useless prototypes is used to eliminate noise interrupting the input data. Unlike other LVQ-based methods, the learning result won't be affected by the sequence of input patterns that come into the ILVQ. Results of experiments described herein show that the proposed ILVQ can accommodate the non-stationary data environment and can provide good recognition performance and storage efficiency.

1 Introduction

Learning Vector Quantization (LVQ) [1] is a supervised learning technique for nearest prototype classification. During recent years, many variants [2] [3] of LVQ have been proposed. These methods have been used as effective pattern recognition tools in a variety of applications.

However, LVQ and LVQ-based methods have some shortcomings. In the first place, it needs a user to initialize the value of the prototypes. Therefore, the learning performance will be affected by the initial values. What's worse, the online learning can not be indeed realized since many people often initialize them with part of training samples. Additionally, the number of prototypes for every class is arbitrarily predetermined without any valuable prior knowledge. Last but not least, these methods can't handle the incremental data set.

* This work was supported in part by the China NSF grant (#60573157, #60723003, and #60775046) and New Energy and Industrial Technology Development Organization (NEDO) of Japan.

T. Theeramunkong et al. (Eds.): PAKDD 2009, LNAI 5476, pp. 1046–1053, 2009.

As described herein, we propose an autonomous learning method named incremental learning vector quantization (ILVQ) for supervised classification. The goal is to develop a network that can operate autonomously in a non-stationary data environment. We need not predetermine the number or value of prototypes before learning. The proposed ILVQ can grow dynamically, learning the number of prototypes for each class needed to solve a current issue and adjusting the number and value of prototypes during training based on a given application. Actually, ILVQ can realize an incremental learning task in the meanings of within-class incremental learning and between-class incremental learning. Within-class incremental learning means that the network learns new knowledge incrementally within the same class, whereas between-class incremental learning means that it incrementally learns new knowledge that comes from a new incoming class. In other words, ILVQ not only adds prototypes within a class (within-class incremental learning); it also incrementally adds the number of classes during training (between-class incremental learning). Furthermore, ILVQ can eliminate the noise from input data. Consequently, the sequence of input patterns in the training set will impart little influence on the ILVQ performance.

2 Incremental Learning Vector Quantization (ILVQ)

In this section, we introduce the basic idea of the proposed incremental learning vector quantization (ILVQ), and give the detailed algorithm.

Shown in Figure 1 is the overall structure of ILVQ. It's a three-layer network. The first layer is input layer that is used to input patterns $\mathbf{x}$ into the network. The second layer is competitive layer. Prototypes are stored in this layer to conduct competitive learning with each other. The competitive result within a class i is:

$$\mathbf{r}_i = (\|\mathbf{w}_1^i - \mathbf{x}\|, \|\mathbf{w}_2^i - \mathbf{x}\|..., \|\mathbf{w}_{n_i}^i - \mathbf{x}\|) \tag{1}$$

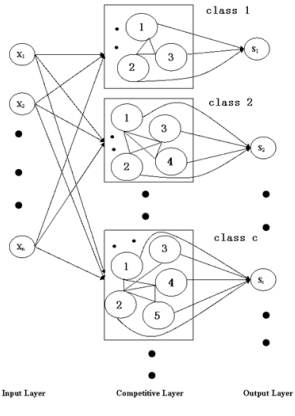

Fig. 1. ILVQ has three layers: input layer, competitive layer and output layer

where n_i is the number of prototypes in class i and $\mathbf{w}_j^i$ is the value of the jth node in class i. The last layer is used to output the result of the network. The competitive result $\mathbf{s}$ that records the winner prototype in each class is

$$\mathbf{s} = (\mathbf{s_1}, \mathbf{s_2}, ..., \mathbf{s_c}) \tag{2}$$

where s_i is the prototype in class i that is nearest to $\mathbf{x}$

$$s_i = \arg \min_{\mathbf{w}_j^i \in classi} r_{ij} = \arg \min_{\mathbf{w}_j^i \in classi} \|\mathbf{w}_j^i - \mathbf{x}\| \tag{3}$$

And we obtain the computed class $label_{\mathbf{x}}$ of the input pattern $\mathbf{x}$

$$label_{\mathbf{x}} = \arg \min_{label_{s_i}} \|\mathbf{w_{s_i}} - \mathbf{x}\| \tag{4}$$

In this network, not only the number of prototypes n_i for each class i but also the class number c are incrementally learned and adjusted. The number and value of prototypes for each class will be reported when the learning is finished. The complete algorithm of ILVQ is given in Algorithm 1.

To fulfill the incremental task, the network should be dynamically growing. Thus we need to sequentially insert proper nodes into the network. However, permanent insertion policy may result in interminable growth of network and overfitting. Therefore, we must decide when and how to insert a new prototype and when insertion is to be stopped. Here, we adopt a threshold method like this: a pattern from new classes that haven't been learned before should be obviously inserted into the network. But for patterns from learned classes, firstly we find the two prototypes "winner" and "runner-up" that are nearest to the input pattern $\mathbf{x}$. Then the insertion operation can be conducted when the distance between $\mathbf{x}$ and $\mathbf{w}_{winner}$ is larger than a certain threshold or distance between $\mathbf{x}$ and $\mathbf{w}_{runnerup}$ is larger than threshold of runner-up. The network may be interrupted by probable noises if we simply take the distance between "winner" and $\mathbf{x}$ into account. So the location of "runner-up" is also considered. This scheme ensures ILVQ to automatically generate proper number of prototypes needed for each class.

However, if the threshold T is too small, all samples will form an isolated prototype. To the contrary, each input pattern will be attributed to only one prototype. To determine T needs some prior knowledge, but we often can not avail them. Furthermore, the threshold should not be constant during training. Otherwise the prototypes in the network may increase endlessly. According to the above discussion, we propose a new method to dynamically compute the threshold distance T in Algorithm 2. We set the average distance between i and other prototypes in the same label with i as i's within-class distance, and set the distance between i and prototypes whose label is different from i's as its between-class distance.

From the discussion in [4], establishing reasonable topological structure is beneficial for us to detect probable noise in the network. We apply the topological representing rule to connect two prototypes when they become the winner and

1. Initialize prototype set G to contain first two input data from the training set, s_1 and s_2.
2. Initialize edge set E storing connection between prototypes to the empty set.
3. Input a new pattern $\xi \in R^n$ to the system.
4. Search set G to find the winner s_1 and runner-up s_2 by $s_1 = argmin_{c \in G} \|\xi - \mathbf{w_c}\|$ and $s_2 = argmin_{c \in G \setminus \{s_1\}} \|\xi - \mathbf{w_c}\|$.
5. **if** $N_{label_\xi} < 2 \bigvee \|\xi - \mathbf{w_{s_1}}\| > T_{s_1} \bigvee \|\xi - \mathbf{w_{s_2}}\| > T_{s_2}$ **then**
6. Insert ξ into set G.
7. Goto Step3 to process the next input pattern.
8. **end if**
9. **if** $(s_1, s_2) \notin E$ **then**
10. Add edge (s_1, s_2) to edge set E and set its age to zero.
11. **end if**
12. **for** s_i in the neighbor area of s_1 **do**
13. Update $age_{(s_1,s_i)} \leftarrow age_{(s_1,s_i)} + 1$
14. **end for**
15. Update winner count $M_{s_1} \leftarrow M_{s_1} + 1$.
16. **if** $label_\xi = label_{s_1}$ **then**
17. Update $\mathbf{w_{s_1}} \leftarrow \mathbf{w_{s_1}} + \eta_1(\xi - \mathbf{w_{s_1}})$
18. **for** s_i in the neighbor area of node s_1 and $label_{s_i} \neq label_\xi$ **do**
19. Update $\mathbf{w_{s_i}} \leftarrow \mathbf{w_{s_i}} - \eta_2(\xi - \mathbf{w_{s_i}})$
20. **end for**
21. **else**
22. Update $\mathbf{w_{s_1}} \leftarrow \mathbf{w_{s_1}} - \eta_1(\xi - \mathbf{w_{s_1}})$
23. **for** s_i in the neighbor area of node s_1 and $label_{s_i} = label_\xi$ **do**
24. Update $\mathbf{w_{s_i}} \leftarrow \mathbf{w_{s_i}} + \eta_2(\xi - \mathbf{w_{s_i}})$
25. **end for**
26. **end if**
27. Delete those edges in set E whose age outstrips the parameter *AgeOld*.
28. **if** the iteration step index is the integer multiple of parameter λ **then**
29. Delete the nodes s_i in set G that have no neighbor node.
30. Delete the nodes s_i whose neighbor node is one and $M_{s_i} < 0.5 \sum_{j=1}^{N_G} \frac{M_j}{N_G}$.
31. **end if**
32. Goto step 3 to continue the online learning process.
33. **return** s et G and set E.

Algorithm 1. Online incremental learning vector quantization

the runner-up of an incoming pattern. In an online environment, the prototypes in the network can not remain their neighbor relationship constantly. The prototypes that are neighboring at an early stage won't be neighboring at a more advanced stage. Thus it is necessary to remove connections which have not been updated recently.

We improve the competitive learning rule to update the value of prototypes. First, we divide the prototypes in the neighbor region of winner into two sets C_1 and C_2 according to their label values. The prototypes that have the same label as $\mathbf{x}$ are put into C_1. And the remaining prototypes are put into C_2. And we update the value of prototypes in this way: when $label_{winner_i} = label_x$, we

1. Compute the within-class distance $T_{i_{within}}$ by $T_{i_{within}} = \frac{1}{N_{label_i}} \sum_{(i,j)\in E \bigwedge label_i = label_j} \|\mathbf{w_i} - \mathbf{w_j}\|$.
2. Find the minimum between-class distance $T_{i_{between}} = \min_{(k_1,i)\in E \bigwedge label_i \neq label_{k_1}} \|\mathbf{w_i} - \mathbf{w_{k_1}}\|$.
3. **if** $T_{i_{between}} < T_{i_{within}}$ **then**
4. Set $T_{i_{between}}$ with the second minimum between-class distance:
5. $T_{i_{between}} = \min_{(k_2,i)\in E \bigwedge label_i \neq label_{k_2} \bigwedge k_1 \neq k_2} \|\mathbf{w_i} - \mathbf{w_{k_2}}\|$.
6. **end if**
7. Go to step2 to update $T_{i_{between}}$ until $T_{i_{between}}$ is no less than $T_{i_{within}}$.
8. Set $T_i = T_{i_{between}}$.
9. **return** T_i

Algorithm 2. Compute the distance threshold T_i of prototype vector i

update winner and prototypes in set C_2. On the contrary, we update winner and those prototypes in C_1. In this case, the directions that these prototypes move are exactly different from the winner: if the winner is moved towards ξ, these prototypes will be moved far from ξ. So the distance between each of the prototype in C_2 and winner will be larger. It is beneficial for us to distinguish those patterns from different classes.

Noise may be interfused in the input data. We need to take some efforts to delete those prototypes in the network that are likely to be noise. Here, we take the strategy in [4]: for every several epochs of learning, we try to remove those nodes with only one or no topological neighbor. In [4], this strategy works well for removing nodes caused by noise without adding computing load.

In Algorithm 1, the learning rate η_1 and η_2 affect the extent the "winner" and its neighbors move towards or far away from the input pattern. In this study, we adopt a method used in [5] to adapt the learning rate over training epoch by $\eta_1 = \frac{1}{M_{winner}}$ and $\eta_2 = \frac{1}{100M_{winner}}$. In this case, the learning rate is time varying and we can make the position of prototype more stable when it becomes a winner for more and more times during training. Due to the divergence of the series $\sum_{n=1}^{\infty} \frac{1}{n}$ and the convergence of $\sum_{n=1}^{\infty} \frac{1}{n^2}$, the learning speed of our network satisfies stochastic approximation conditions.

3 Experiment

3.1 Artificial Data Set

In this section, we conduct our experiment on the two dimensional data set as shown in Fig. 2. Details of the artificial data set can be found in [5]. We apply Fig. 2 to simulate real-world data: 50000 patterns are randomly picked up from all classes, and we add some noise (20% of useful data) into training set. For ILVQ, we set parameters as $\lambda = AgeOld = 16$. During the training, the number and value of prototypes are automatically determined by the algorithm itself. When the training is finished, each class has different number of prototypes:

Table 1. Results for stationary environment

Algorithm	#Prototype	Recognition rate
ILVQ	228	97.5±0.5
LVQ	200	96.7±0.5
LVQ	300	97.0±0.3
LVQ	500	97.3±0.4
G-LVQ	200	96.1±0.2
G-LVQ	300	96.4±0.2
G-LVQ	500	96.5±0.1

Table 2. Results for incremental environment

Algorithm	#Prototype	Recognition rate
ILVQ	291	97.7±0.6
LVQ	200	10.0±0.0
LVQ	300	10.0±0.0
LVQ	500	10.0±0.0
G-LVQ	200	10.0±0.0
G-LVQ	300	10.0±0.0
G-LVQ	500	10.0±0.0

Table 3. Recognition rate and compression ratio of ILVQ under different parameters: Here we always give $AgeOld$ and λ the same value

Parameter	200	300	400	450(optimal)	500	600	700
Recognition rate	97.0	97.2	97.1	97.5	97.4	97.1	97.4
Compression ratio	5.25	6.05	6.07	6.00	6.20	6.57	6.65

class 1, 2, 3, 4 and 5 have 8, 9, 11, 6 and 6 prototypes respectively. It means ILVQ can use fewer prototypes to represent a simple class, and use relative more prototypes to represent a complex class. Such prototypes are connected by edges to represent topological structure of input data, as shown in Fig. 3. We can see not only within-class connections but between-class connections are learned. We highlighted between class connections by black line in Fig. 3.

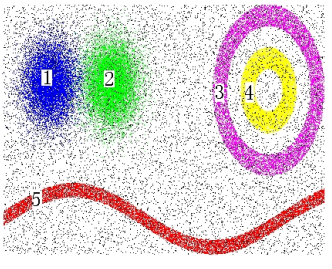

Fig. 2. The artificial data set

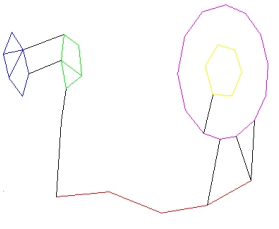

Fig. 3. Topological structure of ILVQ

3.2 Handwritten Digits Recognition

Now we use Optdigits data sets from UCI Machine Learning Repository [6] to test the proposed algorithm. For ILVQ, we apply three times ten-fold cross-validation policy to tune the parameters: $\lambda = AgeOld = 450$. Traditional LVQ and Generalized-Learning Vector Quantization(G-LVQ)[7], an improved version of LVQ, are used to make comparison. For stationary environment and incremental environment, we repeat the algorithms eight times to obtain average results respectively, as shown in Table 1 and Table 2.

Table 4. Recognition rate of experiments: the best and near best performances are highlighted with bold figure

Dataset	ILVQ (Stationary)	ILVQ (Incremental)	LVQ	GLVQ	KMC	NSC	NNC
Glass	**73.3±0.5**	71.4±1.9	68.3±2.0	72.8±0.8	68.8±1.1	70.2±1.5	**72.3±1.2**
Iono	89.5±0.2	89.1±0.2	86.4±0.8	88.6±0.9	87.4±0.6	**91.9±0.8**	86.1±0.7
Iris	**97.1±0.9**	**97.3±0.7**	**96.1±0.6**	**96.7±0.3**	**96.2±0.8**	**96.3±0.4**	**96.7±0.6**
Liver	**67.3±1.3**	**66.6±2.5**	66.3±1.9	**67.4±1.5**	59.3±2.3	62.9±2.3	**67.3±1.6**
Pima	**73.7±1.0**	**73.2±3.3**	**73.5±0.9**	71.1±1.0	58.7±0.9	68.6±1.6	**74.7±0.7**
Wine	73.5±4.1	**73.9±2.5**	72.3±1.5	72.9±4.8	71.9±1.9	**75.3±1.7**	**73.9±1.9**
Average	**79.1±1.3**	**78.6±1.9**	77.2±1.3	**78.3±1.6**	73.7±1.3	77.5±1.4	**78.5±1.1**

Table 5. Parameters and compression ratio (r_c) of experiments

Dataset	ILVQ:Stationary		ILVQ:Incremental		LVQ		GLVQ		KMC		NSC		NNC	
	r_c	(λ,*AgeOld*)	r_c	(λ,*AgeOld*)	r_c	M	r_c	M	r_c	M	r_c	σ^2_{max}	r_c	k
Glass	**25.5**	(786,140)	29.8	(786,140)	45	97	48.7	105	**17**	6	97	0.005	100	1
Iono	25.6	(525,525)	31.7	(530,243)	**6.8**	24	34	120	**4.0**	7	31	1.25	100	2
Iris	19.9	(21,17)	21.9	(66,66)	15	22	22.5	33	**8.0**	4	**7.3**	0.25	100	14
Liver	**6.7**	(16,18)	**4.5**	(5,63)	8.4	29	20.9	72	11	19	**4.9**	600	100	14
Pima	**2.3**	(90,13)	**1.8**	(6,55)	3.4	26	2.6	20	**1.0**	4	**1.7**	2600	100	17
Wine	12	(199,94)	11.3	(124,85)	32	57	10.1	18	29	17	96	4.0	100	1
Average	**15.3**		**16.8**		18.4		23.1		**11.7**		39.7		100	

Table 3 lists recognition rate and compression ratio (the number of prototypes dividing data set size) of ILVQ under different parameters. It means the recognition rate and compression ratio of ILVQ is not sensitive to its parameters. Moreover, we find that in a non-stationary environment LVQ and G-LVQ can only recognize the patterns from the very class (class 10) that are lastly input into the network. The learned patterns (from class 1 to class 9) were destroyed when the LVQ or G-LVQ learned patterns from class 10. But ILVQ can deal with the incremental data environment well because the learned patterns can be always soundly preserved in the network.

3.3 Other UCI Data Sets

Now we sequentially use some other data sets from UCI Machine Learning Repository to test the proposed algorithm. To better evaluate the proposed method, besides LVQ and G-LVQ, we compare it to some other supervised classifiers such as KMC, NNC, and Nearest Subclass Classifier(NSC)[8]. We apply ten-fold cross-validation procedure three times to estimate recognition performance. We list the results of ILVQ in both stationary and incremental environment in Table 4. Because all the above classifiers except NNC can not cope with incremental learning task, we only list the results in stationary environment. The parameters obtained

by ten-fold cross-validation procedure and compression ratio (r_c) are shown in Table 5. We can find that ILVQ realize a best compromise between recognition performance and compression ratio. And the performance of ILVQ in incremental environment is as well as it in stationary environment.

4 Conclusion

In this paper, we propose an online incremental algorithm for supervised learning. The proposed ILVQ applies a threshold-based insertion criterion to grow the network. An innovative method is proposed to adjust the value of the threshold dynamically. Due to the very criterion, the number of prototypes for each class can be generated automatically and adjusted adaptively during learning. Thus users no longer need to predetermine it. ILVQ network can grow gradually and store the learned patterns perfectly so that it can realize the incremental learning well. A proper deletion policy is taken to ensure ILVQ to successfully eliminate the possible noise that comes into the network during learning.

In the experiments, we compare ILVQ with LVQ and other typical prototype-based classifiers. The results show that the recognition rate of ILVQ is better under most conditions. In the respective of compression ratio, the performance of ILVQ is nice as well. Although some algorithms such as KMC and BTS achieve a better compression ratio, we should note ILVQ fulfill the best compromise between classification performance and storage efficiency. Moreover, the experiments demonstrate ILVQ can handle the incremental learning task perfectly.

References

1. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. China Machine Press (2004)
2. De Stefano, C., D'Elia, C., Marcelli, A.: A dynamic approach to learning vector quantization. In: ICPR 2004, vol. 4(4), pp. 601–604 (2004)
3. Pedreira, C.E.: Learning vector quantization with training data selection. IEEE Trans. on PAMI 28(11), 157–162 (2006)
4. Shen, F., Tomotaka, O., Hasegawa, O.: An enhanced self-organizing incremental neural network for online unsupervised learning. Neural Networks 20(8), 893–903 (2007)
5. Shen, F., Hasegawa, O.: An incremental network for on-line unsupervised classification and topology learning. Neural Networks 19, 90–106 (2006)
6. Blake, C.L., Merz, C.J.: Uci repository of machine learning databases. University of California Department of Information, Irvine (1996)
7. Sato, A., Yamada, K.: Generalized learning vector quantization. In: NIPS 1995, pp. 424–429 (1995)
8. Veenman, C.J., Reinders, M.J.T.: The nearest subclass classifier: A compromise between nearest mean and nearest neighbor classifier. IEEE Trans. on PAMI 27(9), 1417–1429 (2005)