Hierarchical LVQ

David Welch
Department of Cognitive Science
University of California San Diego
La Jolla, Ca 92092
david.y.welch@gmail.com

Virginia De Sa
Advisor
desa@ucsd.edu

Abstract

Are there ways we can improve LVQ classification using additional types of data? This question is explored using various modifications to the LVQ algorithm and the SUN attributes database.

1 LVQ1 & LVQ2

1.1 Initialization methods

LVQ 1 can be initialized by choosing input vectors based on pre classification by means of k nearest neighbors, or by some suitable clustering technique such as k-means or self-organizing maps. LVQ 2 requires no initialization but instead takes codebook vectors from LVQ1 as input.

1.2 Learning rule

A basic description of the LVQ1 algorithm found in [1] is as follows: assume we have k codebook vectors, $y_i = (\boldsymbol{z}, t_c)$, learning rate $0 < \alpha(\text{t}) < 1$ and we have inputs $x_m = (\boldsymbol{x}, t_d) \in d$. Then for a given training input **x**, a winning codebook vector **wc** is found, s.t. **wc** = $\{y = (\boldsymbol{a}, t_c) \mid \|\boldsymbol{a} - \boldsymbol{x}\| \leq \|\boldsymbol{b} - \boldsymbol{x}\| \; \forall \boldsymbol{a}, \boldsymbol{b} \in d\}$. It is then updated by the following rule:

(0)
$$\boldsymbol{wc} := \boldsymbol{wc} + \alpha(\mathbf{x}\text{-}\boldsymbol{wc}) \quad \text{if } t_c = t_d$$
$$\boldsymbol{wc} := \boldsymbol{wc} - \alpha(\mathbf{x}\text{-}\boldsymbol{wc}) \text{ otherwise}$$
$$\alpha(t) \textit{ is decreased}$$

This is repeated some number of epochs until optimal error for a given alpha is found. Classification of an input is then achieved using such a trained network by simply finding the winning codebook vector for a given input.

A basic description of the LVQ2 algorithm found in [1] is as follows: this is similar to LVQ1 but takes the codebook vectors of LVQ1 as input. It uses a similar update rule to (0). However, it finds the two nearest codebook vectors $\boldsymbol{wc_1}$ *and* $\boldsymbol{wc_2}$. Then assuming $dist_1$ *and* $dist_2$ are the distances between $\boldsymbol{wc_1}$ *and* $\boldsymbol{wc_2}$ and input **x** respectively. Then if the follow conditions hold it is updated: if $\min(\frac{dist_1}{dist_2}, \frac{dist_2}{dist_1}) > \frac{1-w}{1+w}$, where w is the window size, typically between .2 and .3, $\boldsymbol{wc_1}$ and $\boldsymbol{wc_2}$ belong to different classes but $\boldsymbol{wc_1}$ belongs to the class of **x**.

(1) $$\boldsymbol{wc_1} := \boldsymbol{wc_1} + \alpha(\mathbf{x} - \boldsymbol{wc_1})$$

$$\boldsymbol{wc_2} := \boldsymbol{wc_2} - \alpha(\mathbf{x} - \boldsymbol{wc_2})$$

Then α is decreased by some amount. LVQ2 algorithm is used to fine tune learned classification boundaries found by LVQ1.

2 Hierarchical LVQ

Assumptions about the dataset are as follows: Each sample of dataset 1 is linked one-to-one with a sample of dataset 2. More generally, for any given dataset n, each sample is linked one-to-one with another sample in each of the other n-1 datasets. This means that each linked sample of each dataset corresponds to the same input example. Some modifications of LVQ use specific datasets, which will be noted in the training and initialization sections.

Some notation:

$$\mathrm{D} = \{ d^1, d^2, \dots, d^V \}$$

$$d^v = \{(\boldsymbol{x}_1, t_d^v), (\boldsymbol{x}_2, t_d^v), \dots, (\boldsymbol{x}_m, t_d^v)\}$$

$$C^v = \{ c \mid c^1, c^2, \dots, c^B \}$$

$$c^j = \{y = (\boldsymbol{z}, t) \mid (\boldsymbol{z}_1, t_c^v), (\boldsymbol{z}_2, t_c^v), \dots (\boldsymbol{z}_k, t_c^v)\}$$

$$wc^v(x, d^v) = \{y = (\boldsymbol{a}, t_c^i) \,\big|\, \|\boldsymbol{a} - \boldsymbol{x}\| \leq \|\boldsymbol{b} - \boldsymbol{x}\| \; \forall \boldsymbol{a}, \boldsymbol{b} \in d^v, y \in c^j\}$$

$$LC^j = \{ c^j \mid c^j \in C^v \; \forall v\}$$

$$\alpha(c)$$

D is the set of datasets. $d^v \in D$ is the dataset of feature dataset v=1…V, which contains m samples. C^v is the set of codebooks for feature dataset v=1…V, where B is the number of classes. c^j is the codebook set for class j=1…B, where k is the number of codebooks per class. $wc^v \in C^v$ is the winning codebook vector for feature dataset v. t_d^v is the target of the dataset of feature dataset v. t_c^v is the target of the winning codebook of feature dataset v. LC^j is the set of linked codebooks for class j. $\alpha(c)$ is the learning value of codebook set c^j.

2.1 Initialization

Initialization is achieved by generating a set of codebook vectors per class. These codebook vectors are given by k-means on the subset of data containing this class, where k is the number of desired codebook vectors for a given class. In all cases datasets were normalized to unit length per sample.

There are two methods for linking separate data sets, which will be denoted as type 1 initialization and type 2 initialization respectively. Type 1 initialization involves running k-means separately on each dataset. They are then arbitrarily linked by data point between matching classes. For example, the i'th data point in data set 1 of class 1 is linked to the i'th data point in data set 2 of class 1. The set of links between codebooks of class j is defined as follows: $LC^j = \{ c^j \,\big|\, c^j \in C^v \; \forall v\}$.

Type 2 initialization chooses a suitable data set and initializes it with the standard k-means initialization method. The cluster membership of each data point contained in this initial codebook vector set per class is then used to

determine the initialization of the other data sets codebook vectors. For example, suppose sub data set 1 of class 3 contains 5 points and is clustered into 3 clusters. Further the following membership by indices is obtained for these 3 clusters: cluster 1 = {1, 2}, cluster 2 = {3, 4}, cluster 3 = {5}. Then the other data sets of class 3 are initialized with codebook vectors 1, 2, and 3 corresponding to the means indexed by the respective k-means clusters. In particular, cluster 1 in another data set would contain the mean of the dataset samples with indices 1 and 2. Then each cluster is linked between data sets in their respective class, $LC^j = \{ c^j \big| c^j \in C^v \; \forall v \}$.

Generally it was found that choosing the best single fitting data set as the initial data set for type 2 initialization improved performance over less single fitting data sets.

2.2 Multi update LVQ1/LVQ2

2.2.1 Training & Initialization

Initialization can be of type 1 or type 2. Training works with the standard weight update rule and learning rate α of LVQ1. For input x we find the winning codebook vector wc^v for v=1…V, and for each winning codebook vector of v we apply the following update rules simultaneously:

(2)
$$\boldsymbol{wc^v} := \boldsymbol{wc^v} + \alpha(\mathbf{x} - \boldsymbol{wc^v}) \quad \text{if } t_c^v = t_d^v$$

$$\alpha(c) := \frac{\alpha(c)}{1 + \alpha(c)}$$

$$\boldsymbol{wc^v} := \boldsymbol{wc^v} - \alpha(\mathbf{x} - \boldsymbol{wc^v}) \text{ otherwise}$$

$$\alpha(c) := \frac{\alpha(c)}{1 - \alpha(c)}$$

$$c^j := \boldsymbol{wc^v}$$

Alpha value is shared between all data sets. For example, suppose we have a dataset consisting of two or more different feature extraction methods. Then choosing a random input, the winning codebook vector is found by minimum Euclidean distance to the input for each input simultaneously from the two or more separate datasets. That is, $\boldsymbol{wc^v}(x, d^v)$ is found for each feature dataset v. Note that they may be codebook vectors which are linked to separate codebook vectors in the other codebook vector set.

Then for each winning codebook in each separate data set, check the class against the input and update both that codebook vector and the codebook vector(s) it was linked too in the other data space towards or away from the corresponding input vector set.

Multi update LVQ2 works in a similar manner according to the same rules as the standard LVQ2 algorithm. That is, if the two closest winning codebook vectors fall within a window, one belongs to the input class and one does not, then they are updated towards and away respectively for each codebook vector in the set of linked codebook vectors LC^j.

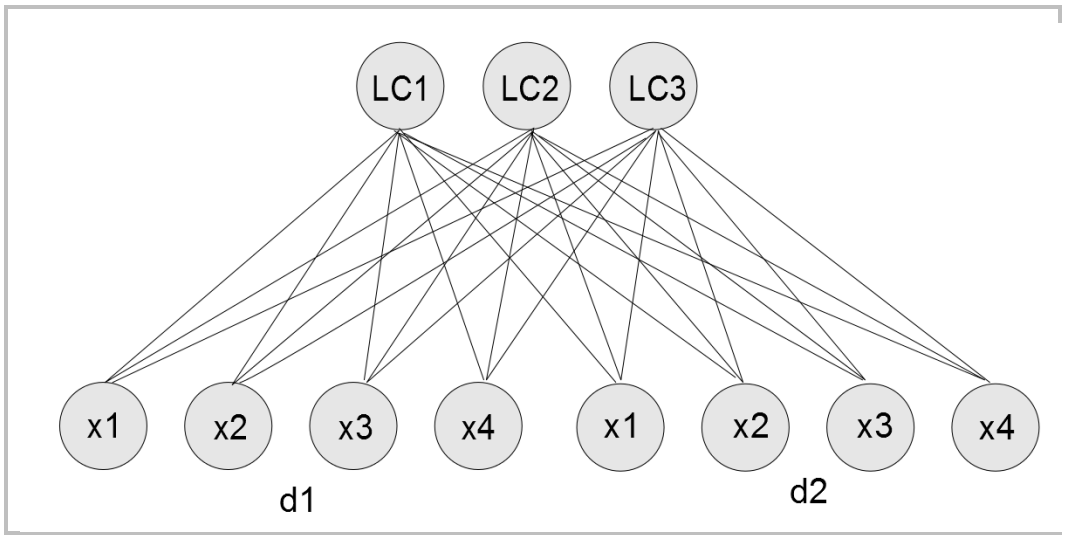

Figure 1: Multi update LVQ

Example multi update LVQ with 3 codebook classes and 4 inputs from each dataset, where d1 and d2 correspond to feature dataset 1 and 2, where each respective set of inputs is from them. LC1,2,3 refer to the set of linked codebook vectors.

2.3 Multitask LVQ

2.3.1 Training & Initialization

Initialization is type 2 only using some feature data set and a suitable attribute dataset. Training works with the standard weight update and learning rate rule of LVQ1. Alpha values are separate for each data set. Multitask LVQ updates a weight in the following manner with respect to the input: a random input is received and a winning codebook vector is found. It is first updated in the regular manner on the first feature data set as in equations (1). Then the following is determined:

$$t(ind) = \max\{x_j \in d^v \; \forall j = 1 \dots m\}$$

Then, the maximum value t(ind) in the attribute vector corresponding to the second dataset of the winning codebook is found, where ind is the index of this value. If t(ind) >= d(ind) where d(ind) is the index of the j'th feature of the input dataset, then it is updated towards by some small alpha, otherwise away, and alpha is decremented similar to (0).

2.4 Multilayer LVQ

2.4.1 Training & Initialization

Initialization is type 1 or 2. Training works with the standard weight update and learning rate rule of LVQ1. Alpha values are reset at every layer, and are shared in the last layer. Multilayer LVQ is constructed as follows: the first (hidden) layer consists of single LVQ networks with a given data set. Each of these LVQ networks feeds its set of codebook vectors C^i as input to a single Multi update LVQ, which is then trained on these inputs. Classification is achieved by considering only the top layer codebook for classification.

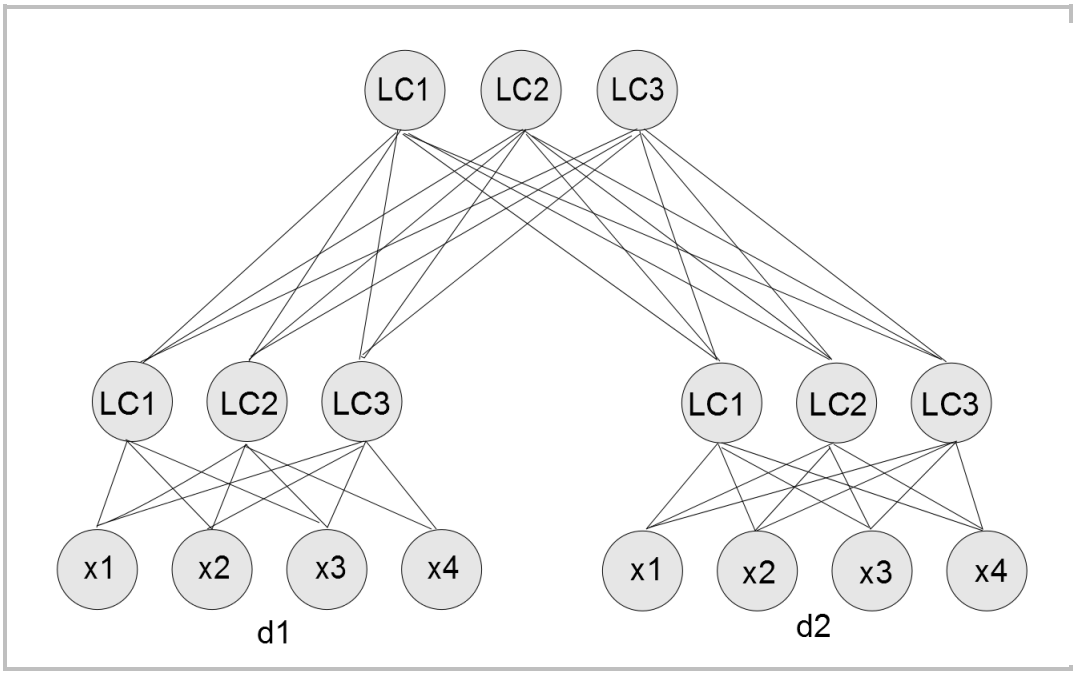

Figure 2: Multilayer LVQ

Example Multilayer LVQ with 3 codebook classes in the output and hidden layer and 4 inputs from each dataset, where d1 and d2 correspond to feature dataset 1 and 2, where each respective set of inputs is from them. LC1,2,3 refer to the set of linked codebook vectors.

3 Reduced data Multi update LVQ

It was found that in fact having incomplete data in some datasets still improved classification. In particular, you need approximately 53% of the total inputs as paired inputs to improve classification. That is, if we have a full dataset and a dataset which only contains only 9 of every 17 linked samples from the full dataset, we find improved classification accuracy by at least 1%. Input without paired input was treated as if it were LVQ1 with some separately determined low alpha value, typically much smaller than the standard alpha.

4 Dataset

The dataset used was the SUN attribute dataset [2] for scene recognition. This dataset consists of several feature extracted datasets and 102 worker annotated attributes per sample. The attribute dataset in particular is utilized in the multitask method, but also interestingly yields high performance in the other methods. Each attribute vector consists of 1/3, 2/3, or 1 value corresponding to a rough probability assigned by a human worker towards the attribute label of a given scene. The specific feature datasets that were used for this paper was HoG, SSIM, and attribute dataset. For full details on the individual feature datasets and parameters, see [3]. A 20 class subset of the 700 categories was chosen for classification, each with 20 examples per class.

5 Parameters and Performance

Figure 3 shows performance of all non-multilayer LVQ's using 5 codebooks per class, varying alpha levels and 6000 epochs. All classification was done

using the HoG dataset. Each accuracy calculation was determined using an average of 3 separate 10-fold cross validation tests. The Multilayer LVQ gave good results using the attribute and HoG dataset with the following parameters: 10 codebooks generated in the first level by both networks using .6 and .01 alpha for the attribute and HoG dataset respectively, and 5 codebooks per class at the final layer with .05 alpha. All layers were run 6000 epochs. The performance for this was .4567. The performance after applying Multi update LVQ2 to the Multilayer algorithm with 1000 epochs, .05 learning rate, and a .25 window yielded .4725 accuracy.

Figure 4 shows performance of these same algorithms with given parameters after running Multi update LVQ2 for 1000 epochs, .05 learning rate and .25 window with a given alpha value of the previous algorithm.

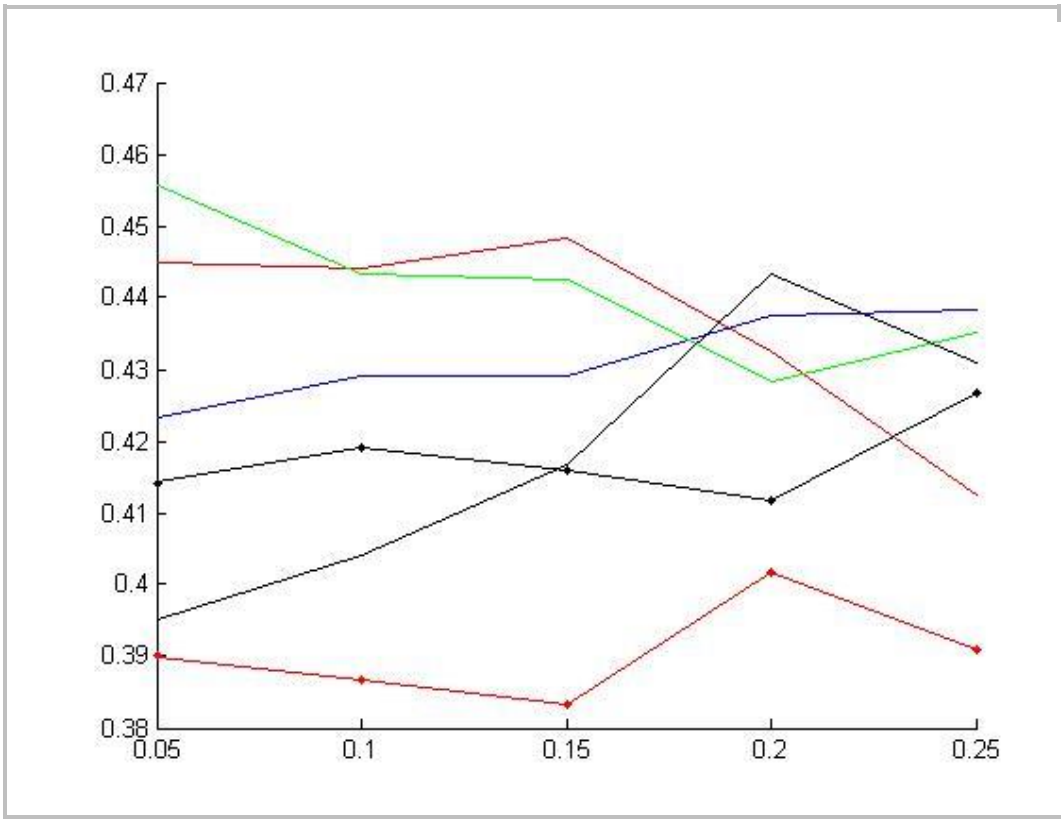

Figure 3: Accuracy vs. Alpha

Graph of accuracy vs. alpha values. The vertical axis is accuracy, the horizontal axis alpha. Each LVQ is initialized with type 1 initialization unless otherwise specified. **Red**: Multi update LVQ w/ HoG/Attributes dataset. **Blue**: Multi update LVQ w/ HoG/SSIM dataset. **Green**: Multi update LVQ w/ HoG/SSIM/Attribute dataset. **Black**: Multitask LVQ w/ HoG/Attribute dataset. **Red dots**: Regular LVQ w/ HoG. **Black dots**: Type 2 Initialized Multi Update LVQ w/ HoG/Attributes dataset.

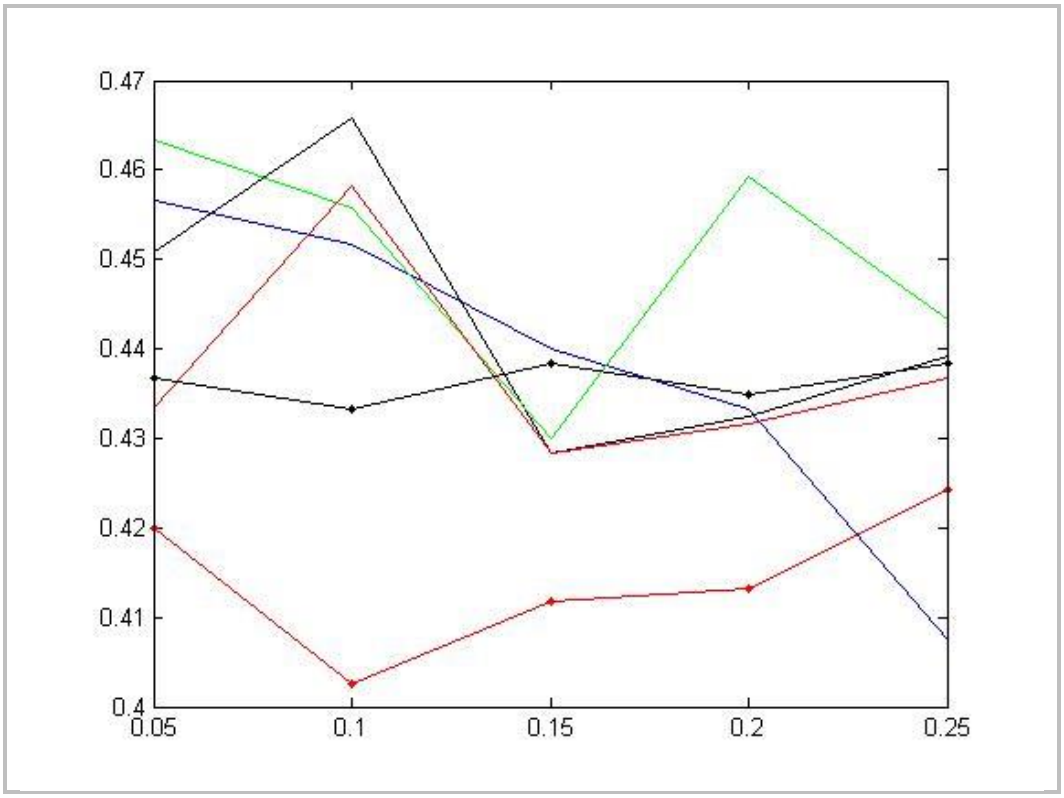

Figure 4: Accuracy vs. Alpha

Graph of multi update LVQ2 fine tuning accuracy. It contains the same legend as figure 3.

6 Discussion

All modifications performed better than the standard LVQ1/2 algorithm. Type 1 initialization performed better than type 2 initialization in all cases. This can be explained by the fact that type 1 initialization averages inputs within each class, so that separation is achieved in each space, while in type 2 initialization one class separation cluster criteria could mean less separable boundaries in the other. That is, cluster membership for a given space may not be the optimal cluster membership for another. Multilayer LVQ performance can be viewed as successive averaging of the boundaries at each layer, thereby increasing generalization. Overall, these algorithmic methods may be helpful in cases were additional data is available yet expensive to gather. In particular, this method improves classification using any particular feature data input; it does not require all feature data inputs after training to give better performance than the traditional LVQ algorithm. However, in cases where all such data is available after training, it would be a simple extension to aggregate votes among individual codebook outputs to improve classification performance.

References

[1] Kohonen, T. (1992) LVQPAK. http://cis.legacy.ics.tkk.fi/hynde/lvq/

[2] Genevieve Patterson, James Hays. SUN Attribute Database: Discovering, Annotating, and Recognizing Scene Attributes. Proceedings of CVPR 2012.

[3] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In CVPR, 2010.