

# Kohonen's Self-Organizing Feature Maps and Linear Vector Quantization: A Comparison

Kiran Bhowmick

Assistant Professor

Department of Computer Engineering,  
D. J. Sanghvi College of Engineering,  
Mumbai, India

Mansi Shah

Student

Department of Computer Engineering,  
D. J. Sanghvi College of Engineering,  
Mumbai, India

## ABSTRACT

Machine learning has evolved over the past years to become one of the major research fields in Computer Science. In simple words, Machine Learning can be described as the process of training a machine to learn from its outputs and improvise itself in order to optimize its outputs. One of the major branch of machine learning is Unsupervised Learning where in the machine is not given any kind of feedback but is expected to learn on its own ("without Supervision"). This paper aims at describing in detail and thus comparing two such neural networks: Kohonen's Self Organizing Feature Maps (KSOFM) and Linear Vector Quantization (LVQ).

## General Terms

KSOFM, LVQ, Unsupervised Learning, Pattern Recognition

## Keywords

KSOFM, LVQ, Machine Learning, Unsupervised Learning, Pattern Recognition, Comparison

## 1. INTRODUCTION

Unsupervised Machine Learning, as the name suggests, is a machine learning technique in which no form of feedback is provided to the machine and instead it is expected to give accurate outputs by learning on its own. Unsupervised machine learning can be easily understood by understanding its major application: Clustering. It groups the given inputs based on their similarity and differences among themselves, without any external help (feedback). Such learning networks are also called Self Organizing networks. In order to ensure that we get one and only one output, the neurons in the net are said to compete among themselves so that eventually one neuron emerges as a winner. Thus, such networks are also called competitive nets.

There are several neural networks that come in this category, such as Maxnet, Mexican Hat, Hamming net, Kohonen's self organizing feature map, Learning vector quantization etc. The learning algorithm used in most of these nets is known as Kohonen learning. In this learning, the units update their weights by forming a new weight vector, which is a linear combination of the old weight vector and the new input vector. Also, the learning continues for the unit whose weight vector is closest to the input vector. The weight updating formula for output cluster unit  $j$  is given as: <sup>[1]</sup>

- of topological neighborhood is also reduced. If the learning rate has decreased considerably such that the network won't be able to learn significantly any more, then the training is stopped. Else the same process is repeated with the new learning rate.

$$w_j (new) = w_j (old) + \alpha [x - w_j (old)]$$

where  $x$  is the input vector,  $w_j$  is the weight vector for unit  $j$  and  $\alpha$  is the learning rate, which decreases (often exponentially) as training progresses. As the learning rate decreases with time, the amount of significant change in the weight also reduces. Thus, a net learns faster initially as compared to gradual and slow learning towards the end.

Having seen the basics, we will now discuss two such networks in detail and compare their performance for an application: Pattern Recognition.

## 2. KOHONEN'S SELF-ORGANIZING FEATURE MAP

Feature Mapping is the process, which converts the patterns of arbitrary dimensionality into a response of one- or two-dimensional arrays of neurons, i.e., it converts a wide pattern space into a typical feature space. <sup>[1]</sup> The network performing such a mapping is called a feature map. The weight vector of the output unit serves as an exemplar vector.

Simple working of this algorithm can be explained in following logical steps:

- Suppose there are  $n$  inputs, which need to be clustered into  $m$  different clusters. Firstly we will initialize  $m$  different weight vectors to random values. The learning rate of the network is also initialized to a predefined value. Also another parameter, which is initialized, is the Radius of topological neighborhood.
- One by one, each input is compared with each of the weight vectors. The vector closest to the current input is chosen for updating by the Kohonen's Learning Rule formula. In case of multiple vectors having equal distance from the input, the first one is chosen for simplicity.
- Along with the "winning" weight vector, those vectors, which lie in the nearby neighborhood defined by the Radius of topological neighborhood, are also updated with the same formula.
- Above two steps are performed for each of the  $n$  inputs.
- The learning rate is decreased by a certain amount dependent on time since training started. The radius

For testing purposes, the weights obtained in the training phase are used. By simply finding out the unit whose weights are most 'similar' to that of the input, we can identify the cluster to which that particular input belongs.

### 3. LINEAR VECTOR QUANTIZATION

Learning Vector Quantization though belongs to the category of competitive networks, it has a touch of supervised learning to it. Both KSOFM and LVQ use Kohonen’s weight updating formula, but both differ in their approach to train the network. And hence their performance varies. As stated earlier, LVQ has a touch of supervised learning to it. Thus, prior knowledge should be known about the input vectors. Unlike KSOFM, LVQ weight vectors are not randomly initialized. For LVQ, each input vector should be associated with a predefined fixed class to which it belongs.

LVQ training can be explained with the following steps:

- Suppose there are n inputs to be classified into m different output classes. Each input for training is associated with one of the output classes. The weight vectors are initialized with the first m input vectors, which belong to different classes.
- For training weights, Euclidean distance is calculated between the input and weight vectors. The node J with minimum distance is chosen as the winning node.
- Next, before updating weights, the output class of the input T is compared with the winning output class  $C_j$

IF  $T = C_j$  THEN

$$w_j(new) = w_j(old) + \alpha [x - w_j(old)]$$

i.e. the weight vector is moved closer to the input vector

IF  $T \neq C_j$  THEN

$$w_j(new) = w_j(old) - \alpha [x - w_j(old)]$$

i.e. the weight vector is moved away from the input vector

- The above two steps are repeated for every input vector
  - After one such iteration, the learning rate is decreased using the formula
- $$\alpha(t) = \alpha(t-1) * 0.5$$
- The training is stopped when the stopping criteria is met. The stopping criteria could be any fixed number iterations or a sufficiently small value of  $\alpha$ .

As compared to KSOFM, LVQ at least theoretically proves to be giving a corrective measure to the network if and when it is found to be performing wrongly. In the next section we shall see, how these two algorithms performed practically.

### 4. IMPLEMENTATION

In order to compare the two algorithms closely, we chose a simple yet effective application whose results could easily be understood and interpreted: Pattern Recognition.

For each of the algorithm, a simple MATLAB code was written to train and test vectors belonging to one of the following patterns:

- Star (\*)
- Cross (+)
- X (x)
- Square
- Diamond

Moreover, to purely compare their performance, both set of weight vectors were initialized to the same values. Each Pattern was represented by a 5\*5 binary matrix, which is represented as a single row vector.

Both algorithms were trained using 10 vectors for 19 iterations. The results obtained thereafter on different test cases are tabulated and discussed in the next section.

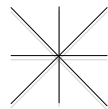
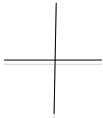
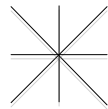
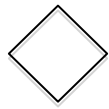
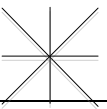
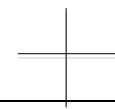
### 5. RESULTS AND DISCUSSIONS

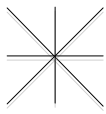

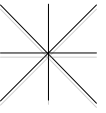
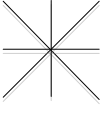
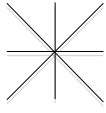
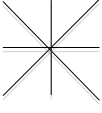
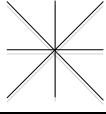
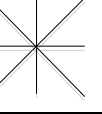
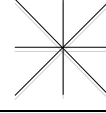
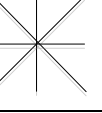
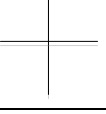
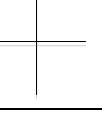




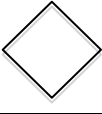
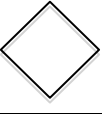
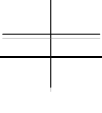
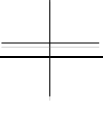
Out of all the test cases tabulated in the table, the first three are of most interest to study the performance of both the networks, as remaining inputs give the same output in both the networks.


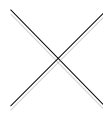

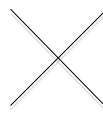
On close observation, following points are inferred from the experiment:

- In the first 4 inputs, KSOFM gives the result as Star, whereas LVQ doesn’t, but subsequently, both give Star as an output for other inputs. In the star pattern representation, if we consider the outermost border of the 5\*5 matrix, then there are 8 bits set to ‘1’. In the first 4 inputs, out of these 8 bits, at most only 4 bits are set to ‘1’, and LVQ doesn’t classify it as Star. Whereas in inputs 5-8, at least 5 bits or more are set to ‘1’, and LVQ successfully labels those patterns as a Star.
- Moreover, in the first input, if we see the pattern, as shown below, it closely resembles a Star and KSOFM rightly labels it so. But LVQ labels it as a Cross (+). This again proves that LVQ gives more importance to the border pixels. For a cross, only 4 border pixels are missing, whereas for a star 8 border pixels are missing. Thus, even though there are 4 extra pixels as compared to an ideal cross (+), LVQ still labels it as a cross.

**Table 1: Results of Pattern Recognition**

Sr. No.	Input Pattern:	Classified as:	
		Using KSOFM	Using LVQ
1	0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0		
2	0 0 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0 1 0 0		
3	0 0 1 0 0 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0		

	00100		
4	10001 01110 01110 01110 10001		
5	10101 01110 01110 01110 10001		
6	10001 01110 11110 01110 10001		
7	10100 01110 11111 01110 00100		
8	10101 01110 11111 01110 00000		
9	00100 00100 10001 00100 00100		
10	10101 00001 10000 00001 10101		
11	10101 01011 10000 01011 10101		
12	00100 01011 10010 01011 10100		
13	00101 00000 11110		

	01001 00100		
14	10001 01110 01110 01110 10001		
15	10001 01110 00100 01110 10001		

- In the second, third and fourth pattern, LVQ labels them as a diamond, cross and X respectively because whole of those patterns are resembled along with some extra pixels. LVQ, because of the border pixel weightage as mentioned above, again doesn't consider the possibility of those patterns being an incomplete star, which KSOFM does.
- Furthermore, one may also question that the third input pattern, has both patterns, a cross and a diamond complete in it then why does LVQ label it as a cross. This is because, for a cross, it just has 4 more extra pixels enabled, whereas for a diamond, the pattern has 5 extra pixels. Thus, cross can be termed as a "closer" label than diamond.
- In the remaining inputs, there isn't much ambiguity and both the networks give the same output as expected.

It can, thus be concluded that, while LVQ appears to be "safe" in predicting labels and needs some threshold beyond which it can consider the pattern to be an incomplete one, on the other hand KSOFM considers all the possibilities and gives much satisfying results even though it being unsupervised in nature. Thus, if not possible to give training data with expected output, KSOFM works out as a much better choice. However, it also depends on the need of the application and whether the inputs will always be in distorted form or not. If there isn't much variance between the trained and testing input, LVQ could also be an option.

## 6. REFERENCES

- [1] Principles of Soft Computing, Wiley, S. N. Sivanandan & S. N. Deepa.