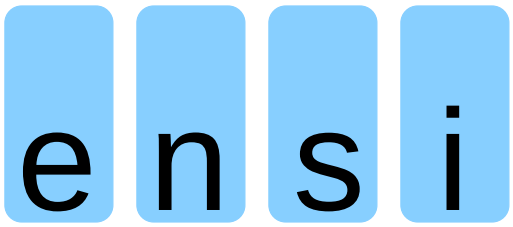
Understanding Kohonen Networks

Imed CHIHI

January 1998



National School for Computer Sciences

.

Abstract

Kohonen neural nets are some kind of competitive nets. The most commonly known variants are the Self-Organizing Maps (SOMs) and the Learning Vector Quantization (LVQ). The former model uses an unsupervized learning, the latter is an efficient classifier.

This paper tries to give, in simple words, a clear idea about the basis of competitive neural nets and competitive learning emphasizing on the SOMs and some of their real-world applications. It should not be considered as an exhaustive reference but as a simple introductory.

Contents

1 Introduction

TEUVO KOHONEN is a Finnish researcher very famous for his work and contributions to neurocomputing. The term Kohonen nets is generally used ambigously because it refers to multiple neural networks especially *Vector Quantization*, *Self Organizing Maps*, and *Learning Vector Quantization*. Indeed, many other networks are known as Kohonen nets, these include:

- DEC−Dynamically Expanding Context,
- LSM−Learning Subspace Method,
- ASSOM−Adaptive Subspace SOM,
- FASSOM−Feedback-controlled Adaptive Subspace SOM,
- Supervised SOM,
- LVQ-SOM.

Almost all of them are competitive nets. In the next we present the major principles of competitive neural nets without going deep with mathematical details. In sections 4, 5 and 6, we give a detailed presentation of the Self-Organizing Maps and the Learning Vector Quantization. Then, we present some real-world applications.

2 Competitive Neural Nets

Competition is one of the most common neurocomputing paradigms. It was first used in Rosenblatt's networks and studied later by Grossberg (1960). To make a long story short, competition consists in letting neurons "fight" till one wins, and then to "reward" the winner.

2.1 Architecture

Competitive nets are structured (in their simplest form) as a couple of layers. One for input, all it does is to reflect the input patterns to the whole network just like the input layer in Multi-Layer Perceptrons (MLPs). The second layer is for competition, each neuron receives some excitation from the input and produces some response. The response of a neuron c to the excitation varies depending on the weights w_{ci} where i covers all the input layer neurons.

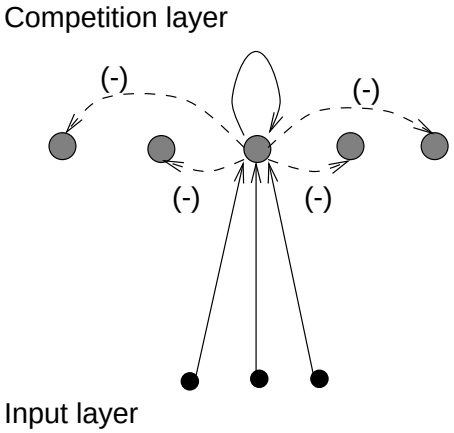


Figure 1. The input layer passively distributes the input vector (pattern) to all the competition layer's neurons. Each neuron is connected to some neighborhood with either excitatory (solid lines) or inhibiting (dashed lines) lateral links.

To be worth called competitive, a neural net should have the following features,

- all the neurons have different activations to a given input pattern. This is generally granted by the initial settings,
- the activation functions are bounded,
- a concurrency mechanism exists. This is granted by negatively weighed connections.

2.2 Learning and Competitive Dynamics

Each cell on the competition layer receives the same set of inputs. If an input pattern x is presented to the input layer, each neuron would calculate its activation as follows,

$$s = \sum_i w_i x_i$$

One of the neurons, say k, will have the highest activation. The lateral connections are the means by which competition occurs. That is, node k will get even more and more active any time the vector x is presented. If this process is iterated with the same vector x, node k will strengthen its activation while others will have their activation falling off to zero. Note that the

lateral connections learn nothing, they are there just to specify the concurrency topology, that is, what neurons should get active together.

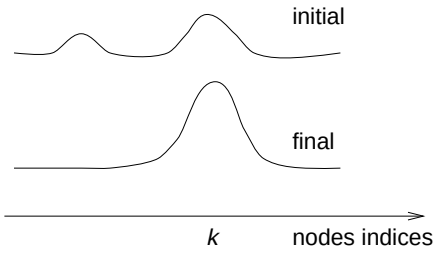


Figure 2. With competitive dynamics, the situation evolves in such a way that the winning cell gets more and more specialized for a given input pattern.

The dynamics are governed by the differential equation:

$$\frac{dw_{ij}}{dt} = -\alpha w_{ij}(t) + \beta f(Y_i)(X_j - w_{ij})$$

The factor $(X_j - w_{ij})$ makes w_{ij} go close to X_j, where $f(Y_i)$ insures that a non-active neuron remains non-active so that only the winning nodes gain more activation.

In the performance phase of a competitive learning network, the index of the winning neuron is typically the output of the network.

3 The Biological Background

It often occurs that sensory inputs may be mapped in such a way that it makes sense to talk of one stimulus being 'close to' another according to some metric property of the stimulus [4]. In the visual area 1 of the mammalian brain, cells are sensitive to the orientation of the presented pattern. That is, if a set of alternating black and white lines is presented, a particular cell will respond the most strongly when the set has a particular orientation. The response will fall quickly as the lines of the set are moved off that preferred orientation to some other ones. This was practically established by the work of Hubel and Weisel (1962). To make the closeness make sense, one should define some metric or measure of the stimuli.

Competitive nets simulate this behavior by making each cell learn to recognize some particular input patterns. An interesting feature with biological nets is that cells trained to recognize the same pattern tend to come close to one another. In the visual cortex, cells tuned to the same orientation are placed vertically below each other perpendicularly to the surface of the cortex.

4 Self-Organizing Maps

The SOM defines a mapping from the input data space $\Re^n$ onto a two-dimensional array of nodes. To every node i, a reference vector $m_i \in \Re^n$ is assigned, in competitive networks jargon this is called a *codebook*. An input vector $x \in \Re^n$ is compared with the m_i and the best match is defined as "response": the input is thus mapped onto this location [2]. The best matched one is the closest to vector x according to some metric.

$$m_i = \begin{pmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{pmatrix}$$

Figure 3. Weights vector associated with a node i from the competition layer. This expression of m_i assumes that the input data space is n-dimensional.

We can say that the SOM is a projection of the probability density function of the high-dimensional input data space onto the two-dimensional display (the map). Let x be an input data vector, it may be compared with all the m_i in any metric, in practice the Euclidean distances $\|x - m_i\|$ are compared to pick up the best matched node (the one that minimizes the distance). Suppose now that the node closest to x is denoted with the subscript c [2].

During learning, the nodes that are topographically close in the array up to a certain distance will activate each other to learn from the same input. Kohonen has shown that the suitable values of m_i are the limit of the convergence of the following learning process (at each presentation of a new input learning vector, m_i's are updated):

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)]$$

where t is the discrete-time coordinate. h_{ci} is the *neighborhood kernel*; it is a function defined

over the competition layer's neurons, usually it looks like $h_{ci}(t) = h(r_c - r_i, t)$, where r_c and $r_i \in \Re^n$ are the radius vectors of nodes c and i, respectively, it is a measure of the topographic distance between i and c. As that distance increases, $h_{ci} \to 0$. The most used neighborhood distributions are *the bubble* and the *Gaussian* forms.

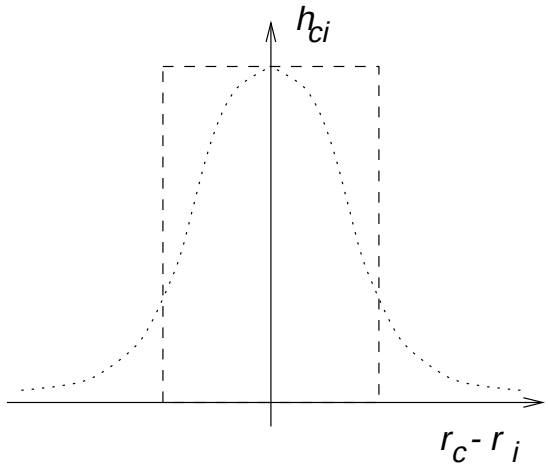
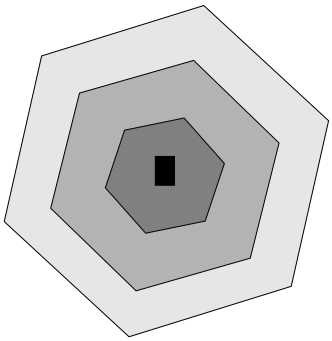


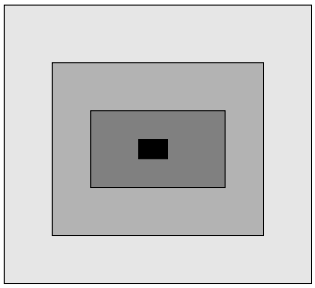
Figure 4. Layout of the neighborhood, bubble (dashed-line) or Gaussian (dotted-line).

In addition to the neighborhood kernel, a SOM user should define the shape of the boundaries of that neighborhood.

By defining the neighborhood kernel, we define the "radius" of the surrounding region into which the nodes should be subject to an update. This is not necessary a circular-shaped; in practice it is either hexagonal or square.



(a)



(b)

Figure 5. Neighborhood boundaries, (a) hexagonal and (b) square.

5 Learning Vector Quantization

Learning Vector Quantization (LVQ) is a technique which improves the performance of self-organizing networks in classification tasks [6]. Typically, it consists in iteratively fine tuning the decision boundaries which are, in some sense, the boundaries of the map regions corresponding to the different classes. Just like shown above, in the basic learning rule, the weight vector w of a neuron is modified in response to an input vector x.

$$w(t+1) = w(t) + \alpha(t)[x(t) - w(t)]$$

Many variations of this basic algorithm were proposed, the most known are LVQ1, LVQ2, LVQ3 and OLVQ, all proposed by Kohonen.

5.1 LVQ1

LVQ1 is very similar to the basic learning algorithm with a slight modification to support classification [6]. For a given training vector x with class c_x, the nearest neighbor w_c, with class c_c is selected as the winning codebook and then updated according to the following equations.

If $c_c = c_x$ (x is classified correctly),

$$w_c(t+1) = w_c(t) + \alpha(t)[x(t) - w_c(t)]$$

If $c_c \neq c_x$ (x is classified incorrectly),

$$w_c(t+1) = w_c(t) - \alpha(t)[x(t) - w_c(t)]$$

One effect of the LVQ1 algorithm is to move the codebooks off the uncertainty region reducing ambiguity. Uncertain regions are located just in between two adjacent portions of the map where distinct classes are mapped.

5.2 LVQ2

The LVQ2 algorithm attempts to better approximate the decision boundaries by making adjustments to pairs of codewords. For each training vector x with class c_x, LVQ2 uses a nearest neighbor selection scheme to choose the closest (winning) codeword w_w with class c_w, and the second closest (*runner up*) codeword w_r, with class c_r. If the class of x is different from the winning class c_w but the same as the runner up class c_r then the codewords are modified according to the following equation.

If $(c_x \neq c_w), (c_x = c_r)$ and x falls within the "window",

$$w_w(t+1) = w_w(t) - \alpha(t)[x(t) - w_w(t)]$$

$$w_r(t+1) = w_r(t) - \alpha(t)[x(t) - w_r(t)]$$

For the above equation to apply, the training vector x must fall within the boundaries of a "window" defined in terms of the relative distances d_w and d_r from w_w and w_r respectively. This "window " criterion is defined below,

$$min(\frac{d_w}{d_r}, \frac{d_r}{d_w}) > s,$$

where

$$s = \frac{1-w}{1+w}$$

5.3 LVQ3

The LVQ2 has the so called *codewords drifting away* disadvantage with lengthy running learning. The LVQ3 is a combination of both LVQ1 and LVQ2 to produce a stable and efficient algorithm. Like LVQ2, for each training vector x with class c_x, LVQ2 uses a nearest neighbor selection scheme to choose the closest two codewords w_i with class c_i, and w_j with class c_j. If the class of x is the same as one of the winning codeword classes and different from the other, an update scheme similar to LVQ2 is employed with the same window criterion. If the classes of both codewords are the same as the class of x, then an update rule similar to that used by LVQ1 is applied.

If $(c_x \neq c_i), (c_x = c_j)$ and x falls within the "window",

$$w_i(t+1) = w_i(t) - \alpha(t)[x(t) - w_i(t)]$$

$$w_j(t+1) = w_j(t) + \alpha(t)[x(t) - w_j(t)]$$

For $k \in \{i, j\}$, and $c_x = c_i = c_j$

$$w_k(t+1) = w_k(t) - \epsilon\alpha(t)[x(t) - w_k(t)]$$

The scalar learning constant ϵ depends upon the size of the window. Reasonable values for ϵ range between 0.1 and 0.5. With LVQ3, codewords placement does not change with extensive learning. It is said to be *self-stabilizing* [6].

6 Applications

6.1 SOMs in High Performance Computing

Tomas Nordström from the Department of Systems Engineering and Mathematics, Lulea University of Technology, Sweden is leading a team working on the design of parallel computer architectures to support neural networks algorithms [7]. They are building a parallel computer especially tuned for SOM, which they called REMAP.

Many other computers were built to do computation with SOM. These architectures are expected to have high broadcasting capabilities. Indeed all it is needed into a SOM is to distribute weights and to calculate a neighborhood, which involves continuous broadcasting.

CNAPS (Connected Network of Adaptive Processors) manufactured by Adaptive Solutions. It is a 256 *processing element* (PE) machine with a broadcast interconnection scheme. CNAPS performs up to 10240 Million Instructions Per Second (MIPS) on dot-product operations and about 183 Million Connection Updates Per Second (MCUPS).

CM (Connection Machine) Manufactured by Thinking Machine Corp., it runs at 48 CUPS (for CM-2). SOMs are basically computation bound[1], that's why in a high-communication variant of SOM where broadcast could not be used efficiently, a 30 node Transputer machine would run at one third of the CM-2 speed.

[1] The useful computation time dominates the time spent on communications.

MasPar Uses between 1024 and 16384 PEs. Achieves a peak performance of 1500 MFlops (on the 16k-PE version).

Warp A one-dimensional array of 10 or more powerful PEs developed at the Carnegie Mellon University. It has a performance of about 10 MFlops per PE.

TInMANN Is an all digital implementation of Kohonen's learning. Indeed, Van den Bout and others have proposed a modification to the SOFM model making it possible to build very simple implementations.

6.2 LVQ in Image Compression

Here are the results of an image compression neural net using a variant of Vector Quantization, the Tree Structure Vector Quantization (TSVQ). According to the its authors, it provides a good tradeoff between the computational complexity of the conventional vector quantization and the quality of the compressed image.

(a) Original image (8 bpp)

(b) Compressed image (0.75 bpp)

(c) Compressed image (0.5 bpp)

Figure 6. Performance of the TSVQ. Even with a compression ratio of 16, the quality loss is undetectable.

7 Practicing SOMs

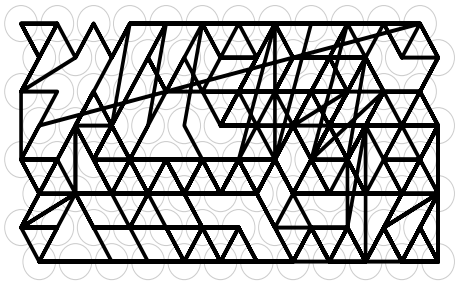
The following experiment was performed using the LVQ_PAK and SOM_PAK program packages, available from anonymous ftp to `cochlea.hut.fi`. A 12 by 8 two-dimensional map is used to build a neural network and train it for monitoring tasks.

The supposed device is some intricate system for which the temperature depends upon a certain number, say 5, of parameters. We aim to build a neural network to detect abnormal situations, it is, anyway, a kind of classification.

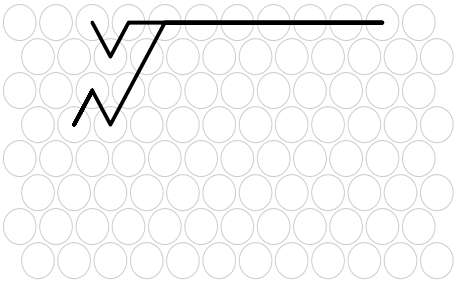
The weight vectors are initialized at random in the space covered by the input data vectors. The neighborhood layout is hexagonal and its function type is bubble-shaped.

The map is then trained using 3840 data vector samples. Two training passes were performed, the first with 1000 iterations and a wide neighborhood ($\alpha = 0.05$ and radius $= 10$), and the second wit 10000 iterations and a tight neighborhood ($\alpha = 0.03$ and radius $= 3$). The average quantization error were about 3.57.

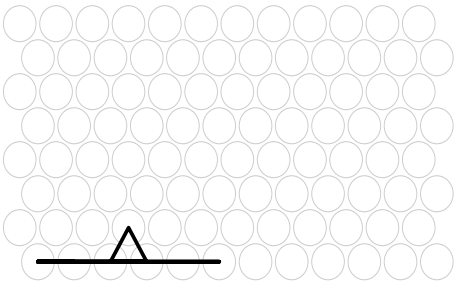
These are some details of the training phase.



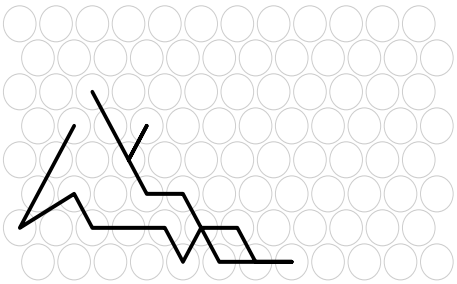
(a)



(b)



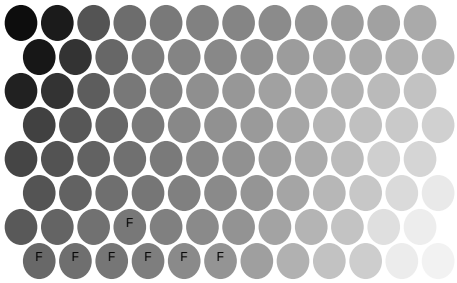
(c)



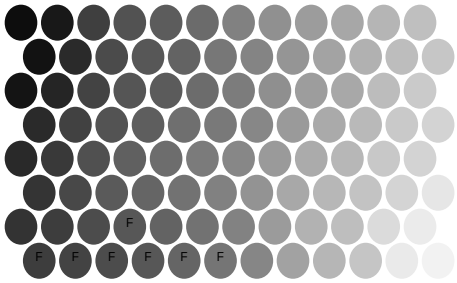
(d)

Figure 7. The best-matching cell trajectory along the training. (a) For all the 3840 training patterns. (b) For vectors representing correct performance of the device. (c) For vectors representing faulty performance status. (d) For vector representing an overheating status (moving into the bad region).

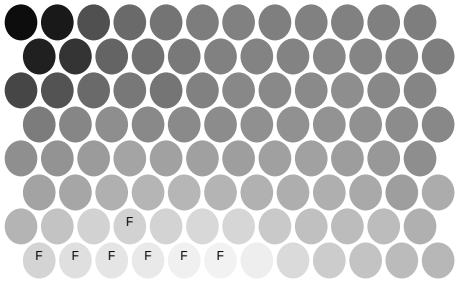
Below are the codebook vectors' components. For a given index k, the $(m_i)_k$ are drawn in gray scales in Figure 8. Note that we give an obvious interpretation of the trajectories in Figure 7: the daily performance vectors (Figure 7. a) were mapped to random regions of the map, the correct performance measurements (Figure 7.b) were mapped far away from the fault region (in the sane region), the faulty status performance vectors (Figure 7.c) were mapped directly into the fault region and when the device is overheating (Figure 7.d) the mappings seemed to "approach" the fault region.



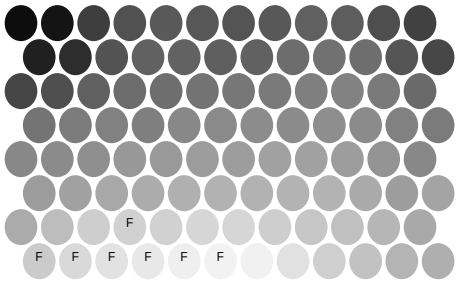
(a) Component plane 1



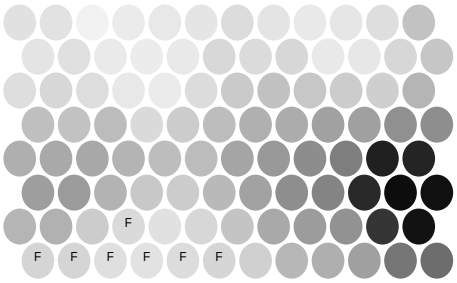
(b) Component plane 2



(c) Component plane 3



(d) Component plane 4



(e) Component plane 5

Figure 8. Distribution of the weights' vectors per plane of components. Cells marked with 'F' denote a faulty performance state.

Looking at Figure 8. e, we can deduce that when the parameter with subscript 5 in the input data vectors is high, the device is likely to operate in the sane region.

8 Conclusion

Competitive nets are a kind of hybrid, where a feedforward structure contains at least one layer with intra-layer recurrence. Each node is connected to a large neighborhood of surrounding nodes by negative or inhibitory weighted inputs, and to itself (and possible a small local neighborhood) via positive or excitatory inputs. These lateral connections are usually fixed in magnitude and do not partake in the learning process. It is also connected to the previous layer by a set of inputs which have trainable weights of either sign. The point of the lateral recurrent links is to enhance an initial pattern of activity over the layer, resulting in the node which was most active being turned fully "on", and the others being turned "off"; hence the label "competitive" or "winner-take-all" being applied to these nets.

The object here is to encode groups of patterns in a 1-out-of-n code, where each class is associated with an active node in the winner-take-all layer. This mechanism, or some minor variant, is a key component in Grossberg's Adaptive Resonance Theory (ART), Fukushima's neo-cognitron and Kohonen's nets that develop topological feature maps.

Acknowledgment

I would like to thank miss Fatma Chaker for her valuable help. This work was supervised by Dr. Rafik Brahem as part of a Master Degree course on Neural Networks at the National School for Computer Sciences, Tunisia.

References

[1] Jean-François Jodouin. *Les réseaux neuromimétiques* (pp. 111-130). Hermes, Paris, 1994.

[2] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen. *SOM_PAK The Self-Organizing Map Program Package*. Helsinki University of Technology, 1995.

[3] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen. *LVQ_PAK The Learning Vector Quantization Program Package*. Helsinki University of Technology, 1995.

[4] Kevin Guerney. *Neural Nets*. Psychology Department, University of Sheffield, 1996.

[5] Warren Sarle. `comp.ai.neural-nets`. Neural nets newsgroup, 1997.

[6] Edward Riegelsberger. *Context-sensitive vowel recognition using the FSCL-LVQ classifier*. Master Degree thesis. Ohio State University, 1994.

[7] Tomas Nordström. *Designing parallel computers for self organizing maps*. Lulea University of Technology, Sweden, 1992.