Neural Networks

Competitive Networks

Dr. Petr Musilek

Department of Electrical and Computer Engineering
University of Alberta

Fall 2019

Competitive Networks

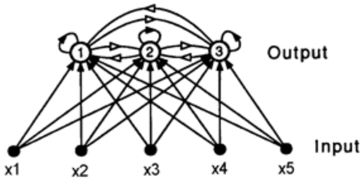
Selforganisation

- Unsupervised learning: system discovers for itself patterns, features, regularities, or categories in the input data and code them in the output
- Units and connections display some degree of selforganisation
- Competitive learning
  - output units compete for being excited
  - only one output unit is active at a time (winner-takes-all mechanism)
- Feature mapping: development of significant spatial organisation in the output layer
- Applications: function approximation, image processing, statistical analysis, combinatorial optimization

Selforganising Network

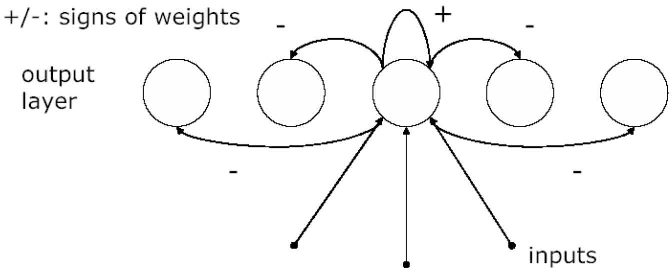
- Goal: to approximate the probability distribution of real-valued input vectors with a finite set of units
- Given a training set T of example patterns (inputs) $\boldsymbol{x} \in \mathcal{R}^n$, and q representatives (outputs) $\boldsymbol{o}_l$, $l = 1, \ldots, q$

- Network topology:
  - weights belonging to one output unit determine its position in the input space
  - lateral inhibition (to facilitate competition)



Competitive dynamics

Single element from a competitive layer: coupling within layer, such that the ‘best’ wins, or ‘the winner takes all’. Achieved by positive feedback to itself, and inhibition of others.

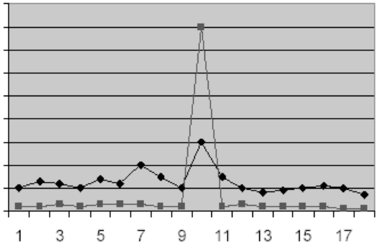


Competitive dynamics

- Suppose a vector ***x*** is presented at the input.
- Each unit computes its net input as follows:

$$tot_j = \sum_i w_{ij} x_i + w_{jj}^{+} o_j + \sum_{k \neq j} w_{jk}^{-} o_k$$

- Winner is found after an iterative process
- Implementation can be much simpler!



Competitive learning

Consider:

- training set where $|\boldsymbol{x}| = 1, \forall \boldsymbol{x}$ (all vectors are on the unit hypersphere)
- competitive layer with normalized set of weight vectors, represented as a 3-node network

Desired state after training:

- each weight vector must respond strongly to one of the clusters of input vectors.

Competitive learning

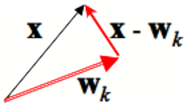
This desired state can be derived by:

- iterative applying vectors and finding the winner
    - node with the largest external input; or (in other words)
    - node with weight vector closest to the input vector (Euclidean distance; using argmin)
- adjusting the weights of the winner

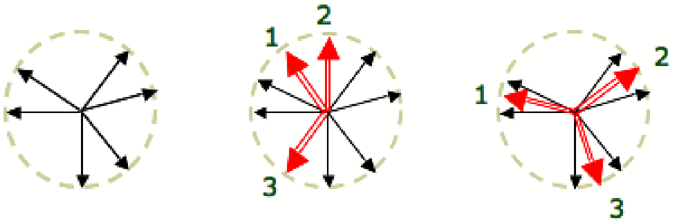
Let this “winning” node have index k and weight vector $\boldsymbol{w}_k$.

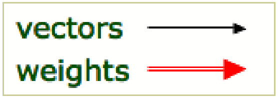
Weight $\boldsymbol{w}_k$ must be rotated towards $\boldsymbol{x}$ by adding a fraction of the difference vector $\boldsymbol{x} - \boldsymbol{w}_k$, i.e. :

$$\Delta \boldsymbol{w}_j = \begin{cases} \eta(\boldsymbol{x} - \boldsymbol{w}_j) & \text{for } j = k \\ 0 & \text{for } j \neq k \end{cases}$$



Example of Competitive learning



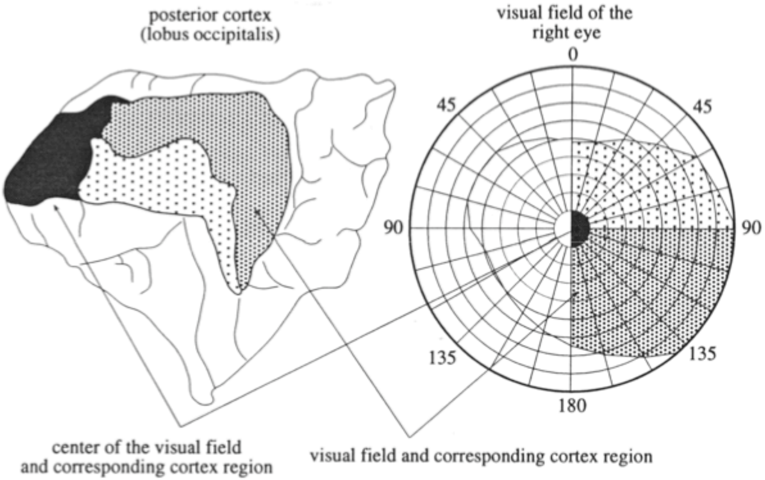


Self-organization in Brain

Formation of feature maps in the brain that have a linear or planar topology

- tonotopic map – sound frequencies are spatially mapped into regions of the cortex in an orderly progression from low to high frequencies.
- retinotopic map – visual field is mapped in the visual cortex (occipital lobe) with higher resolution for the centre of the visual field
- somatosensory map – mapping of touch on the cortex (homunculus)

Mapping of the visual field on the Cortex


posterior cortex
(lobus occipitalis)
visual field of the
right eye
0
45
45
90
90
135
135
180
center of the visual field
and corresponding cortex region
visual field and corresponding cortex region

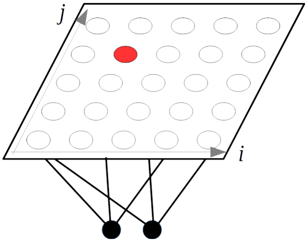
Self-Organizing Maps (SOM)

Purpose: mapping of a multidimensional input space onto a topology preserving map of neurons (often a 2 or 3D space; note the similarity to self-organization in biological systems)

Topology preservation: preserve a topological order so that neighboring output neurons respond to “similar” input patterns

Network structure: similar as described earlier, but:

- no lateral connections
- output units formed in a structure defining neighbourhood
- 1- or 2-dimensional array of units



Self-Organizing Maps (SOM)

- Each neuron is assigned a weight vector with the same dimensionality as the input space
- Input patterns are compared to each weight vector and the closest one wins (Euclidean Distance)

$$k = \text{argmin}_i(\|x_i - w_{ij}\|)$$

- Move neuron *k* and its neighbors closer to the input according to

$$w_{ij} = w_{ij} + \eta(x_i - w_{ij})$$

  for each j such that $D(j, k) < d$ (neighborhood)
- Result of the training process: clustering of data, so that similar inputs appear close in the map space (i.e. implicit categorization discovered without feedback)

Neighborhood function $h_k(j)$

- Neighborhood $N_d(k) = \{j : \; D(j,k) \leq d\}$ defines a set of neurons with distance from k less than d
- In learning algorithm, neighborhood function h_k is used:
  - weight update rule involves neighbourhood relations - the winner as well as the units close to it are changed

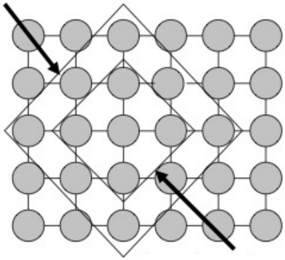
$$w_{ij} = w_{ij} + h_k(j)\big(x_i - w_{ij}\big), j \in N_d(k)$$

  where $h_k(j) = \eta$ if $j \in N_d(k)$ and $h_k(j) = 0$ otherwise.
  - Gaussian function can be used as $h_k(j)$ so closer units are more affected than those further away

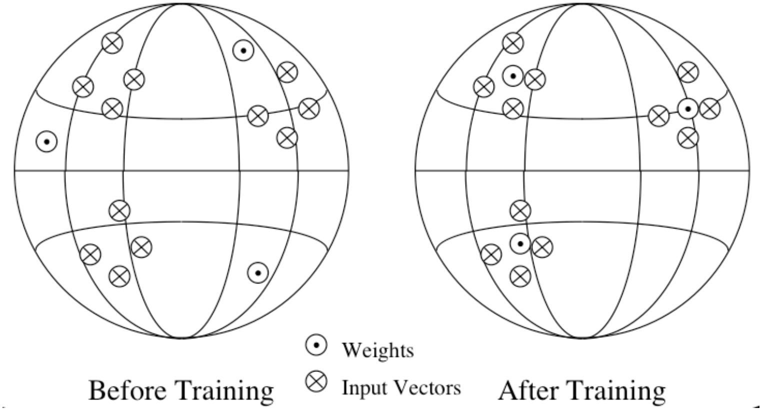
Neighborhood $N_d(k)$

- Neighborhood should get smaller in time (to ensure convergence and specialization of the output nodes), for example when $t_1 < t_2$:

$$t_1 \text{ use } N_2(k), t_2 \text{ use } N_1(k)$$



Typical Convergence

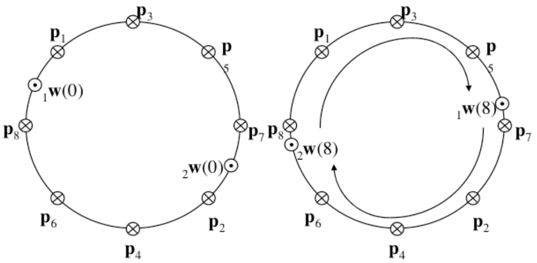


Problems with SOMs

- Stability
- Dead units
- Need to know number of clusters/classes
- Each class must be a convex region

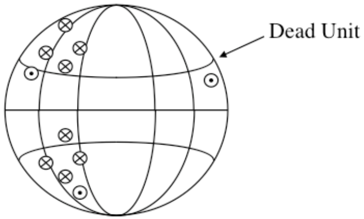
SOM: stability

If the input vectors do not naturally form clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will not converge.



SOM: dead units

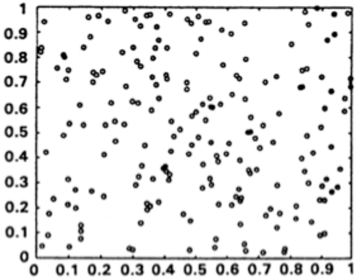
Neurons with initial weights far from any input vector may never win.

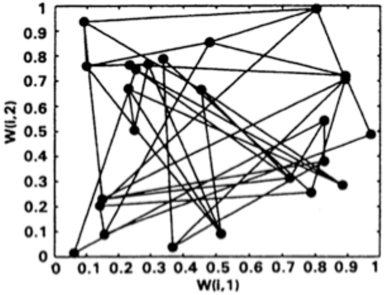


Dead unit remedy: “conscience” - neurons are discouraged from being greedy - their probability of being the winner is influenced by how many times they have won previously.

SOM Example – even distribution

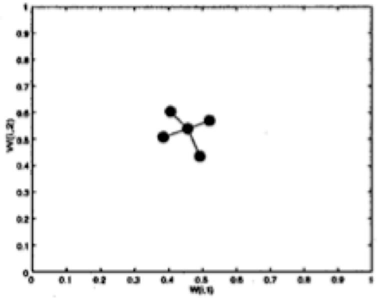
200 vectors chosen randomly are shown in the left picture, to train a net of 25 nodes/units on a 5$\times$5 rectangular grid. The initial weights are shown on the right.





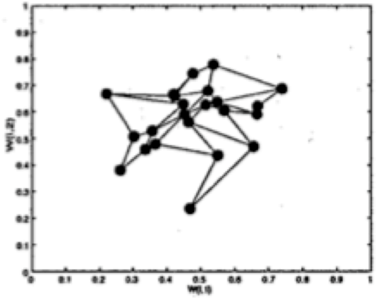
SOM Example – after 60 iterations

All nodes have weights close to the center of the field, which is near the average of the training set. This is the result of selecting a large initial neighborhood, so that nearly all nodes are trained at every step.



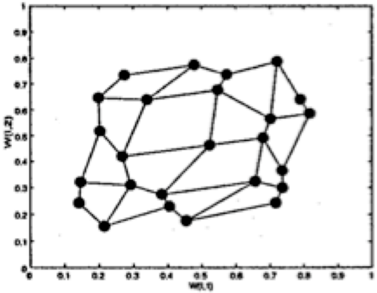
SOM Example – after 300 iterations

As the neighborhood is shrinking, the weight space representation starts to unravel -different parts of the pattern space can start to be encoded by different regions of the net.



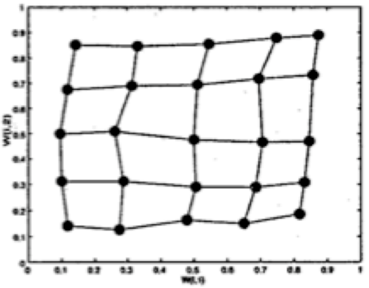
SOM Example – after 600 iterations

The topology is correct – there are no crossing internode connections.



SOM Example – after 6000 iterations

Further training provides fine adjustments, giving the network a better (more even) shape.



Learning Vector Quantization (LVQ)

- SOM algorithm provides an approximate method to compute the Voronoi vectors in an unsupervised manner

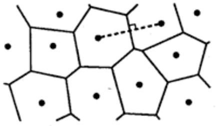


FIGURE 9.4 Voronoi tessellation. The space is divided into polyhedral regions according to which of the prototype vectors (dots) is closest. The boundaries are perpendicular bisector planes of lines joining pairs of neighboring prototype vectors.

- LVQ can be used to add another layer to combine different Voronoi regions to do pattern classification, combining unsupervised and supervised learning (non-convex regions obtained by union of convex regions)

Learning Vector Quantization (LVQ)

- Winning neuron in the competitive layer indicates the subclass which the input vector belongs to
- There may be several different neurons (subclasses) which make up each class
- The second layer of the LVQ network combines subclasses into more complex (non–convex) regions through matrix

$$W^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad \begin{array}{l} \text{Subclasses 1,3 and 4 belong to Class 1} \\ \text{Subclass 2 belongs to Class 2} \\ \text{Subclasses 5 and 6 belong to Class 3} \end{array}$$

LVQ Learning

Combines competitive learning with supervision:

- requires a training set of examples of proper network behavior

$$\boldsymbol{x}_1, \boldsymbol{t}_1, \boldsymbol{x}_2, \boldsymbol{t}_2, \ldots, \boldsymbol{x}_n, \boldsymbol{t}_n.$$

- If the input pattern is classified correctly, then move the winning weight vector toward the input vector according to the Kohonen rule.

$$w_{ij}^{\mathrm{new}} = w_{ij}^{\mathrm{old}} + \eta[x_i - w_{ij}^{\mathrm{old}}]$$

- If the input pattern is classified incorrectly, then move the winning weight away from the input vector.

$$w_{ij}^{\mathrm{new}} = w_{ij}^{\mathrm{old}} - \eta[x_i - w_{ij}^{\mathrm{old}}]$$