

Self-Organizing Networks

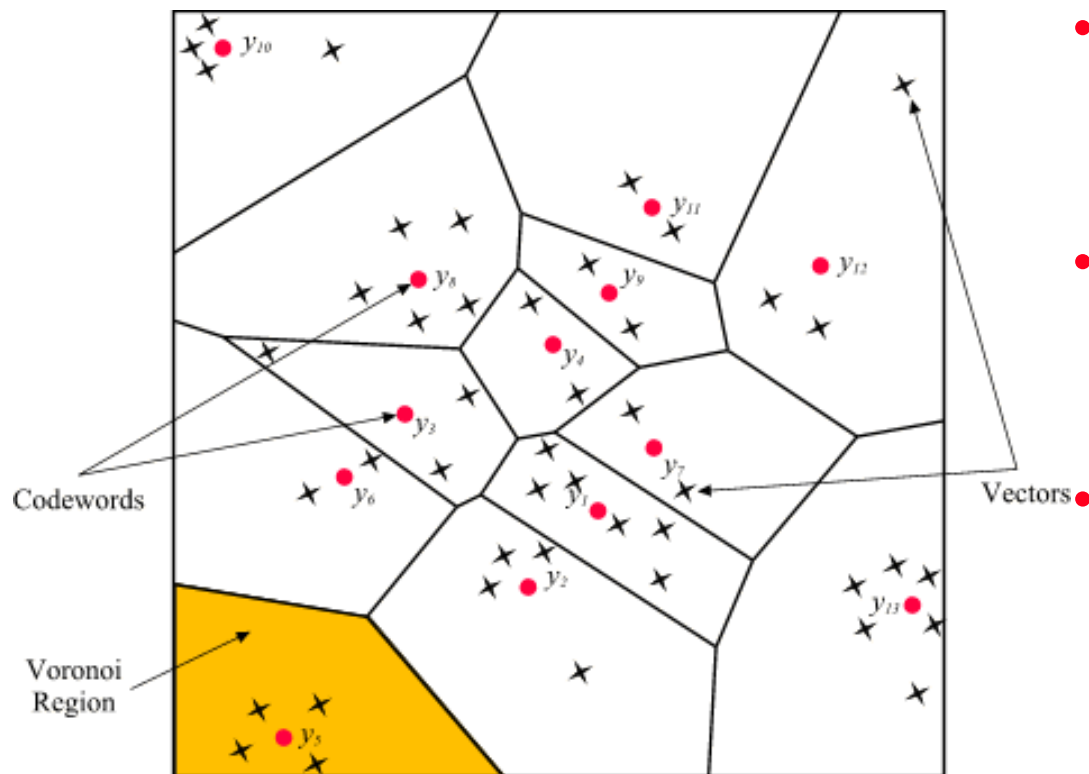
Self-Organizing Networks

- training without external teacher
- only information available is the set of input patterns
- training usually based on competition between neurons
 - competitive learning
 - Kohonen maps
 - ART
 - cognitron

Competitive Learning

- clustering or vector quantization
- dimensionality reduction
- feature extraction

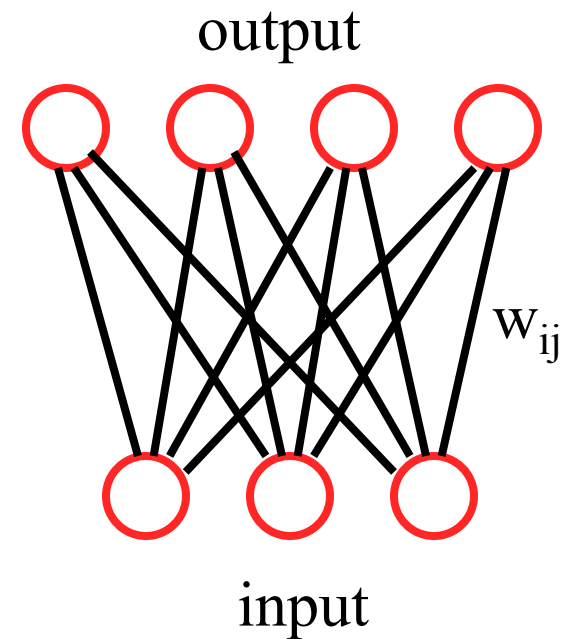
Vector Quantization



- want to quantize the entire input space or find clusters of similar data
- divide input space into a number of disjoint subspaces
- equivalent to tracking the input probability density function

Clustering: Hamming Network

- measures similarity between input vector and the weight vector of each neuron
- each output neuron calculates weighted sum of the input values



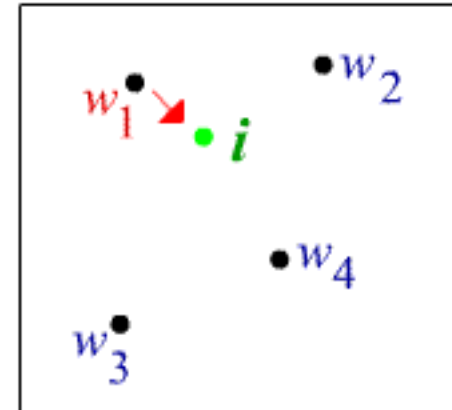
$\sum w_{ij}x_i$ interpreted as $w \cdot x = \|w\| \|x\| \cos \theta$

Algorithm

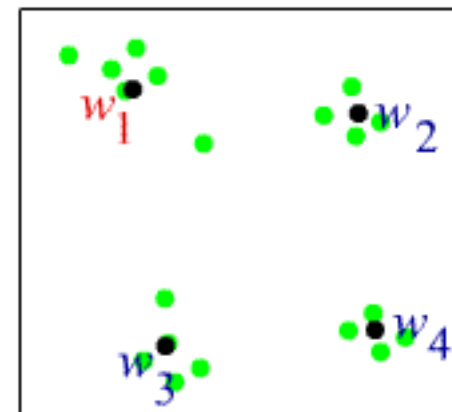
- weights w initialized to random values
- network presented inputs x
- calculate activations: $a_j = \sum w_{ij} x_i$
- select output k (winner) with max activation: $k = \operatorname{argmax}_j (a_j)$
- reset activations such that $a_k = 1$ and $a_j = 0; j \neq k$
- update weights according to $w_k(t+1) = w_k(t) + \gamma(x(t) - w_k(t))$

Hamming Net Learning

- effectively shifts winning weight vector w_k toward input vector x
- all other weights remain unchanged

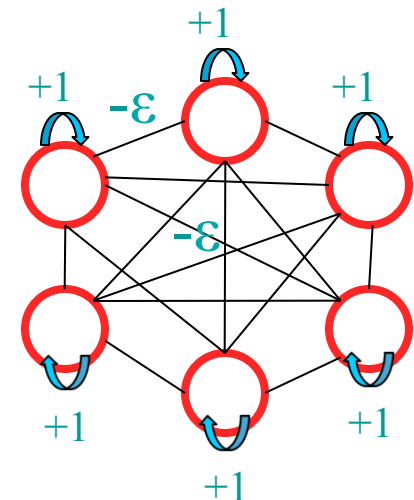


- if input vectors are clustered, weight vectors move to centers of clusters over



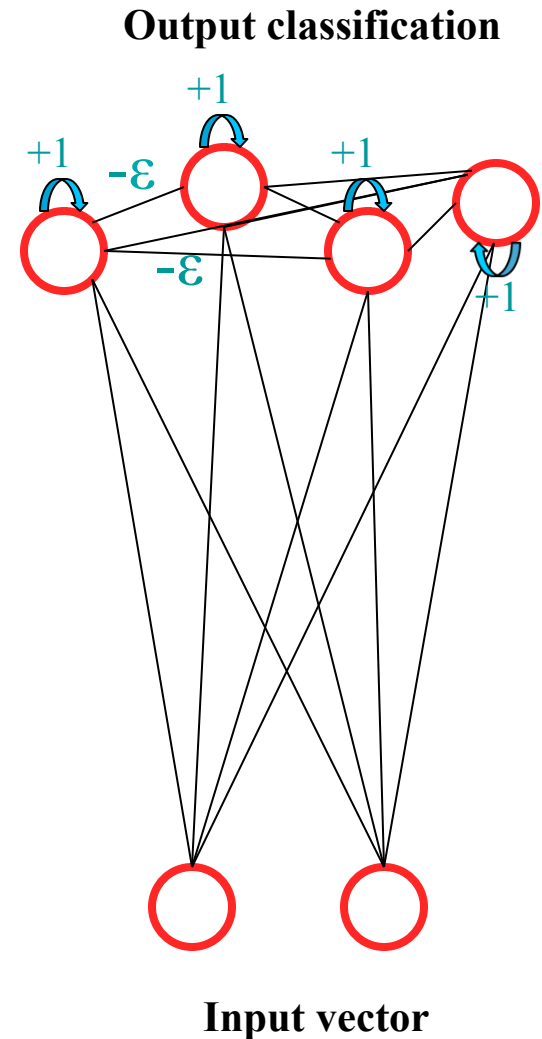
Winner-Take-All (Max Net)

- fully interconnected network with all units connected
 - to others with inhibitory links
 - to itself with an excitory link
- each node competes with every other:
repeat until x_i is stable
for each i
$$x_i = \max(\sum_j w_{ij} x_j, 0)$$
- all nodes converge to 0 except for the node with maximum initial value



Simple VQ network

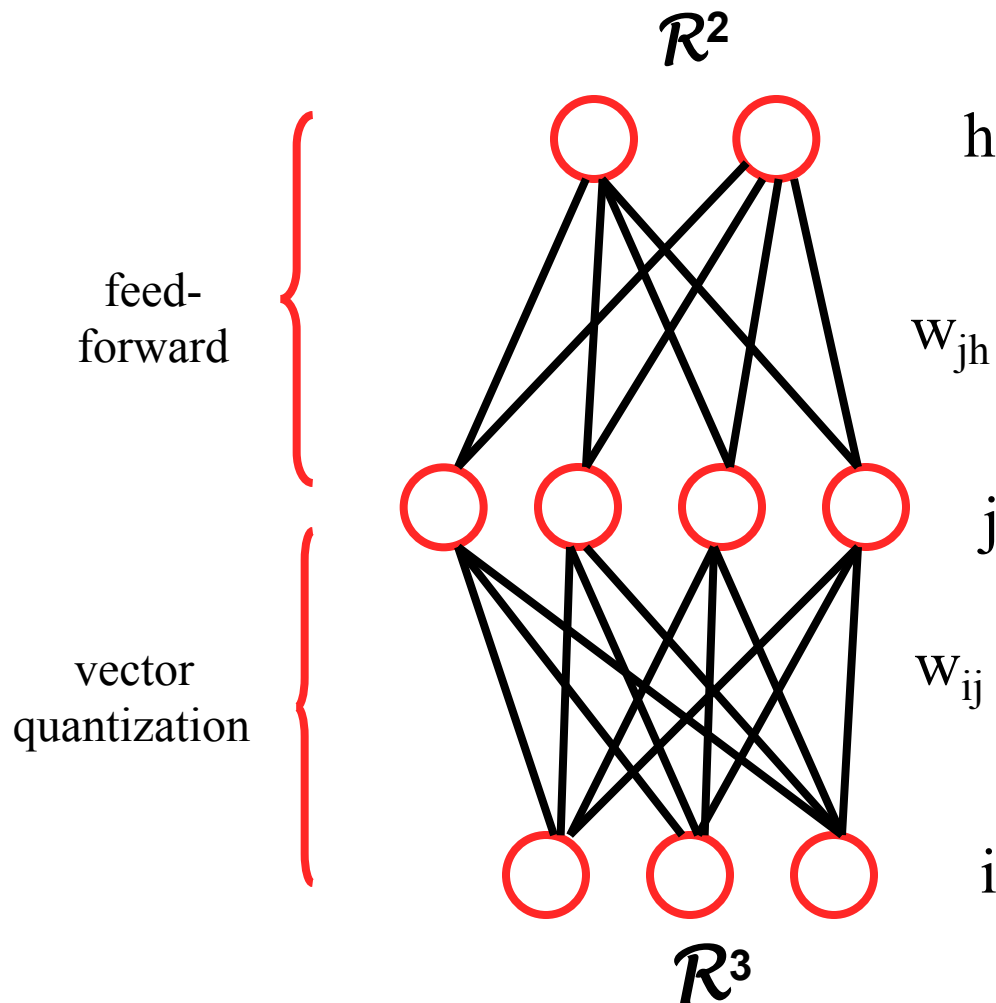
- Hamming net measures similarity (distance) of input vector to each weight vector
- Max net connects all the output layer nodes to find the node whose weight vector is closest to the input (“winner”)



Potential Problem

- in high-dimensional input space, likely that some of the weight vectors will never be chosen “winner” and thus, never moved/used
- Solutions:
 - draw initial weight vectors w_j from input patterns $\{x\}$ at random
 - expand weight update: $w_i(t+1) = w_i(t) + \gamma'(x(t) - w_i(t)) \quad \forall i \neq k$
with $\gamma' \ll \gamma$ known as the *leaky learning rate*
 - frequency selective competitive learning
 - the more often a neuron wins, the less sensitive it becomes
 - neurons that consistently fail increase chances of being selected

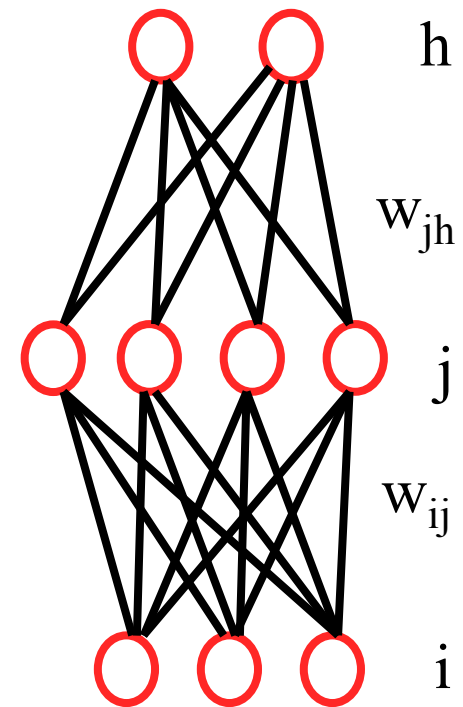
Function approximation



- useful for data encoding
- vector quantization typically combined with another network to map: $\mathcal{R}^N \rightarrow \mathcal{R}^M$
- with each neuron j , associate a function value $[w_{j1} \dots w_{jm}]$, typical of function values $\mathcal{F}(x)$ represented by j

Quantize and Approximate

- present network with input \mathbf{x} and function value $d = \mathbf{F}(\mathbf{x})$
- perform unsupervised quantization step:
 - for each weight vector \mathbf{w}_{ij}
 - calculate distance to input pattern
 - find winner k
 - update $w_{ik}(t+1) = w_{ik}(t) + \gamma(x_i(t) - w_{ik}(t))$
- perform supervised approximation step:
 - $w_{kh}(t+1) = w_{kh}(t) + \gamma(d_h - w_{kh}(t))$
 - which gives $o_h = \sum_j w_{jh} a_j = w_{kh}$ when k is the winning neuron



Issues

- each vector w_j converges to the mean function value over all inputs in the cluster represented by j
- not all functions represented accurately this way (e.g., simple identity or sin/cos combinations better with multilayer backprop)
- good if input expected to be subspace of high-dimensional input space or for functions that are discontinuous at several points
- quantization can be replaced by other methods, e.g., Kohonen nets