PATTERN RECOGNITION AND MACHINE LEARNING SYSTEMS DAY 2A

Dr Zhu Fangming
Institute of Systems Science
National University of Singapore
fangming@nus.edu.sg

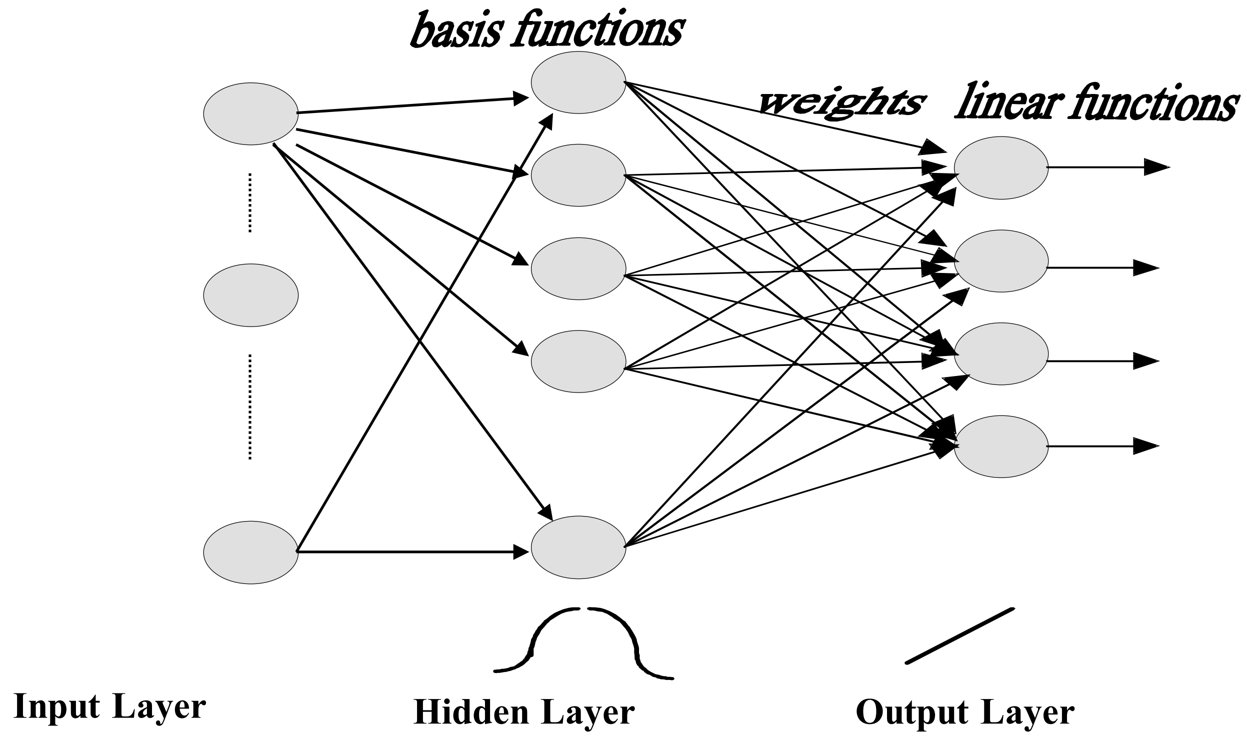
2.1
Neural Network Models and Designs

Topics

- Radial Basis Function Networks
- General Regression Neural Networks
- Self-Organizing Map (SOM, Kohonen) Networks

Radial Basis Function Networks

- Architecture
  - Input layer is *fully* connected to the hidden layer
  - The hidden layer is *fully* connected to the output layer



Architecture of RBF Networks

- **Nodes:**
  - Each node in the input-layer
    - receives the value of an input variable
  - Each hidden node
    - provides a radial basis function of input variables
  - Each node in the output layer
    - corresponds to a non-linear function (mapping) of input variables

- **Weights:**
  - An adjustable weight vector W_i is on the connections between the input nodes and the *i*-th internal node, indicating the centre of the radial basis function *(through unsupervised learning)*
  - Adjustable weights connecting internal nodes with output nodes *(through supervised learning)*

Activation of RBF Networks

- **Activation**
  - Hidden layer (kernel nodes)
    - *j*-th basis function (kernel) gives the same activation value for all inputs that lie within the same radial distance of its kernel centre c_j
    - Kernel nodes often use Gaussian activation basis functions which depend on the distance between the input vector x and the node's centre vector c_i. In this case, the activation value of *i*-th node in hidden layer is calculated by

$$O_i = \exp\left[- \frac{(\mathrm{x} - \mathrm{c_i})^T (\mathrm{x} - \mathrm{c_i})}{2\sigma_i^2} \right]$$

where σ_i is smoothing parameter (also called normalization factor)

  - Output unit activation is usually linear combinations of the kernel outputs

$$O_j = \sum_i W_{ji} O_i$$

where W_{ji} is the weight from hidden node *i* to output node *j*.

RBF Network Learning

- Training requires
  - two parameters (c_i and σ_i) to be found for each kernel node *i*
  - a full weight set for output nodes

- Training is performed in two stages:
  - finding centre and smoothing parameter values for the hidden (kernel) nodes (*unsupervised learning*)
    - Usually Hard C-Means clustering is employed
  - finding weights for the output nodes (*supervised learning*)

RBF Network Learning (cont.)

Learning for Kernel Centre

- If the number of input samples x_i is small, set kernel centre $c_i = x_i$ for each *i* (a form of self growing network).
- If the number of samples is large, do clustering (e.g. *k*-means) first and use each cluster prototype as a kernel centre

$$\mathbf{c}_j = \frac{1}{\mathrm{m}_j} \sum_{\mathbf{x}_i \in \Theta_j} \mathbf{x}_i$$

Learning for Smoothing Parameter

- The smoothing parameters σ_j are usually found from the average distance between the cluster centres and the training patterns

$$\sigma_j^2 = \frac{1}{m_j} \sum_{\mathbf{x} \in \Theta_\mathrm{j}} (\mathbf{x} - \mathbf{c}_\mathbf{j})^T (\mathbf{x} - \mathbf{c}_\mathbf{j}) \qquad j = 1, 2, \ldots, h$$

where m_j is the number of patterns in cluster Θ_j

RBF Network Learning (cont.)

<u>Learning in output layer</u>

- Adjust weights by

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji}$$

where $W_{ji}(t)$ is the weight from hidden node *i* to output node *j* at time *t* (or the *t*-th iteration)

- The weight change is computed by

$$\Delta W_{ji} = \eta \delta_j O_i$$

where η is learning rate, O_i is the output at hidden node *i*, δ_j is the error at the output unit *j*:

$$\delta_j = T_j - O_j$$

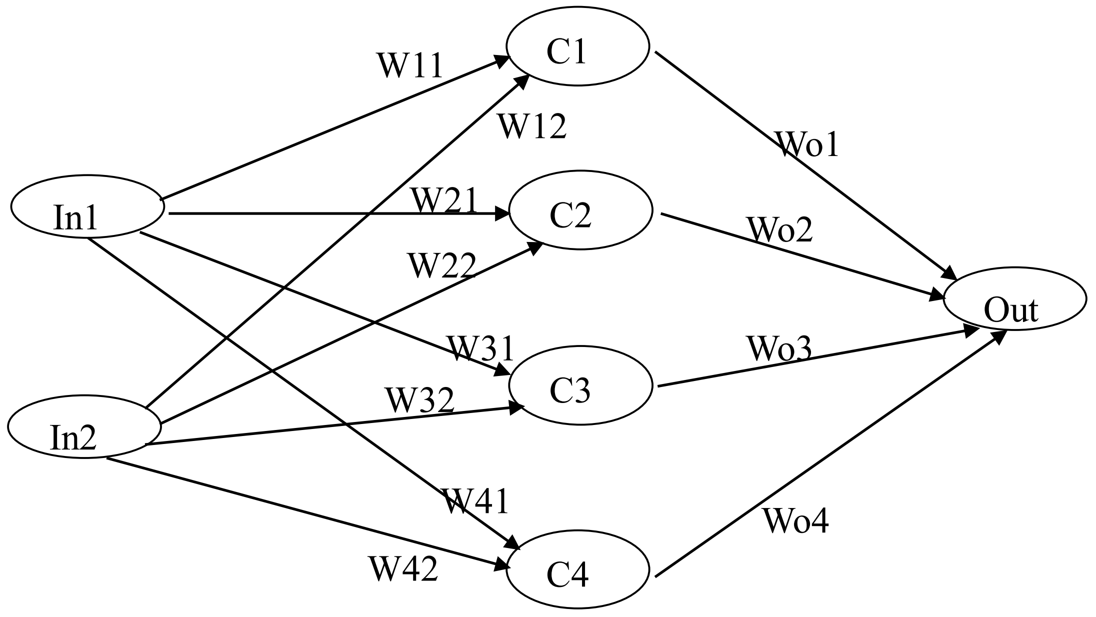
where T_j is the desired (target) output activation and O_j is the actual output activation at output unit *j*.

- Repeat iterations until convergence

A Simple Example of RBF Learning

- Build a RBF network with four hidden units to solve the XOR problem (a simple example to help understanding)
- Let the radial-basis function centers C1, C2, C3, and C4 determined by the four input patterns P1, P2, P3, and P4, respectively:

	In1	In2	Out
P1:	0	0	0
P2:	0	1	1
P3:	1	0	1
P4:	1	1	0



A Simple Example of RBF Learning …

- For simplicity, the following simple radial basis function is used:

$$O_i = \begin{cases} 1-(x-c_i)^T(x-c_i) & when \quad 1-(x-c_i)^T(x-c_i) \geq 0 \\ 0 & otherwise \end{cases}$$

  where c_i is the weight vector of hidden unit Ci (i = 1, 2, 3, 4), x is an input vector (the simple function requires NO smoothing parameter).

- The activation of output node is a linear combination of the kernel outputs: Out = $\Sigma_i W_{oi} O_i$

- The initial weights of the output layer are:

  $W_{o1} = 0.5$, $W_{o2} = 0.5$, $W_{o3} = 0.5$, and $W_{o4} = 0.5$.

- The weight adjustment is based on:

$$W_{oi}(t+1) = W_{oi}(t) + \Delta W_{oi}$$

$$\Delta W_{oi} = \eta(T_{out} - A_{out})O_i$$

  where $\eta = 0.5$ is the learning rate for output layer, T_{out} is the target output, A_{out} is the actual output of the output node, O_i is the output of kernel node *i*.

A Simple Example of RBF Learning …

- Weights for kernel nodes:

  W11 = 0, W12 = 0,

  W21 = 0, W22 = 1,

  W31 = 1, W32 = 0,

  W41 = 1, W42 = 1.

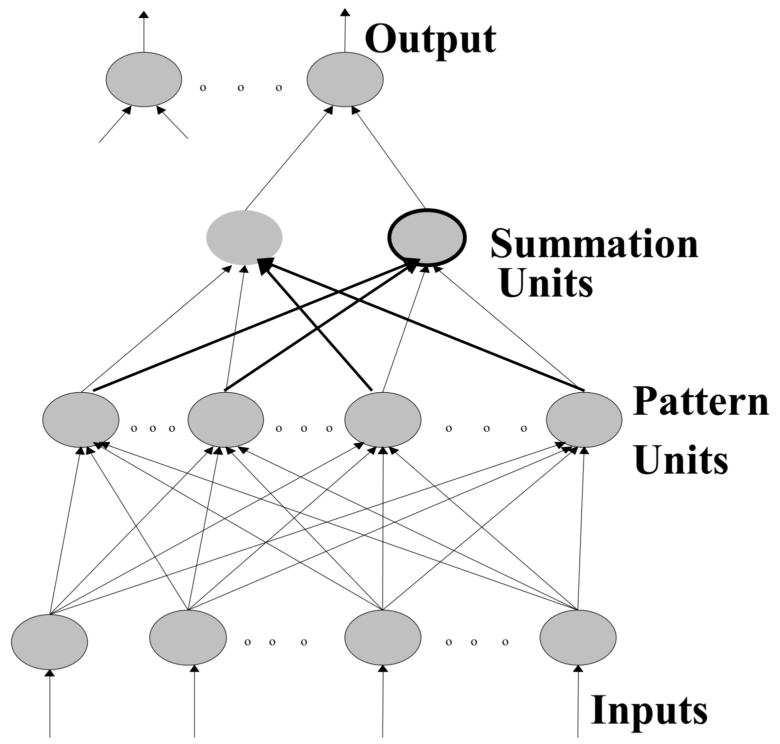
- Weights for output nodes:
  - Input P1:
    - only C1 is activated. Out = 0.5
    - Wo1(t+1) = Wo1(t) + ΔWo1 = 0.5 + 0.5 * (0 – 0.5) * 1 = 0.25
  - Input P2:
    - only C2 is activated. Out = 0.5
    - Wo2(t+1) = Wo2(t) + ΔWo2 = 0.5 + 0.5 * (1 – 0.5) * 1 = 0.75
  - ……

Summary of RBF Networks

- There are many variations possible in learning strategies
  - basis functions types
  - selection of centers
    - fixed centers selected randomly
    - self-organized learning (clustering)
    - supervised selection
- The two-stage training process of RBF permits the use of unlabeled training data (unsupervised training methods) while creating kernel nodes and hence, only a relatively small number of labelled data will be needed to find the output layer parameters.

General Regression Neural Networks (GRNN)

- The GRNN architecture consists of
  - an input-layer
  - three computational layers: pattern, summation, output



Architecture of General Regression NN(cont.)

- **Nodes & connections**
  - The input-layer distributes input patterns to the pattern-layer through fully connected links with adjustable weights.
  - The pattern-layer is fully connected to the summation-layer units through adjustable weights.
  - Each pattern node computes the distance between the input vector and their connection weight vector values.
  - The summation-layer has two types of neurons, type A and type B. They perform a linear summation of weighted input from the pattern layer.
  - The output layer performs a division operation on the output of the two summation-layer units to produce the estimate of the regression of z on x. One output node needs two corresponding summation nodes (each of type A and B).

Activation of General Regression NN

- From mathematical background, after certain rearrangement (details omitted) to the calculation of the regression of z on x, we can obtain

$$\hat{z} = \frac{\sum_{i=1}^{P} A_i \exp\left(\frac{-D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{P} B_i \exp\left(\frac{-D_i^2}{2\sigma^2}\right)}$$

  where $D_i^2 = (x - w_i)^T(x - w_i)$, P is the number of pattern nodes

- Pattern-layer unit *i* computes the distance between its weight vector and the input vector $(x – w_i)$ and transforms this to a scalar activation function value (typically exponential functions - Gaussian).

$$O_i = \exp\left(\frac{-\sum_{j=1}^{n}(x_j - w_{ij})^2}{2\sigma^2}\right)$$

- The activation of each summation unit is a linear sum of weighted inputs from the pattern layer

- The output layer performs a division operation on the output of the two summation-layer units (type A and B) to produce the regression of z on x

Learning of General Regression NN

- In the pattern layer, a new neuron is created for each exemplar pattern (or cluster centre) and the weight values are set equal to the exemplars or cluster centres (centroid values).

- For the weights in summation-layer, A_i and B_i are increased each time a training observation z_j for cluster *i* is seen (after *k* observations):

  $A_i(k) = A_i(k-1) + z_j$

  $B_i(k) = B_i(k-1) + 1$

  where $A_i(0) = 0$, $B_i(0) = 0$

- σ, determined experimentally, is a smoothing constant that defines the decision surface boundary.

GRNN Mapping Surface and Smoothing Constant

- Small values of σ give narrow peaked surfaces that fit well near sample points.
- Larger values of σ result in flatter, smoother surfaces.
- Good generalization requires a trade-off between the two extremes.

Summary of GRNN Networks

Features:

- Memory-based
- One-pass learning algorithm
- A form of self-growing net with *one* pattern unit created for *each* new training pattern or cluster centre.

Applications:

- Good for forecasting, control and similar applications where the training data set is generated by an unknown probability distribution.

Summary of GRNN Networks (cont.)

Advantages:

- The estimate converges to the true regression surface with increasing samples.
- A GRNN has the ability to work with sparse data and in real-time environments since the regression surface is defined everywhere instantly and the network provides smooth transitions from one observation to another.
- Training a GRNN is fast and straightforward with only a single pass through the training set needed. The network can begin to perform the regression after a single training sample has been presented.

Disadvantages:

- *The net may grow to be very large since one pattern unit is added for each pattern in the training set* (unless clustering is used to determine the prototype centres).
- The computation load for the network is relatively heavy.

An Example of GRNN Application

— Adaptive Control —

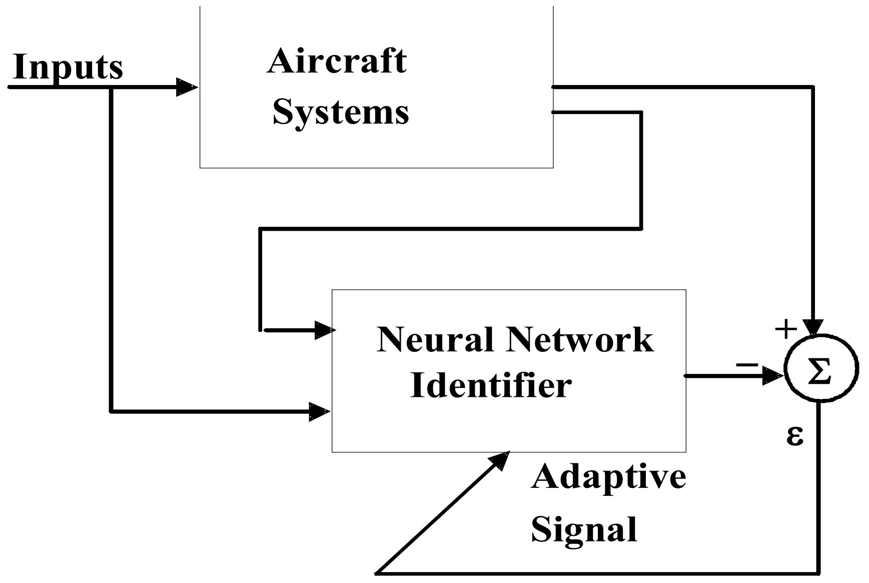
Problem domain:

The aerodynamics of a fighter aircraft are typically very nonlinear in nature. Significant nonlinearities occur as a result of kinematic and inertial couplings, aerodynamic nonlinearities and control deflection rate limitations. Simulations were carried out to see how well different ANN architectures could model the dynamics of a fighter aircraft for two highly nonlinear manoeuvre: *low angles* of attack dynamics and deep-stalls. Five time-dependent variables were input to the network at discrete time points k and two response variables predicted.

An Example of GRNN Application (cont.)

— Adaptive Control —

- **Input variables:**
    - control deflection command: $\delta_c(k)$
    - angles of attack at time *k* and *k-1*: $\alpha(k), \alpha(k-1)$
    - pitch rates at time *k* and *k-1:* $q(k), q(k-1)$
- **Output:**
    - predicted angle of attack at time *k+1*: $\hat{\alpha}(k+1)$
    - predicted pitch rate at time *k+1*: $\hat{q}(k+1)$



An Example of GRNN Application (cont.)

— Adaptive Control —

- Performance comparison was based on learning speed, modelling precision, network flexibility, and complexity:
  - An MLP (two hidden-layer)
  - a Radial basis function network
  - a GRNN

- The MLP required 50 nodes in each hidden-layer and required more than 50 epochs to converge.

- The RBF network used 18 basis nodes to characterize the variables. RBF converged after a few epochs.

- The GRNN was trained in a single pass over the training patterns with 1600 nodes generated for one experiment and 200 for another. σ values were tested for best generalization, any value in the range of 0.1 to 5.0 was satisfactory.

An Example of GRNN Application (cont.)

— Adaptive Control —

- **Approximation errors**
  - for the angle of attack

$$E_a = \left[\sum_k \left(\alpha(k) - \hat{\alpha}(k) \right)^2 \right]^{1/2}$$

  - for the pitch rate

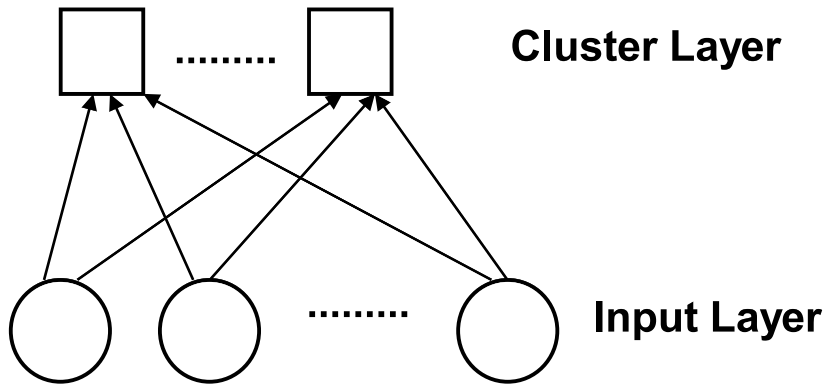
$$E_q = \left[\sum_k \left(q(k) - \hat{q}(k) \right)^2 \right]^{1/2}$$

- **The performance of three networks (six networks have been done, but only three are given here) was satisfactory, but the GRNN was best (compare the errors for angle of attack and pitch rate, E_a and E_q below).**

NN	Case 1E_a	Case 1E_q	Case 2E_a	Case2E_q
MLP	2.8777	2.7712	3.7296	2.7893
RBF	2.1457	1.3625	4.1159	3.2475
GRNN	2.0910	1.2176	2.8319	2.7865

Self-Organizing Map (SOM, Kohonen) Network

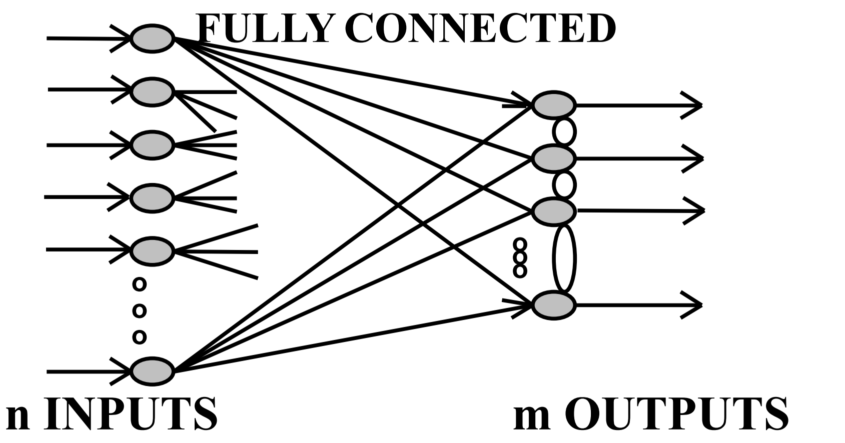
- Neural Network for clustering
  - common strategy: winner-takes-all (a competitive learning)
  - Basic architecture of a clustering network:
    - two layers: input layer and output layer



Competitive Network Operation

- Signals feed forward from input nodes and feed lateral among output neurons, a form of recurrent feedback
- A single-layer network architecture with *n* inputs and *m* output units, one output for each class or category
- *m prototype* weight vectors w_i, $i = 1, 2, \ldots, m$ correspond to the *m* classes. The network classifies each input pattern x as belonging to class *i* iff

$$|w_i - x| \le |w_k - x| \quad k=1, 2, \ldots, m, \; k \neq i$$ (find the most similar class)



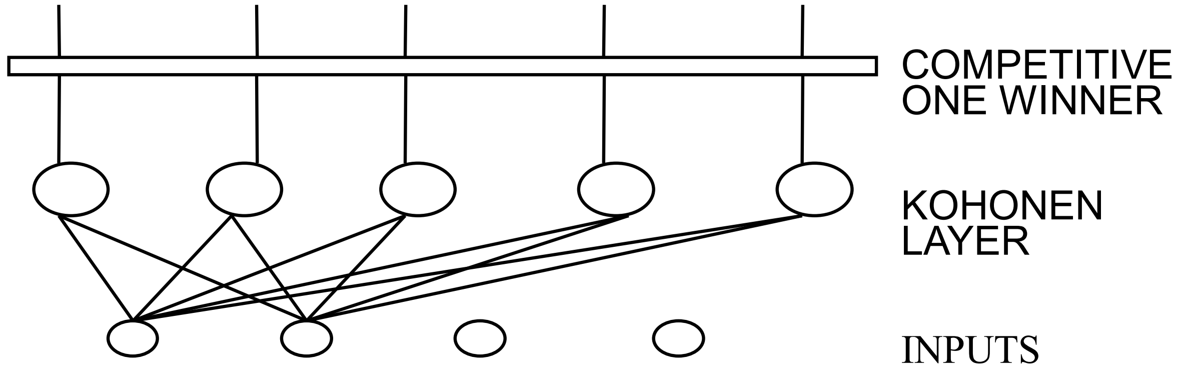
COMPETITIVE "WINNER-TAKES-ALL" NETWROK

Self-Organizing Map (SOM, Kohonen) Network

- Feature mapping converts patterns of arbitrary dimension (the pattern space) into the response of one- or two-dimensional arrays of neurons (the feature space).
- Kohonen learning rule is one approach to design a self-organizing feature map network
- SOM network
  - Consists of a group of geometrically organized neurons in one- or two-dimensions or even higher dimensions.
    - A one-dimensional network is a single layer of units arranged in a row
    - In two-dimensional network, the units are arranged as a lattice array.
  - A form of unsupervised competitive learning where winner shares the gains with neighbours to produce activation zone

Simple SOM (Kohonen) Network
— One-dimension —

- A simple 1-dimensional Kohonen (SOM) network
- Input pattern vector fully connected to all neurons
- Lateral interactions between neurons constrain activations to spatially bounded 'excitation zone'

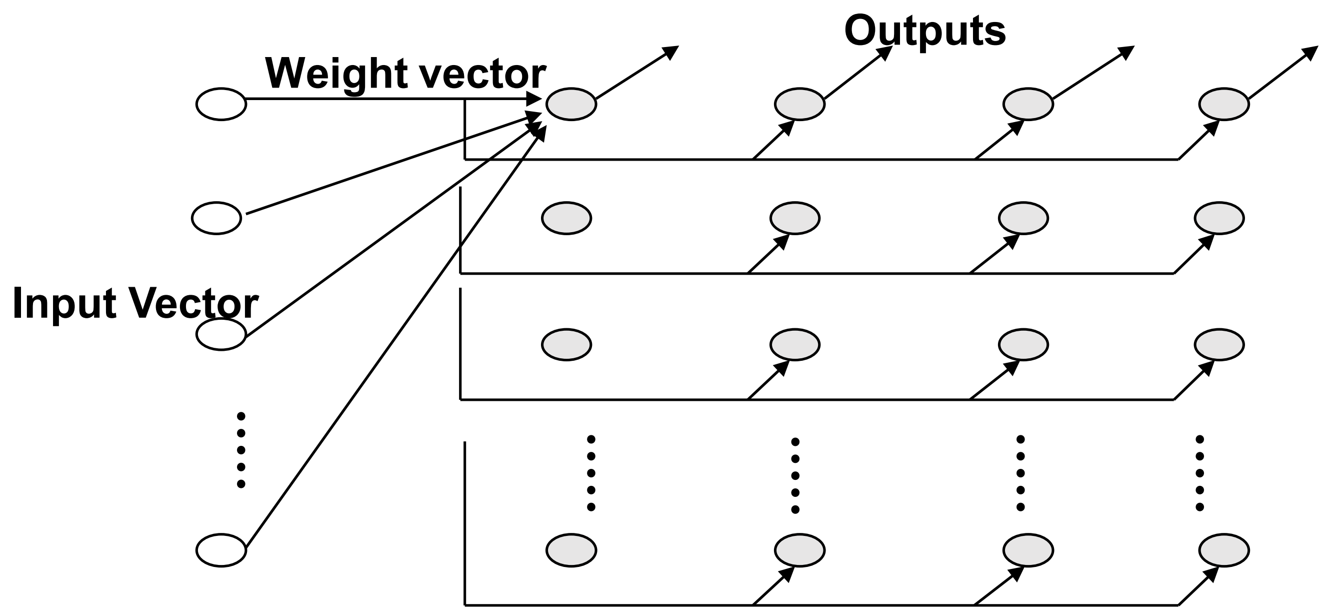


NEURONS COMPETE AND ONLY ONE "WINS"

SOM (Kohonen) Network (cont.)

— Two-dimension —

- A 2-dimensional Kohonen discrete lattice network
- Input vector patterns x are fully distributed to each node in the discrete lattice through adjustable weight vectors w_r
- The node with weight vector closest to the input vector becomes the "excitation centre" winner for the lattice



Kohonen Learning Algorithm

1) Initialize all weights w_r to random numbers following uniform distribution (0,1)

2) Apply an input signal vector x to the network

3) Select the winning output unit as the one with the smallest dissimilarity measure (or largest similarity measure) among all weight vectors w_i and the input vector x

$$\|\mathbf{x}-\mathbf{w}_r\| \leq \min_{r'} \|\mathbf{x}-\mathbf{w}_{r'}\|$$

4) Update

   the weights of the winner unit:

$$\mathbf{w}_{\mathbf{r}}^{\mathbf{new}} = \mathbf{w}_{\mathbf{r}}^{\mathbf{old}} + \alpha \cdot \mathrm{h}_{\mathbf{rr'}} \cdot \left(\mathbf{x}-\mathbf{w}_{\mathbf{r}}^{\mathbf{old}}\right)$$

   the neighbouring

$$\mathbf{w}_{\mathbf{r'}}^{\mathbf{new}} = \mathbf{w}_{\mathbf{r'}}^{\mathbf{old}} + \alpha \cdot \mathrm{h}_{\mathbf{rr'}} \cdot \left(\mathbf{x}-\mathbf{w}_{\mathbf{r'}}^{\mathbf{old}}\right)$$

   where $h_{rr'}$ is a neighborhood function with maximum value centered at the winning neuron *r* and decreases to zero as the distance between *r* and neighboring units r' increase (i.e. $h_{rr'}$ is defined in terms of the distance between r and r').

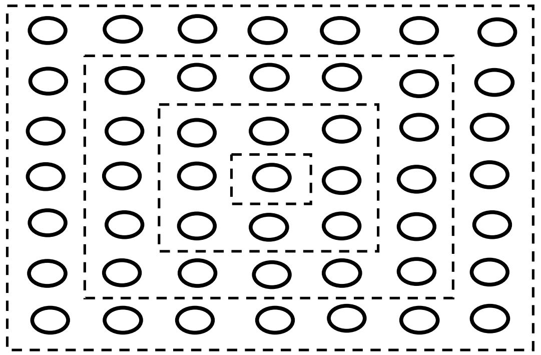
5) Repeat steps 2 through 4 until the network weights stabilize

6) Reduce the neighbourhood and learning rate parameters and iterate steps 2 to 5 if necessary.

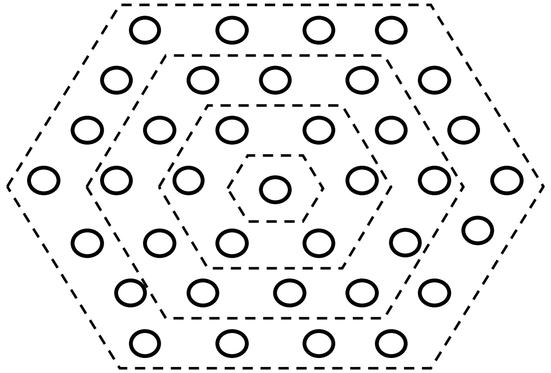
Kohonen Learning Algorithm (cont.)

Activation Neighbourhood can be:

- Square



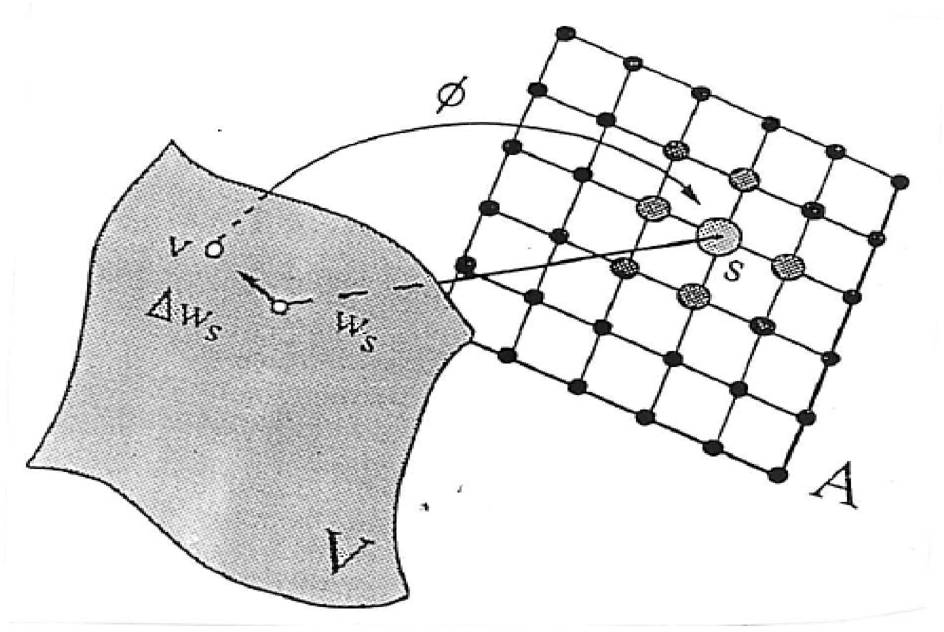
- Hexagon



- Smooth functions such as Gaussian or other types may also be used.

SOM Mapping Operation

- **Mapping the input signal space onto the NN lattice**
- **Weight vectors of neurons in the neighborhood of the winning neuron *s*, are shifted towards the input vector value. Those nearer to *s* have weights shifted more and those farther away shifted less**



Simulations of SOM

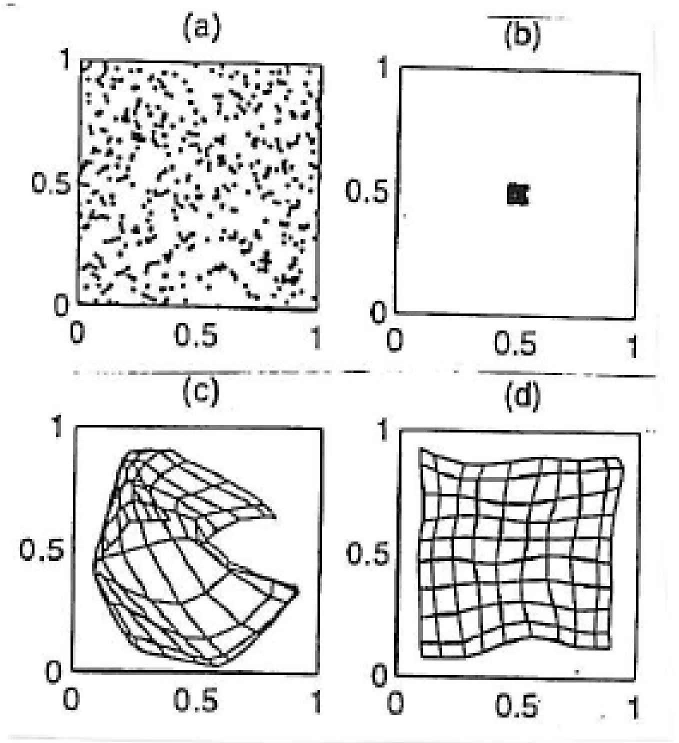
- Mapping two inputs x_1 and x_2 onto a SOM array.
- Input patterns are chosen randomly from the unit square
- Line intersections on the maps specify weight vector values for a single neuron
- Lines between nodes on the graph merely connect weight points for neurons that are topologically nearest neighbors

(a) input data uniformly distributed within [0, 1]× [0, 1]

(b) initial weights

(c) weights after 30 iterations

(d) weights after 1000 iterations



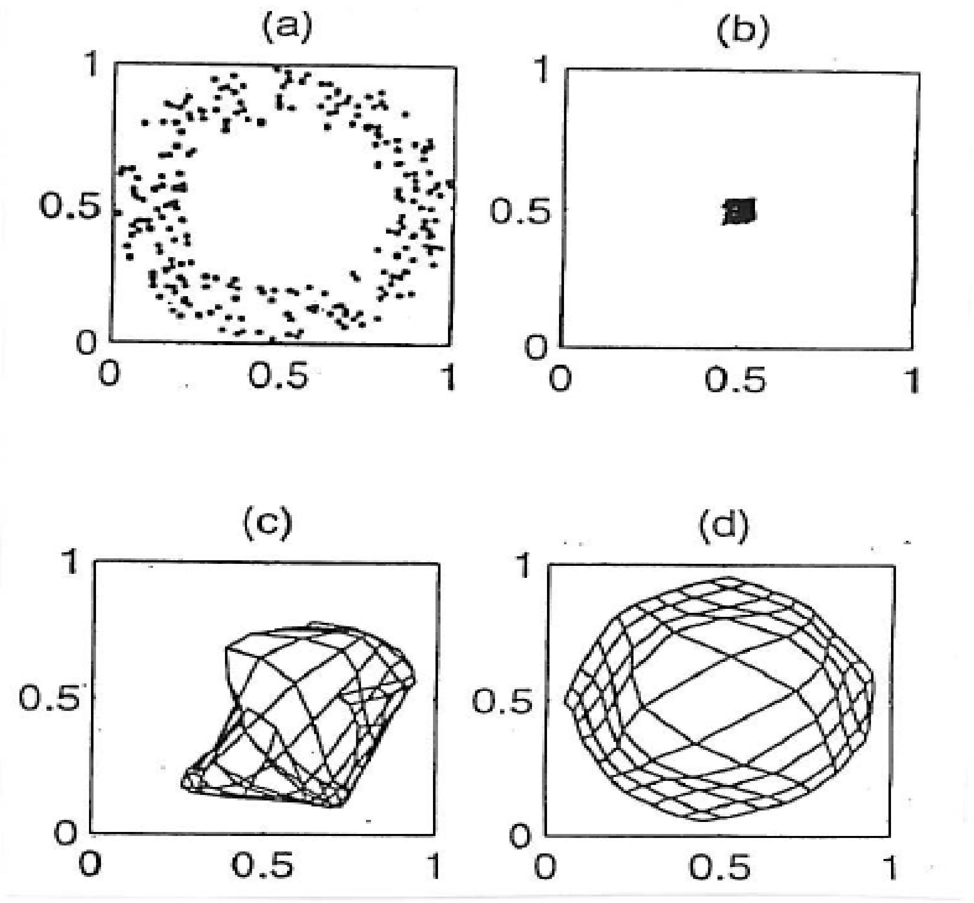
Simulations of SOM (cont.)

(a) input data uniformly distributed within [0, 1]× [0, 1]

(b) initial weights

(c) weights after 30 iterations

(d) weights after 1000 iterations



Advantages of SOM

- It is viable alternative to C-Means clustering
- It does so more elegantly, by removing the main drawback of C-Means - specifying the *number of clusters upfront*
- It provides a method of data compression
- When used for labeled data, this acts as a classifier too.
- *Most importantly*, it provides the user with much wanted *'data visualization'* feature, which is absent in many of the competing algorithms

Application Examples of SOM

Optimization

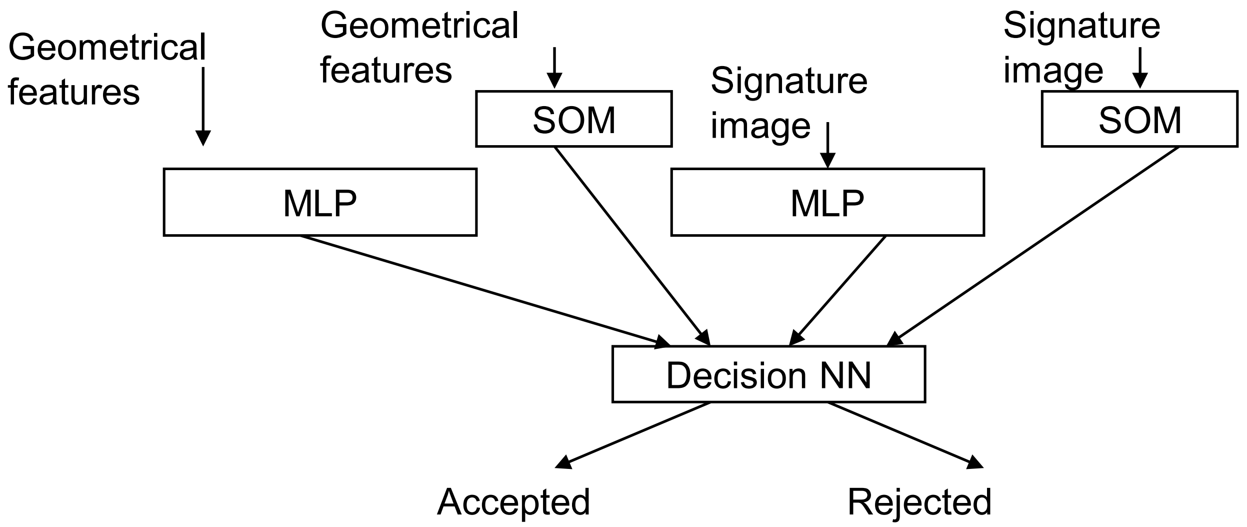
- One-dimensional SOM networks have been used to successfully solve optimization problems such as the traveling salesman problem (TSP)

Control

- Robot arm control
    - as a part of a robot control system, SOM network performs "data fusion" to provide the input data to the robot

Application Examples of SOM (cont.)

- Handwritten signature authentication
  - uses a hybrid neural network system with SOM networks and MLPs
    - 2 SOM networks were used for initial signature clustering into similar sets
    - 2 MLPs for authentication
    - 1 supervised learning NN for final decision



2.2
Workshop: RBF, GRNN & SOM

Workshop: RBF -Weka

- Continue from the Day 1 Workshop on using Weka for MLP

- Launch Weka Explorer
- Open file… …iris.csv
- Change the classifier as weka.classifiers.functions.RBFNetwork
  Note: If RBFNetwork is NOT available under the classifier options, please go to Weka GUIChooser → TOOLS → package manager to install the RBFNetwork package first.
- Click the classifier chosen to launch the properties window
- Mouse over each option to understand its definition or click "More" button to get more explanation
- Choose "Test Options" as "Percentage split 66%"
- Click "Start" to train a RBF network
- Experiment with different numCluster and compare the model performance
- Save the model

Workshop: GRNN & SOM – Python / Neupy

- Open the iPython notebooks provided for this workshop.
- As you go through the notebooks, make sure you understand how the NN models are built. (you can save notes as markdown in the notebook).