Master of Technology

Unit 2/6: Computational Intelligence I

Neural Network Architectures (I)

Dr. Zhu Fangming
Institute of Systems Science
National University of Singapore

Objective

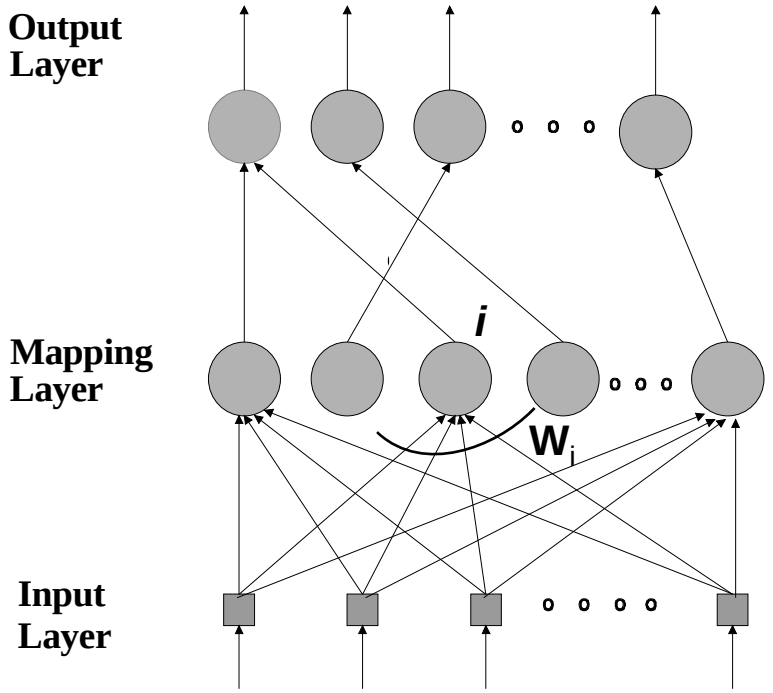
- To introduce important NN architectures with supervised learning

Outline

- Reduced Coulomb Energy Networks (RCE)
- Radial Basis Function Networks (RBF)
- General Regression Neural Networks (GRNN)
- Probabilistic Neural Networks (PNN)

Reduced Coulomb Energy Network

- An example of self-growing network with different activation functions and learning methods.
- Supervised learning of pattern categories separated by non-linear, essentially arbitrary boundaries.
- Architecture:
  - » The input-layer is fully connected to the mapping-layer
  - » Each internal node is connected to a single output node
  - » Output nodes may have input connections from more than one mapping layer nodes



Reduced Coulomb Energy Net

Architecture of RCE Network

- **Nodes:**
  - » **Each node in the input-layer registers the value of a feature in a given pattern**
  - » **Each node in the output layer**
    - ♦ **corresponds to a pattern category.**
    - ♦ **An input pattern is assigned to a category if an output node fires in response to the input. A decision can be *ambiguous* or *unambiguous* corresponding to single or multiple output node firings**
  - » **Each internal node**
    - ♦ **Captures one input pattern or a set of patterns that are close enough in the input space**

- **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 corresponding mapping node**
  - » **Weights connecting internal nodes with output nodes are all equal to +1** (*Updating may also be possible)

Architecture of RCE Network (cont.)

- **Activation function**
  - » **Input⟶ mapping:**
    - ♦ **A function of the input to weight vector distances**
    - ♦ **The activation function defines a hypersphere centered at W_i with radius θ_i. Any input pattern X within this region will cause internal node i to become active**

$$y_i = F_\theta[d(X, W_i)]$$

where $d(X, W_i)$ is a distance metric between the vectors X and W_i (*Cartesian, Hamming, dot product,* etc.), F_θ is a threshold function such that

$$F_\theta(z) = \begin{cases} 1 & \text{if } z \le \theta \\ 0 & \text{if } z > \theta \end{cases}$$

RCE Network Learning

- **Learning involves two operations**
  - » ***Commitment of internal nodes***
  - » ***Setting weights and thresholds***
- ***Node commitment***
  - » **A new internal node is fully connected to the input nodes**
  - » **The internal node is then projected to a single output node**
- ***Setting weights and thresholds***
  - » **Training signals are sent back to control the commitment of nodes and modification of thresholds**
    - ♦ **if an output node is on (i.e. 1) but should be off (i.e. 0), an error signal of –1 is sent back**
    - ♦ **if an output node is off (i.e. 0) but should be on (i.e. 1), an error signal of +1 is sent back**

RCE Network Learning (cont.)

- ***Setting weights and thresholds (cont.)***

  » **If the *k*-th output node produces a +1 error,**

  - ♦ **A new internal (mapping) node is committed and connected to the *k*-th output node**
  - ♦ **The weight vector W_i to the internal node are set equal to the input pattern X values**
  - ♦ **The threshold is set equal to a maximum value or the distance to the nearest node with different class**

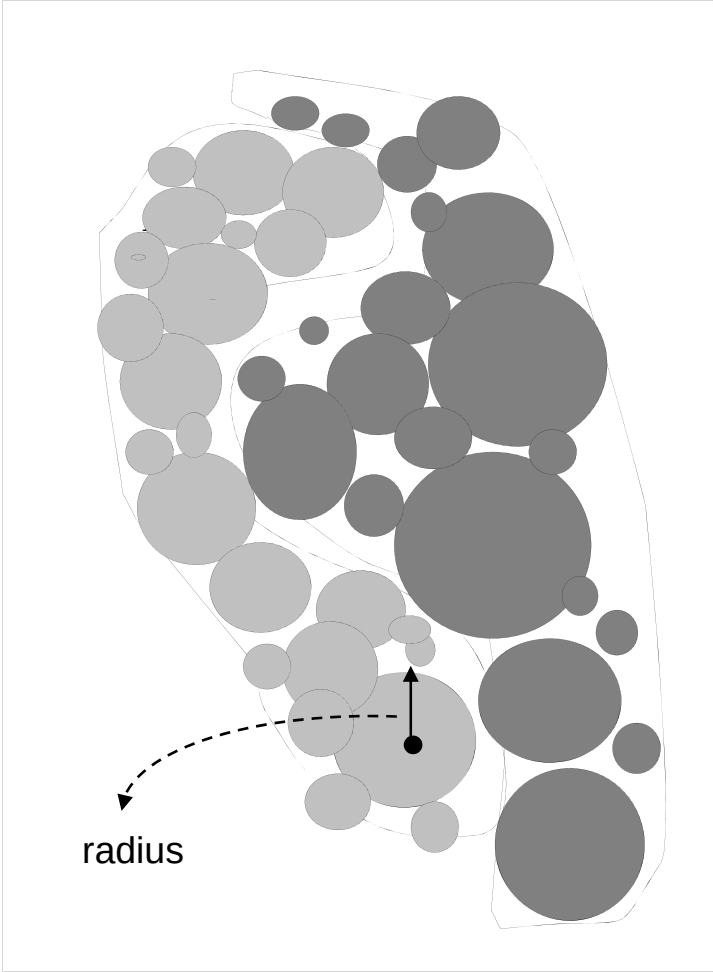
  » **If the *k*-th output node produces a –1 error,**

  - ♦ **the threshold values (radius) of all active internal nodes connected to the *k*-th output node are reduced (to reduce the region of influence and correct the error)**

  » **If the output nodes produce 0 error, no change is made to any internal nodes**

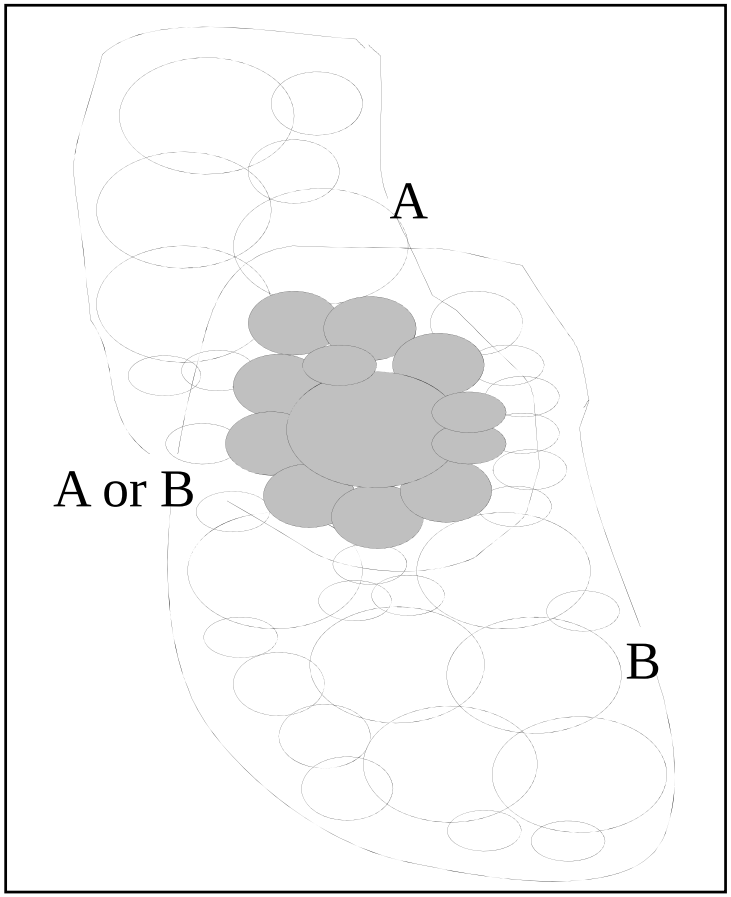
Regions for Two Classes of Patterns (RCE)

- **Network during training on multiple samples to classify patterns into two classes**



Separable and Confusion Regions (RCE)

- **Separable regions mapped deterministically**
- **Confusion region mapped probabilistically**

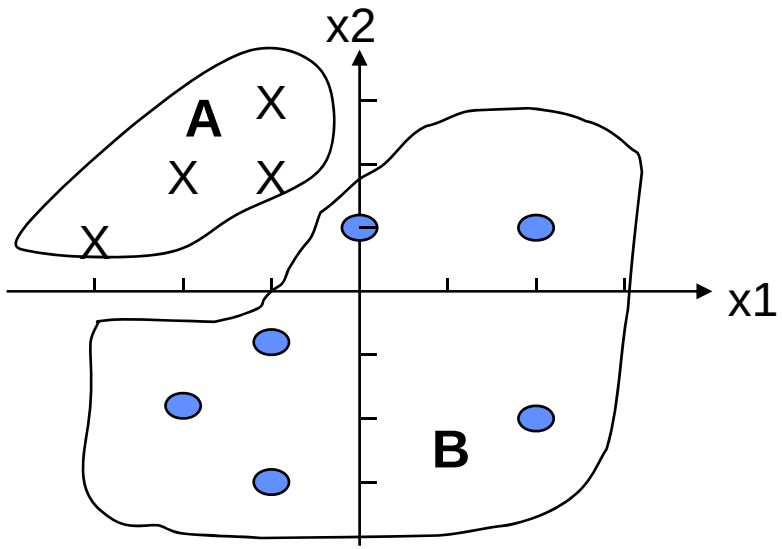


A Simple Example of RCE Learning

- **The previous example for single perceptron:**
  - » **Class A with four patterns**

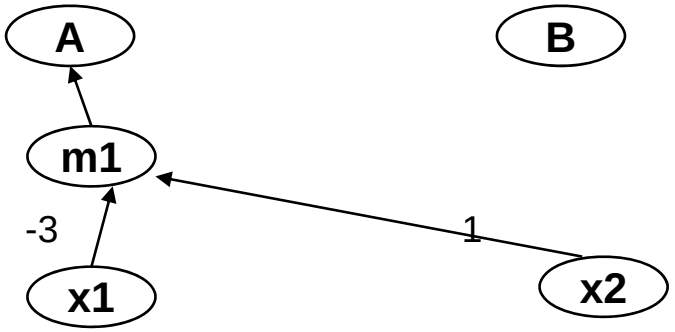
    (-3, 1), (-2, 2), (-1, 2), (-1, 3)
  - » **Class B with six patterns given**

    (-1, -1), (-2, -2), (-1, -3) (2, -2), (2, 1), (0, 1)



A Simple Example of RCE Learning …

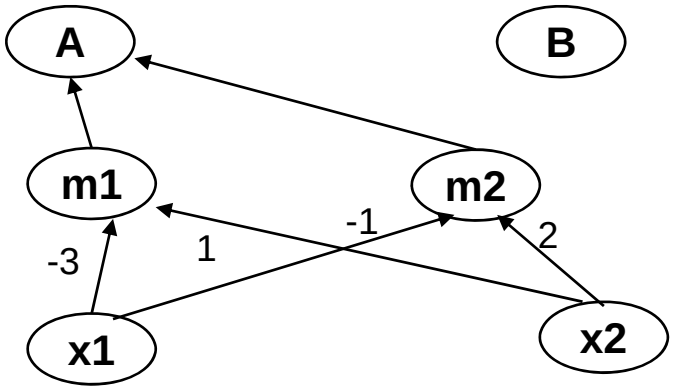
- **Assume**
  - » **Distance measure $[\Sigma(x_i - x'_i)^2]^{1/2}$**
  - » **Distance threshold = 2**
- **Input pattern (-3, 1) -> "A"**
  - » **A new internal node "m1" committed and connected to "A"**



- **Input pattern (-2, 2) -> "A"**
  - » **The activation of node "A"**
    - ♦ **Distance to "m1": $(2)^{1/2} \approx 1.4 <$ threshold**
    - ♦ **"A" is activated, and NO error**

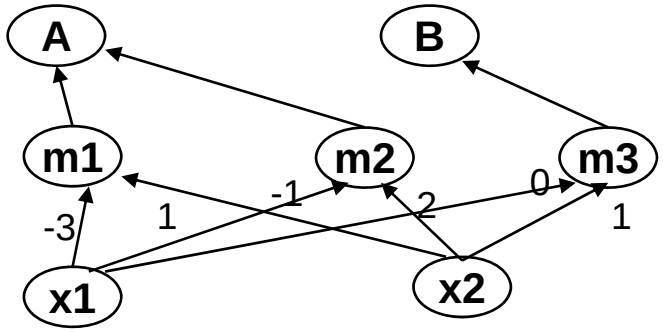
A Simple Example of RCE Learning …

- **Input pattern (-1, 2) -> "A"**
  - » **Distance to "m1": $(5)^{1/2}$ > threshold**
  - » **"A" is not activated, a "+1" error produced**
  - » **A new internal node "m2" committed and connected to "A"**



A Simple Example of RCE Learning …

- **Input pattern (0, 1) -> "B"**
  - » **Distance to "m1":** **3 > threshold**
  - » **Distance to "m2":** **1.4 < 2**
  - » **"A" is activated, a "-1" error produced, so the threshold of "m2" should be reduced**
  - » **At the same time, "B" is not activated, a "+1" error produced, so a new internal node "m3" committed and connected to "B"**



- **… …**

Summary of RCE Networks

- **Characteristics of RCE learning**
  - » **The hidden layer of the network is not pre-configured but dynamically constructed during training (*incremental learning*) – *a definite advantage***
  - » **Learning is through the commitment of prototypes and modification of categorization**
  - » **Learning stops when there is no new prototype created and no modification of prototype**
  - » **The network is not forced to make a decision on novel patterns**

- ***Advantages:***
  - » **Can learn any complex decision region**
  - » **Fast learning**
  - » **Low computational requirements**

- ***Disadvantages:***
  - » **Does not generalize well**
  - » **Requires separate output node for each category**
  - » **May grow large for some problems**

Some Applications of RCE Network

Invariant Object Recognition

Recognition of objects in an image, independent of their position, orientation or even some deformation. An RCE network can be trained to do multiple, invariant object recognition using low dimensional feature training vectors.

Fish counting

Farmers need to track fish by counting the density of minnows in an image sample. Automated methods have been tried, but are not accurate (10% error rate). RCE network solutions were found to be better. Minnows are estimated by summing clumps. Object features used are area, perimeter, dispersion, diameter, etc.

Character recognition on X-Ray Films

The identification of patient's X-ray films using ID numbers on the film. An RCE network performed the recognition process with recognition rates of 99.4% accuracy. Preprocessing of the ID image includes segmentation, rotation and binarization.

Some Applications of RCE Network (cont.)

Liver Diagnosis

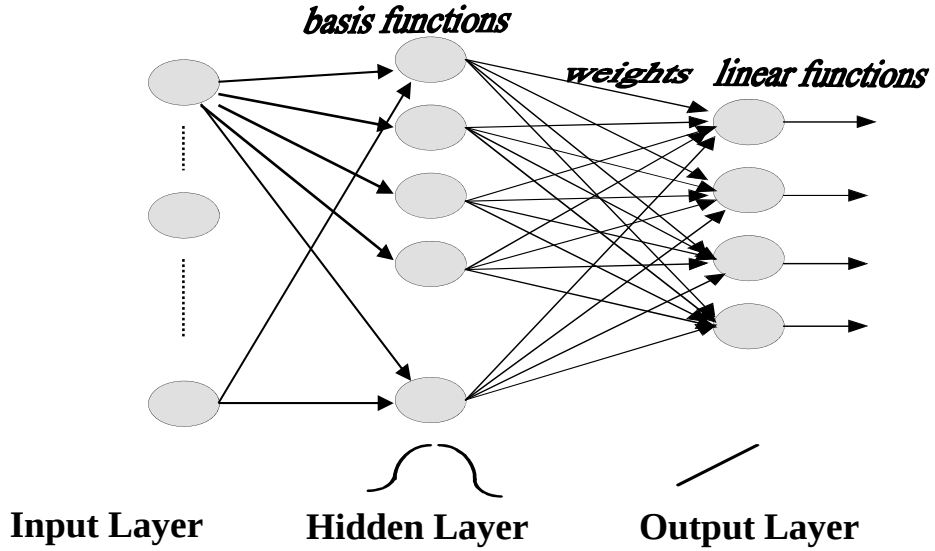
Human livers can be classified as either normal or abnormal through ultrasonic images. Features used to characterize liver images are numerous (mean and mode intensity, maximum gray-level frequency, min and max intensity values, intensity variance and standard deviation, smoothness, skewness and so on). Classification methods were examined and compared: nearest neighbour clustering, linear discriminate analysis, MLFF networks with BP, Counter Propagation networks and RCE networks. None of the methods compared achieved the 90% accuracy attained by the RCE network.

Detection of Surface Defects in Rolled Metal Products

Surface defects are created during the rolling process in metal products. Detection methods use an operator to monitor a video display to warn of potential flaws. An RCE network trained on 19 surface features (geometry of the defect envelope, the distribution of pixel activity, and histograms of pixel intensities) learned to classify some 15 different classes of defects. The system is incrementally trained on new defects as they are encountered.

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

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

      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

$$\sigma_j^2 = \frac{1}{m_j} \sum_{\mathbf{x} \in \Theta_j} (\mathbf{x} - \mathbf{c}_j)^T (\mathbf{x} - \mathbf{c}_j) \qquad j = 1, 2, ..., h$$

where m_j is the number of patterns in cluster Θ_j

RBF Network Learning (cont.)

Learning in output layer

- **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*:

$$\delta_j = T_j - O_j$$

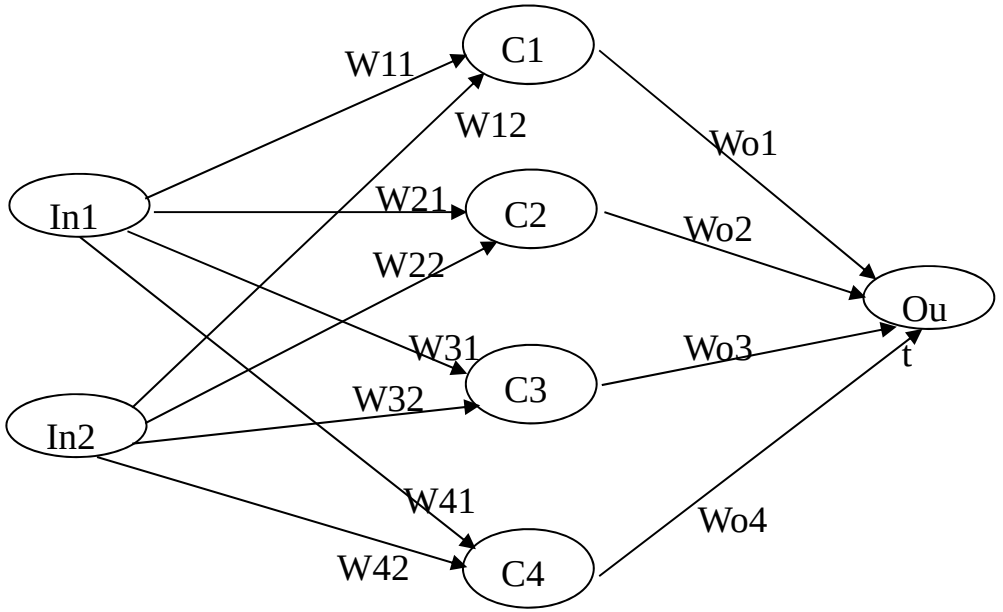
where T_j is the desired (target) output activation and O_j is the actual output activation at output unit *j*.

- **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:** $\mathbf{Out = \Sigma_i W_{oi} O_i}$

- **The initial weights of the output layer are:**
  $\mathbf{W_{o1} = 0.5, W_{o2} = 0.5, W_{o3} = 0.5,}$ **and** $\mathbf{W_{o4} = 0.5.}$

- **The weight adjustment is based on:**

$$\mathbf{W_{oi}(t+1) = W_{oi}(t) + \Delta W_{oi}}$$
$$\mathbf{\Delta W_{oi} = \eta(T_{out} - A_{out})O_i}$$

where η = 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**
    - ♦ **$Wo1(t+1) = Wo1(t) + \Delta Wo1 = 0.5 + 0.5 * (0 – 0.5) * 1 = 0.25$**
  - » **Input P2:**
    - ♦ **only C2 is activated. Out = 0.5**
    - ♦ **$Wo2(t+1) = Wo2(t) + \Delta Wo2 = 0.5 + 0.5 * (1 – 0.5) * 1 = 0.75$**
  - » **……**

Summary of RBF Networks

- **Applications of RBF include function approximation, kernel regression, forecasting and the estimation of class probabilities for classification problems.**

- **RBF nets are often found to be superior to MLFF networks for some applications (e. g. forecasting, or complex mapping tasks).**

- **Comparison of RBF with MLFF+BP:**
  - » **Both can approximate arbitrary non-linear functional mappings between multidimensional spaces. Both are examples of nonlinear multi-layer feedforward networks.**

Summary of RBF Networks …

- **Difference between MLFF+BP and RBF**

Structure

- **MLFF+BP may have one or more hidden layers.**
- **RBF has a single hidden layer.**

Neuronal model

- **MLFF+BP: the computational nodes located in a hidden layer or output layer share a common neuronal model.**
- **RBF: the computational nodes in the hidden layer are quite different and serve a different purpose from those in the output layer.**

Linear / nonlinear

- **MLFF+BP: all layers are usually non-linear.**
- **RBF: The hidden layer is nonlinear, but the output layer is linear.**

Activation function

- **MLFF+BP: each hidden unit computes the *inner product* of the input vector and the weight vector of that unit.**
- **RBF: each hidden unit computes the *Euclidean distance* between the input vector and the centre (weight vector) of that unit.**

Summary of RBF Networks …

- **Difference between MLFF+BP and RBF (cont.)**
  - » **Approximation**
    - ♦ **MLFF+BP: construct global approximations to input-output mapping (sigmoid function assumes nonzero values over an infinitely large region of the input space)**
    - ♦ **RBF: construct local approximations to nonlinear input-output mappings (radial basis function covers only a small region)**
- **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
— Mathematical background —

- **One important goal of *statistical reasoning* is to predict the value of a variable from known values of one or more other variables. Such predictions are important when some variable cannot be observed directly and must be estimated from other relevant variables. The variable to be predicted is called the *dependent variable,* and the variables used as predictors are called the *predictor variables*.**

- ***The principle of least squares* lies at the heart of regression. It minimizes the squared difference between the predicted value and the actual value of the dependent variable using calculus based method.**

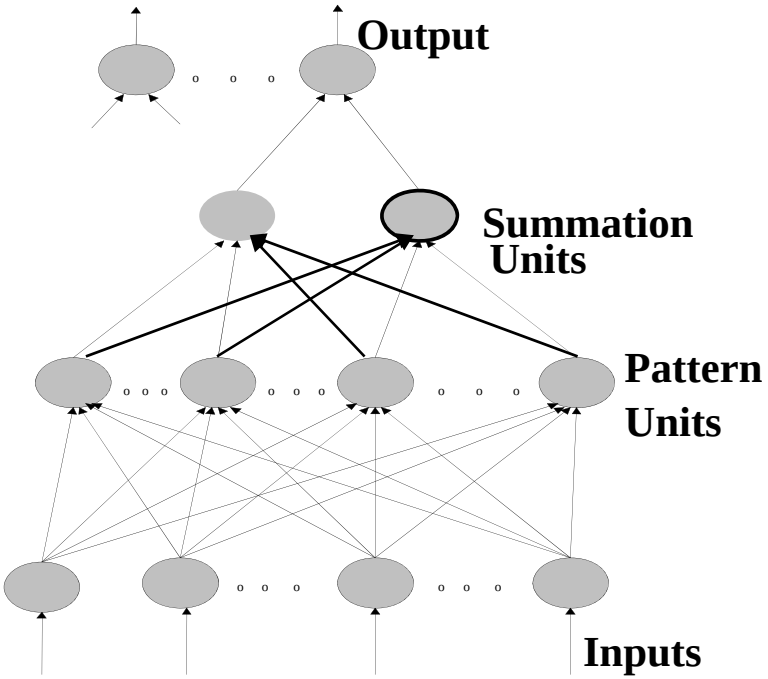
- **Statistical analysis based on least squares is known as *regression analysis.* The regression of z on x is defined as the conditional mean of z given x**

$$E[z \mid \mathbf{x}] = \frac{\int_{-\infty}^{\infty} z f(\mathbf{x}, z)\, dz}{\int_{-\infty}^{\infty} f(\mathbf{x}, z)\, dz}$$

where E[.] is the expected value operator. The joint probability density *f*(x, z) is usually not known, it may be estimated from sample data points of x and z by Parzen window approach.

Architecture of General Regression NN

- **The GRNN architecture consists of**
  - » **an input-layer**
  - » **three computational layers: pattern, summation, output**
- **It is similar to the probabilistic neural networks (PNN). The major difference is GRNN provides estimation of *continuous variables*, but PNN provides only *binary outputs*.**



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.**
- **Performs nonlinear, nonparametric regression (for estimation applications) using Parzen windows, a nonparametric statistical estimation technique.**
- **GRNN will, in general, outperform other estimation methods, even with a sparse number of points .**

***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.**

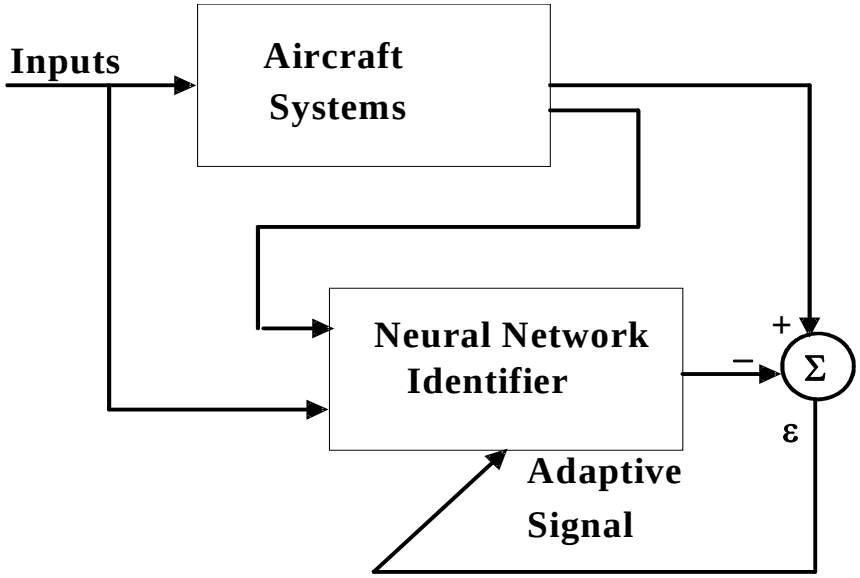
An Example of GRNN Application
— Adaptive Control —

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. In a study conducted by Youssef (1993), 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 MLFF (two hidden-layer) network with BP**
  - » **a Radial basis function network**
  - » **a GRNN**

- **The MLFF 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
MLFF	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

Probabilistic Neural Networks

- PNN are quite similar in architecture to GRNN, but are suitable for classification problems only unlike GRNN

- Donald Specht invented them in 1990

- They can be equated to Kernel Discriminant Analysis. Kernels are also called Parzen windows. These are usually probability density functions such as Gaussian.

- PNN can train quickly on sparse data sets

Probabilistic Neural Networks…

- **Like GRNN, they are also fast, one-pass algorithms**
- **Like GRNN, they are also memory based**
- **PNN are universal approximators for smooth classification problems**
- **Like GRNN, they also suffer from curse of dimensionality – because for every training pattern/sample a separate hidden node is allotted.**
- **Like GRNN, the performance of PNN is totally governed by a smoothing parameter**