

COMP4302/COMP5322, Lecture 8
NEURAL NETWORKS

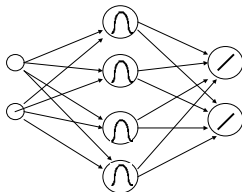
Radial-Basis Function Neural Networks

Outline

- RBF nets for exact approximation
- RBF nets for interpolation
- RBF nets for classification

Radial-Basis Function Neural Networks (RBF)

- Consist of 2 layers
 - Non-linear first layer (Gaussian transfer functions typically used)
 - Linear output layer
- Fully connected network – all neurons of the previous layer are connected with all of the next layer



Interpolation and Approximation

- **RBF networks** has their origins in techniques for performing exact *interpolation* of a set of input-output samples
 - (exact) *interpolation* – after training NN gives the exactly correct outputs for all training data
 - *approximation* – NN relates well the training input vectors to the desired outputs but does not necessary produce the exactly correct outputs
 - **Vacations example**
 - How much you enjoy vacations as a function of their duration
- | Ex. | Duration (days) | Rating (1-10) |
|-----|-----------------|---------------|
| 1 | 4 | 5 |
| 2 | 7 | 9 |
| 3 | 9 | 2 |
| 4 | 12 | 6 |
- We would like to create an interpolation and approximation networks that predict the ratings for other durations
 - Given data is called sample input-output pairs (training data)

COMP4302/5322 Neural Networks, w8, s2 2003

4

Interpolation Task

- **Given:**
 - 1) a black box with n inputs x_1, x_2, \dots, x_n and 1 output t ,
 - 2) a database of k sample input-output $\{x, t\}$ combinations (training examples)
- **Goal:** create a NN that predicts the output given an input vector x :
 - if x is a vector from the sample data, it must produce the exact output of the black box
 - otherwise it must produce output close to the output of the black box

COMP4302/5322 Neural Networks, w8, s2 2003

5

Interpolation with RBF NNs

- We will show that RBF networks enable interpolation if
 - There are as many neurons in the first layer as the number of samples
 - The neurons in the first layer compute the Gaussian of the distance between the current input vector and a sample input vector (i.e. they are centered on the sample data)
 - Each neuron in the output layer adds its inputs together
 - The weights between the two layers are adjusted such that the output of the neuron(s) in the second layer is exactly the desired output for each sample

COMP4302/5322 Neural Networks, w8, s2 2003

6

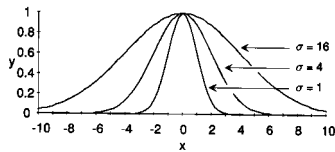
Solution to the Interpolation Task

- **One possible solution: construct $y(x_1, \dots, x_n)$, so that**
 - y is the output of the black box if x is a vector from the sample data
 - y is as close to the output of the black box for other input vectors
- **How to construct y ?**
 - y is a weighted sum of other functions $f_i : y(x_1, \dots, x_n) = \sum_{i=1}^k w_i f_i(x_1, \dots, x_n)$
 - each f_i depends on the distance between the current input vector x and the i -th sample input vector from the database c_i : $f_i(x) = \varphi_i(\|x - c_i\|)$
 - i.e. each sample from the database (training data) is a reference point (center) established by the i -th input-output pair
 - f_i gets its maximum value when x is close to c_i
 - each f_i depends on only one center \Rightarrow it specializes in handling the influence of the i -th sample on future predictions

Solution to the Interpolation Task – cont. 1

- **What function to choose for φ_i ?**
 - Gaussian – its bell shape is easy to control with a parameter σ (width)
 - Larger $\sigma \Rightarrow$ bigger spread out

$$\varphi_i(\|x - c_i\|) = e^{-\frac{\|x - c_i\|^2}{2\sigma}}$$



Solution to the Interpolation Task – cont. 2

- **Thus:** $y(x) = \sum_{i=1}^k w_i e^{-\frac{\|x - c_i\|^2}{2\sigma}}$
- **Can be computed by a RBF network with k neurons in the hidden layer and 1 output neuron**
 - The first layer is similar to competitive layers – each neuron responds vigorously to one sample input vector
 - Centers are the input vectors, σ is a parameter; if small – each input vector will have only local influence, if wide – global influence
 - The output layer adds the weighted outputs of the hidden layer
- **It is easy to see that if the Gaussian bells are extremely narrow, the approximation task is solved**
 - How many Gaussians will contribute substantially to the output for a given input vector?
 - What will be the values of the weights w_i ? How will they relate to the output value of the i -th input-output example?

Solution to the Interpolation Task – cont. 3

- In practice the Gaussians may have substantial reach, i.e. many of them contribute substantially to the output value for a particular input-output example
 - It is still possible (and easy) to compute a set of weights such that the correct output is produced for each input-output examples
- Given σ , we can compute the weights
 - We need to solve a system of linear equations as shown in the next example
- Then the net can be used to predict not only the values of the samples but for any input

Interpolation Net for the Vacation Example

- How many neurons in the first layer?
- How many output neurons?

Equations for the Vacation Example

- Given σ , we can compute the weights
- A system of linear equations
 - Which are the unknowns? How many are they?
 - How many equations?

$$y(x_1) = w_1 e^{-\frac{\|x_1 - x_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|x_1 - x_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|x_1 - x_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|x_1 - x_4\|^2}{2\sigma}}$$

$$y(x_2) = w_1 e^{-\frac{\|x_2 - x_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|x_2 - x_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|x_2 - x_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|x_2 - x_4\|^2}{2\sigma}}$$

$$y(x_3) = w_1 e^{-\frac{\|x_3 - x_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|x_3 - x_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|x_3 - x_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|x_3 - x_4\|^2}{2\sigma}}$$

$$y(x_4) = w_1 e^{-\frac{\|x_4 - x_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|x_4 - x_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|x_4 - x_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|x_4 - x_4\|^2}{2\sigma}}$$

How to Create RBF Approximation Nets

- For a few samples, create an approximation net
- Select a learning rate
- Until performance is satisfactory (error below a threshold or max number of epochs reached):
 - For all sample inputs
 - Compute the resulting outputs
 - Compute the weight change
 - Add up the weight changes for all the sample inputs and update the weights (batch mode) or update the weights after each example (approximate gradient descent)

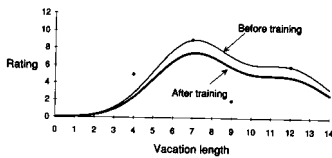
COMP4302/5322 Neural Networks, w8, s2 2003

16

Approximation Net for Vacation Example

- 2 neurons in the first layer
- initialized to example 2 (7 days) and 4 (12 days)
- Learning rate = 0.1, $\sigma=4$
- The weights after 100 epochs

	w1	w2	c1	c2
Initial values	8.75	5.61	7	12
Final values	7.33	4.47	7	12



17

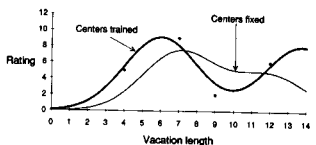
Approximation Nets - Modification

- Not only weights but also centers can be adjusted
- How much? Find the derivatives of the error function with respect to the center coordinates

$$\Delta c_{ij} = \eta \sum_k w_i(t_k - y_k) e^{-\frac{(x_{ij} - c_{ij})^2}{2\sigma^2}} \frac{1}{\sigma} (x_{ij} - c_{ij})$$

- The weights after 100 epochs

	w1	w2	c1	c2
Initial values	8.75	5.61	7	12
Final values	9.13	8.06	6	13.72



• Adjustment of both weights and centers provides a better approximation than adjustment of either of them alone

8, s2 2003

18

RBF NNs for Classification

Cover's Theorem on the Separability of Patterns

- A complex classification problem cast in high dimensional space nonlinearly is more likely to be linearly separable than in a low dimensional space (Cover, 1965)
 - 1. Nonlinear hidden functions
 - 2. Hidden-neuron space > input space (i.e. number of hidden neurons > dimension of input data)
- However, in some cases the use of non-linear mapping (i.e. point 1) may be sufficient to produce linearly separable data without having to increase the dimensionality of the hidden-neurons space, see the next example

RBF NNs on XOR Problem

- XOR problem revisited

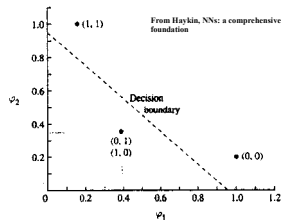
Input	Output
(0, 0)	0
(1, 1)	0
(0, 1)	1
(1, 0)	1

- Define a pair of Gaussian functions

$$c_1 = (1,1), \varphi_1(x) = e^{-\|x-c_1\|^2}$$

$$c_2 = (0,0), \varphi_2(x) = e^{-\|x-c_2\|^2}$$

- After the transformation the classes are linearly separable



- No increase in the dimensionality of the hidden neuron space compared to input space => the nonlinear transformation was sufficient to transform the XOR problem into linearly separable one

Creating RBF Networks for Classification

- 1. Select the locations of the centers c_i
 - Choose them randomly from the training data
 - Use clustering algorithm to find them, e.g. k-means, fuzzy c-means, SOM
 - K-means
 - choose randomly k samples from the data to initialise the seeds
 - run the k-means algorithm
 - Once it converges, set the centers of the RBF network to the centroids of the clusters

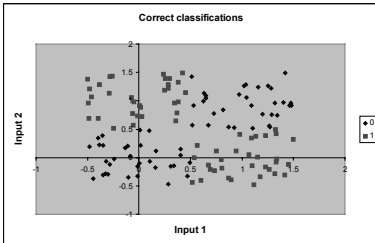
How to Create RBF Networks – cont.

- 2. Fit the RBFs, i.e. set the width of the Gaussians
 - Set their width using the p-nearest neighbor method
 - σ_j of a center j is the root mean squared distance from that center to the p nearest centers
 - p is set by trial and error, typically p=2
 - Small p => narrower Gaussians (may be expected to emphasize the nature of the training data, greater risk of overtraining)
 - Higher p => will broaden the Gaussians, may be expect to suppress the distinction between clusters
 - Other heuristics: $\sigma_j = \sqrt{d_j}$ where d_j is the distance to the nearest center
- 3. Gradient descent to train the linear layer
 - Typically one output neuron for each class => i.e. binary encoding of the targets

Classification Example – Noisy XOR Problem

• Noisy XOR truth table:

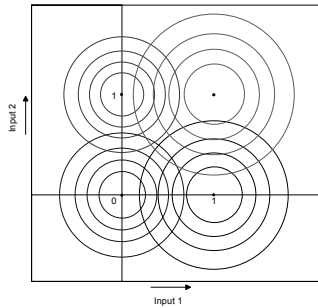
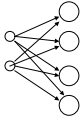
Input 1	Input 2	Output
<0.5	<0.5	0
<0.5	>0.5	1
>0.5	<0.5	1
>0.5	>0.5	0



Example taken from <http://www.uea.ac.uk/~kb/>

Noisy XOR Problem – cont. 1

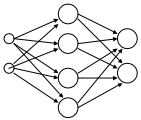
1. Clustering the data using k-means ($k=4$)
2. Locating the centers of the RBF by setting them to the centroids
3. Determining their widths
4. The corresponding RBF net
 - 2 input and 4 RBF neurons



Gaussian RBF fitted at the centroids (different widths)

Noisy XOR Problem – cont. 1

5. Create an output neuron for each class and use binary encoding for the targets (10 for class 1 and 01 for class 2)

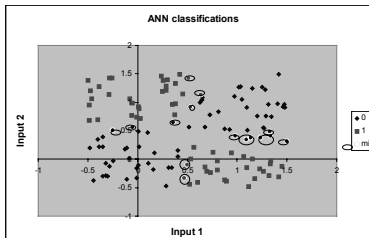


6. Initialize the weights between first and output layer to small random values
7. Train the weights between the first and output layer using the LMS algorithm

Noisy XOR Problem – cont. 2

Classification by the RBF network (misclassifications are circled)

- Where are the misclassifications?
- Why RBF nets are not perfect fit for this data?



RBF networks and MLPs - Similarities

- **Similarities - both MLP and RBF NNs are:**
 - examples of non-linear, layered, feedforward networks
 - universal approximators
 - For the universality of MLPs see lectures 4,5
 - Universality of RBF NNs: Any function can be approximated to arbitrary small error by an RBF NN given there are enough RBF neurons (Park and Sandberg, 1991)
 - Hence, there always exist an RBF network capable of accurately mimicking a specified MLP, and vice versa

RBF networks and MLPs – Differences

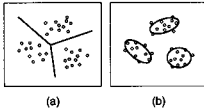
- An RBF network has a simple architecture (2 layers: nonlinear and linear) with single hidden layer; MLP may have 1 or more hidden layers.
- MLPs may have a complex pattern of connectivity (not fully connected), RBF is a fully connected network.
- The hidden layer of an RBF net is non-linear, and the output layer is linear. The hidden and output layers of an MLP used as a classifier are usually nonlinear. When MLPs are used for nonlinear regression, a linear output layer is preferred. Thus, MLPs may have variety of activation functions.
- Hidden and output neurons in a MLP share a common neuron model (nonlinear transformation of linear combination). Hidden neurons in RBF nets are quite different and serve a different purpose than the output neurons.

RBF networks and MLPs – Differences (cont. 1)

- **Computation at hidden neurons**
 - The argument of the activation function of a hidden neuron in RBF nets computes the Euclidean distance between the input vector and the center of the neuron.
 - The activation function of a hidden neuron in MLP computes the inner product of the input vector and the weight vector of that neuron.
- **Representation formed (in the space of hidden neurons activations with respect to the input space)**
 - A MLP is said to form a distributed representation since for a given input vector many hidden neurons will typically contribute to the output value.
 - RBF nets form a local representation as for a given input vector only a few hidden neurons will have significant activation and contribute to the output.

RBF networks and MLPs – Differences (cont. 2)

- Differences in how the weights are determined
 - In MLP - usually at the same time as a part of a single global training strategy involving supervised learning.
 - A RBF net is typically trained in 2 stages with the RBFs being determined first by unsupervised technique using the given data alone, and the weights between the hidden and output layer being found by fast linear supervised methods.
- Differences in the separation of the data
 - Hidden neurons in MLP define hyper-planes in the input space (a)
 - RBF nets fit each class using kernel functions (b)



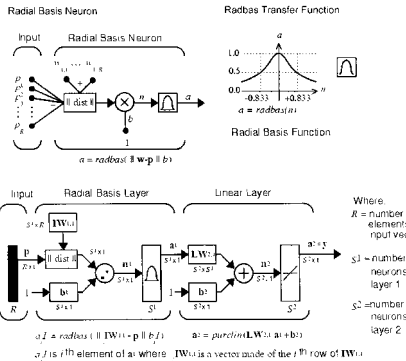
COMP4302/5322 Neural Networks, w8, s2 2003

RBF networks and MLPs – Differences (cont. 3)

- Number of neurons
 - MLPs typically have less number of neurons than RBF NNs. This is because they respond to large regions of the input space.
 - RBF nets may require many RBF neurons to cover the input space (larger number of input vectors and larger ranges of input vectors => more RBF neurons are needed.) However, they are capable of fast learning and have reduced sensitivity to the order of presentation of training data.

COMP4302/5322 Neural Networks, w8, s2 2003

RBF networks in Matlab



RBFs in Matlab -cont.

- `net=newrbe(P,T,spread)` – used for exact approximation, creates as many RBF neurons as there are input examples
- `net=newrb(P,T,goal,spread)` – used for interpolation; starts with one RBF neuron and grows them incrementally until the error falls below the `goal`
