technische universität dortmund

Computational Intelligence

Winter Term 2012/13

Prof. Dr. Günter Rudolph

Lehrstuhl für Algorithm Engineering (LS 11)

Fakultät für Informatik

TU Dortmund

Plan for Today

- Application Fields of ANNs
  - Classification
  - Prediction
  - Function Approximation
- Radial Basis Function Nets (RBF Nets)
  - Model
  - Training
- Recurrent MLP
  - Elman Nets
  - Jordan Nets

Application Fields of ANNs

Classification

given: set of training patterns (input / output) — input: $\tilde{x}_i$, output: $\tilde{y}_i$

output = label
(e.g. class A, class B, ...)

$$\underset{\text{input (unknown)}}{f(x};\ \overbrace{\underbrace{(\tilde{x}_1,\tilde{y}_1),\ldots,(\tilde{x}_m,\tilde{y}_m)}_{\text{training patterns (known)}},\ \underbrace{w_1,\ldots,w_n}_{\text{weights (learnt)}}}^{\text{parameters}}) \to \underset{\text{output (guessed)}}{\hat{y}}$$

phase I:

train network

phase II:

apply network to unkown inputs for classification

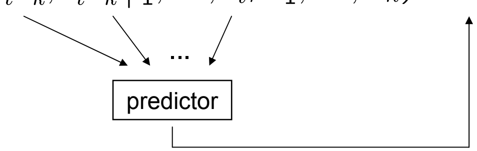
Application Fields of ANNs

Prediction of Time Series

time series $x_1, x_2, x_3, \ldots$ (e.g. temperatures, exchange rates, ...)

task: given a subset of historical data, predict the future

$$f(x_{t-k}, x_{t-k+1}, \ldots, x_t; w_1, \ldots, w_n) \to \hat{x}_{t+\tau}$$



training patterns:

historical data where true output is known;

error per pattern = $(\hat{x}_{t+\tau} - x_{t+\tau})^2$

phase I:

train network

phase II:

apply network to historical inputs for predicting <u>unkown</u> outputs

Prediction of Time Series: Example for Creating Training Data

given: time series 10.5, 3.4, 5.6, 2.4, 5.9, 8.4, 3.9, 4.4, 1.7

time window: k=3

(10.5, 3.4, 5.6) 2.4 first input / output pair

known input known output

further input / output pairs:

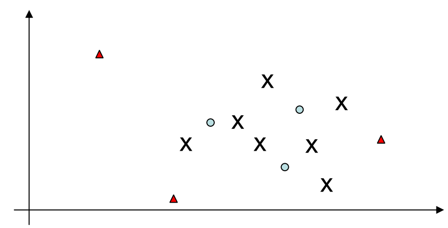
(3.4, 5.6, 2.4) 5.9
(5.6, 2.4, 5.9) 8.4
(2.4, 5.9, 8.4) 3.9
(5.9, 8.4, 3.9) 4.4
(8.4, 3.9, 4.4) 1.7

Function Approximation (the general case)

task: given training patterns (input / output), approximate unkown function

→ should give outputs close to true unkown function for arbitrary inputs

- values between training patterns are **interpolated**
- values outside convex hull of training patterns are **extrapolated**



x : input training pattern

○ : input pattern where output to be interpolated

▲ : input pattern where output to be extrapolated

Definition:
A function $\phi : \mathbb{R}^n \to \mathbb{R}$ is termed **radial basis function** iff $\exists\, \varphi : \mathbb{R} \to \mathbb{R} : \forall x \in \mathbb{R}^n : \phi(x; c) = \varphi(\, || x - c ||\,)$. □

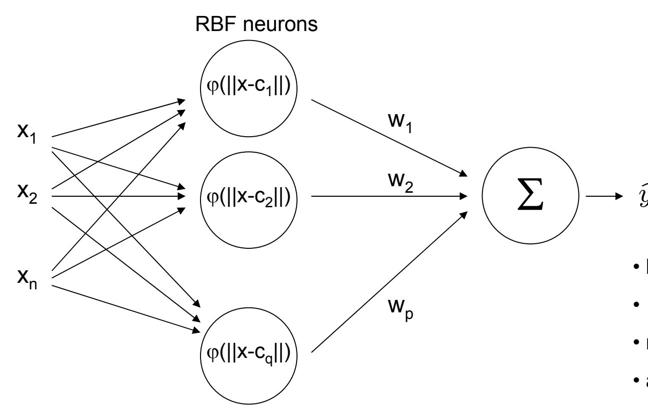
Definition:
RBF **local** iff
$\varphi(r) \to 0$ as $r \to \infty$ □

typically, || x || denotes Euclidean norm of vector x

examples:

$\varphi(r) = \exp\left(-\frac{r^2}{\sigma^2}\right)$	Gaussian	unbounded	local
$\varphi(r) = \frac{3}{4}(1 - r^2) \cdot 1_{\{r \le 1\}}$	Epanechnikov	bounded	local
$\varphi(r) = \frac{\pi}{4}\cos\left(\frac{\pi}{2} r\right) \cdot 1_{\{r \le 1\}}$	Cosine	bounded	local

Definition:
A function $f: \mathbb{R}^n \to \mathbb{R}$ is termed **radial basis function net (RBF net)**
iff $f(x) = w_1\, \varphi(|| x - c_1 ||) + w_2\, \varphi(|| x - c_2 ||) + \ldots + w_p\, \varphi(|| x - c_q ||)$ □



- layered net
- 1st layer fully connected
- no weights in 1st layer
- activation functions differ

given : N training patterns (x_i, y_i) and q RBF neurons

find : weights $w_1, ..., w_q$ with minimal error

solution:

we know that $f(x_i) = y_i$ for i = 1, ..., N and therefore we insist that

$$\sum_{k=1}^{q} w_k \cdot \varphi(\|x_i - c_k\|) = y_i$$

w_k ↓ unknown; $\varphi(\|x_i - c_k\|)$ = p_{ik} known value; y_i ↓ known value

$$\Rightarrow \sum_{k=1}^{q} w_k \cdot p_{ik} = y_i$$

⇒ N linear equations with q unknowns

in matrix form: P w = y with $P = (p_{ik})$ and P: N x q, y: N x 1, w: q x 1,

case N = q: $w = P^{-1} y$ if P has full rank

case N < q: many solutions but of no practical relevance

case N > q: $w = P^{+} y$ where P^{+} is Moore-Penrose pseudo inverse

P w = y | · P‘ from left hand side (P‘ is transpose of P)

P‘P w = P‘ y | · $(P'P)^{-1}$ from left hand side

$(P'P)^{-1} P'P\, w = (P'P)^{-1} P'\, y$ | simplify

$(P'P)^{-1} P'P$: unit matrix; $(P'P)^{-1} P'$: P^{+}

complexity (naive)

$w = (P'P)^{-1} P' y$

P‘P: $N^2 q$ inversion: q^3 P‘y: qN multiplication: q^2

$O(N^2 q)$

remark: if N large then inaccuracies for P‘P likely

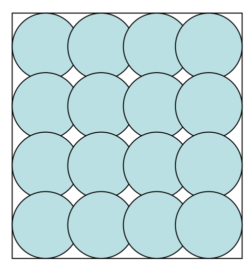
⇒ first analytic solution, then gradient descent starting from this solution

(gradient descent) requires differentiable basis functions!

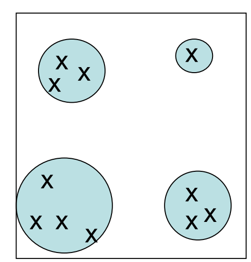
so far: tacitly assumed that RBF neurons are given

⇒ center c_k and radii σ considered given and known

how to choose c_k and σ ?



uniform covering



if training patterns inhomogenously distributed then first cluster analysis

choose center of basis function from each cluster, use cluster size for setting σ

advantages:

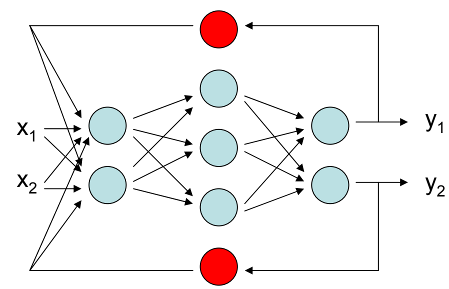
- additional training patterns → only local adjustment of weights
- optimal weights determinable in polynomial time
- regions not supported by RBF net can be identified by zero outputs

disadvantages:

- number of neurons increases exponentially with input dimension
- unable to extrapolate (since there are no centers and RBFs are local)

Jordan nets (1986)

- **context neuron**:
  reads output from some neuron at step t and feeds value into net at step t+1



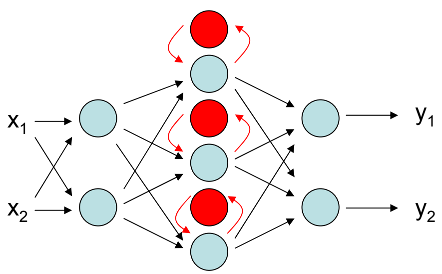
Jordan net =

MLP + context neuron for each output, context neurons fully connected to input layer

Elman nets (1990)

Elman net =

MLP + context neuron for each hidden layer neuron‘s output of MLP, context neurons fully connected to emitting MLP layer



Training?

⇒ unfolding in time (“loop unrolling“)

- identical MLPs serially connected (finitely often)
- results in a large MLP with many hidden (inner) layers
- backpropagation may take a long time
- but reasonable if most recent past more important than layers far away

Why using backpropagation?

⇒ use *Evolutionary Algorithms* directly on recurrent MLP!

later!