Neural Networks
(Chapter 9)

Kai Goebel, Bill Cheetham
GE Corporate Research & Development
goebel@cs.rpi.edu
cheetham@cs.rpi.edu

1

Outline

Introduction
Categories
Hopfield Net
Perceptron
- Single Layer
- Multi Layer

2

Introduction

Human brain is superior to digital computer at many tasks

+ e.g., processing of visual information

+ robust and fault tolerant (nerve cells in the brain die every day)

+ flexible; adjusts to new environment

+ can deal with information that is sparse, imprecise, noisy, inconsistent

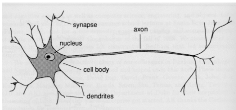
+ highly parallel

+ small, compact, dissipates very little power

- slower in primarily (simple) arithmetic operations

3

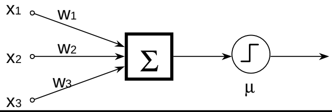
Neurons



McCulloch & Pitts (1943)

- simple model of neuron as a binary threshold unit

- uses step function to "fire" when threshold μ is surpassed



4

Real Neurons

Real Neurons

- use not even approximately threshold devices

- it is assumed they use a non-linear summation method

- produce a sequence of pulses (not a single output level)

- do not have the same fixed delay (t-> t+1)

- are not updated synchronously

- amount of transmitter substance varies unpredictably

5

Issues

What does that leave us with?

What is the best architecture?

(layers, connections, activation functions, updating, # units?)

How can it be programmed?

(can it learn, # examples needed, time to learn, amount of supervision, real-time learning)

What can it do?

(how many tasks, how well, how fast, how robust, level of generalization)

6

Neural Nets: Categorization

Supervised Learning
- Multilayer perceptrons
- Radial basis function networks
- Modular neural networks
- LVQ (learning vector quantization)

Reinforcement Learning
- Temporal Difference Learning
- Q-Learning

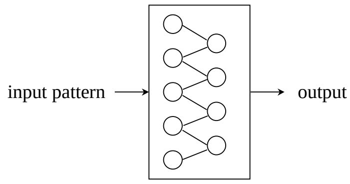
Unsupervised Learning
- Competitive learning networks
- Kohonen self-organizing networks
- ART (adaptive resonant theory)

7

Supervised Neural Networks

Requirement:

known input-output relations



8

Hopfield Model

Associative Memory is considered the "fruit fly" of this field.

It illustrates in the simplest possible manner the way that collective computation can work.

Store a set of patterns in such a way that when presented with a new pattern, the network responds by producing the closest stored pattern.

Conventional approach:

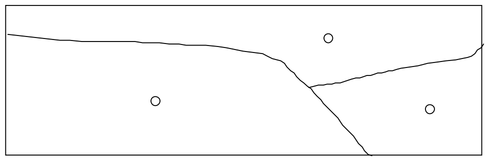
store a list of patterns, compute the Hamming distance, find the smallest, et voila!

9

Hopfield Network Operation

Picture is pattern; stored as attractor in the configuration space.

From arbitrary starting points, one attractor will be found

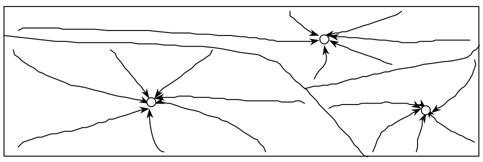


10

Hopfield Network Operation

Picture is pattern; stored as attractor in the configuration space.

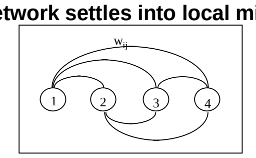
From arbitrary starting points, one attractor will be found



11

Hopfield Architecture

- Recurrent Network
- Symmetric Architecture
- Evaluation until no more changes are observed
  i.e., network settles into local minimum config.



- local minimum corresponds to "energy function"

$$E = -\frac{1}{2}\sum_{i,j} w_{ij}x_i x_j$$

12

Hopfield Network Equations

The operative equation, i.e., the network output at each step is

$$y_i = \operatorname{sgn}\left(\sum_i w_{ij} x_j\right)$$

where

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

13

Learning in Hopfield Models

Learning Rule:

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} + x_i x_j$$
$$w_{ii} = 0$$

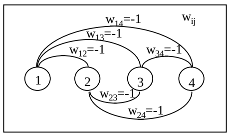
14

Hopfield Example

Learn x=[1 1 -1 -1]

which gives us the weight matrix

w=[0 1 -1 -1
1 0 -1 -1
-1 -1 0 1
-1 -1 1 0]



Now let's check the slightly corrupted pattern

p=[1 1 -1 1]

which will restore the pattern found close

y=[1 1 -1 -1]

with an energy level of E=-6

15

Hopfield Example

Learn second pattern x=[-1 -1 1 1]

which gives us the new weight matrix

w=[0 2 -2 -2
2 0 -2 -2
-2 -2 0 2
-2 -2 2 0]

Now let's check the slightly corrupted pattern

p=[-1 -1 -1 1]

which will restore the pattern

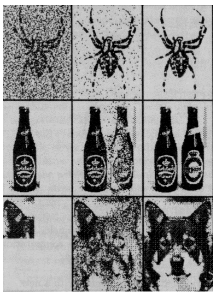
y=[-1 -1 1 1]

with an energy level of E=-12

16

More Complex Hopfield Examples

Reconstruction of Images

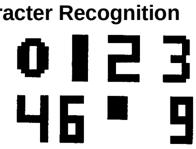


binary images are 130x180 pixels

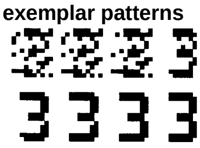
17

Hopfield Book Example

Character Recognition



eight exemplar patterns



18 output pattern for noisy "3" input

Hopfield: Issues

- Other memories can get lost
- Memories are created that were not supposed to be there
- crosstalk: if there are many memories, they might interfere
- no emphasis on learning; rather handcrafting to get desired properties
- goes towards optimization

19

Perceptrons

-Rosenblatt: 1950s

-Input patterns represented is binary

-Single layer network can be trained easily

-Output o is computed by

$$o = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

where

w_i is a (modifiable) weight

x_i is the input signal

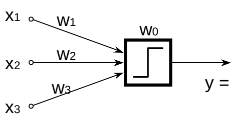
θ is some threshold (weight of constant input)

$f(\cdot)$ is the activation function $f(x) = \mathrm{sgn}(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases}$

20

Single-Layer Perceptrons

Network architecture



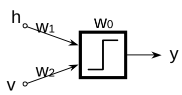
y = signum($\Sigma w_i x_i + w_0$)

21

Single-Layer Perceptron

Example: Gender classification (according to Jang)

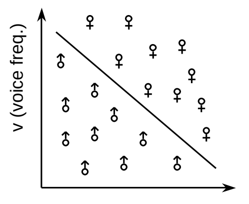
Network Arch.



y = signum($hw_1 + vw_2 + w_0$)

$= \begin{cases} -1 \text{ if female} \\ 1 \text{ if male} \end{cases}$

Training data



22

Perceptron

Learning:

select an input vector

if the response is incorrect, modify all weights

$$\Delta w_i = \eta\, t_i x_i$$

where

t_i is a target output

η is the learning rate

If a set of weights for converged state exists, then a method for tuning towards convergence exists (Rosenblatt, 1962)

23

ADALINE

single layer network (conceived by Widrow and Hoff)

output is weighted linear combination of weights

$$o = \sum_{i=1}^{n} w_i x_i - w_0$$

error is described as

$E_p = (t_p - o_p)^2$ (for pattern p)

where

t_p is the target output

o_p is the actual output

24

ADALINE

To decrease the error, the derivative wrt the weights is taken

$$\frac{\partial E_p}{\partial w_i} = -2\left(t_p - o_p\right)x_i$$

The delta rule is:

$$\Delta_p w_i = \eta\left(t_p - o_p\right)x_i$$

Intuitive appeal:

if $t_p > o_p$, boost o_p by increasing $w_i x_i$

increase w_i if x_i is positive

decrease w_i is x_i is negative

ADALINE and MADALINE

+ Simplicity of learning procedure

+ Distributed learning; can be performed locally at node level

+ on-line (pattern by pattern) learning

+ connect several ADALINEs to MADALINEs to deal with XOR problem

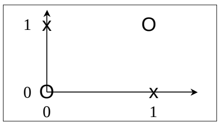
+ were used for noise cancellation, adaptive inverse control

- only one layer; no suitable training method for multi-layer perceptron … why?

XOR

Minsky and Papert reported a severe shortcoming of single layer perceptrons, the XOR problem…

x1	x2	output
0	0	0
0	1	1
1	0	1
1	1	0

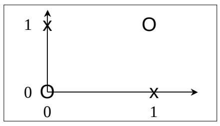


not linearly separable

XOR

Minsky and Papert reported a severe shortcoming of single layer perceptrons, the XOR problem…

x1	x2	output
0	0	0
0	1	1
1	0	1
1	1	0



not linearly separable

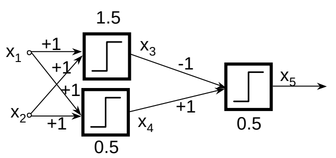
$$0w_1 + 0w_2 + w_0 \le 0 \Leftrightarrow w_0 \le 0$$
$$0w_1 + 1w_2 + w_0 > 0 \Leftrightarrow w_2 > -w_0$$
$$1w_1 + 0w_2 + w_0 > 0 \Leftrightarrow w_1 > -w_0$$
$$1w_1 + 1w_2 + w_0 \le 0 \Leftrightarrow w_1 + w_2 \le -w_0$$

Enter the Dark Ages of NNs

…which (together with a lack or proper training techniques for multi-layer perceptrons) all but killed interest in neural nets in the 70s and early 80s.

Multilayer Perceptrons

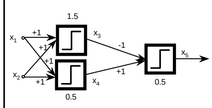
Two-layer perceptron

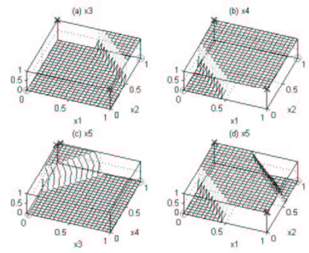


31

Two-Layer Perceptron: XOR

Node output as surface of their two inputs

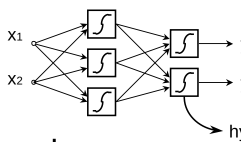




32 **note location of “o” and “x”**

Multilayer Perceptrons (MLPs)

Network architecture



Learning rule:

- **Steepest descent (Backprop)**
- **Conjugate gradient method**
- **All optim. methods using first derivative**
- **Derivative-free optim.**

33

Multi-Layer Perceptrons

-Recall the output

$$o = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

-and the squared error measure

$E_p = (t_p - o_p)^2$ **which is amended to** ${}^pE_k = \sum_{k=1}^{n} \left({}^pt_k - {}^po_k\right)^2$

-and the activation function

$f(x) = \frac{1}{1+e^{-x}}$ **or** $f(x) = \frac{1-e^{-x}}{1+e^{-x}} = \tanh\left(\frac{x}{2}\right)$ **or** $f(x) = x$

34 **then the learning rule for each node can be derived using the chain rule...**

Multi-Layer Perceptrons

-Recall the output

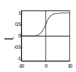
$$o = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

-and the squared error measure

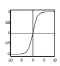
$E_p = (t_p - o_p)^2$ **which is amended to** ${}^pE_k = \sum_{k=1}^{n} \left({}^pt_k - {}^po_k\right)^2$

-and the activation function

$f(x) = \frac{1}{1+e^{-x}}$ **or** $f(x) = \frac{1-e^{-x}}{1+e^{-x}} = \tanh\left(\frac{x}{2}\right)$ **or** $f(x) = x$



squashing functions



35

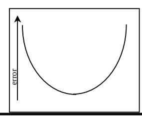
Backpropagation

make incremental change in the direction dE/dw to decrease the error.

The learning rule for each node can be derived using the chain rule...

...to propagate the error back through a multi-layer perceptron. $\frac{\partial E}{\partial parameters}$

$$\Delta w_{ki} = -\eta \sum_p \frac{\partial E_p}{\partial w_{ki}}$$



36

Soft Computing: Neural Networks

Back-prop procedure

1. Initialize weights to small random values
2. Choose a pattern and apply it to input layer
3. Propagate the signal forward through the network
4. Compute the deltas for the output layer
5. Compute the deltas for the preceding layers by propagating the error backwards
6. Update all weights
7. Go back to step 2 and repeat for next pattern
8. Repeat until error rate is acceptable

37

Soft Computing: Neural Networks

Step 3

Propagate signal forward through the network

$$O_i^m = g\left(\sum_j w_{ij}^m O_j^{m-1}\right)$$

until all outputs have been calculated

For m=0 (input layer), the output is the pattern.

38

Soft Computing: Neural Networks

Step 4

Compute the deltas for the output layer

$$\delta_i^M = g'\left(h_i^M\right)\left[t_i^p - O_i^M\right]$$

by comparing the actual output O
with the target output t
for the pattern p considered

39

Soft Computing: Neural Networks

Step 5

Compute the deltas for the preceding layers by propagating the errors backwards

$$\delta_i^{m-1} = g'\left(h_i^{m-1}\right)\sum_j^n w_{ij}^m \delta_j^m$$

for m=M, M-1, M-2, …
until a delta has been calculated for every unit

40

Soft Computing: Neural Networks

Step 6

Use

$$\Delta w_{ij}^m = \eta \delta_i^m O_j^{m-1}$$

to update all connections to

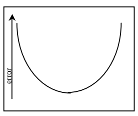
$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$

41

Soft Computing: Neural Networks

Step Size, Initial Weights

Step size:
- too big
- too small
- variable:
  - compute error
  - backpropagate
  - compute error again
  - if error bigger, reduce step size (0.5)
  - otherwise, increase a little (1.1)



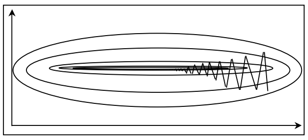
Initial weights: randomize (± 0)

42

Momentum

If error minimum in long narrow valley,

then updating can happen to zig-zag down the valley

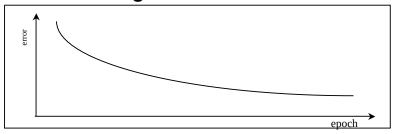


smoothes weight updating

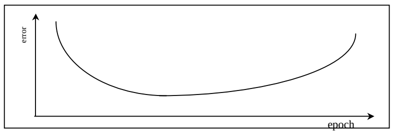
can speed learning up $\Delta w = -\eta \nabla_w E + \alpha \Delta w_{prev}$

Overfitting

Error on learning cases



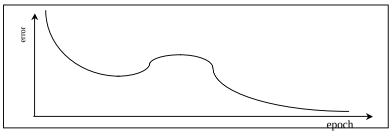
Error on validation cases



trained things that are accidental and unimportant

Local Minima

There is no guarantee that the algorithm converges to a global minimum



- check with different initial conditions (different weights, etc.)
- perturb the system (data) with noise to improve result

Architectures and other Techniques

Normalize weights

move weights same Euclidean distance

each epoch

Data scaling

Input scaling: allows weights to have

same order of magnitude

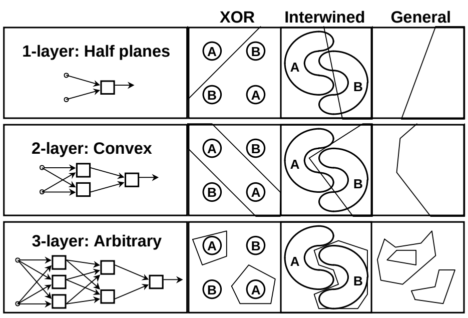
Output scaling: let target go between +- 0.9

to avoid saturation

What number of nodes per layer?

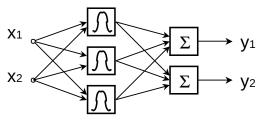
How many layers?

MLP Decision Boundaries



Radial Basis Function (RBF) Networks

Network architecture



Each node is described by a bell shaped function

$$o_i = \exp\left(-\frac{\|x - c_i\|^2}{2\sigma_i^2}\right)$$

where

c_i is the center of the curve

RBF

Output:
- **weighted sum**
- **weighted average**
- **linear combination**

Location of Center:
- **Use (fuzzy) k-means clustering**

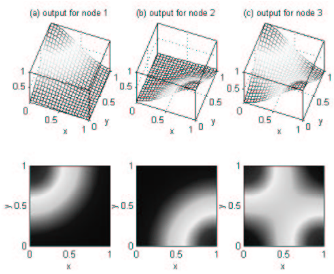
Size of Variance:
- **Use knn-classifier and take average distance**

49

Soft Computing: Neural Networks

XOR, revisited

x	y	output
0	0	0
0	1	1
1	0	1
1	1	0

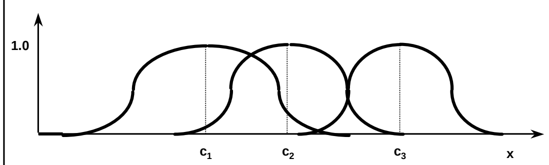


50

Soft Computing: Neural Networks

RBF and FIS

Consider the radial basis functions:



and a linear combination of the output variables

$y_i = \vec{a}_i \vec{o} + b_i$

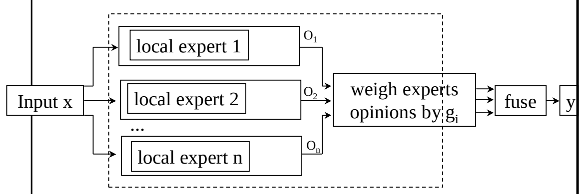
then the response is equivalent to ...

51

Soft Computing: Neural Networks

Modular Networks

Task decomposition
Local Experts
Fuse information



52

Soft Computing: Neural Networks

last slide

53