Neural Networks

(Chapter 9)

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

Outline

Introduction
Categories
Hopfield Net
Perceptron
Single Layer
Multi Layer
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

Neurons

McCulloch & Pitts (1943)
- simple model of neuron as a binary threshold unit
- uses step function to “fire” when threshold $\mu$ is surpassed
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

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)
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)

Supervised Neural Networks

Requirement:
known input-output relations

![Diagram of a neural network with input pattern and output]
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!

Hopfield Network Operation

Picture is pattern; stored as attractor in the configuration space. From arbitrary starting points, one attractor will be found.
Hopfield Network Operation

Picture is pattern; stored as attractor in the configuration space. From arbitrary starting points, one attractor will be found.

\[
E = -\frac{1}{2} \sum_{i,j} w_{ij} x_i x_j
\]
Hopfield Network Equations

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

\[ y_i = \text{sgn} \left( \sum_j w_{ij} x_j \right) \]

where

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

Learning in Hopfield Models

Learning Rule:

\[ w_{ij}^{(n+1)} = w_{ij}^{(n)} + x_i x_j \]

\[ w_{ii} = 0 \]
Hopfield Example

Learn \( x = [1 \ 1 \ -1 \ -1] \)

which gives us the weight matrix

\[
\begin{bmatrix}
0 & 1 & -1 & -1 \\
1 & 0 & -1 & -1 \\
-1 & -1 & 0 & 1 \\
-1 & -1 & 1 & 0
\end{bmatrix}
\]

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 \( E = -6 \)

---

Hopfield Example

Learn second pattern \( x = [-1 \ -1 \ 1 \ 1] \)

which gives us the new weight matrix

\[
\begin{bmatrix}
0 & 2 & -2 & -2 \\
2 & 0 & -2 & -2 \\
-2 & -2 & 0 & 2 \\
-2 & -2 & 2 & 0
\end{bmatrix}
\]

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 \( E = -12 \)
More Complex Hopfield Examples

Reconstruction of Images

binary images are 130x180 pixels

Hopfield Book Example

Character Recognition

\[
\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 \\
4 & \quad 6 & \quad \cdot & \quad 9
\end{align*}
\]

eight exemplar patterns

\[
\begin{align*}
3 & \quad 3 & \quad 3 & \quad 3 \\
3 & \quad 3 & \quad 3 & \quad 3
\end{align*}
\]

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

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

\( \theta \) is some threshold (weight of constant input)

\( f(\cdot) \) is the activation function

\[
f(x) = \text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}
\]
Single-Layer Perceptrons

Network architecture

\[ y = \text{signum}(\sum w_i x_i + w_0) \]

Example: Gender classification (according to Jang)

Network Arch.

\[ y = \text{signum}(hw_1 + vw_2 + w_0) \]

-1 if female
1 if male

Training data

\[ h \ (\text{hair length}) \]

\[ v \ (\text{voice freq.}) \]
Soft Computing: Neural Networks

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
  - \( \eta \) is the learning rate

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

---

Soft Computing: Neural Networks

ADALINE

- single layer network (conceived by Widrow and Hoff)
- output is weighted linear combination of weights
  \[ o = \sum_{i=1}^{n} w_i x_i - w_o \]
- 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
ADALINE

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

$$\frac{\partial E}{\partial w_i} = -2(t_p - o_p)x_i$$

The delta rule is:

$$\Delta w_i = \eta(t_p - o_p)x_i$$

Intuitive appeal:

- If $t_p > o_p$, boost $o_p$ by increasing $w_ix_i$
- Increase $w_i$ if $x_i$ is positive
- Decrease $w_i$ if $x_i$ 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?
Minsky and Papert reported a severe shortcoming of single layer perceptrons, the XOR problem...

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

not linearly separable

$0w_1 + 0w_2 + w_0 \leq 0 \iff w_0 \leq 0$

$0w_1 + 1w_2 + w_0 > 0 \iff w_2 > -w_0$

$1w_1 + 0w_2 + w_0 > 0 \iff w_1 > -w_0$

$1w_1 + 1w_2 + w_0 \leq 0 \iff w_1 + w_2 \leq -w_0$
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

Two-Layer Perceptron: XOR

Node output as surface of their two inputs

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.

Network architecture:

\[ X_1 \rightarrow \sum \rightarrow y_1 \]

\[ X_2 \rightarrow \sum \rightarrow y_2 \]

Learning rule:
- Hyperbolic tangent or logistic function

Multi-Layer Perceptrons

- Recall the output
- and the activation function
- and the squared error measure

\[ o = f\left( \sum w_i x_i + \theta \right) \]

\[ E = \frac{1}{2} (t - o)^2 \]

which is amended to

\[ E_i = \frac{1}{2} (t_i - o_i)^2 \]

- and the activation function

\[ f(x) = \frac{1}{1 + e^{-x}} \quad \text{or} \quad f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} = \tanh\left( \frac{x}{2} \right) \quad \text{or} \quad f(x) = x \]

then the learning rule for each node can be derived using the chain rule...
Multi-Layer Perceptrons

- Recall the output
  \[ o = f \left( \sum_{i} w_i x_i - \theta \right) \]

- and the squared error measure
  \[ E_p = (t_p - o_p)^2 \]

- which is amended to
  \[ E_p = \sum_{i} (t_i - o_i)^2 \]

- 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

Backpropagation

make incremental change in the direction \( \frac{\partial E}{\partial w} \) 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.

\[ \Delta w = -\eta \sum_{p} \frac{\partial E}{\partial w} \]
**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

**Step 1**

1. Initialize weights to small random values

Example:

<table>
<thead>
<tr>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{11}^1$</td>
<td>-0.4</td>
</tr>
<tr>
<td>$w_{12}^1$</td>
<td>-0.3</td>
</tr>
<tr>
<td>$w_{21}^1$</td>
<td>-0.1</td>
</tr>
<tr>
<td>$w_{22}^1$</td>
<td>-0.1</td>
</tr>
<tr>
<td>$w_{11}^2$</td>
<td>-0.6</td>
</tr>
<tr>
<td>$w_{12}^2$</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1^2$</td>
<td>0.2</td>
</tr>
<tr>
<td>$t_2^2$</td>
<td>0.5</td>
</tr>
<tr>
<td>$t_3^3$</td>
<td>-0.1</td>
</tr>
</tbody>
</table>
Step 2

2. Choose a pattern and apply it to input layer

<table>
<thead>
<tr>
<th>x1</th>
<th>x1</th>
<th>Target output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 3

Propagate signal forward through the network

\[ o_i^m = \frac{1}{1 + \exp(-2*β*(0*(-0.4)+1*(-0.3)+0.2))} \]

until all outputs have been calculated

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

Example:

\[ x_1 = \bar{Q}_1 = 0 \]
\[ w_{11} = -0.4 \]
\[ w_{12} = 0.3 \]
\[ x_2 = \bar{Q}_2 = 1 \]
\[ 1 \]
\[ O = \frac{1}{1 + \exp(-2*β*(0*(-0.4)+1*(-0.3)+0.2))} \]
Step 4

Compute the deltas for the output layer

\[ \delta^M = g'(h^M) [d^p - O^M] \]

by comparing the actual output \( O \) with the target output \( t \) for the pattern \( p \) considered.

Example:

\[ x_1 = q_1 \]
\[ x_2 = q_1 \]
\[ t^1 = -0.1 \]
\[ O^1 = 0.1321 \]
\[ h_1^2 = 0.2745 \]
\[ \delta_1^2 = f'(h_1^2) \cdot (t^1 - O^1) \]

Step 5

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

\[ \delta_{j}^{m-1} = g'(h_{j}^{m-1}) \sum_{i} w_{ij} \delta_{i}^{m} \]

for \( m=M, M-1, M-2, \ldots \)

until a delta has been calculated for every unit.
**Step 6**

Use

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

to update all connections to

\[ w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij} \]

---

**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)
Momentum

If error minimum in long narrow valley, then updating can happen to zig-zag down the valley

\[ \Delta w = -\eta \nabla E + \alpha \Delta w_{prev} \]

smoothes weight updating can speed learning up

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**

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>XOR</th>
<th>Interwined</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-layer: Half planes</td>
<td>![Diagram of 1-layer MLP]</td>
<td>![Diagram of 1-layer MLP]</td>
<td>![Diagram of 1-layer MLP]</td>
</tr>
<tr>
<td>2-layer: Convex</td>
<td>![Diagram of 2-layer MLP]</td>
<td>![Diagram of 2-layer MLP]</td>
<td>![Diagram of 2-layer MLP]</td>
</tr>
<tr>
<td>3-layer: Arbitrary</td>
<td>![Diagram of 3-layer MLP]</td>
<td>![Diagram of 3-layer MLP]</td>
<td>![Diagram of 3-layer MLP]</td>
</tr>
</tbody>
</table>

**Radial Basis Function (RBF) Networks**

Network architecture

\[ y_i = \exp \left( -\frac{1}{2\sigma^2} \| x - c_i \|^2 \right) \]

Each node is described by a bell shaped function

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

---

XOR, revisited

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) output for node 1   (b) output for node 2   (c) output for node 3

---

Page 26
Consider the radial basis functions:

\[ y_i = \sigma(x_i) + b_i \]

and a linear combination of the output variables

\[ y = \alpha \sigma + b \]

then the response is equivalent to ...

Modular Networks

Task decomposition
Local Experts
Fuse information

Input \( x \)

local expert 1 \( O_1 \)

local expert 2 \( O_2 \)

... \( O_n \)

local expert n

weigh experts opinions by \( g \)

fuse \( y \)
last slide