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
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 (\( t \rightarrow t+1 \))
- 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

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

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”

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

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

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

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:

\[ w_{ij}^{(t+1)} = w_{ij}^{(t)} + \alpha_j x_j \]

\[ w_{ij}^{(0)} = 0 \]

**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 \( E = -6 \)

**More Complex Hopfield Examples**

**Hopfield Book Example**

Character Recognition

eight exemplar patterns

\[
\begin{bmatrix}
0 & 1 & 2 & 3 \\
4 & 6 & 8 & 9 \\
3 & 3 & 3 & 3
\end{bmatrix}
\]

output pattern for noisy "3" input

**Reconstruction of Images**

binary images are 130x180 pixels
Hopfield: Issues

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

Perceptron

- Rosenblatt: 1950s
- Input patterns represented is binary
- Single layer network can be trained easily
- Output \( o \) is computed by
  \[ o = f \left( \sum w_i x_i - \theta \right) \]
  where
  \( w_i \) is a (modifiable) weight
  \( x_i \) is the input signal
  \( \theta \) is some threshold (weight of constant input)
  \( f() \) is the activation function

Single-Layer Perceptrons

Network architecture

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

Example: Gender classification (according to Jang)

Network Arch.

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

\[ y = \begin{cases} -1 & \text{if female} \\ 1 & \text{if male} \end{cases} \]

Training data

\[ h \] (hair length)

ADALINE

Single layer network (conceived by Widrow and Hoff)
output is weighted linear combination of weights

\[ o = \sum x_i w_i \]

error is described as

\[ e_p = (t_p - o_p) \] (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

\[ \frac{dE}{dw_i} = -2(y_i - o_i) \cdot x_i \]

The delta rule is:

\[ \delta = \frac{dE}{dw_i} \]

Intuitive appeal:
- if \( t_i > o_i \) boost \( o_i \) by increasing \( w_i x_i \)
- 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?

**XOR**

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

\[ 0w_1 + 0w_2 + w_3 \leq 0 \iff w_3 \leq 0 \]
\[ 0w_1 + 1w_2 + w_3 > 0 \iff w_2 > -w_3 \]
\[ 1w_1 + 0w_2 + w_3 > 0 \iff w_1 > -w_3 \]
\[ 1w_1 + 1w_2 + w_3 \leq 0 \iff w_1 + w_2 \leq -w_3 \]

**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

\[ x_1 + 1 \rightarrow \sigma \rightarrow y_1 \]
\[ x_2 + 1 \rightarrow \sigma \rightarrow y_2 \]

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

Multi-Layer Perceptrons

- Recall the output
  \[ a = \sigma \left( \sum w_i y_i - \theta \right) \]
- and the squared error measure
  \[ e_i = (y_i - a_i) \] which is amended to
  \[ e_i = \sum (y_i - a_i) \]
- and the activation function
  \[ \sigma(z) = \frac{1}{1 + e^{-z}} \] or \[ \sigma(z) = \frac{1}{1 + e^{-z}} \cdot \tanh \left( \frac{1}{4} \right) \] or \[ \sigma(z) = u \]

Multi-Layer Perceptrons (MLPs)

Network architecture

Multi-Layer Perceptrons

Backpropagation

make incremental change in the direction
\[ \frac{dE}{dw} \] 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.

\[ \Delta w = - \eta \frac{dE}{dw} \]
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 3
Propagate signal forward through the network
\[ \omega^m = \sum_{j} w_{ij} \delta^j \]
until all outputs have been calculated
For \( m=0 \) (input layer), the output is the pattern.

Step 4
Compute the deltas for the output layer
\[ \delta^m = \frac{1}{\eta} (O^m - t^m) \]
by comparing the actual output \( O \)
with the target output \( t \)
for the pattern \( p \) considered

Step 5
Compute the deltas for the preceding layers by propagating the errors backwards
\[ \delta^{m-1} = \frac{1}{\eta} \sum_{j} w_{ij} \delta^j \]
for \( m=M, M-1, M-2, \ldots \)
until a delta has been calculated for every unit

Step 6
Use
\[ \Delta w_{ij}^m = \eta \delta^m O^i \]
to update all connections to
\[ w_{ij}^{m+1} = w_{ij}^m + \Delta w_{ij} \]

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>XOR</th>
<th>Interwined</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="1-layer: Half planes" /></td>
<td><img src="image" alt="2-layer: Convex" /></td>
<td><img src="image" alt="3-layer: Arbitrary" /></td>
</tr>
</tbody>
</table>

**Radial Basis Function (RBF) Networks**

Network architecture

Each node is described by a bell shaped function

\[ a = \exp \left( -\frac{1}{2\sigma^2} \right) \]

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

XOR, revisited

RBF and FIS

Consider the radial basis functions:

\[ y_i = \sum w_i \delta(x - c_i) + b_i \]

and a linear combination of the output variables

then the response is equivalent to ...

Modular Networks

Task decomposition
Local Experts
Fuse information

last slide