A general introduction to artificial neural networks (ANN) is given assuming no previous knowledge in the field. Properties of the multilayer perceptron, feature maps, the Hopfield model and the Boltzmann machine are discussed in some detail. It is discussed how these approaches are related to conventional classifiers and statistical methods. Also novel methods of finding good solutions of difficult optimization problems with feed—back networks and so-called deformable templates are described. Throughout the lectures practical hints on how to use the algorithms are given. Potential hardware implementations, both VLSI and optical, are briefly mentioned.

The power of the artificial neural network approach is illustrated in different high energy physics applications - quark jet tagging, mass reconstruction, track finding and process control in accelerator physics.
Artificial Neural Networks [ANN]

Carsten Peterson
University of Lund

1. Introduction
   Structure of the Central Nervous System
   ANN Generics

2. Feed-forward Networks
   Perceptrons
   Function Approximators

3. Self-organization
   Feature Maps

4. Feed-back Networks
   The Hopfield Model
   The Boltzmann Machine

5. Optimization Problems
   Feed-back ANN
   Deformable Templates
   Graph Bisection, Traveling Salesman, Track Finding

6. Hardware Implementations
   VLSI
   Optics
Literature:


Software:

- JETNET 2.0 [F77] *
  available from denni@thept.lu.se

- Commercial packages .....  

* Computer Physics Communications 70, 167 (1992)
1. Introduction

A. Structure of Central Nervous System

- Details revealed ~1940; electron microscope

- All neurons have the same basic parts regardless of size and shape.

  - cell body: 10–80\,\mu m
  - dendrite (d): 1–2\,\mu m
  - neuron: 0.01\,mm – 1\,m

interneuron  neuron-neuron
motor neuron  neuron - muscle fiber
receptor neuron muscle, receptor \rightarrow neuron

dendrite: receptor of signals from other neurons
axon: transmitter of generated activity to other cells

- cell body: amplifier [electrical]
Synaptic joint: 200 nm wide gap
neurotransmitters released [chemical]
gates
________________________ dendrite

more gates open
stronger coupling

typical response time: \( \sim \) msec

human brain: \( 10^{11} - 10^{12} \) neurons

connectivity: "a few" – \( O(10^5) \)
\( O(10^3) \) average in cerebral cortex

\( \Rightarrow 10^{15} \) synapses / brain

For excitatory/inhibitory synapses \( 2 \times 10^{15} = 10^{14} \) possible configuration.

Power dissipation: \( \leq 100 \) W

Very fault tolerant: many neurons die daily.

Parallel asynchronous execution.

Cf conventional computer:

- response time: \( \sim \) nsec
- memory: 1 Gbyte [Cray]
  \( \sim 10^{10} \) transistors
- logical functions: \( \sim 10^8 \) transistors
- power dissipation: \( \sim 10^5 \) W
"Mammal" computer

**Good at:**

- Pattern Recognition
- Adaption
- Process Control
- Optimization
  [Quick and Dirty]
- Robust
- Fault tolerant

- $\approx$ ms
- Parallel Execution

Dove $\approx$ PC AT

[# of amplifiers]
1. Basic Elements

- neuron $v_i \in [0,1]$

$\left(v_i \in [-1,1]\right)$

- synapse connection $\omega_{ij}$

Each neuron performs summation + amplification

$$\sum \omega_{ij} v_j$$

Local updating rule (common to all models)

$v_i = g \left( \sum \omega_{ij} v_j + \Theta_i \right)$; $\Theta_i$ threshold

$g(x) = \frac{1}{2} \left[ 1 + \tanh \left( \frac{x}{T} \right) \right]$ “sigmoid”

$g(x) = \tanh \left( \frac{x}{T} \right)$ if $v_i \in [-1,1]$

$\frac{1}{T} = \text{gain}$

$T = "\text{temperature}" \ (\text{more later})$

$T \text{ small}$

$T \text{ large}$
Sometimes (e.g. in self-organizing networks) it is advantageous to use a Gaussian activation function $G(x)$.

- Encode computational problems with these ingredients

- Narrow gap to VLSI hardware

- Inspired by biology; major abstraction, key elements and functionality kept.

Encouraging experience from theoretical with systems containing many degrees of freedom, e.g. magnets. Details of individual atoms lumped into “effective” variables - spins. Good descriptions of macroscopic phenomena emerge (phase transitions etc.)
2. Architectures

i) Feed-forward

\[
\text{output nodes}
\]

\[
\text{input nodes (sensors)}
\]

at each node \( v_i = g(\sum_j w_{ij} v_j + \theta_i) \) once

Final state: The output layer has been reached

Perceptron

[Back-propagation]

ii) Feed-back

a. Fixed Point Networks

- \( w_{ij} = w_{ji} \)
- input/output node distinction possible
- layered structure possible
- more versatile

Each node is updated, \( v_i = g(\sum_j w_{ij} v_j + \theta_i) \), until a stable state (fixed point) has been reached; the \( v_i \)'s don't move.

Hopfield Model
Boltzmann Machine
Mean Field Learning
Optimization
b. Non-Fixed Point Networks

- $w_{ij} \neq w_{ji}$

- self-interactions

\[ w_{ii} \neq 0 \]

delayed signals

non-stationary solutions (limit cycles)

\[ V_1 \]

\[ V_2 \]

time-sequenced data
e.g. speech
3. Applications

i) Feature Recognition

\[ \text{feature} = f(\text{observed data}) \]

Find function \( f \) using an ANN parametrization:

\[ v_i = g(\sum_j w_{ij} v_j + \Theta_i) \]

some \( v_i \) reserved for features (outp.)

some for data (inp.)

Fit \( w_{ij} \) to a set of features/data [training set]

Feed the ANN with data it has never seen before [test set]

\( \Rightarrow \) features

Cf curve-fitting

\[ \begin{array}{c}
\text{\textbullet training set} \\
\text{\textendash fitted "}w_{ij}" \\
\times \text{test set}
\end{array} \]

Procedure for fitting: LEARNING ALGORITHM

This is an expansion in terms of \( g(x) \sim \sum \). Cf fitting

with \( G(x) = \sum \). Sigmoid efficient.

\[ \sum \sim \sum + \tau \]
ii) Real Function Approximators

e.g. time-series prediction, system identification (process control)

Ex. \( x_t = f(x_{t-1}, x_{t-2}) \)

- \( x_t \): output (linear model)
- \( x_{t-1}, x_{t-2} \): input

find \( w_{ij} \)

iii) Associative Memory

fit \( w_{ij} \) to "words" \((v_1, v_2, v_3, \ldots, v_n)\)

no distinction between input and output nodes

feed network with incomplete "words" \((v_1, ?, v_2, \ldots, ?)\)

and let it complete the pattern – find \( ?'s \).

iii) Optimization

- feedback networks only

\( v_i = g(\sum_j w_{ij} v_j + \Theta_i) \) are eq. of motions corresponding to

minimizing some energy [Lyapunov energy]

\[ E = \frac{1}{2} \sum_{ij} w_{ij} v_i v_j \]

if \( g = \) step function [\( T \rightarrow 0 \)] (more later)

Map a problem (e.g. Traveling Salesman) onto \( E \) by clever choice of \( w_{ij} \); fixed once and for all
9. Feedforward Networks - Learning Algorithms

A. The Simple Perceptron

\[ \omega_{ik} \]
\[ x_k \text{ input nodes} \]
\[ o_i \text{ output nodes (classes)} \]

For each pattern \( p \) there is a target value \( t_i(p) \).

Determine \( \omega_{ik} \) in

\[ o_i = g\left( \sum_k \omega_{ik} x_k + \theta_i \right) \]

such that \( o_i(p) \approx t_i(p) \) for all patterns

[in what follow suppress \( p \)]

Minimize e.g.

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

Simplest method \hspace{1cm} Gradient Descent

\[ \Delta \omega_{ik} = -\eta \frac{\partial E}{\partial \omega_{ik}} \quad \text{Learning rule} \]

\[ \text{step size} \]

\[ \frac{\partial E}{\partial \omega_{ik}} = \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial \omega_{ik}} = (o_i - t_i) g'(\ldots) x_k = \delta_i x_k \]

on-line \hspace{1cm} pattern by pattern

off-line \hspace{1cm} all patterns
Logical exclusive-OR problem $[\text{XOR}]$ of parity

$$0 = \text{XOR}(x_1, x_2) = 1 \quad \text{if one of } x_1 \text{ and } x_2 \text{ true } (=1)$$

but not both

$[-1,1]$ notation

$$o = \text{sgn} (\omega_1 x_1 + \omega_2 x_2 + \Theta)$$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$t$</th>
<th>$o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>$\omega_1 + \omega_2 + \Theta &lt; 0$</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>$\omega_1 - \omega_2 + \Theta &gt; 0$</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>$-\omega_1 + \omega_2 + \Theta &gt; 0$</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>$-\omega_1 - \omega_2 + \Theta &lt; 0$</td>
</tr>
</tbody>
</table>

(8) + (3) $\Theta > 0$
(1) + (4) $\Theta < 0$ cannot be satisfied

[3 parameters - 4 patterns]

Input space corners of hypercube

XOR problem not linearly separable

One line $[N-1]$-dim hyperplane] not enough to separate the regions.

The simple perception is a linear separator!
B. Multilayered Perceptrons - The Back-propagation Learning Rule

Introduce a layer of hidden units

XOR again

Example of architecture with weights

\[
\Theta = 2\omega \\
\Theta_1 = \omega \; ; \; \Theta_2 = -\omega \\
h_1 = \text{sgn} (\omega x_1 + \omega x_2 + \omega) \\
h_2 = \text{sgn} (\omega x_1 + \omega x_2 - \omega) \\
o = \text{sgn} (\omega h_1 - 2\omega h_2 - 2\omega)
\]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(t)</th>
<th>(h_1)</th>
<th>(h_2)</th>
<th>(o)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

OR AND gates

- Generalize Perceptron Learning to include hidden units
- Many recognition problems are not linearly separable
One hidden layer

\[ h_j = g(\sum_k w_{jk} x_k + \Theta_j) \]
\[ o_i = g(\sum_j w_{ij} h_j + \Theta_i) \]

The thresholds are also parameters, can be absorbed into \(\omega_{jk}\) formalism

\[ \Theta_j = \omega_{jo} x_0 \]
\[ \Theta_i = \omega_{io} h_0 \]

where \(x_0\) and \(h_0\) are neurons that are always "on" (+1). In what follows

\[ h_j = g(\sum_k w_{jk} x_k) \]
\[ o_i = g(\sum_j w_{ij} h_j) \]

Learning rule

minimize \[ E = \frac{1}{2} \sum_i (t_i - o_i)^2 \]

Gradient descent

\[ \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \]
\[ \Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} \]

\[ \frac{\partial E}{\partial w_{ij}} = \delta_i h_j \]

where \[ \delta_i = (o_i - t_i) g'(\sum_j w_{ij} h_j) \]
as in the simple Perceptron
\[
\frac{\delta E}{\delta w_{jk}} = \sum_i \frac{\delta E}{\delta o_i} \frac{\delta o_i}{\delta h_j} \frac{\delta h_j}{\delta w_{jk}} = \sum_i \delta_i w_{ij} g'(c) x_k
\]

**Back-propagation learning rule**

- Can be generalized to any # of hidden layers.
- The error \( \delta_i \) "propagates" down.
- No guarantee that global minima of \( E \) is reached.

What is going on?

Hidden units divide input space in regions

\[ g(\sum_k w_{jk} x_k) = \text{sigmoid}(\sum_k w_{jk} x_k) \]

\[ \sum_k w_{jk} x_k = 0 \quad \text{hyperplane} \]

1st hidden layer cuts out convex volumes

2nd hidden layer cuts out any volume
Interpretation of Output

When estimated accurately output values can be interpreted as Bayesian probabilities provided

1. 1-of-M output

2. Restricted class of E-functions
   
   \[ \sum (e_i - o_i)^2 \]

3. Training data reflects actual distributions

\[
P(c_i | x) = \frac{P(x | c_i) p(c_i)}{p(x)} \]

\(C_i\)  \(i = 1, ..., M\) classes

\(x\)  input data
Refinements

- Alternative Error Measures
- Potts Neurons
- Finding the Best Minimum [Quickly]
- Choice of Architecture
- Preprocessing of Input Data

Many applications do amazingly well with "vanilla" version
Alternative Error Measures

\[ E = - \sum_i \left[ t_i \log o_i + (1-t_i) \log(1-o_i) \right] \]

Cross Entropy

\[ \frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial \omega_{ij}} = \left[ -\frac{t_i}{o_i} + \frac{(1-t_i)}{1-o_i} \right] g'(\eta) h_j = \]

\[ = \frac{o_i-t_i}{o_i(1-o_i)} g'(\eta) h_j = \frac{1}{2} \cdot \frac{2}{1-g} (o_i-t_i) h_j \]

\[ [g'(\eta) = \frac{1}{2} g(1-g)] \text{ absorbed into } \eta \]

\[ \Delta \omega_{ij} = -\eta (o_i-t_i) h_j = -\eta \hat{\delta}_i h_j \]

lower layers identical to \( \frac{1}{2} (\cdot)^2 \) error

Exclusive Classification - Potts Neurons

For each pattern only one neuron should be "on" (=1). The others \( o \) (0).

E.g. recognize handwritten digits 0, ..., 9

With binary neurons winner-takes-all interpretation
K-valued Neurons [Potts representation]

Consider state-space of 8 binary neurons

Confine dynamics to plane

\[ a_i = \sum_j \omega_{ij} h_j \]

\[ o_i = g(a_i) \]

\[ o_i = \frac{e^{a_i}}{\sum_j e^{a_j}} \]

\[ \sum o_i = 1 \]

[\[ \sum t_i = 1 \]

Derive BP with Kullback error measure

\[ E = \sum_k t_k \log \frac{t_k}{o_k} \]

\[ \frac{\delta E}{\delta w_{ij}} = \sum_k \frac{\delta E}{\delta o_k} \frac{\delta o_k}{\delta a_i} \frac{\delta a_i}{\delta w_{ij}} \]

\[ \frac{\delta E}{\delta o_k} = -\frac{t_k}{o_k} \]
\[ = \delta_{ik} o_k - o_i o_k \]

\[ \frac{\partial a_i}{\partial w_{ij}} = h_j \]

\[ \Rightarrow \frac{\partial E}{\partial w_{ij}} = -\sum_k t_k \frac{1}{o_k} (\delta_{ik} o_k - o_i o_k) h_j = -\sum_k t_k (\delta_{ik} - o_i) h_j \]

\[ = \left[ t_i - (\sum_k t_k) o_i \right] h_j = (o_i - t_i) h_j = \delta_{i} h_j \]

Again for lower layers no change.
Finding the Minima

- **Gradient Descent**

\[ \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \]

\( \eta \) has to be tuned by hand

Could get trapped in local minima

Potential of being slow

"on-line" improves

Random pattern ordering improves

- **Hessian Methods**

\[ \Delta w_{ij} = -\eta H^{-1} \frac{\partial E}{\partial w_{ij}} ; \quad H = \frac{\partial^2 E}{\partial w_{ij} \partial w_{ki}} \]

Probes natural metric - Individual \( \eta \)'s.

Slow! Requires inversion of non matrix

Cf. Stiff ODE solvers

Never used in its full form
• Conjugate Gradient

\[ \tilde{w} = \tilde{w}_0 + \lambda \tilde{d} \]

\( \lambda \) chosen to minimize \( E(\tilde{w}) \)

\[ \tilde{d} = - \nabla E(\tilde{w}_0) \] stepest descent

Successive steps

\[ 0 = \frac{\partial}{\partial \lambda} E(\tilde{w}_0 + \lambda \tilde{d}_{\text{old}}) = \tilde{d}_{\text{old}} \cdot \nabla E_{\text{new}} \]

zigzag path

Compromise

\[ d_{\text{new}} = - \nabla E_{\text{new}} + \beta d_{\text{old}} \]

Choose \( \beta \) to spoil as little as possible of previous achievements

\[ d_{\text{old}} \cdot \nabla E(\tilde{w}_0 + \lambda d_{\text{new}}) = 0 \]

\( \Rightarrow \beta \)

Conjugate Gradient (off-line)

Almost never yields speedup compared to BP

• Momentum Term

Poor man's conjugate gradient

\[ \Delta w_{ij}^\text{new} = -\eta \frac{\partial E}{\partial w_{ij}} + \gamma \Delta w_{ij}^\text{old} \]

Often effective
• **Langevin Updating**
  
  Avoid getting stuck in local minima
  
  - on-line updating [random order]
  - add noise to inputs
  
  Be more systematic — put $E(\bar{\omega})$ in temperature environment
  
  $P \sim e^{-E(\bar{\omega})/T}$  
  anneal $T$ [more later]
  
  Generate $P$ with Metropolis, Heat Bath, Langevin
  
  Langevin [Brownian motion] natural in our case
  
  \[
  \frac{\partial}{\partial t} \bar{\omega}(t) = -\eta \frac{\partial}{\partial \bar{\omega}} E(t) + \sigma(t)
  \]
  
  \[
  \Delta \bar{\omega} = -\eta \frac{\partial E}{\partial \bar{\omega}} + \xi \quad \text{stochastic noise}
  \]
  
  decrease width during learning

  Very robust and easy to control

  Fast

  Good Performance

=\\

# of hiddens
Choice of Architecture

Good generalization performance requires \# of weights < \# of training patterns

Decrease \# of weights

(i) Intelligent Preprocessing [Application Specific]

(ii) Encode Symmetries

- Explicitly
  Fourier transforms etc.

- Implicitly
  A. Local receptive fields - Linked weights

9-D patterns

First hidden layer has limited connectivity with inputs -
local receptive fields
efficient for picking up local features

translation invariance of features
A plane of hidden for each possible feature

On each H-plane
- looks at the same receptive field

Back-propagate errors all the way down to I-plane

For each H-plane (feature) share weight (after update) by averaging between $o$.

$h_j$ and $h_{j'} \in H_1$

$$h_j = \frac{g}{R_j} \left( \sum \omega_{jk} x_k \right)$$

$$h_{j'} = \frac{g}{R_{j'}} \left( \sum \omega_{j'k} x_k \right)$$

$\omega_{jk} = \omega_{j'k}$ for $k$ at the "same" position in $R$
B. Complexity Terms in $E$

\[ E \rightarrow E' = E + \lambda \sum_{ij} \omega_{ij}^2 \quad \text{(weight decay)} \]

\[- \log P(\omega) \quad \text{where} \quad P(\omega) \quad \text{Gaussian} \quad \sigma = \frac{1}{\lambda} \]

\[ \left( \frac{\omega}{2} \right)^2 + \left( \frac{\phi}{2} \right)^2 < \omega^2 + \phi^2 \]

It would be better to have

\[ P(\omega) \sim \begin{array}{c} \hline \hline \hline \end{array} \quad + \quad \begin{array}{c} \hline \hline \hline \end{array} \]

or more general

\[ E' = E - \sum_{ij} \log P_{ij} \quad \text{[M Gaussians]} \]

where

\[ P_{ij} = \prod_{k=1}^{M} \frac{1}{\sqrt{2\pi} \sigma_k} e^{-\frac{(\omega_{ij} - \mu_k)^2}{2 \sigma_k}} \]

$\mu_k$, $\sigma_k$ additional parameters

Squeezes weights into common means - clustering

Effective $\#$ of degrees of freedom reduced
(L11) Linear Dimensional Reduction

\[
\begin{align*}
\omega_{kk} & \quad \rightarrow \quad \mathbf{y}_k \\
M & \quad \downarrow \quad N
\end{align*}
\]

Compute \( C_{ke} = (x_k - \langle x_k \rangle)(x_e - \langle x_e \rangle) \)

Diagonalize \( C_{ke} \)

Inspect eigenvalue spectrum

Keep \( M \) largest \( \lambda_k \)

Freeze \( \omega_{kk} \) \( N \times M \) weights

Use \( y_k \) for ANN processing

Beware that this is linear!
How to do these things?

Example: 2-D Munsen symmetry

Patterns symmetrical around vertical or horizontal line

$N^{1/2}$ patterns of each class

- Create $N$ training and $N$ test patterns

- $o = 1 \ (v) \ ; \ o = 0 \ (H)$

- $n_h \geq 5$

- 16 input pixels

- Initialize weights $w_{ij} = \text{rand} (-0.1, 0.1)$

- Pick a training pattern

- Forward pass

- Back-propagate $\Rightarrow \Delta w_{ij}, \Delta w_{jk}$

- After $K$ passes update

$N$ patterns = one "epoch"
(A) training set with labels
(B) validation set
(C) test set

Test on \((A,B,C)\) with some score criteria
e.g. \(\theta \geq 0.5\)

[Alternatively measure error]

![Graph showing training set and validation set performance over epochs.](image)

stopping criteria overlearning

[ Cf curve-fitting ]
0. Initialize weights \( w \in [-1, 1] \)

1. Until \( \omega(t+1) = \omega(t) \)
   
   A. Pick pattern \( p \)

   B. Compute \( o_i \) and \( E = \frac{1}{2} (o_i - t_i)^2 \)

   C. Compute \( \Delta \omega = -\eta \frac{\partial E}{\partial \omega} \)

   D. Update \( \omega(t+1) = \omega(t) + \Delta \omega \)
(1) If no hidden nodes present
MLP reduces to Linear Discriminant

(a) MLP (with hidden) hits Bayesian limit, when the latter can be estimated

\[ Z_{1-p} \]

Benchmark
2 overlapping Gaussians\( \left( \mu_i \sigma_i \right) \) in 8 dimensions

<table>
<thead>
<tr>
<th>Exact</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>93.8%</td>
<td>93.2%</td>
</tr>
<tr>
<td>91.0%</td>
<td>90.7%</td>
</tr>
<tr>
<td>73.6%</td>
<td>73.8%</td>
</tr>
</tbody>
</table>

(3) Classification \( \equiv \) carving out areas in high-dimensional input space

- Sigmoids - hyperplanes
- Gaussians - spheres

more economical
HEP Application Areas

(1) Off-line feature recognition
   e.g. quark (gluon) identification given hadron information. Use intelligent variables \([S, m_1^2, \ldots]\) rather than raw (calorimeter) data

(2) On-line triggers on features - raw data.

(3) Mass Reconstruction

(4) Process control in accelerator physics *)
   System identification + set-point optimisation

(5) Track-finding *)
   Combinatorial optimisation + parameter fitting

*) Off-line on-line

(1), (2), (3) MC model sensitivity
b-quark Identification

Efficiency / Purity

- Lepton tagging  
  \[ \frac{10\%}{100\%} \]  
  \[ \frac{[5]}{[95]} \]

- Vertex detectors
  \[(c+b) \quad \frac{25\%}{80\%} \quad 92 \text{ GeV} \]
  \[ b \quad \frac{25\%}{60\%} \]

- ANN method on raw MC data

  - b/non-b

  \[ \begin{array}{c}
    \text{Jetnet 2.0} \\
    \text{F77}
  \end{array} \]

  \[ \text{10 hidden} \]

  \[ \text{E}_{\text{tot}} \text{ Plot} \]

  \( (E, \theta, \phi) \) of leading 6 particles

  \[ \{ \text{20 inputs} \} \]

  Results comparable with vertex detectors

- Use intelligent variables
ALEPH analysis
[ J. Conway, J. Jacobsen, Y. Bin Pan, S.L. Wu]

Input nodes:

- Jet sphericity
- \( \frac{n}{\log E} \) for jet
- \( \sum p_t^2 \) relative to jet axis
- Out of 4 leading particles take all pairs, "threes", + all 4
  Compute \( S, m^2 \)
  2 x 11
  \( \Sigma = 25 \) inputs

- \( b/\text{non-}b \)
  5 hidden units
  25 hidden units
  25 inputs

\( \Rightarrow \) 2-jet events only

150 \( \cdot 10^3 \) \( b \)
50 \( \cdot 10^3 \) non-\( b \)
\( \Rightarrow \frac{1}{2} + \frac{1}{2} \)

(30% of all \( b \)-events)

Include 3-jet events. Remove \( q \) by \( \text{WJ} \)
Figure 9: Distributions of network output for $b$ jets (solid line) and light quark jets (dashed line) from two-jet events. Both are normalised to unit area.

Choice of threshold purity / efficiency
Figure 10: Efficiency $E$ vs. $1 - K$ for two-jet Monte Carlo events.
Conclusion:

- Very powerful and inexpensive method for tagging b's
  61% [80%]

- Independent of other tagging methods
  - secondary vertex
  - leptonic decay

Compare

1. Histogram cuts
2. Linear Discriminant
3. ANN
Figure 2
Efficiency / Purity
Representation of Continuous Functions

\[ O_i = \sum_j w_{ij} h_j \quad \text{(linear)} \]

\[ h_j = g(\sum_k w_{jk} x_k) \]

Sigmoid functions \([g(x)]\) powerful function approximators

\([\text{Cf. Gaussians}]\)

Example

Time series

Sun spot data

\[ x_t = \text{annual activity} \quad \text{(period 11 yrs)} \]

\[ x_t = f(x_{t-1}, x_{t-2}, \ldots, x_{t-12}) \]

1700 - 1920 learning set
1921 - 1979 test set

"single-step" \((x_{t-1}, \ldots, x_{t-12})\) real data

"multi-step" \(\ldots\) predicted data
Finite precision sets limits
Can Sigmoidals Fake Gaussians?

\[ o_i = \sum_j w_{ij} h_j \]

linear output

hidden layer(s)

\[ f(x) = g(x) - g(x-c) \]

1 dimension

In general

\[ f(x) = g[ g(wx-c_1) - g(wx-c_2) ] \]

One hidden layer does this

How about 2 dimensions?

\[ f(x, y) = f_1(x)f_2(y) \]

network does summations not multiplications

Add an extra hidden layer.
\( f_1(x) \) and \( f_2(y) \) are "ridges" in 2 dimensions. Intersection a weak bump.

\[
f_1(x) + f_2(y) = \eta \left[ g(\omega x - c_1) - g(\omega x - c_2) + g(\omega y - c_3) - g(\omega y - c_4) \right]
\]

Process through another non-linearity \( g(\cdot) \) to clean up bump 
\( \Rightarrow \) second hidden layer

\[
f(x, y) = g \left[ f_1(x) + f_2(y) - \Theta \right]
\]

Choose large \( \eta \)

\( \Theta \in [0, 2\eta] \)

For higher dimensions add more contributions to argument in 
\( f(x, x_2, \ldots, x_n) = g[\ldots] \)

"Anything that can be approximated with Gaussians can be dealt with by feed-forward networks with two hidden layers."
Again $W \to \text{jets} \ [\text{UA2}]$

1. Conventional Method

\[
\begin{align*}
0 & \rightarrow \cdots, \\
2 & \leftarrow \cdots
\end{align*}
\]

- Sweep with cone-window
- Find 2 largest jets
- $|y| < 0.7 \implies M_W$

2. NN approach

- Select 30 largest towers $(E_T, \eta, \phi)_1, \ldots$

- Linear output $(M_W)$

- 20 hidden

- $30 \times 3$ input

- Train on MC-generated $pp \to W$ events with masses in range 50, 51, 52, ..., 150

- Test on MC-generated events with $M = M_W$

- $D$ better peaked
- Less low mass tail - bremsstrahlung taken care of
- Facilitates real-time reconstruction
How well does training data cover phase space?

Makes sense to interpolate within "sweetspots".

* Estimate confidence levels \([CL] \) from cluster analysis

* Do k-means clustering on training set
  Determine membership function \(V_{ia} \) for each exemplar \(i \) in cluster \(a\)
  \[\sum_{a} V_{ia} = 1\]

  For each \(a\)
  \[\sigma_{a}^{2} = \sum_{i=1}^{N} V_{ia} E_{i}^{2}\]
  \[E_{a} = \text{network error for pattern-}i \text{ [validation set]}\]
  \[CL_{a} = t_{95} \cdot \sigma_{a}\]
  \(CL_{a}\) properly normalized

  \[CL_{i} = \sum_{a=1}^{K} V_{ia} CL_{a} \]
   test set - "real world"

* There exists non-parametric methods
  VC-dimensions ...

But very tight bounds
(2) Optimize $\sigma_i$ with respect to $c_k$

$$E = \frac{1}{2} \sum_i (\sigma_i - \sigma_i^{\text{opt}})^2$$

Back-propagate errors to $c_k$ (bpa)

$$\Delta c_k = -\eta \frac{\partial E}{\partial c_k}$$

Dynamical range of $c_k$ ?

$$E \rightarrow E' = E + f(c_k)$$

For real-time operations both $\uparrow$ and $\downarrow$ on chip.

Very competitive algorithm in industrial applications
Non-linear capacity important - small improvement in $\sigma_i$ important for control part.

Cf. pattern recognition.

- In finding $F$ beware of stiff updating equations.

- Can we trust $c_k$ determination ?
Problem: Optimize output [quality] with respect to control settings.

9 steps: (1) Model the data [System Identification].

(2) Use the model to set control parameters in an optimal way.

\[ O_i = F(c_{1}, c_{2}, \ldots, c_{n_c}; s_{1}, s_{2}, \ldots, s_{n_s}; O_{i}(t-1), \ldots) \]

Find F with good experience with "synthetic" strongly non-linear systems.

E.g. Mackey-Glass equation

Also sunspot data etc..
HEP applications using ANN for process control

- Ion source [Pb, S] at CERN SPS
  [very non-linear]

- Wake-field corrections at SLC:
  - model simulations
  - proper and versatile data recordings important
PS/CERN Sulphur Beam Source

Feasibility Study

Intensity

Figure 2: A typical evolution of the S contents in the PS during three days

Function of

\[
\begin{align*}
  c_1 & : B1 & \text{Back solenoid field} \\
  c_2 & : B2 & \text{Center solenoid field} \\
  c_3 & : B3 & \text{Extraction solenoid field} \\
  c_4 & : \text{RF/Voltage} & \text{Microwave power} \\
  c_5 & : \text{RF/Delay} & \text{Pulse length} \\
  c_6 & : V_c & \text{Cathode voltage} \\
  c_7 & : %O^{in} & \text{Inlet gas pressure, \% of full scale} \\
  c_8 & : %S^{in} & \text{Inlet gas pressure, \% of full scale} \\
  s_1 & : \text{Temp} & \text{Temperature of hexapole} \\
  s_2 & : t & \text{Second afterglow}
\end{align*}
\]

32 data points \( \leq \) training \( \leq \) test

PCA used to lower input dimension
Consider the problem as static [no $o_{t-1}, \ldots$]

$$E = \frac{((o - t)^2)}{((t - t)^2)} \quad \text{"monkey-index"}$$

**Linear** (no hidden layers)

$$E = 0.3 - 0.4$$

**Non-linear** (with hidden layers)

$$E = 0.2 - 0.3$$

Pb beam
- Measure more frequently
- Include $o_{t-1}, o_{t-2}$
- No PCA
- Optimize set points
The $\delta$-test

- Which variables are really needed [dependencies]?
  Find embedding dimension
- Estimate noise level in a system

\[
x_t = f(x_{t-1}, x_{t-2}, \ldots, x_{t-d}) + r_t
\]

Approach not limited to time series

Represent function as a series of points in $(d+1)$-dim. space.

\[
(x_0(t), x_1(t), \ldots, x_d(t))
\]

\[
\ell_k(i,j) = |x_k(i) - x_k(j)| \quad k = 0, 1, \ldots, d
\]

Construct from data

\[
P_d(\varepsilon | \delta_1, \ldots, \delta_d) \equiv P(\ell_0 < \varepsilon | \ell_1 \leq \delta_1, \ldots, \ell_d \leq \delta_d)
\]

not as tedious as it looks
- For a completely random process
  \[ P_0(\epsilon) = P_1(\epsilon|\delta_1) = \ldots = P_d(\epsilon|\delta_1, \ldots, \delta_d) \]

- If a continuous map exists (no noise) then for any \( \epsilon > 0 \) there exists a \( \delta_\epsilon \) such that
  \[ P_d(\epsilon|\delta) = 1 \text{ for } \delta \leq \delta_\epsilon \text{ and } d \geq d_0 \]
  [smallest \( d_0 = \text{embedding dimension} \)]

- In presence of noise

\[
|x_0 - x_0'| = |f(x_1, \ldots) - f(x_1', \ldots) + r - r'|\]

\[
\begin{align*}
P_d(\epsilon|\delta) & \\
\Delta r_{\text{max}} & \\
\end{align*}
\]

\[
\begin{align*}
|x_0 - x_0'| & \\
\Delta r & \\
\end{align*}
\]
Dependability index

\[
\lambda_d (e) = 1 - \frac{P_d (e)}{P_{d+1} (e)}
\]

\[
\lambda_d (e) = 0 \implies \lambda_t \text{ does not depend upon } \lambda_{t-1}
\]

Examples:

(1) Logistic map

\[
x(t) = \eta \times (t-1) \left[ 1 - x(t-1) \right] + r
\]

<table>
<thead>
<tr>
<th>( r = 0 )</th>
<th>( r = 0.1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \eta )</td>
<td>( \lambda_d )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

\( \epsilon = 0.55 \) \( \epsilon = 1.0 \) \( \epsilon = 0 - 0.16 \)

(2) Henon map

\[
x(t) = 1 - ax(t-8)^2 + bx(t-16)
\]

<table>
<thead>
<tr>
<th>( d )</th>
<th>( P_d (e) )</th>
<th>( \lambda_d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-7</td>
<td>0.29</td>
<td>( \Theta (10^{-3}) )</td>
</tr>
<tr>
<td>8</td>
<td>0.86</td>
<td>0.66</td>
</tr>
<tr>
<td>9-15</td>
<td>0.86</td>
<td>( \Theta (10^{-3}) )</td>
</tr>
<tr>
<td>16</td>
<td>1.0</td>
<td>0.134</td>
</tr>
<tr>
<td>17-19</td>
<td>1.0</td>
<td>0</td>
</tr>
</tbody>
</table>
Self Organizing Networks

Supervised Learning (e.g. BP)

I. Advantages:

- Good performance; close to Bayes limit.
- Fairly parameter insensitive.

II. Limitations:

- Requires prejudice about what features there are
- Somewhat cumbersome to analyze what the hidden nodes are doing.

Competitive Learning

\[ h_j = g \left( \sum_k w_{jk} x_k \right) = g(\tilde{w}_j \cdot \tilde{x}) \]

For each pattern presented pick a winner

\[ h_m = \max_j \left[ h_j \right] \]
\[ \theta = \frac{\langle \bar{\omega}_j, \bar{x} \rangle \cos \theta}{\| \bar{\omega}_j \| \| \bar{x} \|} \]

should be normalized.

**Update \( \bar{\omega}_j \) for the winner**

\[ \Delta \omega_{jk} = \eta h_j (v_k - \omega_{jk}) \]

\[ \omega_{jk} \rightarrow \frac{\omega_{jk}}{\sqrt{\sum_k \omega_{jk}^2}} \]  \hspace{1cm} (2)

**Redo (1)-(2) for all patterns over many epochs**

- Different hidden \( h_j \) pick out various features
- Simple to analyze - information in \( \bar{\omega}_j \)

- Cf. Adaptive K-means

**Unnormalized data?**

Gaussian nodes

\[ h_j = e^{-(\bar{\omega}_j - \bar{x})^2 / \tau} \]
This procedure corresponds to minimizing the error

\[ E_m = \frac{1}{2} \sum_a V_{am} (\hat{x}^a - \hat{w}_m)^2 \]

for the winner node \( h_m \), where

\[ V_{am} = 1 \text{ if pattern a activated node } m \text{ as a winner} \]
\[ 0 \text{ otherwise} \]

\[ \Delta \omega_{mk} = -\eta \frac{\partial E}{\partial \omega_{mk}} = \eta \sum_a V_{am} (\hat{x}^a - \omega_{mk}) \]

Spatial Neighborhoods

topological maps

A plane of hidden
feed-back lateral connections
local (or semi-local) connections only defines geometry

"Mexican hat"

\[ \Lambda_{jm} = e^{-d_{jm}^2 / \lambda^2} (1 - d_{jm}^2 / \lambda^2) \]

Supports the formation of local regions of nodes responding to similar stimuli

Dimensional reduction of data; data compression
Modified updating rule

$$\Delta w_{jk} = \eta \Lambda_{jm} (x_k - w_{jk})$$

Short-cut version $\Lambda_{jm} = 1$ for $d_{jm} \leq \lambda$

Also Potts updating

$$V_{am} = \frac{e^{a_m}}{\sum_{n'} e^{a_{n'}}}$$

relevance factor [more later]

Learning Vector Quantization [LVQ]

Supervised self organization

fine-tuning of feature maps

training vectors $x$ with known classifications $j$

$$\Delta w_{jk} = \eta (x_k - w_{jk}) \text{ if correct}$$

$$= -\eta (x_k - w_{jk}) \text{ if wrong}$$

Cf Gaussian classifiers
Phonetic typewriter

Filter A/D FFT

15 input nodes

[ Kohonen ]
• In general MLP are always more efficient than Gaussian classifiers [e.g. LVA] for high dimensional problems.

  more efficient = less units needed.

Easier to cut with planes than fill with spheres.

• Also true in general when underlying distributions contain outliers.