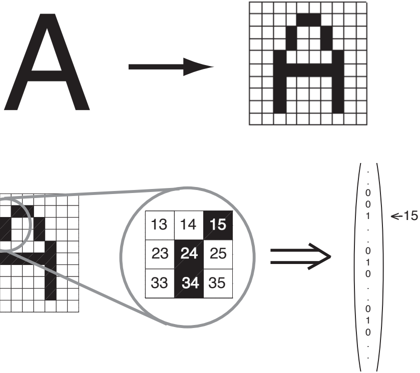
Neuroinformatics

April 18, 2013

Lecture 8: Feed Forward Networks

Digital representation of a letter



Optical character recognition: Predict meaning from features.
E.g., given features $\mathbf{x}$, what is the character $\mathbf{y}$

$$f : \mathbf{x} \in \mathbf{S}_1^n \rightarrow \mathbf{y} \in \mathbf{S}_2^m$$

Examples given by lookup table

Boolean AND function

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

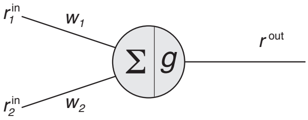
Look-up table for a non-boolean example function

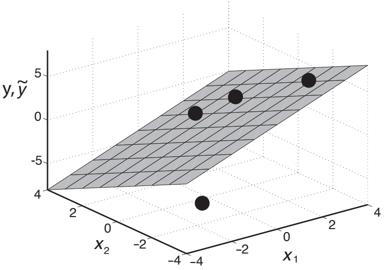
x_1	x_2	y
1	2	-1
2	1	1
3	-2	5
-1	-1	7
...	...	...

The population node as perceptron

Update rule: $\mathbf{r}^{\text{out}} = g(\mathbf{w}\mathbf{r}^{\text{in}})$ (component-wise: $r_i^{\text{out}} = g(\sum_j w_{ij} r_j^{\text{in}})$)
For example: $r_i^{\text{in}} = x_i$, $\tilde{y} = r^{\text{out}}$, linear grain function $g(x) = x$:

$$\tilde{y} = w_1 x_1 + w_2 x_2$$





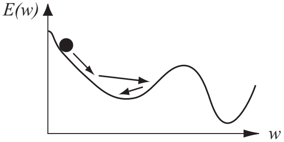
How to find the right weight values?

Objective (error) function, for example: mean square error (MSE)

$$E = \frac{1}{2}\sum_i (r_i^{\mathrm{out}} - y_i)^2$$

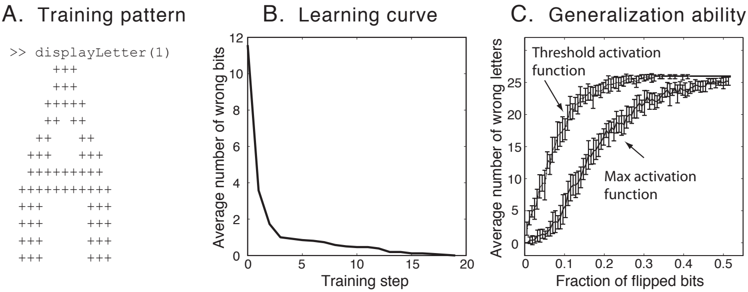
Gradient descent method: $w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}}$

$$= w_{ij} - \epsilon(y_i - r_i^{\mathrm{out}})r_j^{\mathrm{in}}$$ for MSE, linear gain



Initialize weights arbitrarily
Repeat until error is sufficiently small
- Apply a sample pattern to the input nodes: $r_i^0 = r_i^{\mathrm{in}} = \xi_i^{\mathrm{in}}$
- Calculate rate of the output nodes: $r_i^{\mathrm{out}} = g(\sum_j w_{ij} r_j^{\mathrm{in}})$
- Compute the delta term for the output layer: $\delta_i = g'(h_i^{\mathrm{out}})(\xi_i^{\mathrm{out}} - r_i^{\mathrm{out}})$
- Update the weight matrix by adding the term: $\Delta w_{ij} = \epsilon \delta_i r_j^{\mathrm{in}}$

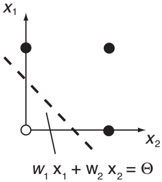
Example: OCR

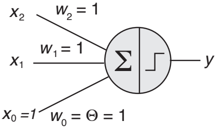

A. Training pattern
>> displayLetter(1)
+++
+++
+++++
++ ++
++   ++
+++   +++
+++++++++
+++++++++++
+++   +++
+++   +++
+++   +++
+++   +++
B. Learning curve
Average number of wrong bits
Training step
C. Generalization ability
Average number of wrong letters
Fraction of flipped bits
Threshold activation function
Max activation function

Example: Boolean function

A. Boolean OR function

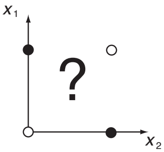
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

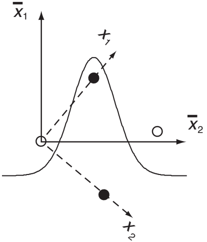




B. Boolean XOR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





perceptronTrain.m

```
%% Letter recognition with threshold perceptron
clear; clf;
nIn=12*13; nOut=26;
wOut=rand(nOut,nIn)-0.5;

% training vectors
load pattern1;
rIn=reshape(pattern1', nIn, 26);
rDes=diag(ones(1,26));

% Updating and training network
for training_step=1:20;
    % test all pattern
    rOut=(wOut*rIn)>0.5;
    distH=sum(sum((rDes-rOut).^2))/26;
    error(training_step)=distH;
    % training with delta rule
    wOut=wOut+0.1*(rDes-rOut)*rIn';
end

plot(0:19,error)
xlabel('Training step')
ylabel('Average Hamming distance')
```

Percepton as Linear Classifier: ML approach

- Assume a binary classification problem, i.e. $S = \{s_1, s_2\}$.
- One discriminant function $g(\vec{x})$ enough: classify $y = \begin{cases} s_1, & \text{if } g(\vec{x}) > 0; \\ s_2, & \text{otherwise.} \end{cases}$
- we will estimate $\vec{b}, c$ directly from the given sample $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2) \ldots (\vec{x}_m, y_m)\}$.
- We want $\left(\vec{b}^t \vec{x}_i + c\right) > 0$ if $y_i = s_1$ and $\left(\vec{b}^t \vec{x}_i + c\right) < 0$ otherwise.
- Same as requesting $\left(\vec{b}^t \vec{z}_i + c\right) > 0$ for all z_i, where $z_i = x_i$ if $y_i = s_1$ and $z_i = -x_i$ otherwise.
- Let formally $z_i^{n+1} = 1\ \forall i$ and $\vec{w} = [\vec{b}, c]$ (add c as the last component of $\vec{w}$).
- Thus we can write simply $g(\vec{z}) = \vec{w}^t \vec{z}$ and request $\vec{w}^t \vec{z}_i > 0$ for all z_i .
- Let

$$E(\vec{w}) = \sum_{\vec{z}_i \in M} -\vec{w}^t \vec{z}_i$$

where M is the set $\vec{z}_i$ that are misclassified.

Percepton ML view

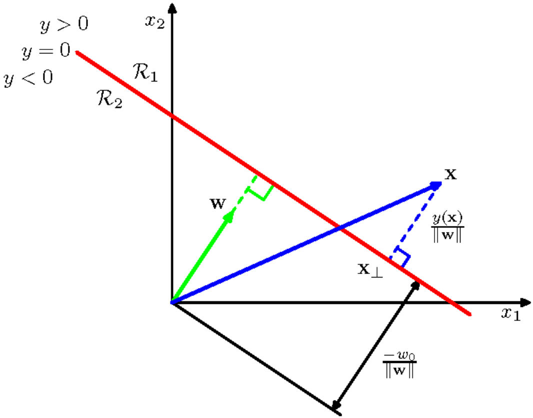
- $E(\vec{b}, c)$ is always non-negative.
- If $E(\vec{w}) = 0$ then all examples in D are correctly classified and D is linearly separable. We want to find the minimum of $E(\vec{w})$.
- $E(\vec{w})$ is piece-wise linear. A gradient algorithm can be used to search a minimum.
- Gradient algorithm: go towards a minimum by making discrete steps in $\Re^{n+1}$ in the direction opposite to the gradient of $E(\vec{w})$.

$$\nabla(E(\vec{w})) = \left(\frac{\partial E(\vec{w})}{\partial w_1}, \frac{\partial E(\vec{w})}{\partial w_2}, \dots \frac{\partial E(\vec{w})}{\partial w_{n+1}}\right) = \sum_{z_i \in M} -\vec{z}$$

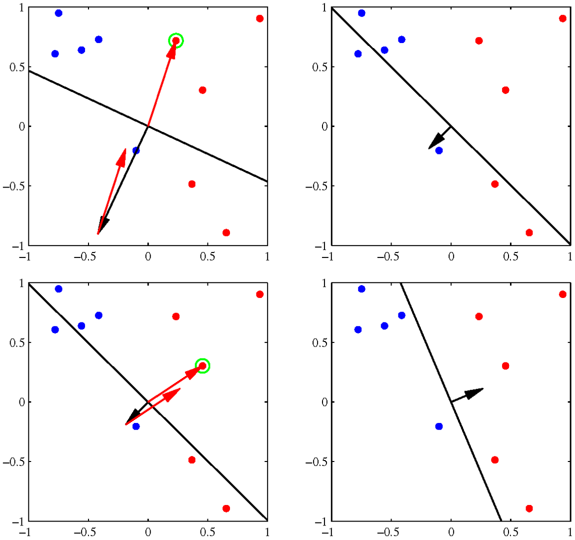
- The perceptron gradient algorithm:
  1. $k = 0$. Choose a random $\vec{w}$.
  2. $k \leftarrow k + 1$
  3. $\vec{w} \leftarrow \vec{w} + \eta(k) \sum_{z_i \in M_k} \vec{z}$
  4. if $|\nu(k) \sum_{z_i \in M_k} \vec{z}| > \theta$ go to 2
  5. return $\vec{w}$
- η - the learning rate, θ - an error threshold.

Percepton graphical representation

- $y(\vec{x}) = \vec{w}^t\vec{x} + w_0$, $y(\vec{x_a}) = y(\vec{x_b})$
- $\vec{x_a}$ a $\vec{x_b}$ is on decision surface, hence $\vec{w}^t(\vec{x_a} - \vec{x_b}) = 0)$
- w is orthonomal to decision surface, $\omega_0(b)$ is translation [Bishop]



Percepton learning

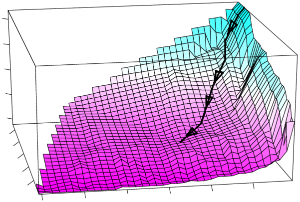


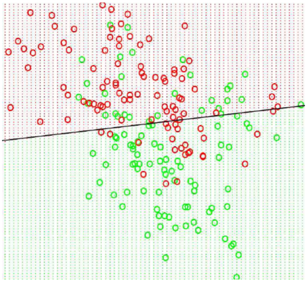
Percepton - linear separability

- If the two classes are linearly separable, the perceptron algorithm will terminate in a finite number of steps with zero training error.
- A problem that is linearly non-separable in $\Re^n$ may be separable after being transformed to $\Re^{n'}$ $n' > n$. For example, new coordinates may contain all quadratic terms:

$$[x(1), \ldots x(n), x^2(1), x(1)x(2), x(1)x(3), \ldots x^2(n)]$$

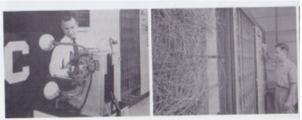
- A linear separation method such as the perceptron may be applied in the extended space, generating nonlinear separation in the original space.





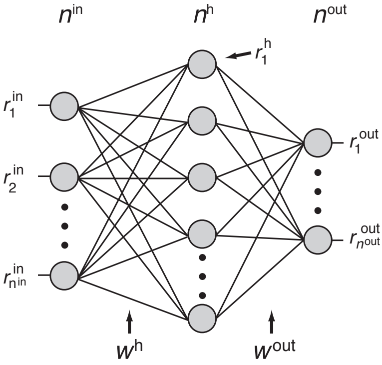
Percepton - history

- Frank Rosenblatt - HW realization of percepton in 1958



- Learning of simple symbols and alphabet - inspiration by brain nets
- Character was illuminated by powerful lights, image focused onto 20 x 20 array of cadmium sulphide photocells giving 400 pixel image
- Patch board - different configuration of input features
- Rack of adaptive weights, each weight rotary variable resistor driven by electric motor - weights were adjusted automatically by the elarning algorithm
- MARK 1 computer (Harvard - IBM): 765000 parts, 16 m long, 2.4 m height, 2 m wide, 3 operation per second, multiplication took 6 sec

The multilayer Perceptron (MLP)



Update rule: $\mathbf{r}^{\mathrm{out}} = g^{\mathrm{out}}(\mathbf{w}^{\mathrm{out}} g^{\mathrm{h}}(\mathbf{w}^{\mathrm{h}} \mathbf{r}^{\mathrm{in}}))$

Learning rule (error backpropagation): $w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}}$

The error-backpropagation algorithm

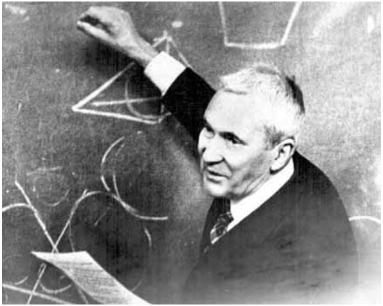
Initialize weights arbitrarily
Repeat until error is sufficiently small
    Apply a sample pattern to the input nodes: $r_i^0 := r_i^{\text{in}} = \xi_i^{\text{in}}$
    Propagate input through the network by calculating the rates of nodes in successive layers l: $r_i^l = g(h_i^l) = g(\sum_j w_{ij}^l r_j^{l-1})$
    Compute the delta term for the output layer: $\delta_i^{\text{out}} = g'(h_i^{\text{out}})(\xi_i^{\text{out}} - r_i^{\text{out}})$
    Back-propagate delta terms through the network: $\delta_i^{l-1} = g'(h_i^{l-1}) \sum_j w_{ji}^l \delta_j^l$
    Update weight matrix by adding the term: $\Delta w_{ij}^l = \epsilon \delta_i^l r_j^{l-1}$

MLP as universal approximator

- Hidden layer enables realization of complicated non-linear fces
- Each neuron can have its own activation fce
- We suppose that we have only ONE type of activation fce
- **QUESTION: Can 3-forward layer approximate any non-linear function?**
- **ANSWER: YES- thanks to A.Kolmogorov**
  Any continuous fce can be implemented by 3-layes net under assumption of sufficient number of n_H hidden neurons,suitable non-linearities and weights w.

Andrej Kolmogorov

- He constructed p̈erpetuum mobileïn high school, his teacher could not discover the trick
- First he studied history in Moscow university
- He published the first scientific work on realities in Novgorod area during 15. a 16. centurary
- The biggest contribution in probability field



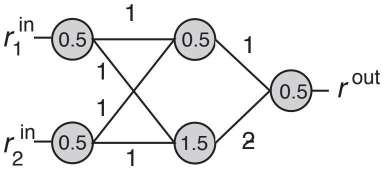
mlp.m

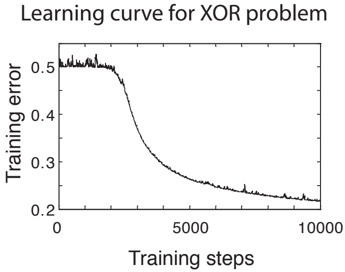
```
%% MLP with backpropagation learning on XOR problem
clear; clf;
N_i=2; N_h=2; N_o=1;
w_h=rand(N_h,N_i)-0.5; w_o=rand(N_o,N_h)-0.5;

% training vectors (XOR)
r_i=[0 1 0 1 ; 0 0 1 1];
r_d=[0 1 1 0];

% Updating and training network with sigmoid activation function
for sweep=1:10000;
  % training randomly on one pattern
    i=ceil(4*rand);
    r_h=1./(1+exp(-w_h*r_i(:,i)));
    r_o=1./(1+exp(-w_o*r_h));
    d_o=(r_o.*(1-r_o)).*(r_d(:,i)-r_o);
    d_h=(r_h.*(1-r_h)).*(w_o'*d_o);
    w_o=w_o+0.7*(r_h*d_o')';
    w_h=w_h+0.7*(r_i(:,i)*d_h')';
  % test all pattern
    r_o_test=1./(1+exp(-w_o*(1./(1+exp(-w_h*r_i)))));
    d(sweep)=0.5*sum((r_o_test-r_d).^2);
end
plot(d)
```

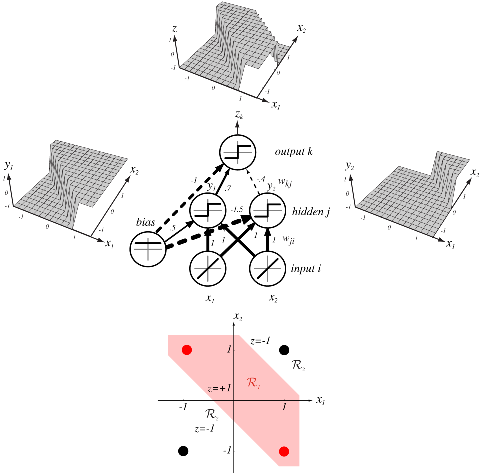
MLP for XOR function





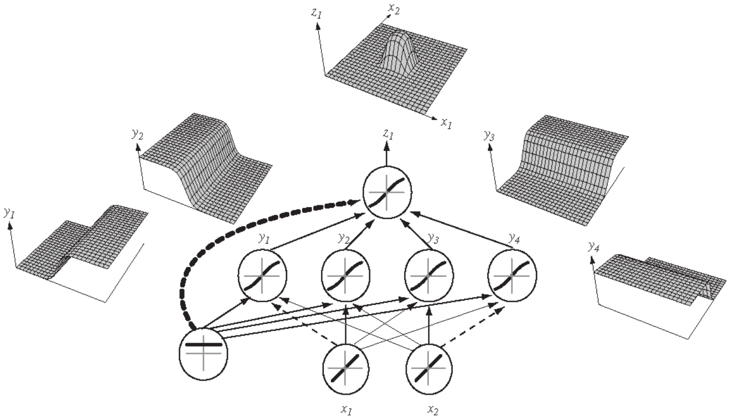
Example 3-layer neural net - XOR problem

- $0 \bigoplus 0 = 0$, $1 \bigoplus 1 = 0$, $1 \bigoplus 0 = 1$, $0 \bigoplus 1 = 1$
- $-1 \bigoplus -1 = -1$, $1 \bigoplus 1 = -1$, $1 \bigoplus -1 = 1$, $-1 \bigoplus 1 = 1$

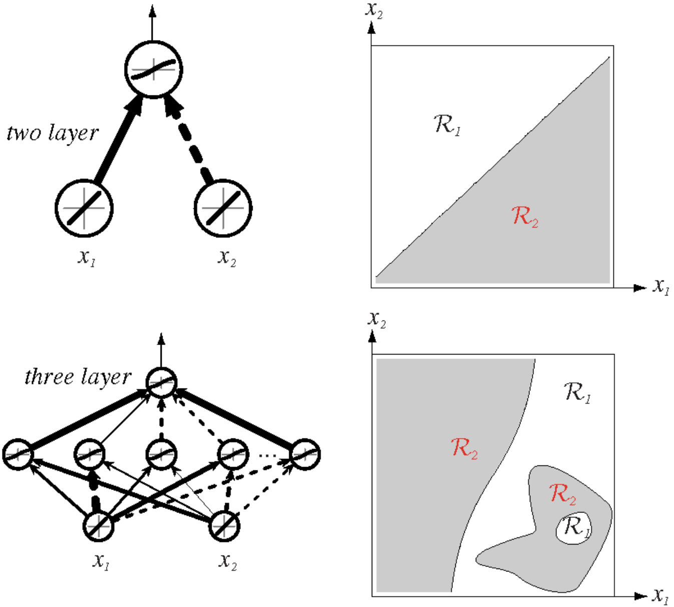


Non-linear fce approximation

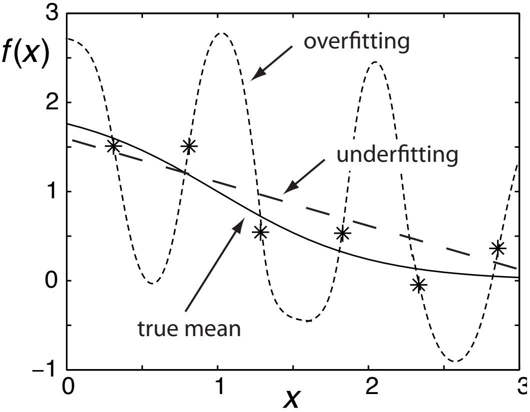
Fourier transform ANALOGY



Comparision of 2-layer and 3-layer net

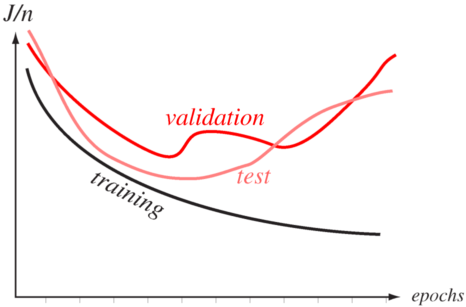


MLP, generalization, overfitting



Validation

- error of training set in monotonic-decreasing fce because of gradient algorithm optimization
- we divide data to training and validation set We use validation as stopping criteria (e.g. the first minimum)
- DEMO - Neural Network Toolbox Matlab
  `http://www.mathworks.com/products/neuralnet/`
- netlab -Bishop
  `http://www1.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/`

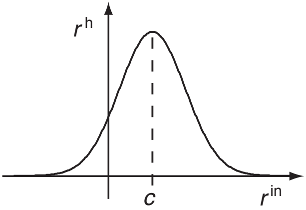


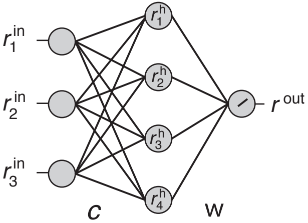
MLP biological plausibility

1. universal approximator $\rightarrow$ small number of hidden neurons $\rightarrow$ smooth solution & big number of hidden layers in biological systems
2. problematic training with error-back propagation, some exchange between postsynaptic and presynaptic neurons is possible, however
3. inclusion of derivative terms??
4. non-locality of the algorithm, neuron must gather the back-propagated errors from all other nodes to which it projects

Kernel machine

- better recognition after transformation of feature space $x_1 x_2, x_i^2$, $x \to \Theta(x)$, $w \to \Theta(w)$
- the net input of node $h = \sum_i (w_i r_i) = wr$, node in the network, $h = \Theta(w)\Theta(r) = K(w, r)$
- K is kernel function, special case is Gaussian kernel function $K(w, x) = \frac{(w-x)^2}{2\delta^2}$, FITS tuning curve
- Radial basis networks





Advance learning

- shallow part of errro function, very slow convergence, using momentum term

$$\Delta w_{ij}(t+1) = \eta \frac{\partial E}{w_{ij}} + \alpha \Delta w_{ij}(t)$$

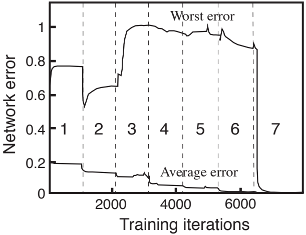
- Acceleration of learning process, other fce than MSE: entropic error function

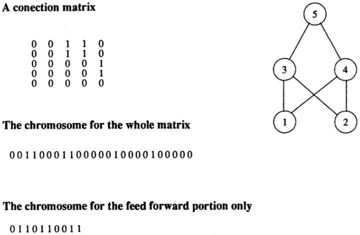
$$E = \frac{1}{2} \sum_{\mu,i} [(1+y_i^\mu) \log \frac{1+y_i^\mu}{1+r_i^{out}} + (1-y_i^\mu) \log \frac{1-y_i^\mu}{1-r_i^{out}}]$$

- measure information content of the output, even less computation of delta term: $g(x) = tanh(x), \delta_i = y_i - r^{out}$
- more sophisticated training using higher-order gradients: in MATLAB Levenberg-Marquardt. The relation of such sophistivate technique to biological learning is,so far, unclear!
- random search $\rightarrow$ stochastic processes, stochastic annealing, genetic algorithms

Self-organizing network architectures

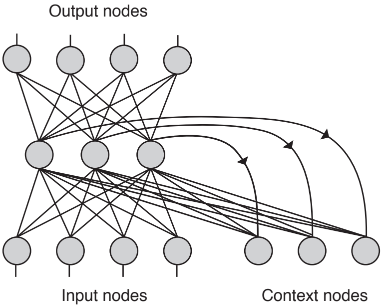
- how many nodes we need? too few $\rightarrow$ not good mapping, too many $\rightarrow$ reduction of generalization abilities, how the nodes should be connected?
- node creation algorithm $\rightarrow$ adding more and more nodes
- pruning algorithms $\rightarrow$ starting with large number of ones, e.g. weight decay, $w_{ij}(t+1) = w_{ij}(t) + \delta w_{ij} - \epsilon^{decay} w_{ij}(t)$
- genetics algorithm $\rightarrow$ vector [0010001] indicating presence of connection, biological inspiration $\rightarrow$ development of major structure of the central nervous system





Recurrent mapping networks - context units

- Elman net - simple recurrent net, physical back-projections
- short-term memory - input is connected to context units - remember the inputs from the previous time steps
- training of sequence of inputs e.g. predicting the next output (time series)



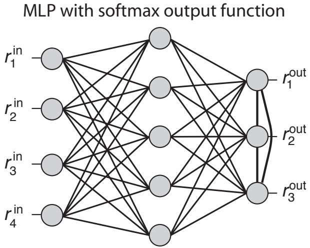
Probabilistic MLP

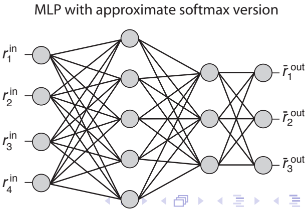
- data classification, n^{out} classes probability of the membership of the object
- all outputs nodes to 1, r^out firing rate of output node

$$\sum_i r_i^{out} = 1$$

- output layer competing for the output $\rightarrow$ collateral inhibitory connections, strong inhabitation - winner take all
- confidence of membership - soft competition:

$$r_i^{out} = \frac{e^{r_i^{out}}}{\sum_j r_j^{out}}$$

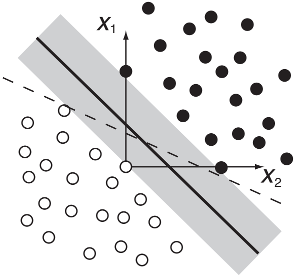


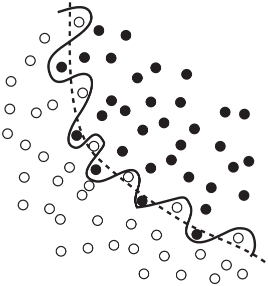


Support Vector Machines

- MLP: good interpolators, bad extrapolaters, local problem minima, slow convergence
- margin: distance from the middle line to the border, large-margin classifiers: more robust than percepton

Linear large-margine classifier





Margin

- distance of the line to the origins: $\frac{(\theta+1)}{|w|}$, $\frac{(\theta-1)}{|w|}$
- distance between the lines: $d = \frac{2}{|w|}$, minimizing weights subject to constrains

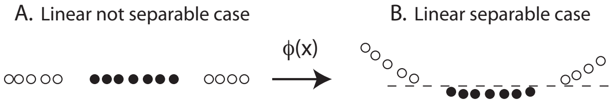
$$
\begin{aligned}
w_1 x_1 + w_2 x_2 - \theta &= 0 \\
w_1 x_1 + w_2 x_2 - \theta &= 1 \\
w_1 x_1 + w_2 x_2 - \theta &= -1 \\
y(wx - \theta - 1) &< 0
\end{aligned}
$$

- Lagrange formalism, constraints are added with multiplies α
- L_P is quadratic optimization problem, equivalent to dual problem L_D, data points on margine $\rightarrow$ support vector

$$
L_P = \frac{1}{2}|w|^2 + \sum_i \alpha_i y_i (wx_i - \theta) + sum_i \alpha_i
$$

SVM: Kernel trick

- non-linear separable data! Transformation $\phi(x) = (x, x^2)$, Kernel function $\phi)(x_i)\phi(x_j) = K(x_i, x_j)$
- right choice of kernel $\rightarrow$ convex optimization problem:
  $K(x_i, x_j) = e^{\frac{(x_i, x_j)^2}{2\sigma^2}}$



Further Readings

Simon Haykin (1999), **Neural networks: a comprehensive foundation**, MacMillan (2nd edition).

John Hertz, Anders Krogh, and Richard G. Palmer (1991), **Introduction to the theory of neural computation**, Addison-Wesley.

Berndt Müller, Joachim Reinhardt, and Michael Thomas Strickland (1995), **Neural Networks: An Introduction**, Springer

Christopher M. Bishop (2006), **Pattern Recognition and Machine Learning**, Springer

Laurence F. Abbott and Sacha B. Nelson (2000), **Synaptic plasticity: taming the beast**, in **Nature Neurosci. (suppl.)**, 3: 1178–83.

Christopher J. C. Burges (1998), **A Tutorial on Support Vector Machines for Pattern Recognition** in **Data Mining and Knowledge Discovery** 2:121–167.

Alex J. Smola and Bernhard Schölhopf (2004), **A tutorial on support vector regression** in **Statistics and computing** 14: 199-222.

David E. Rumelhart, James L. McClelland, and the PDP research group (1986), **Parallel Distributed Processing: Explorations in the Microstructure of Cognition**, MIT Press.

Peter McLeod, Kim Plunkett, and Edmund T. Rolls (1998), **Introduction to connectionist modelling of cognitive processes**, Oxford University Press.

E. Bruce Goldstein (1999), **Sensation & perception**, Brooks/Cole Publishing Company (5th edition).