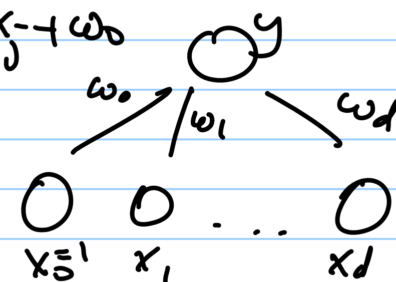
Chp 11 (1)
Alpaydin

Multilayer Perceptrons

4/27/2008

Analogy to Neural Networks in the Brain.
— probably misguided.

<u>perceptron.</u> $y = \sum_{j=1}^{d} w_j x_j + w_0$



<u>idealized neuron</u>

threshold function.

hard $\quad S(a) = \begin{cases} 1, & \text{if } a > 0 \\ 0, & \text{otherwise.} \end{cases}$ $\quad$ || linear discriminant.

soft. $\quad y = \sigma(\underline{w}^T \underline{x}) = \dfrac{1}{1 + e^{-\underline{w}^T \underline{x}}}.$ $\quad \sigma(.)$ sigmoid function.

There are a variety of different algorithms to train a perceptron from labelled examples

<u>Example</u>: Quadratic error.

$$E(\underline{w} \mid \underline{x}^t, r^t) = \frac{1}{2}(r^t - y^t)^2 \qquad y = \underline{w}^T \underline{x}$$

update rule $\quad \Delta w_j^t = \eta (r^t - y^t) x_j^t.$

$$E(\{\underline{w}_i\} \mid \underline{x}^t, r^t) = -\sum_i \{ r_i^t \log y_i^t + (1 - r_i^t) \log(1 - y_i^t) \}$$

$$y^t = \text{sigmoid}(\underline{w}_i^T \underline{x}^t).$$

update rule $\quad \Delta w_j^t = \eta (r^t - y^t) x_j^t.$

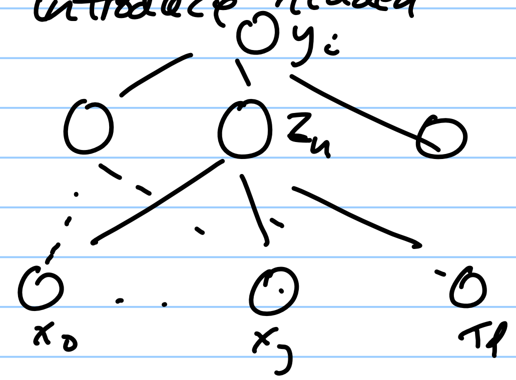
Update = Learning Factor. (Desired Output − Actual Output) × Input.

(2)

Multilayer Perceptron.

A single layer perceptron can only approximate liner function of the input — i.e. the set of perceptrons has limited capacity.

Multilayer perceptrons were invented to increase the capacity — introduce hidden units (nodes)



$z_h = \sigma(\underline{w}_h^T \underline{x})$

$= \dfrac{1}{1 + \exp\{-\sum_{j=1}^{d} w_{hj} x_j + w_{i0})\}}$,

$h = 1, .. H.$

Output $y_i = \underline{v}_i^T z = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}.$

Other output function — e.g. $y_i = \sigma(\underline{v}_i^T \underline{z})$.

Many levels can be specified.

What do the hidden units represent?

Unclear., but many people have tried to explain them

Any input-output function can be represented as a multilayer perceptron with enough hidden units

— infinite capacity.

(3)

How to train a multilayer perceptron?

Unknown parameters – the weights ω_{hj}, v_{ij}.

Define an error function:

e.g. $$E[\omega, v] = \sum_i \left(y_i - \sum_h v_{ih} \, \mathrm{sigmoid}\left(\sum_j \omega_{hj} x_j\right) \right)^2$$

update
$$\Delta \omega_{hj} = \frac{-\partial E}{\partial \omega_{hj}}.$$
$$\Delta v_{ik} = \frac{-\partial E}{\partial v_{ih}}$$

these gradients can be computed by the chain rule of differentiation
$$\frac{\partial E}{\partial \omega_{hj}} = \frac{\partial E}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_h} \cdot \frac{\partial z_h}{\partial \omega_{hj}}$$

For historical reasons, this is known as the backpropagation algorithm.
→ the errors would be projected back to the hidden units. It was ~~thought~~ /hoped that this might be biologically plausible.

(4)

More details.

quadratic error.

For the 2nd layer.

$$\Delta v_{ih} = \eta \sum_t (r^t - y^t) z_h^t$$

For the 1st layer

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = -\eta \sum_t \frac{\partial E}{\partial y^t} \cdot \frac{\partial y^t}{\partial z_h^t} \cdot \frac{\partial z_h^t}{\partial w_{hj}}$$

$$= -\eta \sum_t -(r^t - y^t)\, v_h \, z_h^t (1 - z_h^t) x_j^t$$

$$= \eta \sum_t \underbrace{(r^t - y^t) v_h}_{\text{error term for the hidden units}} z_h^t (1 - z_h^t) x_j^t$$

Batch learning versus stochastic learning
→ advantages of stochastic learning.

Training Procedures:

momentum $\quad \Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$ $\quad$ t - epoch.

overtraining - .

(5) Multilayer Perceptrons and Nonlinear Regression

multilayer perceptrons can be thought as non linear regression — and forget about the neural interpretation.

Output function

$$y_i(\underline{x}) = \sum_h v_{ih}\, \sigma\Big(\sum_j \omega_{hj} x_j\Big)$$

regression:

$$y_i(\underline{x}) = \sum_h v_{ih}\, \bar{\sigma}\Big(\sum_j \omega_{hj} x_j\Big) + \underbrace{\epsilon}_{\text{zero mean Gaussian noise}}.$$

Cost function.

$$\sum_{t=1}^{N} \sum_i \Big\{ y_i^t - \sum_h v_{ih}\, \bar{\sigma}\Big(\sum_j \omega_{hj} x_j^t\Big)\Big\}^2$$

(-log of probability) Gaussian

Since this is nonlinear, we cannot obtain an analytic solution for the coefficients $\{v_{ih}\}, \{\omega_{hj}\}$.

Instead, we have to minimize the cost function algorithmically.

(6) Multilayer Perceptron and Steepest Descent.

Strategy: Batch Mode.
use all the data, i.e. full cost function.

steepest descent.

E cost function

$$\frac{dv_{ih}}{dt} = -\frac{\partial E}{\partial v_{ih}}$$

$$\frac{dw_{hj}}{dt} = -\frac{\partial E}{\partial w_{hj}}$$

discrete formulation

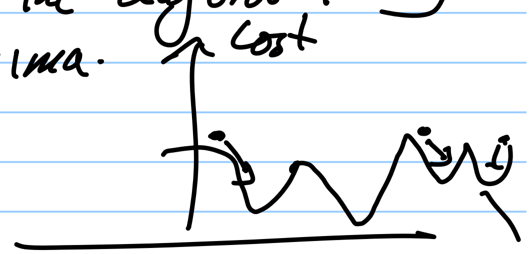
$$v_{ih}(t+\Delta t) = v_{ih}(t) - \Delta t \frac{\partial E}{\partial v_{ih}}.$$

$$w_{hj}(t+\Delta t) = w_{hj}(t) - \Delta t \frac{\partial E}{\partial w_{hj}}.$$

(page 304).

The cost function will usually have many local minima, so the algorithm may get stuck in a local minima.

Final result depends on initial conditions.



(7) Multilayer Perceptron & Stochastic Descent

Perform one iteration from each data sample

Current state: $\{v_{ih}, w_{hj}\}$

select data sample y_i^t, x_j^t.

set $v_{ih} \to v_{ih} + \Delta v_{ih}$

$w_{hj} \to w_{hj} + \Delta w_{hj}$

with $\Delta v_{ih} = -\frac{\partial}{\partial v_{ih}} \sum_i \left\{ y_i^t - \sum_h v_{ih}\, \sigma\left(\sum_j w_{hj} x_j^t\right) \right\}^2$

$\Delta w_{hj} = -\frac{\partial}{\partial w_{hj}} \sum_i \left\{ y_i^t - \sum_h v_{ih}\, \sigma\left(\sum_j w_{hj} x_j^t\right) \right\}^2$

This is "stochastic" because the algorithm has a random element – the choice of which data sample to use.

Stochastic descent is able to escape from some local minima (theoretical results).

(8) Critical Issues for multilayer perceptrons

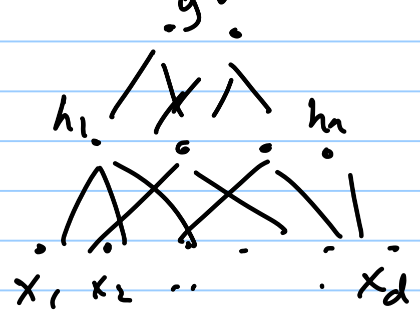
— how many hidden units to use?

Book desribes several techniques for dealing with this → for example, having more hidden units than you need and then penalizing the weights.

E.G. Add term $\sum_{h,j} (w_{hj}^2) + \sum_{i,h} (v_{ih}^2)$ to the cost function → intuition, if the weights are small to a hidden unit, then the hidden unit is not used.

In practice, some of the most effective multi layer perceptrons are those which the structure was hand designed.

(9) Multilayer Perceptrons / SVM / AdaBoost (next lecture).



$$y_i = \sum_h v_{ih} h_h$$
$$h_h = \sum_j \sigma\left(\sum w_{hj} x_j\right).$$

<u>SVM</u> can also be represented in this way

$$y = \text{sign}\left(\sum_\mu \alpha_\mu y_\mu \underline{x}_\mu \cdot \underline{x}\right)$$

hidden units responce $\underline{x}_\mu \cdot \underline{x} = h_\mu$.

$$y = \text{sign}\left(\sum_\mu \alpha_\mu y_\mu h_\mu\right)$$

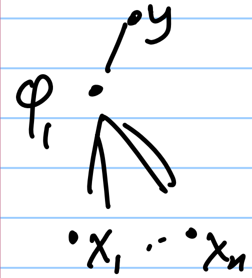
Advantage of SVM — number of hidden units is given by the no. of support vectors.

$\rightarrow \{\alpha_\mu\}$ specified by minimizing the primal problem (well defined algorithm to perform this minimization).

(10) Multilayer Perceptrons / SVM / AdaBoost (next lecture).

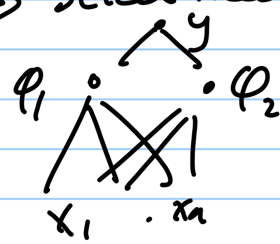
AdaBoost (next lecture)

Build a multilayer perceptron incrementally

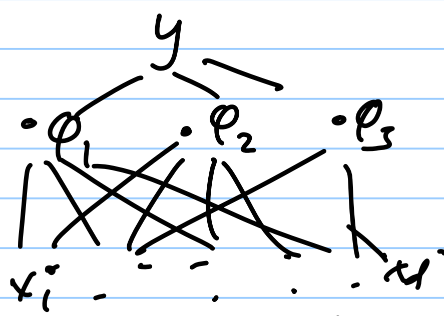


one hidden unit. $\varphi_1(\underline{x})$

→ select new hidden unit $\varphi_2(\underline{x})$



→ new unit $\varphi_3(x)$



→ and so on.

AdaBoost selects the hidden units incrementally by minimizing a criterion.