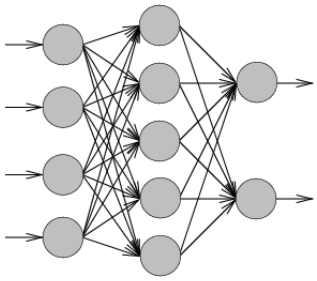
CMSC35000-1 Introduction to Artificial Intelligence **Winter 2005**

Lecture 4: Jan/16th/2007

Lecturer: Partha Niyogi *Scribe: Hector Villafuerte*

So far we've seen: *Linear methods:* perceptron learning algorithm (PLA); *Pattern recognition:* Bayes discriminant function (BDF). Recall also that in practice we don't have the underlying probability distribution, so instead of minimizing the *true error* we go after the *empirical error*.



Today's topic is *Multilayer perceptrons* (aka *(Artificial) Neural networks*). Natural questions that arise when we have a multilayer perceptron like the one shown are: how to set (adapt) the weights? is there an algorithm? Note that in the '60s there was no satisfactory solution, and that is why AI shifted to other things (e.g. expert systems) in the '70s. But then in 1986 it became popular again with a paper by Rumelhart, Hinton, Williams, *"Learning internal representations by error propagation"*.

And now the *backpropagation algorithm*:

$$(x_i, y_i),\ i = 1..n \tag{4.1}$$

$$h(x) = \theta(\sum_{k=1}^{n} w_k f_k(x)) \tag{4.2}$$

Note that 2 main things were done: (1) θ became σ (the sigmoid), (2) the error rate was written as $\sum_{i=1}^{n}(y_i - h(x))^2$. And then they used: $\min_w \left(\sum_{i=1}^{n}(y_i - h(x))^2\right)$. Lets now describe the sigmoid. $\sigma : \mathbb{R} \to (0,1)$, $\sigma(z) = \frac{1}{1+\exp(-\alpha z)}$. Let $o(\cdot)$ be the output, D the layer, we then have:

$$h(x) = o(x) = \sigma\left(\sum_{i=1}^{n_D} w_{i1}^{D+1} o_i^D(x)\right) \tag{4.3}$$

$$o_j^D(x) = \sigma\left(\sum_{i=1}^{n_{D-1}} w_{ij}^D o_i^(D-1)(x)\right) \tag{4.4}$$

Now we would like to adapt the weights: w_{ij}^k. The main idea is simple: just use *gradient descent*. The objective function is $J = (y - o(x))^2$.

$$\bar{w}_0 \to \bar{w}_1 \tag{4.5}$$

$$\bar{w}_{t+1} = \bar{w}_t - \varepsilon \nabla_{\bar{w}} J \tag{4.6}$$

Now the question is how to calculate $\nabla_{\bar{w}} J$? The rest of the lecture is an exercise of the *chain rule*.

$$J = \sum_{i=1}^{n} (y - o(x_i))^2 \tag{4.7}$$

$$\frac{\partial J}{\partial w} = \sum_{i=1}^{n} \frac{\partial J_i}{\partial w} \tag{4.8}$$

Where $J_i = (y_i - o(x_i))^2$. Note that for the rest of the discussion we'll drop the index i: $J = (y - o(x))^2$.

$$\frac{\partial J}{\partial w_{ij}^k} = \sum_{i=1}^{n_k} \frac{\partial J}{\partial o_l^k} \frac{\partial o_l^k}{\partial w_{ij}^k} \tag{4.9}$$

Also note that $\frac{\partial o_l^k}{\partial w_{ij}^k} = 0$ if $j \neq l$. Therefore:

$$\frac{\partial J}{\partial w_{ij}^k} = \frac{\partial J}{\partial o_j^k} \frac{\partial o_j^k}{\partial w_{ij}^k} \tag{4.10}$$

We now have the next recurrence relation:

$$\frac{\partial J}{\partial o_j^k} = \sum_{l=1}^{n_{k+1}} \frac{\partial J}{\partial o_l^{k+1}} \frac{\partial o_l^{k+1}}{\partial o_j^k} \tag{4.11}$$

$$\frac{\partial o_l^{k+1}}{\partial o_j^k} = \sigma \left(\sum_{m=1}^{n_k} w_{ml}^{k+1} o_m^k \right) \tag{4.12}$$

$$= \frac{d\sigma}{dz} \frac{\partial z}{\partial o_j^k} \tag{4.13}$$

where $z = \sum_{m=1}^{n_k} w_{ml}^{k+1} o_m^k$,

$$= w_{jl}^{k+1} \frac{d\sigma}{dz} \tag{4.14}$$

$$\frac{\partial o_l}{\partial o_j^k} = \alpha w_{jl}^{k+1} o_l^{k+1} (1 - o_l^{k+1}) \tag{4.15}$$

$$\frac{\partial o_j^k}{\partial w_{ij}^k} = \alpha o_j^k (1 - o_j^k) o_i^{k-1} \tag{4.16}$$

$$o_j^k = \sigma \left(\sum_{m=1}^{n_{k-1}} w_{mj}^k o_m^{k-1} \right) \tag{4.17}$$

So we can see that backpropagation is easy conceptually and numerically, and also this approach has had an important social role to revive probabilistic and numerical methods in the AI community.