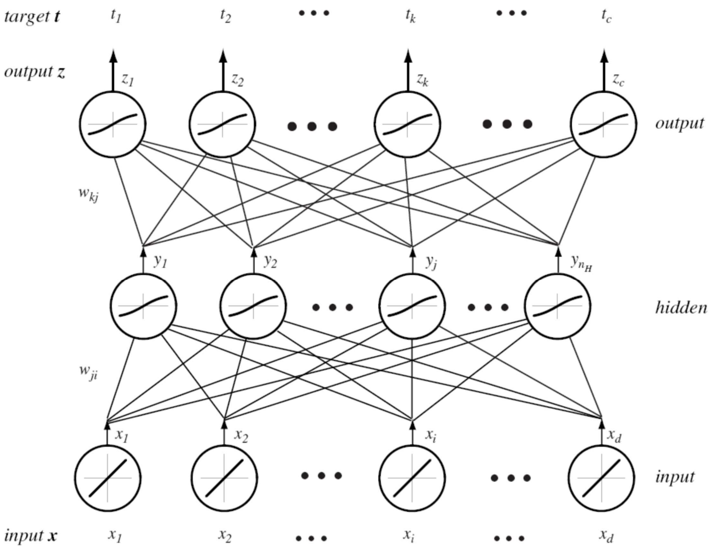
Backpropagation

- **1 Feedforward Operation**
- **2 Backpropagation**
- **3 Improving backpropagation**
- **4 Other techniques**

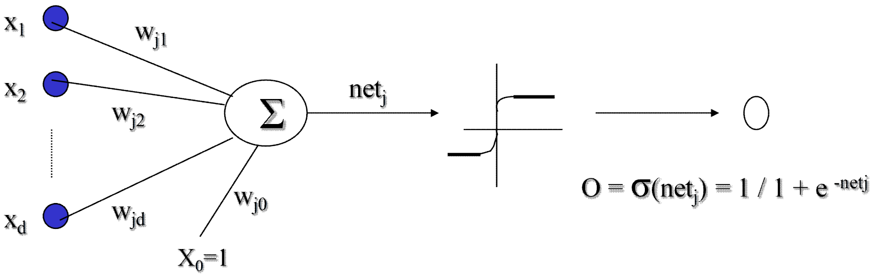
----

1 Feedforward Operation

Turn to the problem for a three-layer net



As for One Single sigmoid Unit



1.1 Each hidden unit performs the weighted sum of its inputs to form its (scalar) net activation or simply net.

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = W_j^t X$$

$$O = \sigma(net_j) = 1 / 1 + e^{-netj}$$

Function σ is called the sigmoid or logistic function.

$d\sigma(x) / dx = \sigma(x)\ (1 - \sigma(x))$

1.2 从以上作为从输入层到隐藏层的前向传播, 接下来进行从隐藏层到输出层的传播, 同理

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = W_k^t Y$$

$z_k = f(net_k)$

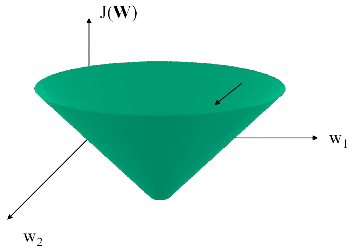
Induction, Feedforward Operation And Classification

$$g_k(\mathbf{x}) \equiv z_k = f\left(\sum_{j=1}^{n_H} w_{kj}\ f\left(\sum_{i=1}^{d} w_{ji}x_i + w_{j0}\right) + w_{k0}\right)$$

2 Backpropagation

Next To learn the weights for all links in an interconnected multilayer network.

The idea is to use again a gradient descent over the space of weights to find a global minimum (no guarantee).



2.1 First define our measure of error

$$J(W) = 1/2 \sum_{k=1}^{c} (t_k - z_k)^2 = 1/2(T - Z)^2$$

T and Z are the target and the network output vectors of length c;
W represents all the weights in the network.

The weights are initialized with random values, and are changed in a direction that will reduce the error

$$\Delta W = -\eta \frac{\partial J}{\partial W}$$

where h is the learning rate

at iteration m updating it as

$$W(m+1) = W(m) + \Delta W(m)$$

2.2 Consider first the hidden-to-output weights, w_{kj}

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

where the sensitivity of unit k is defined to be

$$\delta_k = -\frac{\partial J}{\partial net_{kj}} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

Taken together, these results give the weight update (learning rule) for the hidden-to-output weights

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

2.3 The learning rule for the input-to-hidden units.

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$\frac{\partial J}{\partial y_j} = \frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k) f'(net_k) w_{jk}$$

Define the sensitivity for a hidden unit as:

$$\delta_j = f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

Thus the learning rule for the input-to-hidden weights is:

$$\Delta w_{ji} = \eta x_i \delta_j = \eta x_i f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{ji}} = \frac{\partial J(\mathbf{w})}{\partial y_j} \cdot \overbrace{\frac{\partial y_j}{\partial net_j}}^{f'(netj)} \cdot \overbrace{\frac{\partial net_j}{\partial w_{ji}}}^{x_i} \qquad \frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} = f'(net_k) \cdot w_{kj}$$

$$\frac{\partial J(\mathbf{w})}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k) \cdot \frac{\partial z_k}{\partial y_j}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_{ji}} = -\sum_{k=1}^{c}\underbrace{(t_k - z_k) f'(net_k)}_{\delta_k} \cdot w_{kj} = -\sum_{k=1}^{c}\delta_k w_{kj}$$

$$\Rightarrow \Delta w_{ji} = -\eta \frac{\partial J}{\partial w_{ji}} = \eta \underbrace{\left[\sum_{k=1}^{c}\delta_k w_{kj}\right] f'(net_j)}_{\triangleq\, \delta_j} x_i = \eta \cdot \delta_j \cdot x_i$$

BP Algorithm

1. Create a network with n_{in} input nodes, n_H internal nodes, and nout output nodes.
2. Initialize all weights to small random numbers.
3. Until error is small do:

   For each example X do

   - Propagate example X forward through the network
   - Propagate errors backward through the network

$$\delta_k = (t_k - z_k) f'(net_k)$$

$$\delta_j = f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

$$w_{kj} \leftarrow w_{kj} + \eta \delta_k y_j$$

$$w_{ji} \leftarrow w_{ji} + \eta \delta_j x_i;$$

3 Improving backpropagation

Adding Momentum

The weight update rule can be modified so as to depend on the last iteration. At iteration n we have the following:

$$\Delta W_{ji}(n+1) = \eta \delta_j X_{ji} + \alpha \Delta W_{ji}(n)$$

Where

$$\alpha (0 \le \alpha \le 1)$$

is a constant called the momentum.

a. It increases the speed along a local minimum
b. It increases the speed along flat regions.

Remarks on Backpropagation

1. It implements a gradient descent search over the weight space.
2. It may become trapped in local minima.
3. In practice, it is very effective.
4. The more weights the chances to avoid local minima.
5. How to avoid local minima?
   a. Add momentum
   b. Use stochastic gradient descent
   c. Use different networks with different initial values for weights.

4 Other techniques - Training Protocols

Stochastic Training:

Examples are chosen randomly from the training set.
Weights are updated on each example.

Batch Training:

Examples are given to the network before weights are updated.

Online Training

Each example is selected once and only once; we do not store examples in memory.

Algorithm 1 (Stochastic backpropagation)

<u>**begin**</u> <u>**initialize**</u> network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, m \leftarrow 0$
    <u>**do**</u> $m \leftarrow m + 1$
        $\mathbf{x}^m \leftarrow$ randomly chosen pattern
        $w_{ij} \leftarrow w_{ij} + \eta\delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta\delta_k y_j$
    <u>**until**</u> $\nabla J(\mathbf{w}) < \theta$
<u>**return**</u> $\mathbf{w}$
<u>**end**</u>

Algorithm 2 (Batch backpropagation)

<u>**begin**</u> <u>**initialize**</u> network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$
    <u>**do**</u> $r \leftarrow r + 1$ (increment epoch)
        $m \leftarrow 0; \ \Delta w_{ij} \leftarrow 0; \ \Delta w_{jk} \leftarrow 0$
        <u>**do**</u> $m \leftarrow m + 1$
            $\mathbf{x}^m \leftarrow$ select pattern
            $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta\delta_j x_i; \quad \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta\delta_k y_j$
        <u>**until**</u> $m = n$
        $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}; \quad w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$
    <u>**until**</u> $\nabla J(\mathbf{w}) < \theta$
<u>**return**</u> $\mathbf{w}$
<u>**end**</u>