DERIVATION OF THE GRADIENT DESCENT RULE

- To calculate the direction of steepest descent along the error surface:
- This direction can be found by computing the derivative of ***E*** with respect to each component of the vector $\vec{w}$**.**
- This vector derivative is called the ***gradient*** of ***E*** with respect to $\vec{w}$ written $\nabla E(\vec{w})$

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

- Since the gradient specifies the direction of steepest increase of ***E,*** the training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

where

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

- training rule can also be written in its component form

$$w_i \leftarrow w_i + \Delta w_i$$

where

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

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_{d \in D} (t_d - o_d)(-x_{id})
\end{aligned}
$$

So final standard GRADIENT DESCENT

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

- Gradient descent is a strategy for searching through a large or infinite hypothesis space that can be applied whenever

(1) the hypothesis space contains continuously parameterized hypotheses

(2) the error can be differentiated with respect to these hypothesis parameters.

- The key practical difficulties in applying gradient descent are

(1) converging to a local minimum can sometimes be quite slow

(2) if there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

- One common variation on gradient descent intended to alleviate these difficulties is called ***incremental gradient descent,*** or alternatively ***stochastic gradient descent.***

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \quad \text{(T4.1)}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i \quad \text{(T4.2)}$$

- Whereas the s**tandard gradient descent** training rule presented in Equation computes weight updates after summing over ***all*** the training examples in D, the idea behind **stochastic gradient descent** is to approximate this gradient descent search by updating weights incrementally, following the calculation of the error for ***each*** individual example.

$$\Delta w_i = \eta(t - o)\ x_i$$

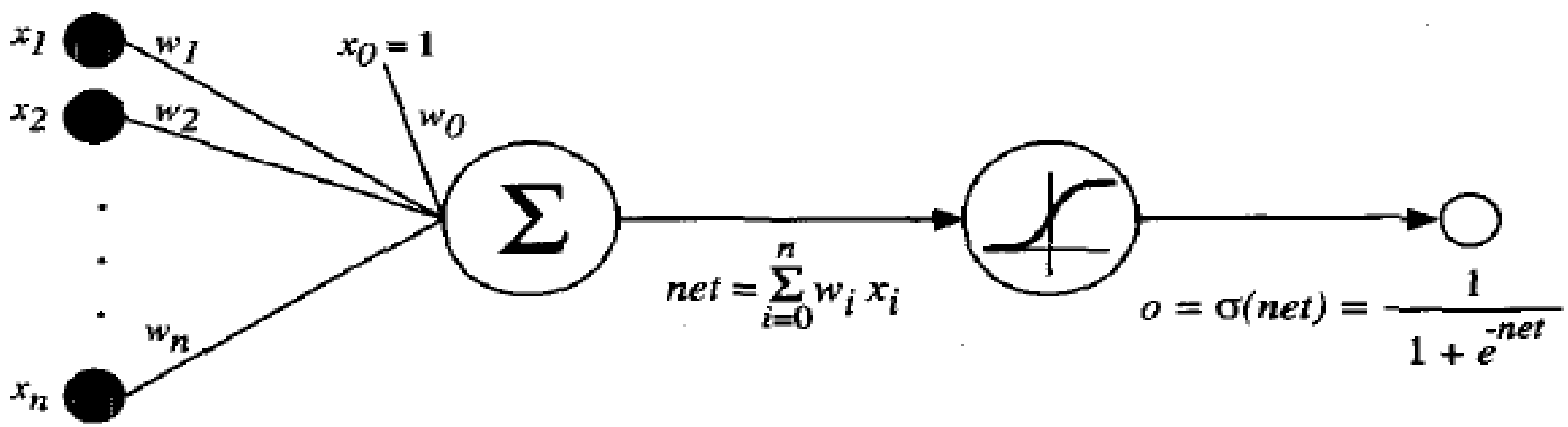
$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$$

MULTILAYER NETWORKS AND THE BACKPROPAGATION ALGORITHM

- single perceptrons can only express linear decision surfaces.

A Differentiable Threshold Unit

The sigmoid threshold unit



- the sigmoid unit computes its output ***o*** as

$$o = \sigma(\vec{w} \cdot \vec{x})$$

where

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))]$$

The BACKPROPAGATIAOIN Algorithm

- we begin by redefining E to sum the errors over all of the network output units:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and $.05$).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in $training\_examples$, Do

    Propagate the input forward through the network:

    1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

    Propagate the errors backward through the network:

    2. For each network output unit k, calculate its error term δ_k

    $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

    3. For each hidden unit h, calculate its error term δ_h

    $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

    4. Update each network weight w_{ji}

    $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

    where

    $$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \tag{T4.5}$$

ADDING MOMENTUM

$$\Delta w_{ji}(n) = \eta\,\delta_j\,x_{ji} + \alpha\,\Delta w_{ji}(n-1)$$

LEARNING IN ARBITRARY ACYCLIC NETWORKS

$$\delta_r = o_r (1 - o_r) \sum_{s \in layer\, m+1} w_{sr} \delta_s$$

$$\delta_r = o_r (1 - o_r) \sum_{s \in Downstream(r)} w_{sr} \delta_s$$

Derivation of the BACKPROPAGATION Rule

- The specific problem we address here is deriving the stochastic gradient descent rule implemented by the algorithm

- Stochastic gradient descent involves iterating through the training examples one at a time, for each training example d descending the gradient of the error $\boldsymbol{E_d}$ with respect to this single example.

- In other words, for each training example ***d*** every weight $\boldsymbol{w_{ji}}$ is updated by adding to it $\nabla \boldsymbol{w_{ji}}$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- x_{ji} = the ith input to unit j
- w_{ji} = the weight associated with the ith input to unit j
- $net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)
- o_j = the output computed by unit j
- t_j = the target output for unit j
- σ = the sigmoid function
- $outputs$ = the set of units in the final layer of the network
- $Downstream(j)$ = the set of units whose immediate inputs include the output of unit j

1.

- To begin, notice that weight $\boldsymbol{w_{ji}}$ can influence the rest of the network only through $\boldsymbol{net_j}$.

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ji} \qquad (4.22)$$

Case 1: Training Rule for Output Unit Weights.

- ***net**$_j$* can influence the network only through ***o**$_j$*

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

$$
\begin{aligned}
\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2 \\
&= \frac{1}{2}2(t_j - o_j)\frac{\partial (t_j - o_j)}{\partial o_j} \\
&= -(t_j - o_j)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial o_j}{\partial net_j} &= \frac{\partial \sigma(net_j)}{\partial net_j} \\
&= o_j(1 - o_j)
\end{aligned}
$$

- So,

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j)$$

- Finally, we have the stochastic gradient descent rule for ***output units***

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta \ (t_j - o_j) \ o_j(1 - o_j)x_{ji}$$

Case 2: Training Rule for Hidden Unit Weights

- ***net_j*** can influence the network outputs (and therefore E_d) only through the units in ***Downstream(j).***

$$
\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \; w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \; w_{kj} \; o_j(1 - o_j)
\end{aligned}
$$

Rearranging terms and using δ_j to denote $-\frac{\partial E_d}{\partial net_j}$, we have

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \; w_{kj}$$

.

- So,

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$