Gradient Descent – BEP –

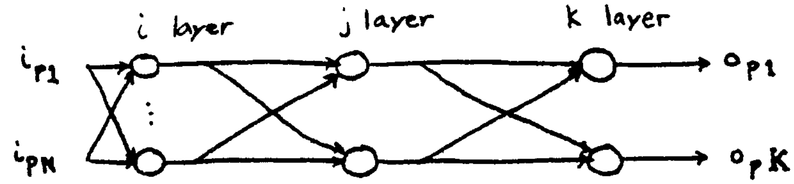
5 May 1989

Neil E Cotter

Chain rule and delta rule

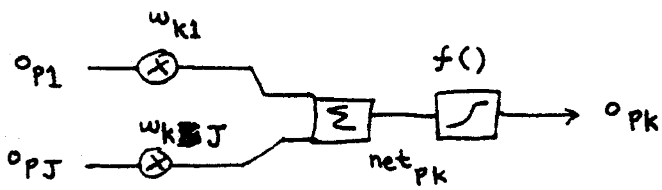
We need to calculate $\Delta_p w_{ij} = -\eta \dfrac{\partial E_p}{\partial w_{ij}}$.

$w_{ij}$ can be anywhere in a neural neural network of, say, three layers:




Each circle represents one neuron, complete with synaptic weights.

Consider first a neuron on the <u>output layer</u>:




$$\Delta_p w_{kj} = -\eta \frac{\partial E_p}{\partial w_{kj}}$$

1) $$\frac{\partial E_p}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \frac{1}{2} (t_{pk} - o_{pk})^2$$

$t_{pk}$: desired output; $o_{pk}$: actual output

$$= (t_{pk} - o_{pk}) \frac{\partial}{\partial w_{kj}} (t_{pk} - o_{pk})$$

• by chain rule $\frac{\partial}{\partial x} f(g(x)) = f'(g(x)) g'(x)$

$$= -(t_{pk} - o_{pk}) \frac{\partial}{\partial w_{kj}} o_{pk}.$$

Now use the chain rule to compute $\dfrac{\partial}{\partial w_{kj}} o_{pk}$.

May 1989
Neil E Cotter

Gradient Descent – BEP –
Chain rule and delta rule (cont.)

2) $$\frac{\partial}{\partial w_{kj}} o_{pk} = \frac{\partial}{\partial w_{kj}} f(net_{pk})$$

$$= f(net_{pk})\left[1 - f(net_{pk})\right] \frac{\partial}{\partial w_{kj}} net_{pk}$$

· Recall that $f'(x) = f(x)\left[1 - f(x)\right]$
· Also use chain rule

Now compute $\frac{\partial}{\partial w_{kj}} net_{pk}$:

3) $$\frac{\partial}{\partial w_{kj}} net_{pk} = \frac{\partial}{\partial w_{kj}}\left(\sum_{j=1}^{J} w_{kj}\, o_{pj} + \theta_k\right)$$

$$= \frac{\partial}{\partial w_{kj}}\left(\sum_{j=1}^{J} w_{kj}\, o_{pj}\right)$$

$= ?$ Here we have to be careful about notation. "$\frac{\partial}{\partial w_{kj}}$" refers to a particular synaptic weight such as $w_{23}$, say. "$\Sigma w_{kj}\, o_{pj}$" refers to the sum over various different $w_{kj}$ synapse values.

$\frac{\partial}{\partial w_{kj}} w_{kj} = 0$ unless both $j$'s have the same value.

$\therefore \frac{\partial}{\partial w_{kj}} \sum_{j=1}^{J} w_{kj}\, o_{pj} = o_{pj}$ · One term, $w_{kj}\, o_{pj}$, is extracted from the $\Sigma$

ex: $$\frac{\partial}{\partial w_{12}} \sum_{j=1}^{2} w_{1j}\, o_{pj} = \frac{\partial}{\partial w_{12}}\left(w_{11}\, o_{p1} + w_{12}\, o_{p2}\right) = \frac{\partial}{\partial w_{12}} w_{12}\, o_{p2}$$

$$= o_{p2}$$

$$\therefore \frac{\partial}{\partial w_{kj}} net_{pk} = o_{pj}$$

May 1989
Neil E Cotter

Gradient Descent - BEP -
Chain rule and delta rule (cont.)
Put results together:

Output layer learning rule:

$$\nabla_p w_{kj} = -\eta \frac{\partial E_p}{\partial w_{kj}} = -\eta (-1)(t_{pk} - o_{pk})\, f(net_{pk}) \left[1 - f(net_{pk}\right. \cdot o_{pj}$$

↑
input from previous layer outputs

$$= \eta \underbrace{(t_{pk} - o_{pk})\, f(net_{pk}) \left[1 - f(net_{pk})\right]}_{\equiv\ \delta_{pk}} o_{pj}$$

- We identify a quantity, $\delta_{pk}$, that is a function of things that happen after signals pass through a synapse. The $\delta_{pk}$ information is therefore propagated back through the network to the synapse. Unfortunately, no physiological mechanism for this has been identified. But it works. It learns by gradient descent.
- The $o_{pj}$ is the input to the synapse, $(w_{kj})$ and $o_{pj}$ comes from the previous layer of the network.
- $\eta$ is the step size for the gradient descent.

A subtle point: Our gradient descent changes the weights of the network so as to minimize $E_p$ for a particular pattern p. What we really want to do is minimize the average error over all patterns: Ave $E_p = \frac{1}{P} \sum_{p=1}^{P} E_p$.

The theory is that using a small value for $\eta$ and changing to a different learning pattern p at each step of the gradient descent will give small values of averaged error.

Gradient Descent — BEP —
Chain rule and delta rule (cont.)

15 May 1989
Neil E Cotter

Neurons in hidden layers, learning rule

- A hidden layer is any layer other than the output layer.

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$$

Proceed as before for steps (1) and (2). We hit a snag at step (3):

3) $$\frac{\partial}{\partial w_{ji}} net_{pk} = \frac{\partial}{\partial w_{ji}} \left( \sum_{j=1}^{J} w_{kj}\, o_{pj} + \theta_k \right)$$

$$= \frac{\partial}{\partial w_{ji}} \left( \sum_{j=1}^{J} w_{kj}\, o_{pj} \right)$$

$$= \, ?$$

Here, $o_{pj}$ is a function of $w_{ji}$. Furthermore, changing $w_{ji}$ will change every output in the output layer since the output $o_{pj}$ goes to every neuron in the ~~outp~~ output layer.

$\therefore$ Total error is $E_p = \sum_{k=1}^{K} \frac{1}{2} (t_{pk} - o_{pk})^2$

Start over.

1) $$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_{k=1}^{K} \frac{1}{2} (t_{pk} - o_{pk})^2$$

$$= \sum_{k=1}^{K} (t_{pk} - o_{pk}) \frac{\partial (-o_{pk})}{\partial w_{ji}}$$

$$= -\sum_{k=1}^{K} (t_{pk} - o_{pk}) \frac{\partial}{\partial w_{ji}} o_{pk}$$

2) $$\frac{\partial}{\partial w_{ji}} o_{pk} = f(net_{pk}) \left[ 1 - f(net_{pk}) \right] \frac{\partial}{\partial w_{ji}} net_{pk}$$ as before

Gradient Descent – BEP –
Chain rule and delta rule (cont.)


[?] May 1989
Neil E Cotter


3) $$\frac{\partial}{\partial w_{ji}} \text{net}_{pk} = \frac{\partial}{\partial w_{ji}} \left( \sum_{j=1}^{J} w_{kj}\, o_{pj} \right)$$

$$= \frac{\partial}{\partial w_{ji}} w_{kj}\, o_{pj}$$ • Picks out one term

$$= w_{kj} \frac{\partial}{\partial w_{ji}} o_{pj}$$ • $w_{kj}$ is <u>not</u> a function of $w_{ji}$. • $o_{pj}$ <u>is</u> a function of $w_j$

4) $$\frac{\partial}{\partial w_{ji}} o_{pj} = \frac{\partial}{\partial w_{ji}} f(\text{net}_{pj})$$

This has the same form as a previous derivation, and we get:

$$\frac{\partial}{\partial w_{ji}} o_{pj} = f(\text{net}_{pj})\left[1 - f(\text{net}_{pj})\right] o_{pi}$$

Putting the above results together, we have

Learning rule for neurons in next-to-last (penultimate) layer:

$$\Delta_p w_{ji} = +\eta \sum_{k=1}^{K} \Big\{ (t_{pk} - o_{pk})\, f(\text{net}_{pk})\left[1 - f(\text{net}_{pk})\right] w_{kj}$$
$$f(\text{net}_{pj})\left[1 - f(\text{net}_{pj})\right] o_{pi} \Big\}$$

$$= \eta\, o_{pi} \underbrace{f(\text{net}_{pj})\left[1 - f(\text{net}_{pj})\right] \sum_{k=1}^{K} (t_{pk} - o_{pk})\, f(\text{net}_{pk})\left[1 - f(\text{net}_{pk})\right] w_{kj}}_{\equiv\ \delta_{pj}}$$

- Same general form as output layer learning rule
- Notice that we have used the chain rule to propagate the effects of output errors back through the network. Hence the name "Backward Error Propagation".

Gradient Descent - BEP-
Chain rule and delta rule (cont.)

15 May 1989
Neil E Cotter

Now for the amazing result:

$$\delta_{pj} = f(net_{pj}) \left[1 - f(net_{pj})\right] \sum_{k=1}^{K} \delta_{pk} w_{kj}$$

Amazing? Yes. We can compute the $\delta$'s for this layer if we know the $\delta$'s for the next layer. So start at the output layer and work backwards, calculating $\delta$'s as you go.

Even more amazing is the similarity between the calculation of $\delta$'s and the calculations performed by the neural net. We have weighted sums of $\delta$'s just as we have weighted sums of inputs in the forward direction. The backward error propagation computation is nearly equivalent in complexity to the forward computation performed by the network.

Best of all, learning requires only information that a synapse might actually have access to:
$o_{pi}$, the incoming signal from the previous layer
$\delta_{pj}$, the $\delta_{pj}$ is the same for all synapses on this neuron (neuron $j$) and $\delta_{pj}$ is computed from $\delta_{pk}$ on the next layer and from $w_{kj}$ synapses connecting this neuron to the next layer.

By recursion we get the general delta rule for Backward Error Propagation:

$$\nabla_p w_{ji} = \eta \, \delta_{pj} \, o_{pi} \qquad \text{where} \quad \delta_{pj} = \sum_{k=1}^{K} \delta_{pk} w_{kj}$$

Sum over next layer.

$o_{pi}$ = previous layer's output is this layer's input

Gradient Descent – BEP –
Chain rule and delta rule (cont.)

6 May 1988
Neil E Cotter

Learning Rule

- *Backward Error Propagation*
- *Is a type of delta rule*

$$w_{ji}\text{ new} = w_{ji}\text{ old} + \underbrace{\Delta_p w_{ji}}_{\text{change in weight when input pattern } p \text{ is training vector}}$$

synapse connects neuron $i$ in previous layer to neuron $j$ in this layer

Simplistic $\delta$ rule

Use $\underbrace{\Delta_p w_{ji}}_{\text{change in weight}} = \eta \underbrace{(t_{pj} - o_{pj})}_{\substack{\text{desired out} \\ -\text{ actual out} \\ = \text{error} = \delta_{pj}}} o_{pi} \equiv \eta\ \delta_{pj}\ o_{pi}$

$o_{pi}$: input to neuron $j$ from previous layer neuron $i$

- *$\eta$ is a constant that controls step size in this gradient descent algorithm.*

Now $o_{pj} \propto w_{ji}\, o_{pi}$

- $\therefore$ if $o_{pi} > 0$ then larger $w_{ji} \Rightarrow$ more pos (more positive) $o_{pj}$
- i.e. if $o_{pi} > 0$ and $w_{ji}\uparrow$ then $o_{pj}\uparrow$

| In $o_{pi}$ | $\Delta w_{ji}$ | out $o_{pj}$ | error that gave $\Delta w_{ji}$: $t_{pj} - o_{pj}$ | result: $\lvert t_{pj} - o_{pj}\rvert$ |
|---|---|---|---|---|
| $>0$ | $\uparrow$ | $\uparrow$ | $>0$ | $\downarrow$ |
| $>0$ | $\downarrow$ | $\downarrow$ | $<0$ | $\downarrow$ |
| $<0$ | $\downarrow$ | $\uparrow$ | $>0$ | $\downarrow$ |
| $<0$ | $\uparrow$ | $\downarrow$ | $<0$ | $\downarrow$ |

- move this col here; makes more sense

$\lvert\text{error}\rvert$ reduced in every case

Generalized $\delta$ rule (*One actually used in Backward Error Prop*)

def: $E_p \equiv \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \equiv$ total squared error for pattern $p$ as input

$\sum$ over all output layer neurons

- *Remember that* $t_{pj} \equiv$ *desired (or target) output for neuron* $j$
  $o_{pj} \equiv$ *actual output for neuron* $j$

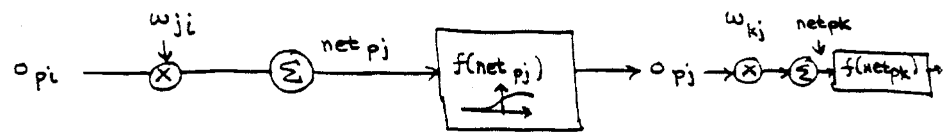
Then use $\delta_{pj} \equiv -\partial E_p / \partial \text{net}_{pj}$

Gradient Descent – BEP –
Chain rule and delta rule (cont.)

May 1988
*Neil E Cotter*

Calculation of $\delta_{pj}$

prelims:




• *Next layer (if $o_{pj}$ not output lay*

eqn's: $\text{net}_{pj} = \sum_i w_{ji} o_{pi} + \theta_j$

$$o_{pj} = f(\text{net}_{pj})$$

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2$$

$\delta_{pj}$ for output-layer neuron $j$

$$\delta_{pj} \equiv -\frac{\partial E_p}{\partial \text{net}_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial \text{net}_{pj}}$$

• *Use the chain rule for differentiation*

1) $$\frac{\partial E_p}{\partial o_{pj}} = \frac{\partial}{\partial o_{pj}} \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \overset{?}{=} \sum_j \cancel{\frac{\partial}{\partial o_{pj}} \frac{1}{2} (t_{pj} - o_{pj})^2}$$

• *Be careful*

$$= \frac{1}{2} 2 (t_{pj} - o_{pj})(-1)$$

$$= -(t_{pj} - o_{pj})$$

• *The only term in the $\Sigma$ th isn't constant relative to $o_{pj}$ is the $j^{th}$ term.*

• *Note the difference between $o_{pj}$ inside and outside the $\Sigma$. Inside the $\Sigma$ the $j$ is a dummy variable (similar to dummy variables of integra*

2) $$\frac{\partial o_{pj}}{\partial \text{net}_{pj}} = \frac{\partial f(\text{net}_{pj})}{\partial \text{net}_{pj}} = f'\Big|_{\text{net}_{pj}} = f(\text{net}_{pj})(1 - f(\text{net}_{pj}))$$

$$= o_{pj}(1 - o_{pj})$$

• *Recall that for $f(u) = 1/(1+e^{-u})$, $\partial f/\partial u = f(1-f)$*

∴ Output layer learning rule:

$$\Delta_p w_{ji} = \eta\, \delta_{pj}\, o_{pi} = \eta \underbrace{(t_{pj} - o_{pj})\, o_{pj}(1 - o_{pj})}_{\delta_{pj}} o_{pi}$$