Gradient Descent – BEP – Algorithm

7 Mar 1990 ~~Logistic Networks~~ Backward Error Propagation

Neil E Cotter

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$ (Gradient Descent

where $\Delta w_{ji} \equiv$ change in weight w_{ji}

$\eta \equiv$ learning rate or step size

$E \equiv$ output error $\frac{1}{2}(t_k - o_k)^2$

Equivalent to:

$$\Delta w_{ji} = \eta \, \delta_j \, o_i$$ (Delta Rule)

where $\Delta w_{ji} \equiv$ change in weight w_{ji}

$\eta \equiv$ learning rate or step size

$o_i \equiv$ input to synapse w_{ji} (output of neuron i

$$\delta_j \equiv f'(\text{net}_j) \sum_k \delta_k w_{kj} = o_j(1-o_j) \sum_k \delta_k w_{kj}$$

$o_j \equiv$ output of j

$\delta_k \equiv \delta$ for next layer downstream, neuron k

$w_{kj} \equiv$ synaptic weights from neuron j to neuron k in next layer

δ's defined recursively ~~working~~ (propagating) backward from error at output layer:

$$\delta_k = (t_k - o_k) f'(\text{net}_k)$$ for output layer

$$\delta_j = f'(\text{net}_j) \sum_k \delta_k w_{kj}$$ for penultimate (n· to last) layer j

$$\delta_i = f'(\text{net}_i) \sum_j \delta_j w_{ji}$$ for second to (c layer i

We usually have only three layers.

$$\boxed{\text{Backward Error Propagation} = \text{Delta Rule} = \text{Gradient Descen}}$$

δ's come from application of chain rule for derivative $\frac{\partial E}{\partial w}$.

Apr 1990 Gradient Descent — Backward Error Propagation (BEP) – Algorithm (cont.)

Neil E Cotter

Error in output is $E = \frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2$

Last Layer Update Rule:

$$\xcancel{\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} = -\eta \frac{\partial}{\partial w_{kj}} \sum_{k=1}^{K} \frac{(t_k - o_k)^2}{2}}$$

Bad notation because k in w_{kj} easily confused with k in $\sum_k$. My solution is to use $\boldsymbol{k}$ for the particular synaptic weight whose update rule we are finding. PDP book uses same k for both and is confusi. Same story for $\boldsymbol{j}$ vs j.

$$\Delta w_{\boldsymbol{k}\boldsymbol{j}} = -\eta \frac{\partial E}{\partial w_{\boldsymbol{k}\boldsymbol{j}}}$$

$$= -\eta \frac{\partial}{\partial w_{\boldsymbol{k}\boldsymbol{j}}} \frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2$$

$$= +\eta \frac{1}{2} \sum_{k=1}^{K} 2(t_k - o_k) \frac{\partial o_k}{\partial w_{\boldsymbol{k}\boldsymbol{j}}}$$

Now $\frac{\partial o_k}{\partial w_{\boldsymbol{k}\boldsymbol{j}}}$ is the change in output o_k when weight $w_{\boldsymbol{k}\boldsymbol{j}}$ changes.

But o_k will not change unless $w_{\boldsymbol{k}\boldsymbol{j}}$ is in neuron k.

$$\therefore \frac{\partial o_k}{\partial w_{\boldsymbol{k}\boldsymbol{j}}} = 0 \quad \text{unless} \quad k = \boldsymbol{k}$$

So we only get one term in the sum:

$$\Delta w_{\boldsymbol{k}\boldsymbol{j}} = +\eta (t_{\boldsymbol{k}} - o_{\boldsymbol{k}}) \frac{\partial o_{\boldsymbol{k}}}{\partial w_{\boldsymbol{k}\boldsymbol{j}}}$$

Now find $\frac{\partial o_{\boldsymbol{k}}}{\partial w_{\boldsymbol{k}\boldsymbol{j}}}$:

20 Apr 1990 Gradient Descent — BEP–algorithm (cont.)

Neil E Cotter

$$\frac{\partial o_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} f(net_k)$$

$$= \frac{\partial f(net_k)}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$

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

logistic sigmoid $f' = f(1-f)$

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_{j=1}^{J} w_{kj}\, o_j$$

o_j is input to w_{kj} synapse

Again we only get the term where $j = j$:

$$= \frac{\partial}{\partial w_{kj}} w_{kj}\, o_j$$

$$= o_j$$

Putting it all together:

$$\Delta w_{kj} = \eta \underbrace{(t_k - o_k)\ \overset{o_k}{\overset{\shortparallel}{f(net_k)}}\ \overset{1-o_k}{\overset{\shortparallel}{\left[1 - f(net_k)\right]}}}_{\equiv\ \delta_k}\ o_j$$

We define δ_k as shown, and we call our weight update rule a "delta rule." We shall find that weight updates for any layer can be written as a delta rule:

$$\Delta w_{kj} = \eta\, \delta_k\, o_j$$

Better yet, δ's for previous layers are defined recursively in terms of δ's for later layers, (consequence of chain rule for der

Penultimate Layer Update Rule:
(Next-to-last)

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

$$= -\eta \frac{\partial}{\partial w_{ji}} \frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2$$

Since the output of neuron o_j goes to every neuron on the last layer, w_{ji} affects every o_k and we get all the terms in the $\sum_k$:

$$= \eta \sum_{k=1}^{K} (t_k - o_k) \frac{\partial o_k}{\partial w_{ji}}$$

Now we repeatedly apply the chain rule to $\frac{\partial o_k}{\partial w_{ji}}$. The first few steps are the same as for the last layer update, and we can immediately write down:

$$\frac{\partial o_k}{\partial w_{ji}} = f(net_k)\left[1 - f(net_k)\right] \frac{\partial\, net_k}{\partial w_{ji}}$$

$$\frac{\partial\, net_k}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_{j=1}^{J} w_{kj}\, o_j$$

Since w_{ji} affects only o_j we get only the o_j term:

$$= \frac{\partial}{\partial w_{ji}} w_{kj}\, o_j$$

$$= w_{kj} \frac{\partial o_j}{\partial w_{ji}}$$

This is similar to $\partial o_k / \partial w_{kj}$, and we get:

Apr 1990 Gradient Descent — BEP-algorithm (cont.)
Neil E Cotter

$$\frac{\partial o_j}{\partial w_{ji}} = f(net_j)\left[1 - f(net_j)\right] o_i$$

Putting it all together:

$$\Delta w_{ji} = \eta \underbrace{\sum_{k=1}^{K} \underbrace{(t_k - o_k)\, f(net_k)\left[1 - f(net_k)\right]}_{\delta_k} w_{kj}\, f(net_j)\left[1 - f(net_j)\right]}_{\overset{|||}{\delta_j}}$$

Notice that δ_k is embedded in the update rule. We can write:

$$\Delta w_{ji} = \eta \underbrace{\sum_{k=1}^{K} \delta_k w_{kj}\, \overset{o_j}{\overset{\shortparallel}{f(net_j)}}\left[\overset{1-o_j}{\overset{\shortparallel}{1 - f(net_j)}}\right]}_{\delta_j} o_i$$

$$= \eta \qquad \delta_j \qquad o_i$$

$$= \eta\, \delta_j\, o_i$$

So we have a delta rule once again. The δ_j is defined in terms of δ_k and synaptic weights from neuron j to neuron k:

$$\delta_j = \overset{o_j}{\overset{\shortparallel}{f(net_j)}}\left[\overset{1-o_j}{\overset{\shortparallel}{1 - f(net_j)}}\right] \sum_{k=1}^{K} \delta_k w_{kj}$$

(We can take $f(1-f)$ out of sum since it does not depend on k.)

Previous Layer Update Rule:

We discover by calculation that the update rule for any layer can be written in terms of δ's for the layer after it.

The form of result for Δw_{ji} carries over to all previous layers. In particular, for Δw_{ih} we obtain:

$$\Delta w_{ih} = \eta\, \delta_i\, i_h$$

We use i_h rather than o_h because we are on the first layer of our network, and the inputs to our neuron do not actually come from a previous layer. (Just notation.)

where $$\delta_i \equiv \underbrace{f(\text{net}_i)}_{=o_i}\left[\underbrace{1 - f(\text{net}_i)}_{=1-o_i}\right] \sum_{j=1}^{J} \delta_j\, w_{ji}$$

Now we see where the name "backward-error-propagation" comes from: we start with the output error, $t_K - o_K$, and propagate it back through the network via the recursive definition of the δ's.

The computation of the update rule is very similar in form to the neural network computation. Hence, the complexity of weight updates is roughly the same as the complexity of the network computation. Learning slows us down by about half.