

Error backpropagation

goal: compute 1st & 2nd derivatives of  $E(\vec{w})$  [wrt  $\vec{w}$ ] efficiently.

Consider a general NN with arbitrary feed-forward topology, arbitrary activation f's and a general error f'n:

$$E(\vec{w}) = \sum_{n=1}^N E_n(\vec{w})$$

Consider first a linear model:

$$\begin{cases} y_k = \sum_i w_{ki} x_i, \\ E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2. \end{cases}$$

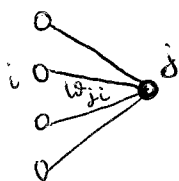
"  $y_k(\vec{x}_n, \vec{w})$  "

Then 
$$\frac{\partial E_n}{\partial w_{ji}} = \sum_k (y_{nk} - t_{nk}) \underbrace{\frac{\partial y_{nk}}{\partial w_{ji}}}_{\delta_{kj} x_{ni}} =$$

$$= \underbrace{(y_{nj} - t_{nj})}_{\text{"error signal"}} \underbrace{x_{ni}}_{\text{"input"}}$$

More generally,

$$a_j = \sum_i w_{ji} z_i$$



activation of unit  $i$  connected to unit  $j$

Then  $z_j = h(a_j)$   
 $\uparrow$  activation of unit  $j$

Now consider  $\frac{\partial E_n}{\partial w_{ji}} = \underbrace{\frac{\partial E_n}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$  (no sum!)  
 $\ominus \delta_j z_i$

$\delta_j z_i$   
 ( $z_i = 1$  if we are dealing with bias)

If  $j$  is an output unit,

$\delta_j = \frac{\partial E_n}{\partial a_j} = y_j - t_j$  for regression OR classification  
 $n$  index omitted for simplicity

Indeed, for regression

$y_k = a_k$  (unit activation function)

$\hookrightarrow \frac{\partial E}{\partial a_k} = y_k - t_k$

For  $k=2$  classification,

$\frac{\partial E_n}{\partial a_j} = - \left[ \frac{t_j}{y_j} \frac{\partial y_j}{\partial a_j} + (1-t_j)(-1) \frac{1}{1-y_j} \frac{\partial y_j}{\partial a_j} \right] \ominus$

$\rightarrow y_j(1-y_j)$

$y_j = \sigma(a_j)$ , sigmoid

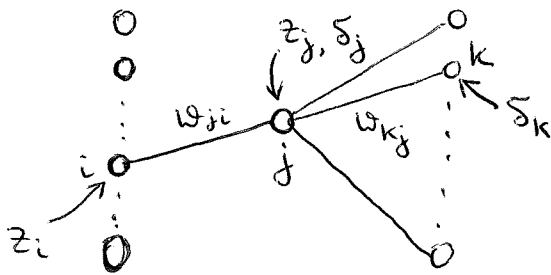
$\ominus - [t_j(1-y_j) - (1-t_j)y_j] = y_j - t_j$

Same for  $N$   $K=2$  classif's and  $K>2$  classif'n.

If  $j$  is a hidden unit,

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad \diamond$$

↑  
over all "future" units, hidden or output, to which  $j$  is connected



$$\begin{aligned} \diamond \sum_k \delta_k \frac{\partial}{\partial a_j} \left( \sum_i w_{ki} h(a_i) \right) &= \sum_{k,i} \delta_k w_{ki} \underbrace{\frac{\partial h(a_i)}{\partial a_j}}_{h'(a_j) \delta_{ij}} = \\ &= h'(a_j) \sum_k w_{kj} \delta_k. \quad (*) \end{aligned}$$

We can use (\*) by starting from output units (for which  $\delta_j$  is easily computed) and computing  $\delta_j$ 's for hidden units in a backpropagation pass.

Finally, use  $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$  to compute all gradients.

If needed, finish with

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}$$

The above derivation can be easily generalized to several types of activation functions  $h(\cdot)$ .

Ex. Consider a two-layer network as before, and focus on regression:

output units  $\Rightarrow y_k = a_k$

hidden units  $\Rightarrow h(a_k) = \tanh(a_k)$

Note that  $\frac{d \tanh(a)}{da} = 1 - \tanh^2(a)$

For pattern  $n$ ,  $E_n = \frac{1}{2} \sum_{k=1}^K (y_k - t_k)^2$

Forward pass:

$$\left\{ \begin{array}{l} a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i \\ z_j = \tanh(a_j) \\ y_k = \sum_{j=0}^M w_{kj}^{(2)} z_j \end{array} \right.$$

$\uparrow$  inputs

$\leftarrow$  # hidden units

Next, compute  $\delta_k = y_k - t_k$

Use (\*) to find

$$\delta_j = (1 - z_j^2) \sum_{k=1}^K w_{kj}^{(2)} \delta_k \quad \text{for hidden units}$$

Finally, compute

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j \quad (**)$$

and sum over  $n$ .

Note that backpropagation is  $\Theta(w)$ , where  $w$  is the total # weights & biases. Most effort goes into evaluating

$$a_j = \sum_i w_{ji} z_i$$

Forward propagation (computing  $E_n$ ) is  $\Theta(w)$  as well. Note that finite difference:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \frac{\epsilon}{2}) - E_n(w_{ji} - \frac{\epsilon}{2})}{\epsilon} + \mathcal{O}(\epsilon^2)$$

is  $\Theta(w^2)$  since each  $E_n$  computation is  $\Theta(w)$  & one needs  $w$  of those. However, it can be used for numerical checks.

We can check these results by evaluating derivatives of  $E_n$  without using backpropagation:

$$E_n = \frac{1}{2} \sum_{k=1}^K \underbrace{(y_k - t_k)^2}_{\text{index } n \text{ omitted for clarity}}, \text{ where}$$

$$y_k = \sum_{j=0}^M \omega_{kj}^{(2)} \tanh\left(\sum_{i=0}^D \omega_{ji}^{(1)} x_i\right).$$

$$\text{Then } \frac{\partial E_n}{\partial \omega_{k'j'}^{(2)}} = \sum_{k=1}^K (y_k - t_k) \frac{\partial y_k}{\partial \omega_{k'j'}^{(2)}} =$$

$$= \sum_{k=1}^K (y_k - t_k) \underbrace{\sum_{j=0}^M \frac{\partial \omega_{kj}^{(2)}}{\partial \omega_{k'j'}^{(2)}}}_{\delta_{kk'} \delta_{jj'}} \underbrace{\tanh\left(\sum_{i=0}^D \omega_{ji}^{(1)} x_i\right)}_{z_j} \quad (\ominus)$$

$$\ominus \underbrace{(y_{k'} - t_{k'})}_{\delta_{k'}} z_{j'} = \delta_{k'} z_{j'}, \text{ same as } (**)$$

$$\text{Next, } \frac{\partial E_n}{\partial \omega_{j'i'}^{(1)}} = \sum_{k=1}^K (y_k - t_k) \frac{\partial y_k}{\partial \omega_{j'i'}^{(1)}} =$$

$$= \sum_{k=1}^K \underbrace{(y_k - t_k)}_{\delta_k} \sum_{j=0}^M \omega_{kj}^{(2)} \underbrace{\tanh'(a_j)}_{1 - z_j^2} \sum_{i=0}^D x_i \underbrace{\frac{\partial \omega_{ji}^{(1)}}{\partial \omega_{j'i'}^{(1)}}}_{\delta_{jj'} \delta_{ii'}} \quad (\boxminus)$$

$$\equiv \sum_{k=1}^K \delta_k \omega_{kj}^{(2)} (1 - z_j^2) x_{ij} =$$

$$= x_{ij} (1 - z_j^2) \underbrace{\sum_{k=1}^K \delta_k \omega_{kj}^{(2)}}_{\delta_j} = \delta_j x_{ij}, \text{ same as (**)}$$

## Jacobian matrix

$J_{ki} = \frac{\partial y_k}{\partial x_i}$  local sensitivity of outputs to changes in inputs

$$\Delta y_k \approx \sum_i \underbrace{\frac{\partial y_k}{\partial x_i}}_{J_{ki}} \underbrace{\Delta x_i}_{\text{small input perturbations}} \quad \left| \frac{\Delta x_i}{x_i} \right| \ll 1$$

$J_{ki}$  can be evaluated using backpropagation:

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} \underbrace{\frac{\partial a_j}{\partial x_i}}_{w_{ji}} \quad (*)$$

↑  
over all "future" units  $j$   
connected to unit  $i$

Further,  $\frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial a_l}{\partial a_j} \quad \textcircled{=}$

↑  
over all "future" units  $l$  connected  
to unit  $j$

$$\textcircled{=} \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial}{\partial a_j} \sum_i w_{li} \underbrace{h(a_i)}_{z_i} =$$

$$= \sum_{l,i} \frac{\partial y_k}{\partial a_l} w_{li} h'(a_i) \delta_{ij} = h'(a_j) \sum_l w_{lj} \frac{\partial y_k}{\partial a_l}$$

Start from output units, e.g.  
with sigmoid activation functions:



$$\frac{\partial y_k}{\partial a_j} = \sigma'(a_k) \underbrace{\frac{\partial a_k}{\partial a_j}}_{\delta_{kj}} = \sigma(a_j) (1 - \sigma(a_j)) \delta_{kj}$$

Algorithm:

1. Choose  $\vec{x}_n$  and forward-propagate
2. For each  $k$ , start with output units and backpropagate  $\Rightarrow$  find  $\frac{\partial y_k}{\partial a_j}$ ,  $\forall j$
3. Find  $J_{ki}$  using (\*)

Can be checked against finite difference:

$$\frac{\partial y_k}{\partial x_i} = \frac{y_k(x_i + \epsilon) - y_k(x_i - \epsilon)}{2\epsilon} + O(\epsilon^2)$$

needs  $2D$  forward propagations ( $D = \#$  inputs)

Hessian matrix

$$H_{ij} = \frac{\partial^2 E}{\partial w_{\bullet i} \partial w_{\bullet j}}$$

$$i, j = 1, \dots, W$$

↑ total # weights & biases  
 all weights/biases relabeled using one consecutive index.

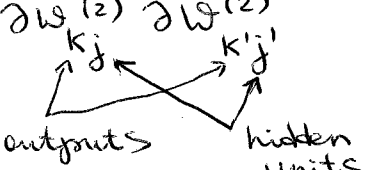
- (1)  $H_{ij}$  used in some non-linear optimization algorithms
- (2)  $H_{ij}$  is used in Bayesian neural networks

Exact evaluation:

Consider a two-layer network for simplicity.

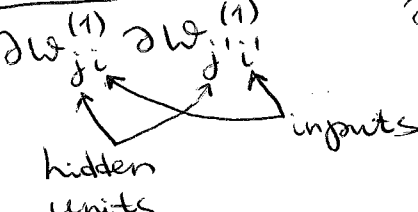
Define  $\delta_k = \frac{\partial E_n}{\partial a_k}$ ,  $M_{kk'} = \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}}$   
 as before here,  $k$  &  $k'$  can be ~~hidden~~ hidden or output nodes  
 (hidden)

(a) Both weights in the second layer:

$$\frac{\partial^2 E_n}{\partial w_{kj}^{(2)} \partial w_{k'j'}^{(2)}} = \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} \frac{\partial a_k}{\partial w_{kj}^{(2)}} \frac{\partial a_{k'}}{\partial w_{k'j'}^{(2)}} =$$


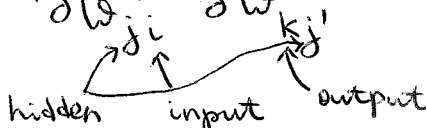
$$= M_{kk'} z_j z_{j'}$$

(b) Both weights in the first (input) layer:

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{j'i'}^{(1)}} = \frac{\partial^2 E_n}{\partial a_j \partial a_{j'}} \frac{\partial a_j}{\partial w_{ji}^{(1)}} \frac{\partial a_{j'}}{\partial w_{j'i'}^{(1)}} \equiv$$


$$\equiv M_{jj'} x_i x_{i'}$$

(c) One weight in each layer:

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{kj'}^{(2)}} = \frac{\partial}{\partial w_{ji}^{(1)}} \left[ \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{kj'}^{(2)}} \right] =$$


$$= \frac{\partial^2 E_n}{\partial a_j \partial a_k} x_i z_{j'} + \frac{\partial E_n}{\partial a_k} \frac{\partial z_{j'}}{\partial w_{ji}^{(1)}} \equiv$$

$$\equiv M_{jk} x_i z_{j'} + \delta_k h'(a_j) \delta_{jj'} x_i$$

Thus, we need to backpropagate  $M_{kk'}$ .

Indeed, 
$$\frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \underbrace{\frac{\partial a_k}{\partial a_j}}_{\substack{\text{downstream of} \\ \text{\& connected to } j}} = h'(a_j) \sum_k \frac{\partial E_n}{\partial a_k} w_{kj}$$

$$\sum_l w_{kl} h'(a_l) \frac{\partial a_l}{\partial a_j} = \delta_{lj}$$

$$= w_{kj} h'(a_j).$$

Now, 
$$\frac{\partial^2 E_n}{\partial a_i \partial a_j} = h''(a_i) \delta_{ij} \sum_k \frac{\partial E_n}{\partial a_k} w_{kj} +$$

$$+ h'(a_j) \sum_{k, k'} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} w_{kj} \frac{\partial a_{k'}}{\partial a_i} \quad \text{⊖}$$

$$\sum_l h'(a_l) \delta_{li} w_{kl} = h'(a_i) w_{k'i}$$

⊖ 
$$h''(a_i) \delta_{ij} \sum_k \frac{\partial E_n}{\partial a_k} w_{kj} + h'(a_i) h'(a_j) \sum_{k, k'} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} w_{kj} w_{k'i}$$

$$\delta_k \quad (**)$$

$$M_{kk'}$$

Compute  $M_{ij}$  recursively starting from output nodes. For ex., for regression we get:

output nodes 
$$\begin{cases} \frac{\partial E_n}{\partial a_j} = \frac{\partial E_n}{\partial y_j} = y_j - t_j, \\ \frac{\partial^2 E_n}{\partial a_i \partial a_j} = \delta_{ij}. \end{cases}$$

$$a_i = y_i \text{ here}$$

Then we can compute  $M_{ij}$  for hidden nodes using (\*\*).

Finally, in the "mixed" case:

$$\frac{\partial}{\partial a_i} \frac{\partial E_n}{\partial y_j} = \frac{\partial}{\partial a_i} (y_j - t_j) = \sum_k \frac{\partial (y_j - t_j)}{\partial a_k} \frac{\partial a_k}{\partial a_i} \quad \textcircled{=}$$

↑ hidden     ↑ output     ↑ output     ↑  $y_k$      ↑  $w_{ki} h'(a_i)$

$$\textcircled{=} \sum_k \delta_{jk} w_{ki} h'(a_i) = \underline{\underline{h'(a_i) w_{ji}}}$$

Thus we can compute all  $M_{ij}$  & therefore all  $H_{ij}$ . This is an  $\mathcal{O}(W^2)$  computation.

0  
This can be checked against finite differences:

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{ek}} = \frac{1}{4\epsilon^2} \left[ E(w_{ji} + \epsilon, w_{ek} + \epsilon) - E(w_{ji} + \epsilon, w_{ek} - \epsilon) - E(w_{ji} - \epsilon, w_{ek} + \epsilon) + E(w_{ji} - \epsilon, w_{ek} - \epsilon) \right] + \mathcal{O}(\epsilon^2)$$

↑ 4 forward propagations with  $\mathcal{O}(W)$  operations each  $\times W^2$  Hessian elements  $\Rightarrow$

$\Rightarrow \mathcal{O}(W^3)$  computation.

However, we can use a mixed approach:

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{ek}} = \frac{1}{2\epsilon} \left[ \frac{\partial E}{\partial w_{ji}} \Big|_{w_{ek} + \epsilon} - \frac{\partial E}{\partial w_{ji}} \Big|_{w_{ek} - \epsilon} \right] + \mathcal{O}(\epsilon^2)$$

compute using backpropagation in  $\mathcal{O}(W)$  steps  $\times W$  weights to be perturbed ( $w_{ek} \pm \epsilon$ ):  $\mathcal{O}(W^2)$  computation just as with explicit backpropagation above.