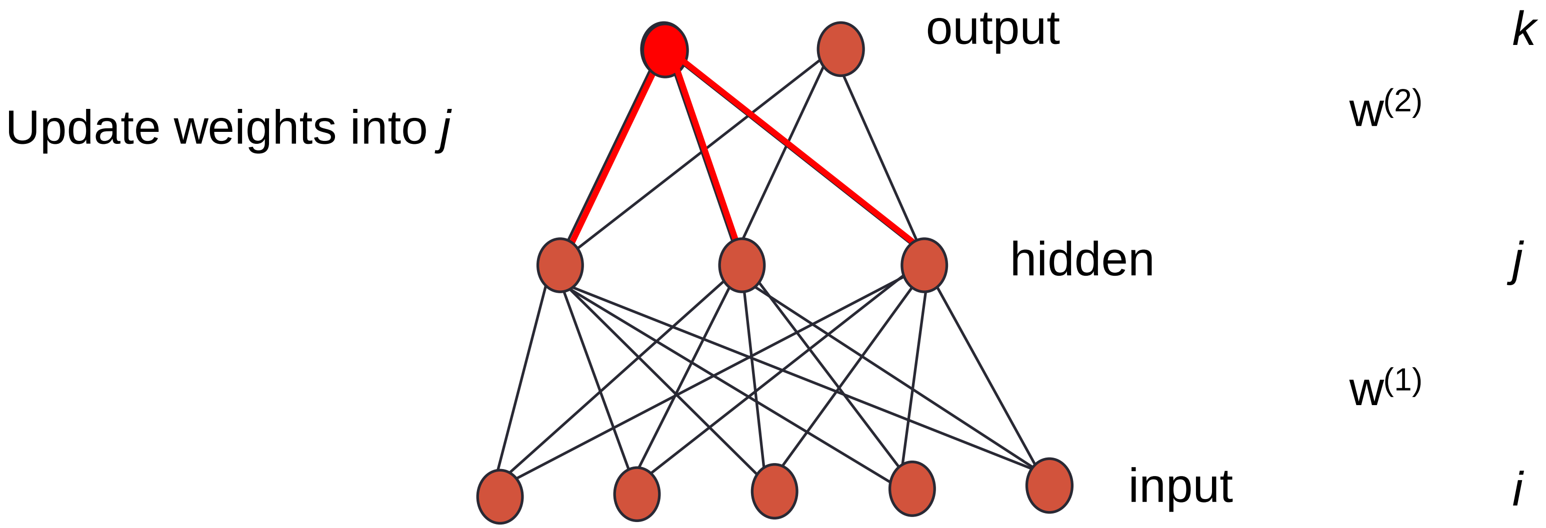
NEURAL NETWORK BACKPROPAGATION

Backpropagation

- A fast algorithm for computing gradients
- Introduced in 1970s
- Based on partial derivatives (Remember the Chain Rule from calculus?)
- How changing the weights and biases changes the overall behaviour of the network ~ Learning

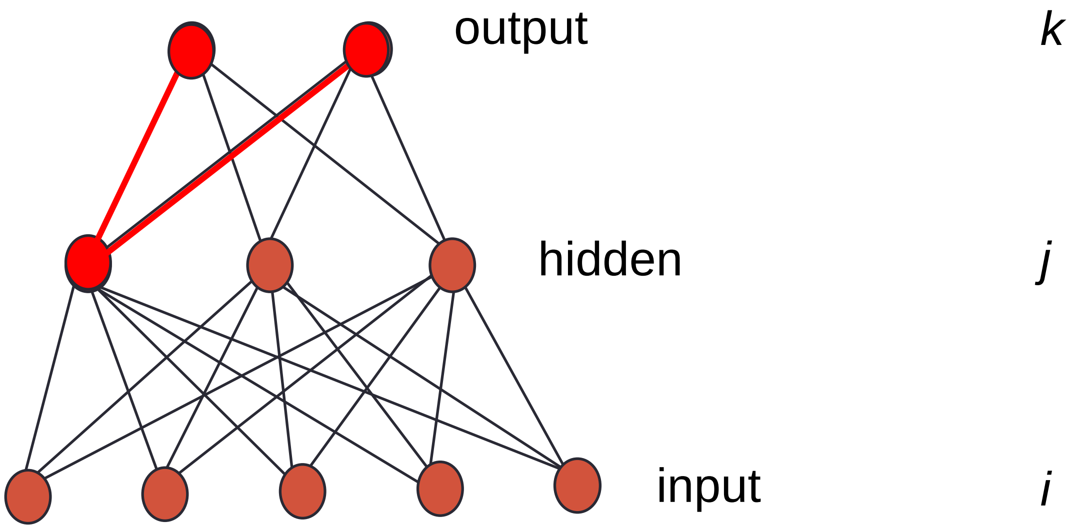
Backpropagation: Graphic example

- First calculate error of output units and use this to change the top layer of weights.



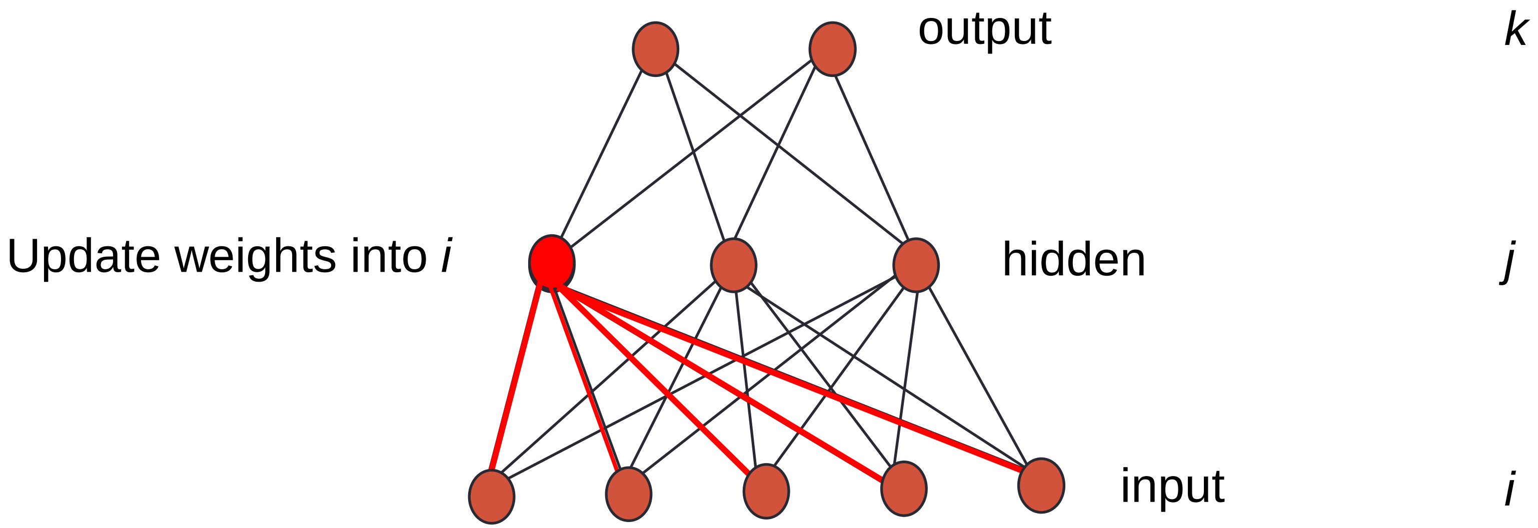
Backpropagation: Graphic example

- Next calculate error for hidden units based on errors on the output units it feeds into.



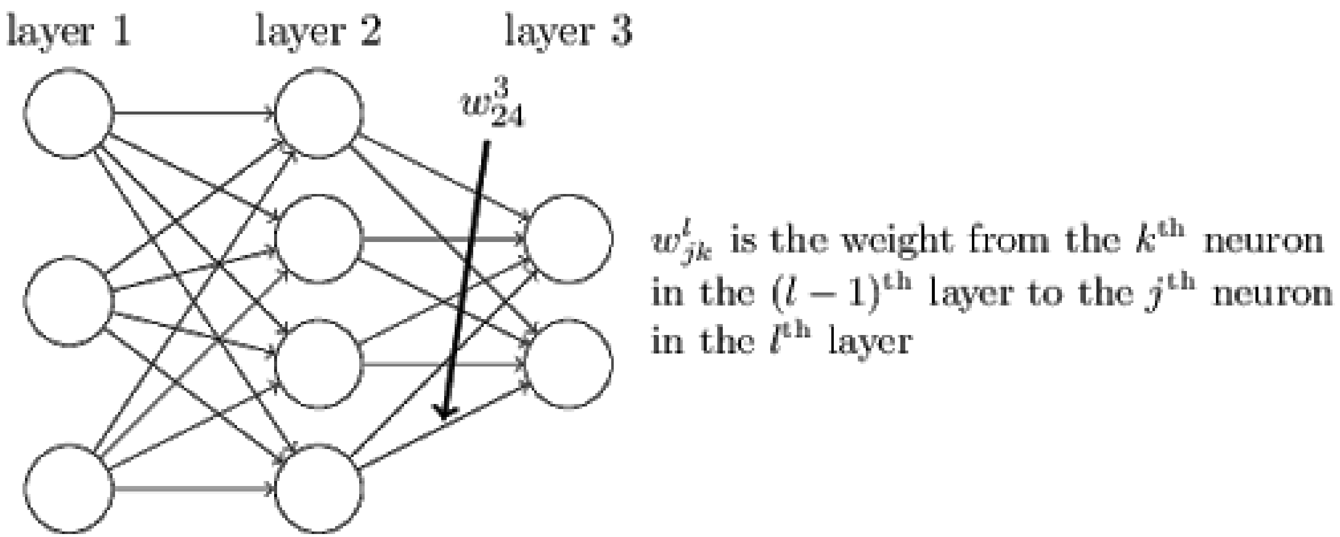
Backpropagation: Graphic example

- Finally update bottom layer of weights based on errors calculated for hidden units.



Backpropagation

- Notation: w^l_{jk} to denote the weight for the connection from the k^{th} neuron in the $(l\ -1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.



Backpropagation

- Based on this notation activation of the j^th neuron in the l^th layer can be written as:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

- In vectorized form

$$a^l = \sigma\left(w^l a^{l-1} + b^l\right)$$

- For simplicity, say

$$z^l = w^l a^{l-1} + b^l \qquad a^l = \sigma\left(z^l\right)$$

Backpropagation - Assumptions

- The cost function can be written as an average over the cost functions of individual training examples:

$$C = \frac{1}{n}\sum_{i} C^{i}$$

- The cost function can be written as a function of the outputs from the NN

$$C(w,b) = \frac{1}{2n}\sum_{i}(y - a(w,b))^{2}$$

Backpropagation

- Backpropagation is about understanding how changing the weights and biases in a network changes the cost function. Ultimately, this means computing the partial derivatives

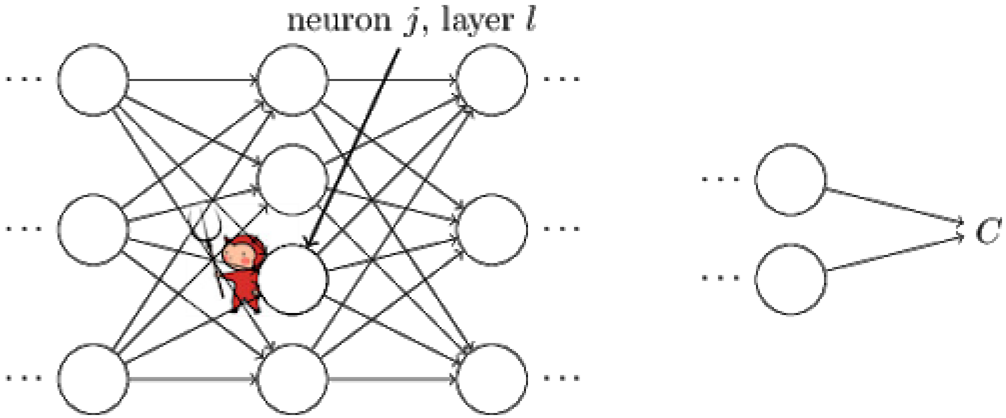
$$\frac{\partial C}{\partial w_{jk}^{l}}, \frac{\partial C}{\partial b_{j}^{l}}$$

- To do that we introduce an error notation, error in the j^{th} neuron in the l^{th} layer

$$\delta_{j}^{l}$$

Backpropagation

- Demon introduces an error to input: Δz_j^l
- Changes the output to: $\sigma(z_j^l + \Delta z_j^l)$
- Resulting cost change: $\dfrac{\partial C}{\partial z_j^l}\Delta z_j^l$



Backpropagation

- Resulting cost change: $$\frac{\partial C}{\partial z_j^l} \Delta z_j^l$$
- Error of neuron: $$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$
- Error of the layer l: $$\delta^l$$

Backpropagation

- Having introduced the notation, we are going backwards from the output layer, L:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}$$

$$a^l = \sigma\left(z^l\right)$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

Backpropagation

- If we write everything in the matrix form:

$$\boxed{\delta^L = \nabla_a C \odot \sigma'(z^L)}$$

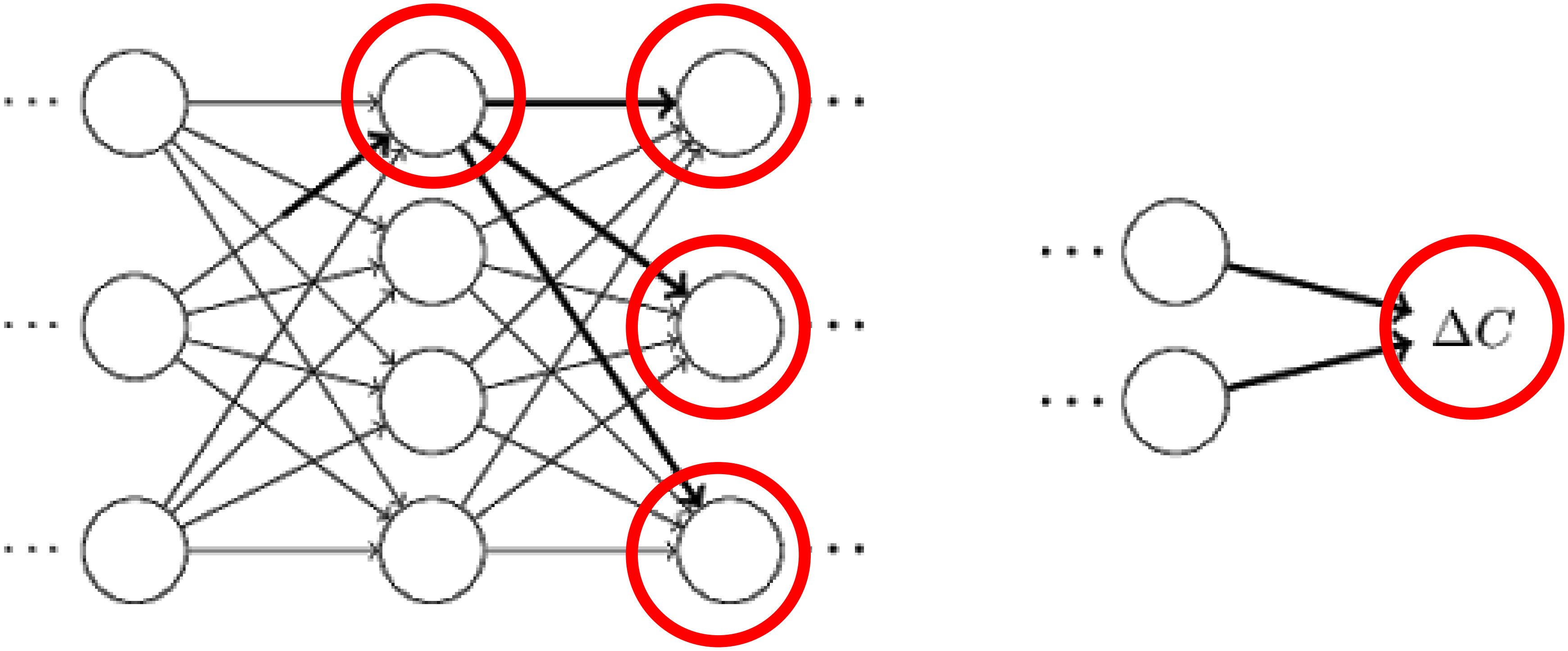
$$C = \frac{1}{2}(y - a^L)^2$$

$$\nabla_a C = (a^L - y)$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

Backpropagation

- Chain effect:



Backpropagation

- Backward updating

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$\delta_j^l = \sum_k \frac{\partial z_j^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

Backpropagation

- Backward updating

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

$$\frac{\partial z_j^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

$$\delta_j^l = \sum_k \frac{\partial z_j^{l+1}}{\partial z_j^l} \delta_k^{l+1} = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

$$\boxed{\delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)}$$

Backpropagation

- Backward updating

$$z^l = w^l a^{l-1} + b^l$$

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$\boxed{\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l}$$

$$\boxed{\begin{aligned}\frac{\partial C}{\partial w_{jk}^l} &= \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l}\\ &= \delta_j^l a_k^{l-1}\end{aligned}}$$

Backpropagation

- Summary

$$\boxed{\delta^L = \nabla_a C \odot \sigma'(z^L)}$$

$$\boxed{\delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)}$$

$$\boxed{\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l}$$

$$\boxed{\begin{aligned}\frac{\partial C}{\partial w_{jk}^l} &= \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \\ &= \delta_j^l a_k^{l-1}\end{aligned}}$$

Backpropagation

1. Input a set of training examples
2. For each training example x: Set the corresponding input activation $a^{x,1}$, and perform the following steps:
   - Feedforward: For each l= $2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   - Output error $\delta^{x,L}$: Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - Backpropagate the error: For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.
3. Gradient descent: For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

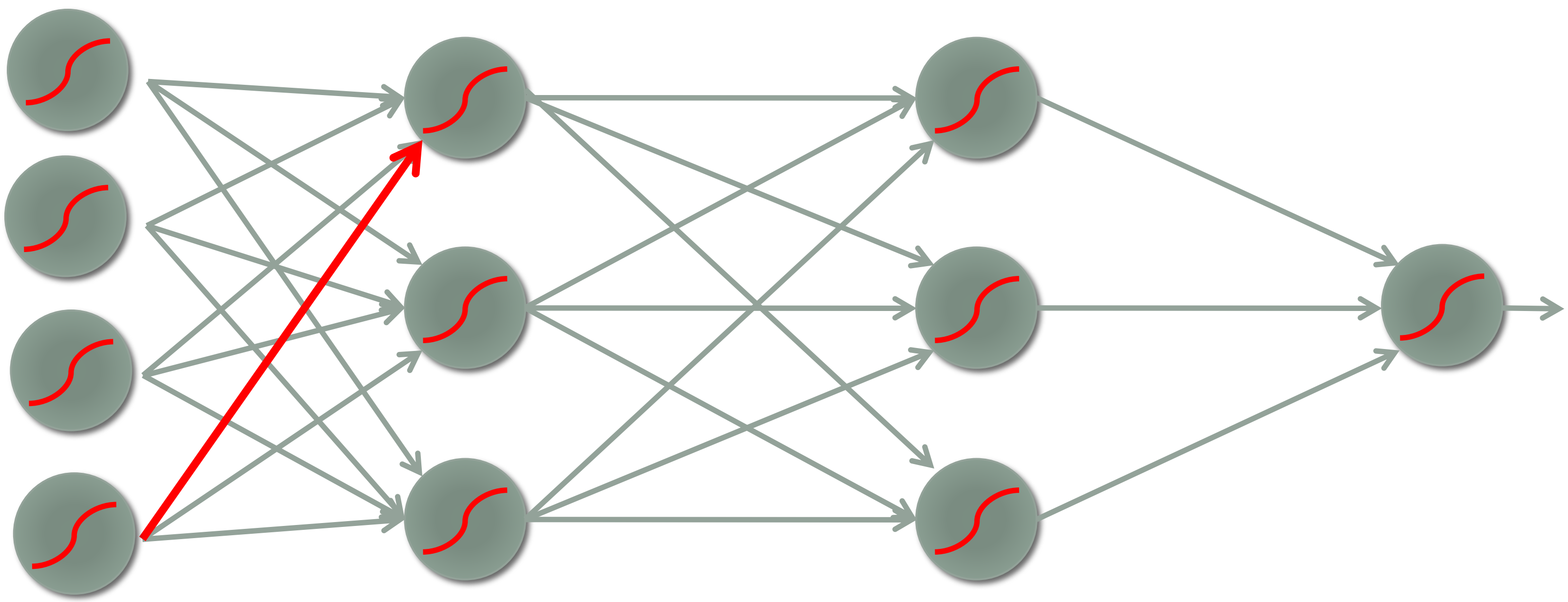
Backpropagation

- Why do we prefer backpropagation:

$$\frac{\partial C}{\partial w} = \lim_{h \to 0} \frac{C(w+h) - C(w)}{h}$$

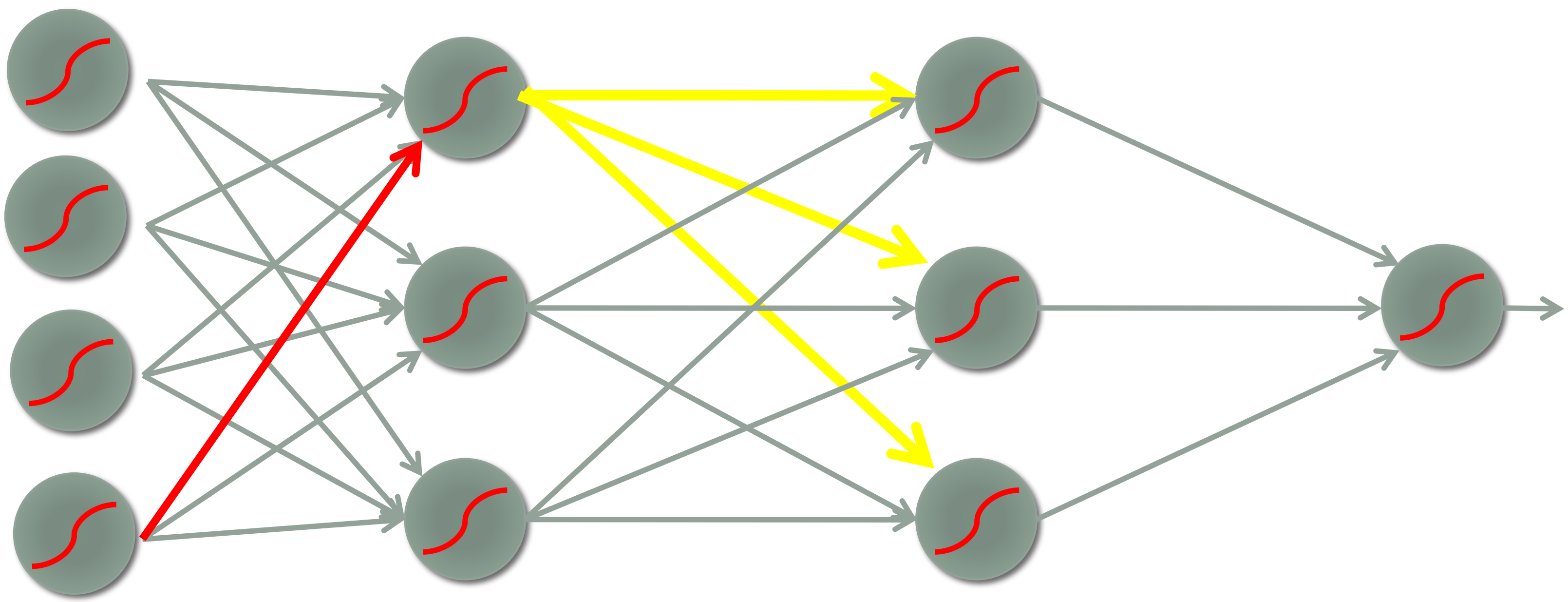
Backpropagation

- Network view



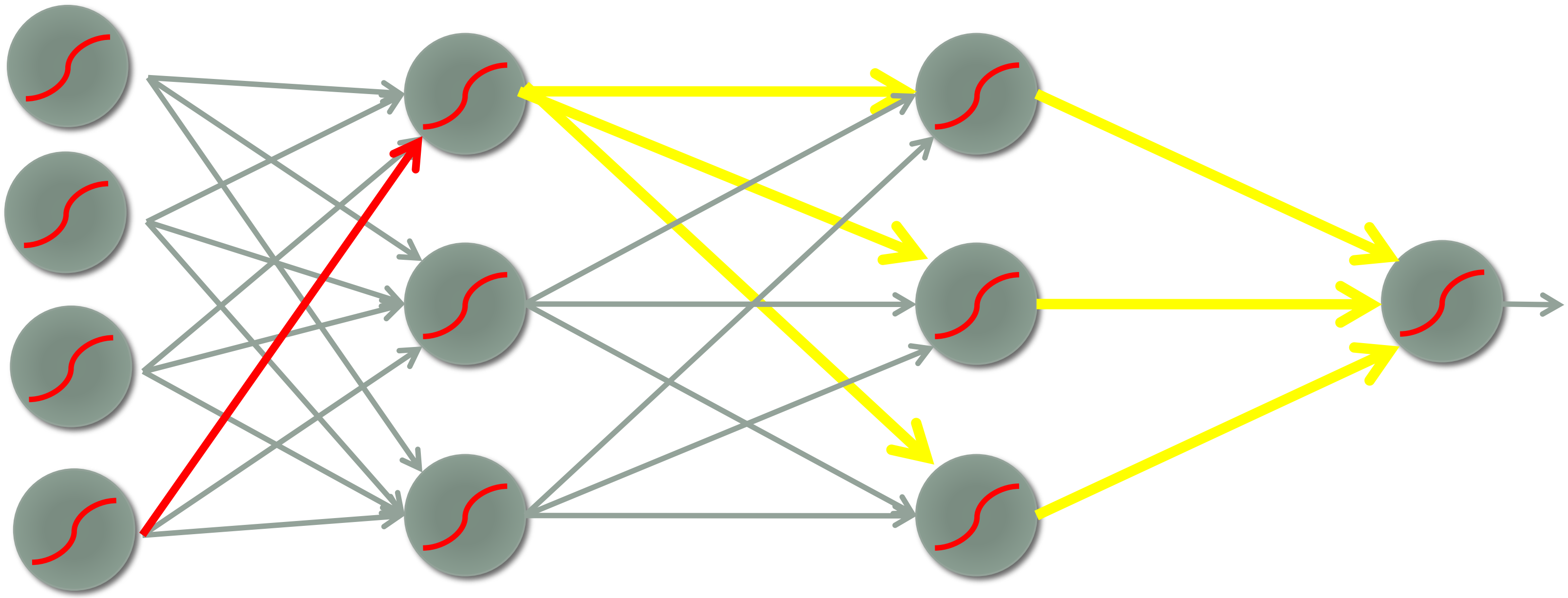
Backpropagation

- Network view



Backpropagation

- Network view



Backpropagation

- Network view