Neuronale Netze (SS 2002), 8.5.

Gradient calculations for feedforward neural networks:

- Derivatives of
$$E = 0.5 \cdot \sum_{p} \sum_{\text{outputs i}} (o_i(\vec{x}^p) - y_i^p)^2$$
for sigmoidal networks with respect to w_{ij}, θ_i.

- Simplifications:
  - it holds $\text{sgd}'(x) = \text{sgd}(x)(1 - \text{sgd}(x))$
  - biases can be simulated as on-neurons
  - derivative with respect to all patterns = sum of the derivatives with respect to one pattern $\rightarrow$ hence we only consider one pattern
  - in a first step: only fully connected multilayered networks

- first try:
$$\frac{E}{\partial w_{ij}} = \sum_{\text{outputs k}} (o_k - y_k) \cdot \frac{\partial o_k}{\partial w_{ij}}$$
where $\frac{\partial o_k}{\partial w_{ij}}$ can be computed recursively for all neurons starting from the input layer up to the output layer as
$$\frac{\partial o_k}{\partial w_{ij}} = \begin{cases} 0 & \text{if neuron } k \text{ lies before weight } w_{ij} \\ o_i \cdot o_k(1 - o_k) & \text{if } j = k \\ \sum_{l \rightarrow j} w_{lk} \cdot \frac{\partial o_l}{\partial w_{ij}} \cdot o_k(1 - o_k) & \text{otherwise} \end{cases}$$
This can be computed via feedforward propagation through the network.
costs: O(W^2), $W =$ number of weights

- alternative – **backpropagation**:
idea: the w_{ij} only contribute through the nettoinputs net_j. Compute the derivatives of E w.r.t. this bottleneck net_j.
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} = \delta_j \cdot o_i$$

where

$$\delta_j := \frac{\partial E}{\partial net_j} = \begin{cases} (o_j - y_j) \cdot o_j(1 - o_j) & j \text{ is output neuron} \\ \sum_{j \to k} \delta_k \cdot w_{jk} \cdot o_j(1 - o_j) & \text{otherwise} \end{cases}$$

This can be computed via backpropagation through the network starting at the output layer up to the first layer.

costs: O(W)

Thereby:

- the terms $o_j(1 - o_j)$ come from the derivative of the activation function for alternative activation f use $f'(net_j)$!
- the term $(o_j - y_j)$ comes from the quadratic error
  for alternative error functions only this part has to be substituted
  it is $\frac{\partial E}{\partial o_j}$ instead of $(o_j - y_j)$

- Backprop is the most popular training algorithm for FNNs.

  It forms the basis for an infinite number of variations which use gradient information. Gradients in FNNs are always computed with the above efficient formula!

- Drawback: no longer biologically plausible –
  - derivative of sgd,
  - global error signals δ_j

  Note: backpropagation computes weighted sum with reversed directions $\rightarrow$ generalized recirculation: biological approximation of backprop for an alternative error function (cross-entropy) and for specific networks where
  - derivative of sgd is integrated into approximation with difference quotient,
  - global error signals δ_j are then substituted by standard synaptic potentials in the network with reversed directions
  - formula: $\triangle w_{ij} = \eta(o_j^+ - o_j^-)o_i^-$
    $o_i^- =$ forward propagated signals
    $o_i^+ =$ backward propagated signals with desired outputs