**Artificial intelligence and deep learning**
Henning Christiansen, September 17, 2017

# The back-propagation and feed-forward algorithms

The various indices in the book chapter's algorithm p. 581 are a bit difficult to follow. Below follows an improved version; we number the layers starting from the input layer, $L_0$ to the output layer $L_N$. For completeness we have also included an abstract version of the RUN-NETWORK algorithm (next page), which is not made explicit in the text.

**function** BACK-PROP-UPDATE(network, examples, $\alpha$) **returns** a network with modified weights
**inputs:** *network*, a multilayer network
*examples*, a set of input-output pairs
$\alpha$, the learning rate

**repeat {**
**for each** $e = \langle \mathbf{I}^e, \mathbf{T}^e \rangle$ **in** examples **do {**
*/* Compute the output for this example, storing values of "$in_i$" and "$a_i$" for each node */*
$\mathbf{O}^e \leftarrow$ RUN-NETWORK$(network, \mathbf{I}^e)$
*/* Compute the error and $\Delta$ for units in the output layer */*
$\mathbf{Err}^e \leftarrow \mathbf{T}^e - \mathbf{O}^e$
**for each** *unit i* **in** *the output layer* **do**
$\Delta_i \leftarrow Err_i^e \times g'(in_i)$
*/* Update the weights leading to the output layer */*
**for each** *unit i* **in** *the output layer* **and** *unit j in the previous layer* **do**
$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
*/* Update the weights in hidden layers */*
**for each** *hidden layer L*: $L_{N-1}, \ldots, L_1$ **do {**
*/* Computer the error term at each node */*
**for each** *node j* **in** *layer L* **do {**
$\Delta_j \leftarrow g'(in_j) \times \sum_i W_{j,i}\Delta_i$, *i running over all nodes in layer* $L+1$
*/* Update weights leading to layer L */*
**for each** *unit k* **in** *layer* $L-1$ **do**
$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$
**}** */* end: for each node j ... */*
**}** */* end: for each hidden layer L ... */*
**}** */* end: for each e ... */*
**}** */* end: repeat */*
**until** *stop-condition*

# The feed forward algorithm

**function** RUN-NETWORK(network, input) **returns** output, plus stores "$in_i$" and "$a_i$" for each unit
  **inputs:** *network*, a multilayer network
    *input*, a sequence of numbers

  */* initialize input nodes with given input */*
  **for each** *unit i in the input layer* **do**
    $a_i \leftarrow I_i$

  */* feed forward through the layers */*
  **for each** *hidden and the output layer* $L$: $L_1, \ldots, L_N$ **do**
    **for each** *node* $j$ *in layer* $L$ **do {**
      $in_i \leftarrow \sum_i W_{j,i} \times a_j$, *i running over all nodes in layer* $L-1$
      $a_i \leftarrow g(in_i)$
    **}**

  */* extract the output */*
  **return** $\langle in_1, \ldots in_n \rangle$, *where* $in_1, \ldots in_n$ *are the values taken from the output layer* $L_N$

Notice that the final output – as read out from the output nodes – is a weighted sum, called "*in*", of the "activations" from then previous layer, the sigmoid function is not applied here.