Neural Networks — April 18, 2024

We want to use a *Neural Network* (NN) to learn, e.g., the XOR function. Apparently, in this case we have two input nodes ($\texttt{inputs} = 2$) and one output node.

Formulas

First we define the sigmoid function g and compute its derivative g':

$$g\colon x \mapsto 1/(1 + e^{-\beta x}) \qquad g'\colon x \mapsto \beta\, g(x)(1 - g(x))$$

Usually we take $\beta = 1$. We can also use other activation functions, like ReLU.
For weights W_j ($j = 0, 1, 2, \ldots, \texttt{hiddens}$) on the edges from hidden layer to output layer (with one output node) the update rule is:

$$W_j \longleftarrow W_j + \alpha \cdot a_j \cdot \Delta \quad \text{with} \quad \Delta = \texttt{error} \cdot g'(\underline{\text{in}})$$

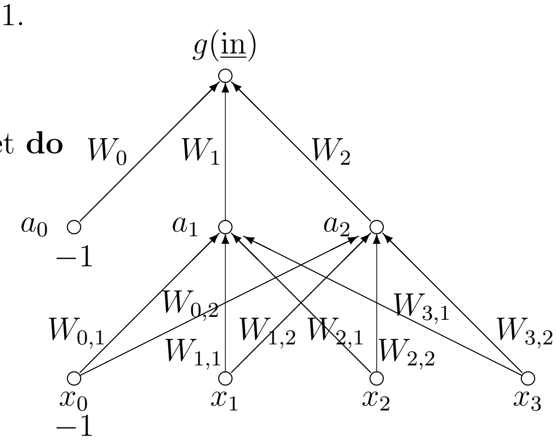
Here α is the learning rate, a_j is the activation of the jth hidden node, and $\underline{\text{in}} = \sum_{\ell=0}^{\texttt{hiddens}} W_\ell\, a_\ell$ is the input for the single output node (in general there can be more than one); $\texttt{error}$ is defined as the target value t minus the net output $g(\underline{\text{in}})$. Always keep the hidden bias node 0 at $a_0 = -1$.
And for weights $W_{k,j}$ ($k = 0, 1, , \ldots, \texttt{inputs}$; $j = 1, 2, \ldots, \texttt{hiddens}$) on the edges from input layer to hidden layer the update rule is:

$$W_{k,j} \longleftarrow W_{k,j} + \alpha \cdot x_k \cdot \Delta_j \quad \text{with} \quad \Delta_j = g'(\underline{\text{in}}_j) \cdot W_j \cdot \Delta$$

Here x_k is the kth input, and $\underline{\text{in}}_j = \sum_{\ell=0}^{\texttt{inputs}} W_{\ell,j}\, x_\ell$ is the input for the jth hidden node, and $a_j = g(\underline{\text{in}}_j)$. Always keep the input bias node 0 at $x_0 = -1$.
Finally, the *Backpropagation algorithm* reads like this:

repeat
  for each $(*)$ $e = (x_1, x_2, \ldots, x_{\texttt{inputs}}, t)$ **in** training set **do**
    compute $\underline{\text{in}}_j$'s, a_j's, $\underline{\text{in}}$ and $g(\underline{\text{in}})$
    compute $\texttt{error}$, Δ and Δ_j's
    update W_j's and $W_{k,j}$'s
until network "converged"

$(*)$ in random order
In the figure we have: $\texttt{inputs} = 3$ en $\texttt{hiddens} = 2$.



Implementation

On the website www.liacs.leidenuniv.nl/~kosterswa/AI/ a simple skeleton program called `nnskelet.cc` is available. The variables are: `input[k]` $\leftrightarrow x_k$, `inhidden[j]` $\leftrightarrow \underline{\text{in}}_j$, `acthidden[j]` $\leftrightarrow a_j$, `inoutput` $\leftrightarrow \underline{\text{in}}$, `netoutput` $\leftrightarrow g(\underline{\text{in}})$, `target` $\leftrightarrow t$, `delta` $\leftrightarrow \Delta$, `deltahidden[j]` $\leftrightarrow \Delta_j$, `inputtohidden[k][j]` $\leftrightarrow W_{k,j}$, `hiddentooutput[j]` $\leftrightarrow W_j$ and finally `ALPHA` $\leftrightarrow \alpha$. Note that $\texttt{inputs} < \texttt{MAX}$ and $\texttt{hiddens} < \texttt{MAX}$.
We use ./nn <inputs> <hiddens> <epochs> <type> <seed>, where we try to learn from <epochs> examples, and <type> determines the activation function.