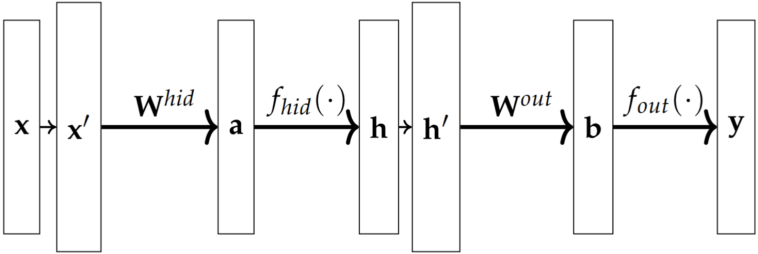
Neural Networks

4. Multi-layer perceptron & Back-propagation

Center for Cognitive Science
Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

Thursday 14th March, 2024

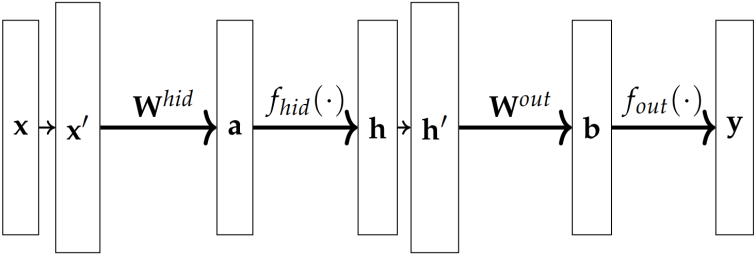
Multi-layer perceptron



$\boldsymbol{x}$ $\qquad$ $\boldsymbol{h} = f_{hid}(\boldsymbol{W}^{hid}\boldsymbol{x}')$ $\qquad$ $\boldsymbol{y} = f_{out}(\boldsymbol{W}^{out}\boldsymbol{h}')$

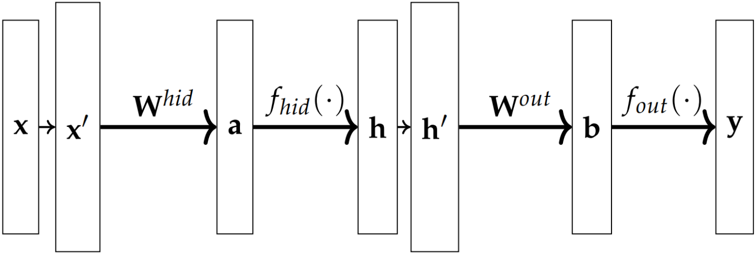
- dimensions:
  - $\boldsymbol{x} : \mathrm{dim}_{\mathrm{in}}$, $\quad \boldsymbol{h} : \mathrm{dim}_{\mathrm{hid}}$, $\quad \boldsymbol{y} : \mathrm{dim}_{\mathrm{out}}$
- add bias terms for both $\boldsymbol{x}$ and $\boldsymbol{h}$:
  - $x'_{\mathrm{dim}_{\mathrm{in}}+1} = h'_{dim_{hid}+1} = 1$

Activation functions



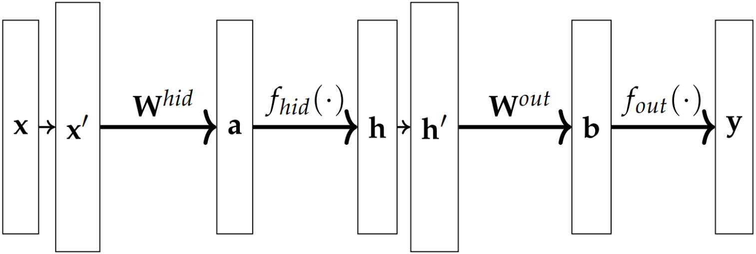
- logistic sigmoid:
  - $\mathrm{logsig}(x) = \frac{1}{1+e^{-x}}$
  - $\mathrm{logsig}'(x) = \mathrm{logsig}(x)(1 - \mathrm{logsig}(x))$
- hyperbolic tangent
- rectified linear units (ReLU)
- linear (makes sense only on output)
- ...

MLP:Back-propagation



- $g_i^{out} = (d_i - y_i) f'_{out}(b_i)$
- $g_k^{hid} = \left(\sum_i w_{i,k}^{out} g_i^{out}\right) f'_{hid}(a_k)$
- $\Delta \boldsymbol{W}^{out} = \boldsymbol{g}^{out} \boldsymbol{h}'^T \qquad \boldsymbol{W}^{out}(t+1) = \boldsymbol{W}^{out}(t) + \alpha \Delta \boldsymbol{W}^{out}(t)$
- $\Delta \boldsymbol{W}^{hid} = \boldsymbol{g}^{hid} \boldsymbol{x}'^T \qquad \boldsymbol{W}^{hid}(t+1) = \boldsymbol{W}^{hid}(t) + \alpha \Delta \boldsymbol{W}^{hid}(t)$
- $\boldsymbol{g}^{hid}$ *and* $\boldsymbol{g}^{out}$ *describe the errors of hidden/output neurons*

MLP:Back-propagation



- $\boldsymbol{g}^{hid}$ and $\boldsymbol{g}^{out}$ describe the errors of hidden/output neurons, hence:
  - $dim(\boldsymbol{g}_{hid}) = dim(\boldsymbol{h}) = dim_{hid}$
  - $dim(\boldsymbol{g}_{out}) = dim(\boldsymbol{y}) = dim_{out}$
- bias on hidden layer is *not* a neuron, thus we do not use its weights $\boldsymbol{W}^{out}_{:,dim_{hid+1}}$ when computing $\boldsymbol{g}^{hid}$.

Algorithm

Initialization:

1. choose model parameters (# of hidden neurons)
2. choose training parameters (learning rate, # epochs)
3. generate random initial weights

Training:

Until stopping criterion (accuracy / # epochs / time, ...):

- with each training sample($\boldsymbol{x}, \boldsymbol{d}$) *in random order*:
    - forward-pass: compute $\boldsymbol{a}, \boldsymbol{h}, \boldsymbol{b}, \boldsymbol{y}$
    - backward-pass: compute $\Delta \boldsymbol{W}^{hid}, \Delta \boldsymbol{W}^{out}$
    - adjust weights $\boldsymbol{W}^{hid}, \boldsymbol{W}^{out}$

Task

Train regressor on 2D data, with one output variable.

1. C04.py
   - data preparation, launcher
   - to-do: data normalization (zero mean, unit variance)
2. mlp.py
   - abstract base class for generic MLP
   - to-do: forward & backward pass (i.e. output computation and weight adjustment)
3. regressor.py
   - derived class for specific regression model
   - to-do: f_{hid}, f_{out} and the training cycle