Advanced Machine Learning
Lecture 6: Neural networks

Sandjai Bhulai
Vrije Universiteit Amsterdam

s.bhulai@vu.nl
22 September 2023

VU VRIJE UNIVERSITEIT AMSTERDAM
Faculty of Science

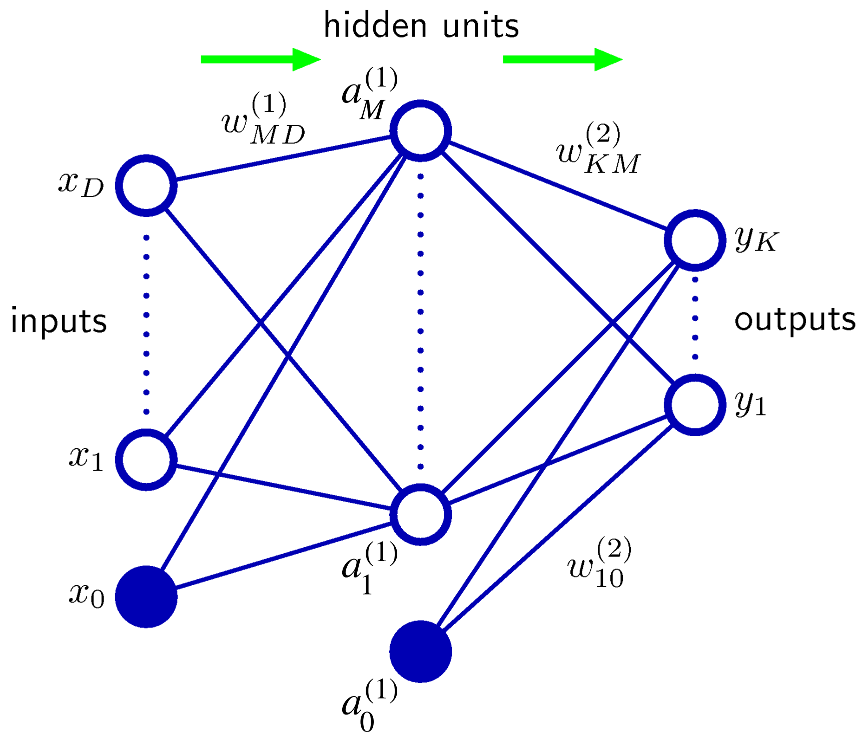
Neural networks

Advanced Machine Learning

Neural networks

- Transform to outputs by activation function σ

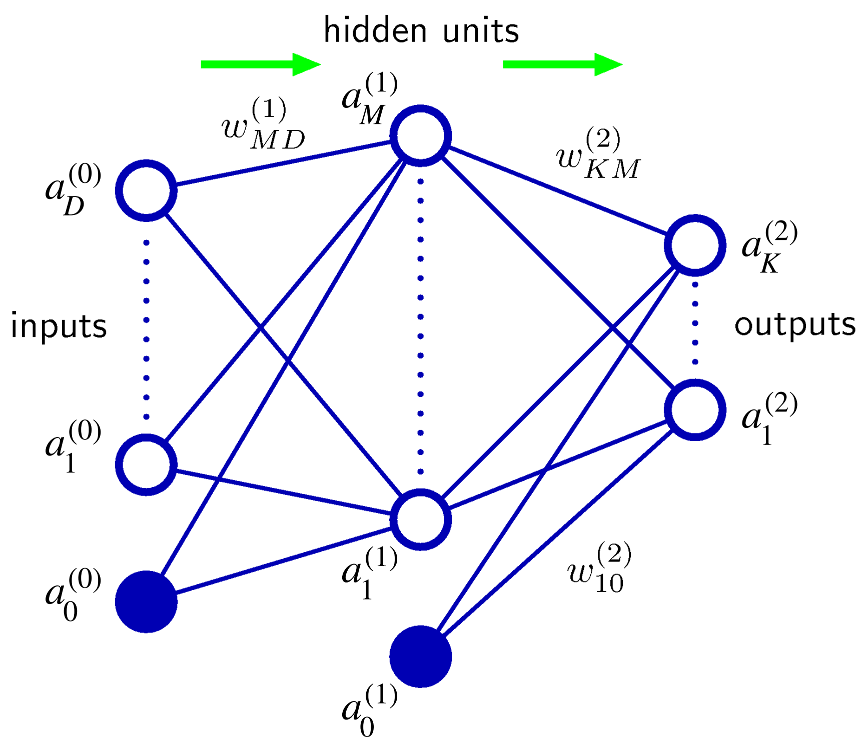
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$



Neural networks

- Transform to outputs by activation function σ

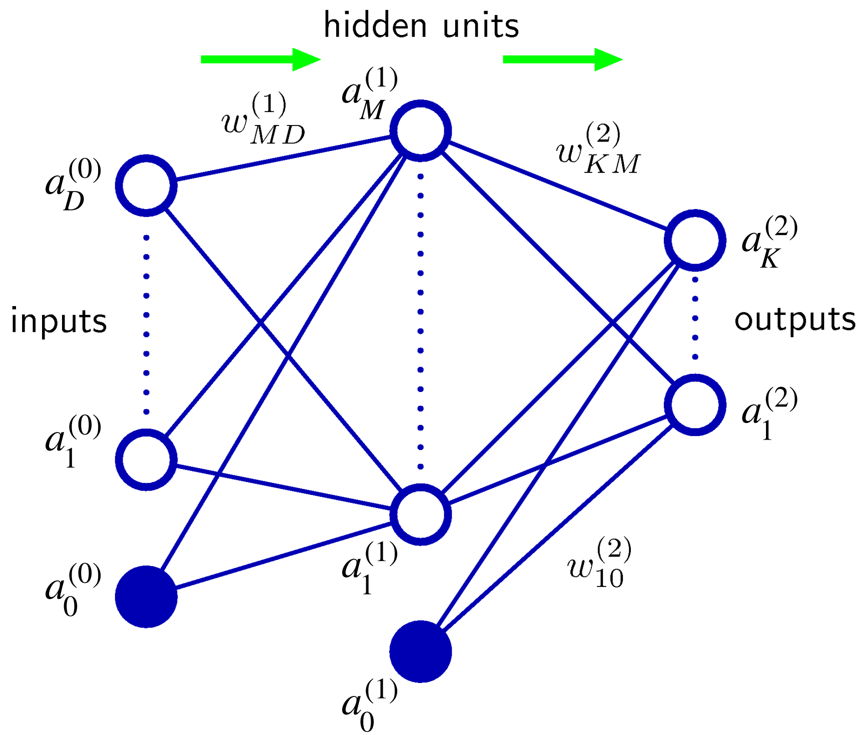
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$



Neural networks

- Transform to outputs by activation function σ

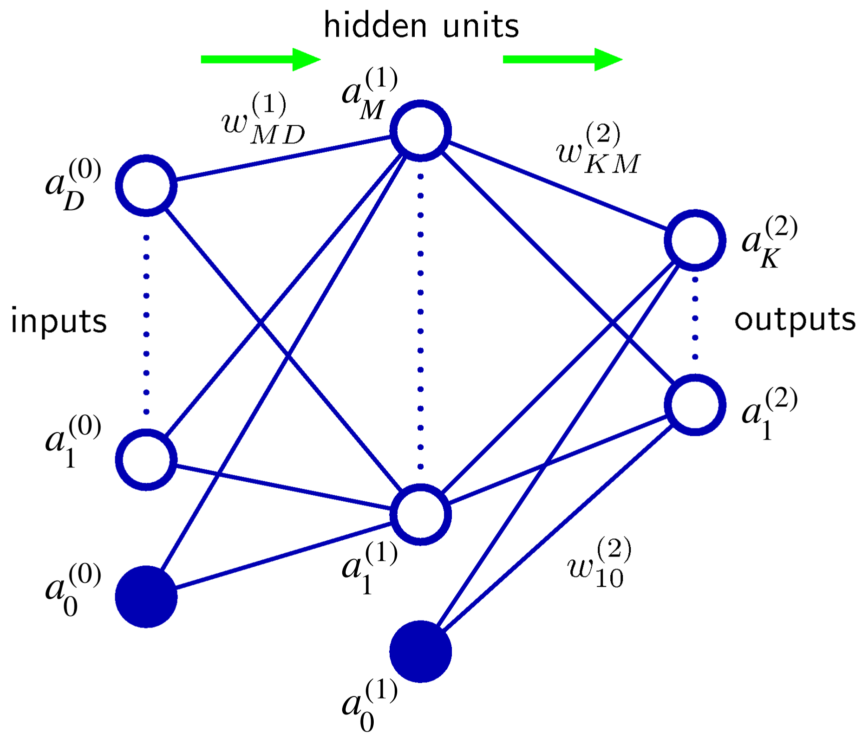
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} a_i^{(0)} + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$



Neural networks

- Transform to outputs by activation function σ

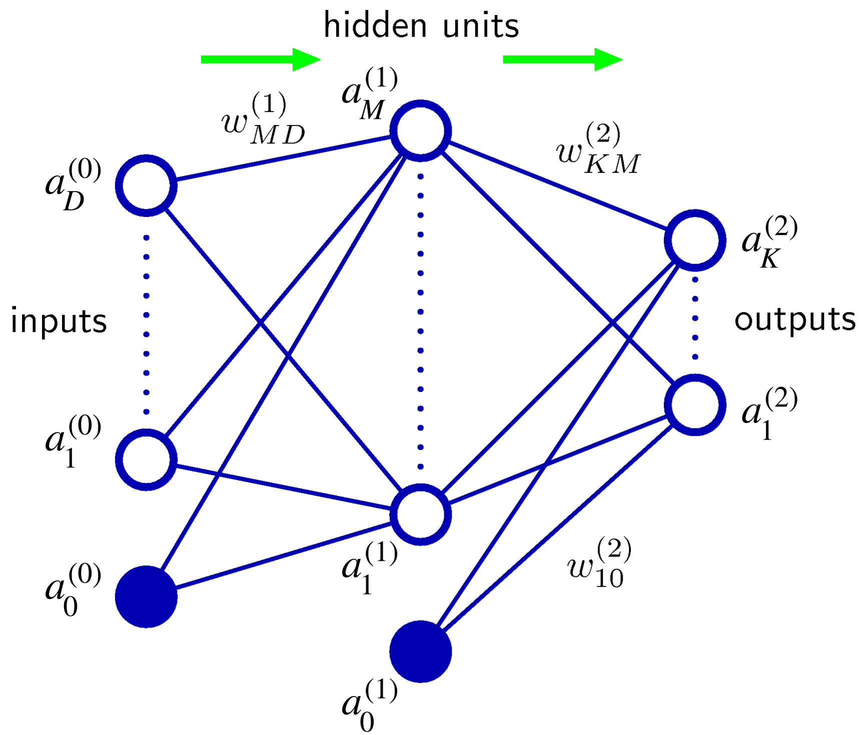
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h(z_j^{(1)}) + w_{k0}^{(2)} \right)$$



Neural networks

- Transform to outputs by activation function σ

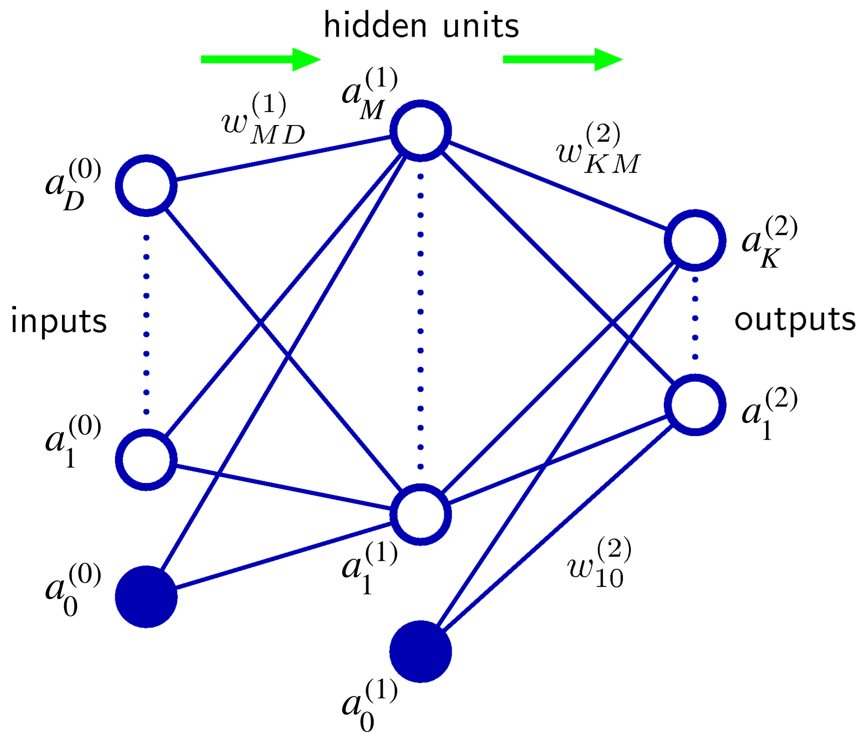
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} a_j^{(1)} + w_{k0}^{(2)}\right)$$



Neural networks

- Transform to outputs by activation function σ

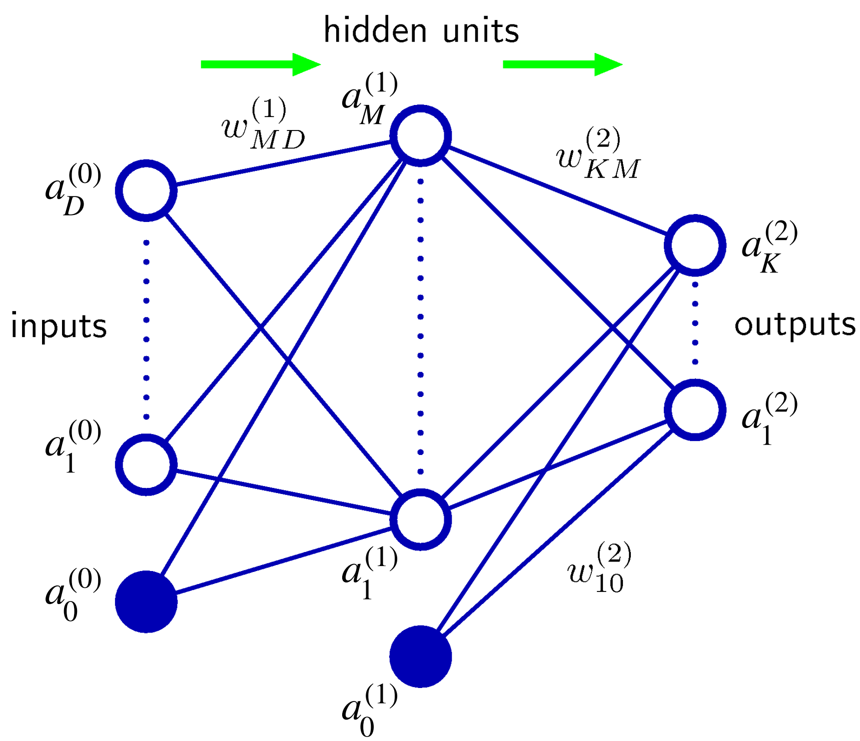
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma(z_k^{(2)})$$



Neural networks

- Transform to outputs by activation function σ

$$y_k(\mathbf{x}, \mathbf{w}) = a_k^{(2)}$$



Neural networks - training

- Thus,

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} a_i^{(0)}$$

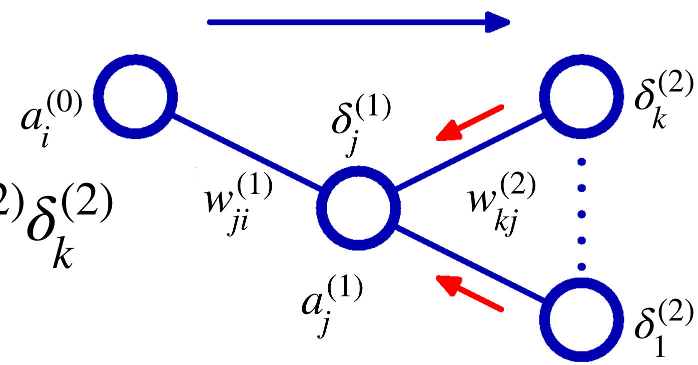
$$\boxed{\frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} = a_i^{(l-1)}}$$

$$\boxed{\delta_j^{(l)} = \frac{\partial E_n}{\partial z_j^{(l)}}}$$

- This has the form as the simple linear model that we considered

- Using $z_j^{(1)} = \sum_i w_{ji}^{(1)} a_i^{(0)}$ and $a_j^{(1)} = h(z_j^{(1)})$, we find

$$\delta_j^{(1)} = \frac{\partial E_n}{\partial z_j^{(1)}} = \sum_k \frac{\partial E_n}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial z_j^{(1)}} = h'(z_j^{(1)}) \sum_k w_{kj}^{(2)} \delta_k^{(2)}$$



Neural networks - training

- Error backpropagation

1. Apply an input vector x_n to the network and forward propagate through the network using $z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)}$ and $a_j^{(l)} = h(z_j^{(l)})$ to find the activations of all the hidden and output units
2. Evaluate the $\delta_k^{(2)}$ for all the output units using $\delta_k^{(2)} = y_k - t_k$
3. Backpropagate the $\delta^{(2)}$'s using $\delta_j^{(1)} = h'(z_j^{(1)}) \sum_k w_{kj}^{(2)} \delta_k^{(2)}$ to obtain $\delta_j^{(1)}$ for each hidden unit in the network
4. Use $\dfrac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$ to evaluate the required derivates

Neural networks - training

- Example: regression with $\tanh$ activation functions
- Note that $h'(z) = 1 - h(z)^2$

$$h(z) = \tanh(z)$$

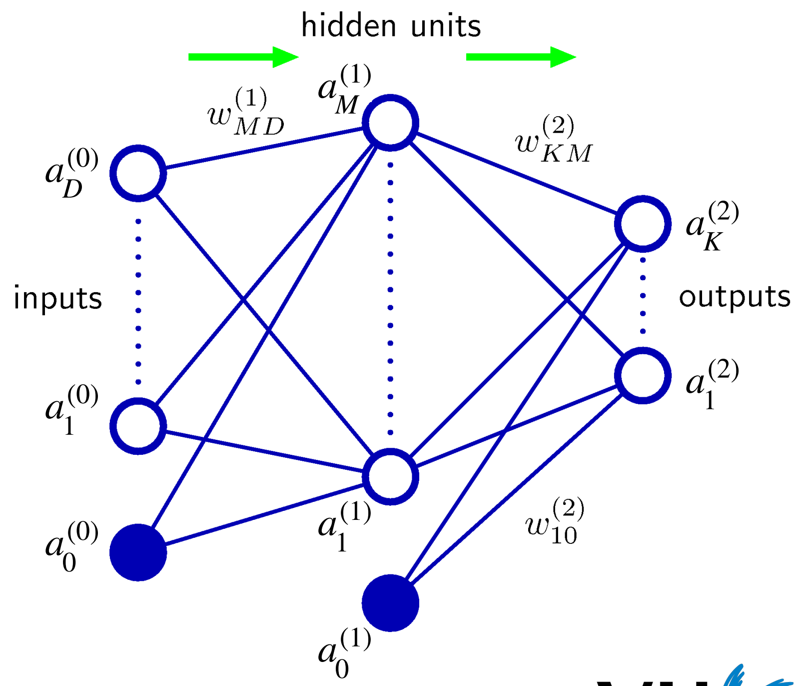
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Forward propagation:

$$z_j^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$a_j^{(1)} = \tanh(z_j^{(1)})$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}$$



Neural networks - training

Errors

- Output:

$$\delta_k^{(2)} = y_k - t_k$$

- Hidden layer:

$$\delta_j^{(1)} = (1 - (a_j^{(1)})^2) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_k^{(2)}$$

- Derivatives:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} a_j^{(1)}$$

$$z_j^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

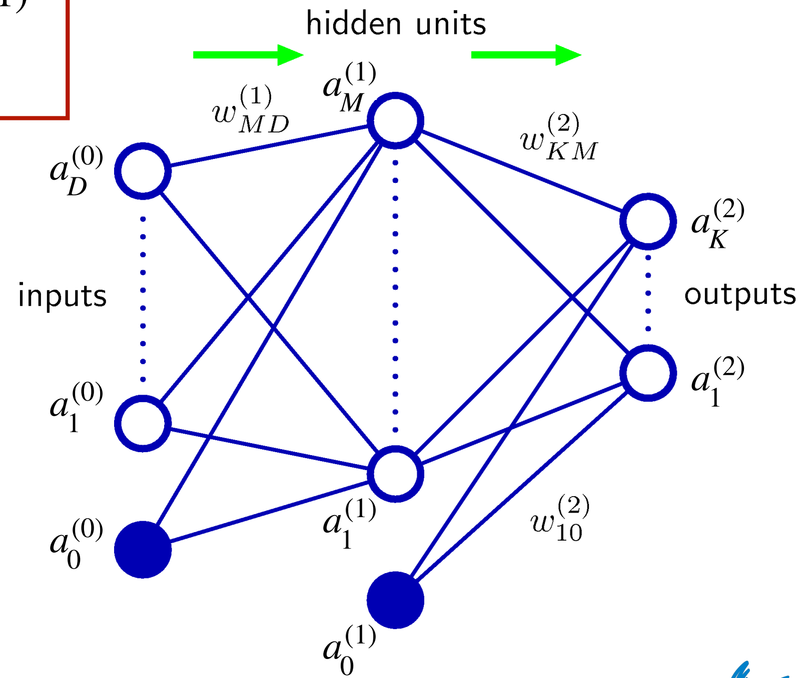
$$a_j^{(1)} = \tanh(z_j^{(1)})$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}$$

$$h(z) = \tanh(z)$$

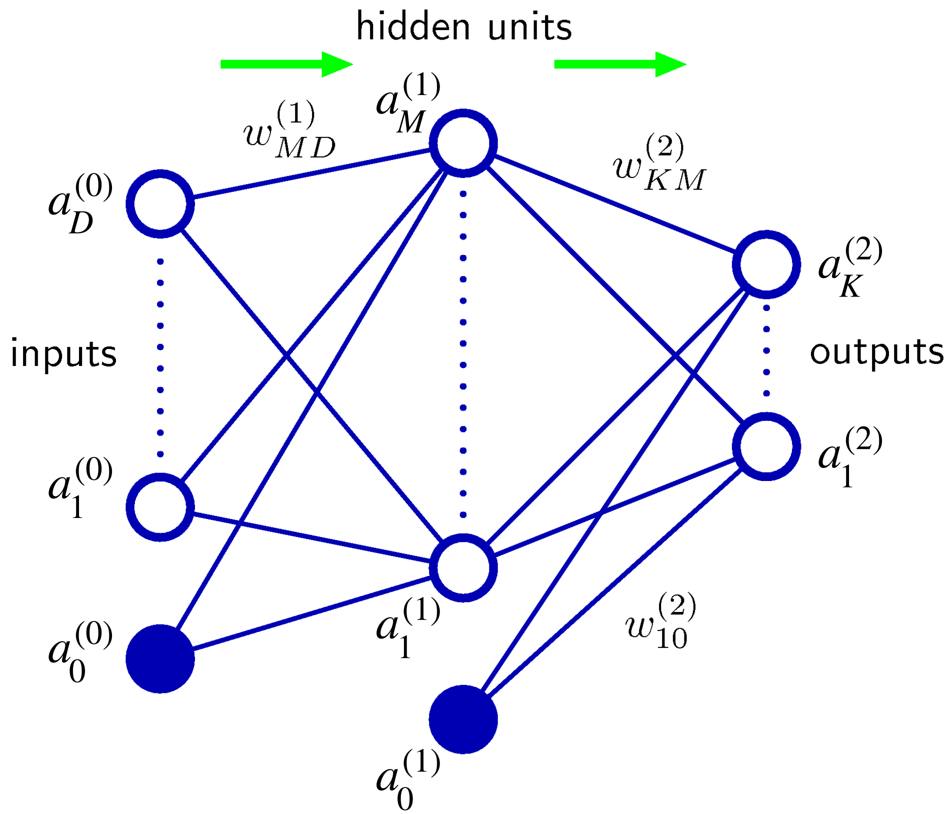
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$h'(z) = 1 - h(z)^2$$



Neural networks - training

$$x_k = a_k^{(0)} \xrightarrow{\sum_{i=0}^{D} w_{ji}^{(1)} a_i^{(0)}} z_j^{(1)} \xrightarrow{\tanh(z_j^{(1)})} a_j^{(1)} \xrightarrow{\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}} z_k^{(2)} = y_k \xrightarrow{\frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2} E$$



Neural networks - training

$$x_k = a_k^{(0)} \xrightarrow{\sum_{i=0}^{D} w_{ji}^{(1)} a_i^{(0)}} z_j^{(1)} \xrightarrow{\tanh(z_j^{(1)})} a_j^{(1)} \xrightarrow{\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}} z_k^{(2)} = y_k \xrightarrow{\frac{1}{2}\sum_{i=1}^{N} (y_i - t_i)^2} E$$

$$\frac{\partial E}{\partial z_k^{(2)}} = \frac{\partial E}{\partial y_k} = \frac{\partial \frac{1}{2} \sum_{i=1}^{N} (y_i - t_i)^2}{\partial y_k} = y_k - t_k = \delta_k^{(2)}$$

Neural networks - training

$$x_k = a_k^{(0)} \xrightarrow{\sum_{i=0}^{D} w_{ji}^{(1)} a_i^{(0)}} z_j^{(1)} \xrightarrow{\tanh(z_j^{(1)})} a_j^{(1)} \xrightarrow{\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}} z_k^{(2)} = y_k \xrightarrow{\frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2} E$$

$$\frac{\partial E}{\partial z_k^{(2)}} = \frac{\partial E}{\partial y_k} = \frac{\partial \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2}{\partial y_k} = y_k - t_k = \delta_k^{(2)}$$

$$\frac{\partial z_k^{(2)}}{\partial a_l^{(1)}} = \frac{\partial \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}}{\partial a_l^{(1)}} = w_{kl}^{(2)}$$

Neural networks - training

$$x_k = a_k^{(0)} \xrightarrow{\sum_{i=0}^{D} w_{ji}^{(1)} a_i^{(0)}} z_j^{(1)} \xrightarrow{\tanh(z_j^{(1)})} a_j^{(1)} \xrightarrow{\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}} z_k^{(2)} = y_k \xrightarrow{\frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2} E$$

$$\frac{\partial E}{\partial z_k^{(2)}} = \frac{\partial E}{\partial y_k} = \frac{\partial \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2}{\partial y_k} = y_k - t_k = \delta_k^{(2)}$$

$$\frac{\partial z_k^{(2)}}{\partial a_l^{(1)}} = \frac{\partial \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}}{\partial a_l^{(1)}} = w_{kl}^{(2)}$$

$$\frac{\partial a_l^{(1)}}{\partial z_l^{(1)}} = \frac{\partial \tanh(z_l^{(1)})}{\partial z_l^{(1)}} = 1 - \tanh^2(z_l^{(1)}) = 1 - (a_l^{(1)})^2$$

Neural networks - training

$$x_k = a_k^{(0)} \xrightarrow{\sum_{i=0}^{D} w_{ji}^{(1)} a_i^{(0)}} z_j^{(1)} \xrightarrow{\tanh(z_j^{(1)})} a_j^{(1)} \xrightarrow{\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}} z_k^{(2)} = y_k \xrightarrow{\frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2} E$$

$$\frac{\partial E}{\partial z_k^{(2)}} = \frac{\partial E}{\partial y_k} = \frac{\partial \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2}{\partial y_k} = y_k - t_k = \delta_k^{(2)}$$

$$\frac{\partial z_k^{(2)}}{\partial a_l^{(1)}} = \frac{\partial \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}}{\partial a_l^{(1)}} = w_{kl}^{(2)}$$

$$\frac{\partial a_l^{(1)}}{\partial z_l^{(1)}} = \frac{\partial \tanh(z_l^{(1)})}{\partial z_l^{(1)}} = 1 - \tanh^2(z_l^{(1)}) = 1 - (a_l^{(1)})^2$$

$$\frac{\partial E}{\partial z_l^{(1)}} = \sum_{k=1}^{K} \frac{\partial E}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial a_l^{(1)}} \frac{\partial a_l^{(1)}}{\partial z_l^{(1)}} = [1 - (a_l^{(1)})^2] \sum_{k=1}^{K} \delta_k^{(2)} w_{kl}^{(2)} = \delta_l^{(1)}$$

Neural networks - training

$$x_k = a_k^{(0)} \xrightarrow{\sum_{i=0}^{D} w_{ji}^{(1)} a_i^{(0)}} z_j^{(1)} \xrightarrow{\tanh(z_j^{(1)})} a_j^{(1)} \xrightarrow{\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}} z_k^{(2)} = y_k \xrightarrow{\frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2} E$$

$$\frac{\partial E}{\partial z_k^{(2)}} = \frac{\partial E}{\partial y_k} = \frac{\partial \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2}{\partial y_k} = y_k - t_k = \delta_k^{(2)}$$

$$\frac{\partial z_k^{(2)}}{\partial a_l^{(1)}} = \frac{\partial \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}}{\partial a_l^{(1)}} = w_{kl}^{(2)}$$

$$\frac{\partial a_l^{(1)}}{\partial z_l^{(1)}} = \frac{\partial \tanh(z_l^{(1)})}{\partial z_l^{(1)}} = 1 - \tanh^2(z_l^{(1)}) = 1 - (a_l^{(1)})^2$$

$$\boxed{\begin{aligned} \frac{\partial E}{\partial w_{kj}^{(2)}} &= \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} a_j^{(1)} \\ \frac{\partial E}{\partial w_{ji}^{(1)}} &= \frac{\partial E}{\partial z_j^{(1)}}\frac{\partial z_j^{(1)}}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i = \delta_j^{(1)} a_i^{(0)} \end{aligned}}$$

$$\frac{\partial E}{\partial z_l^{(1)}} = \sum_{k=1}^{K} \frac{\partial E}{\partial z_k^{(2)}}\frac{\partial z_k^{(2)}}{\partial a_l^{(1)}}\frac{\partial a_l^{(1)}}{\partial z_l^{(1)}} = [1 - (a_l^{(1)})^2]\sum_{k=1}^{K} \delta_k^{(2)} w_{kl}^{(2)} = \delta_l^{(1)}$$

Neural networks - backpropagation

- Note that Jacobian is now easy to derive:

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \frac{\partial z_k^{(2)}}{\partial a_i^{(0)}} = \sum_{j=1}^{M} \frac{\partial z_k^{(2)}}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial a_i^{(0)}}$$

- This is relevant to sensitivity analysis

$$x_k = a_k^{(0)} \xrightarrow{\sum_{i=0}^{D} w_{ji}^{(1)} a_i^{(0)}} z_j^{(1)} \xrightarrow{\tanh(z_j^{(1)})} a_j^{(1)} \xrightarrow{\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}} z_k^{(2)} = y_k \xrightarrow{\frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2} E$$

Neural networks - backpropagation

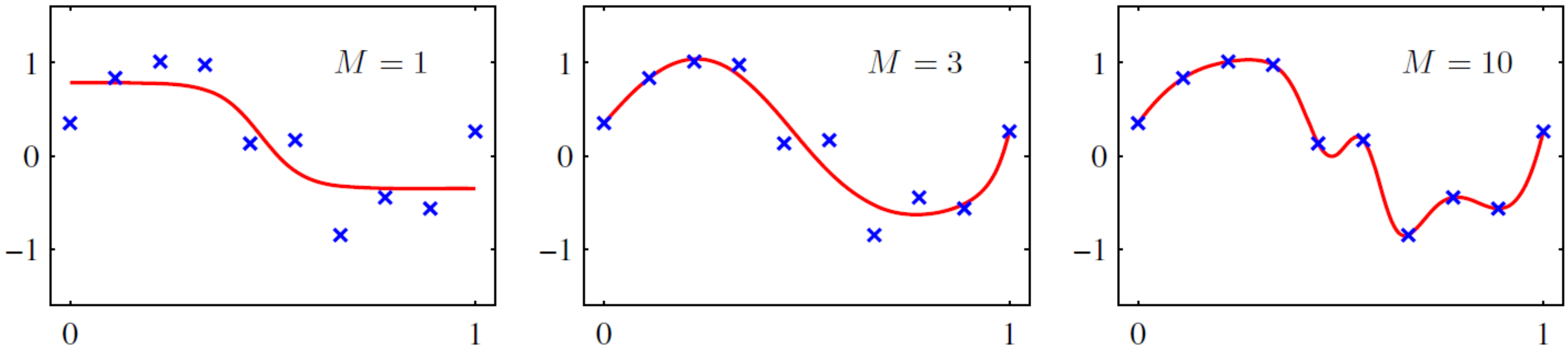
- The Hessian can also be derived similarly:

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}$$

- This is relevant to
  - non-linear optimization techniques
  - re-training feed-forward networks
  - pruning by removing least significant weights
  - Laplace approximation for Bayesian networks

Regularization in neural networks

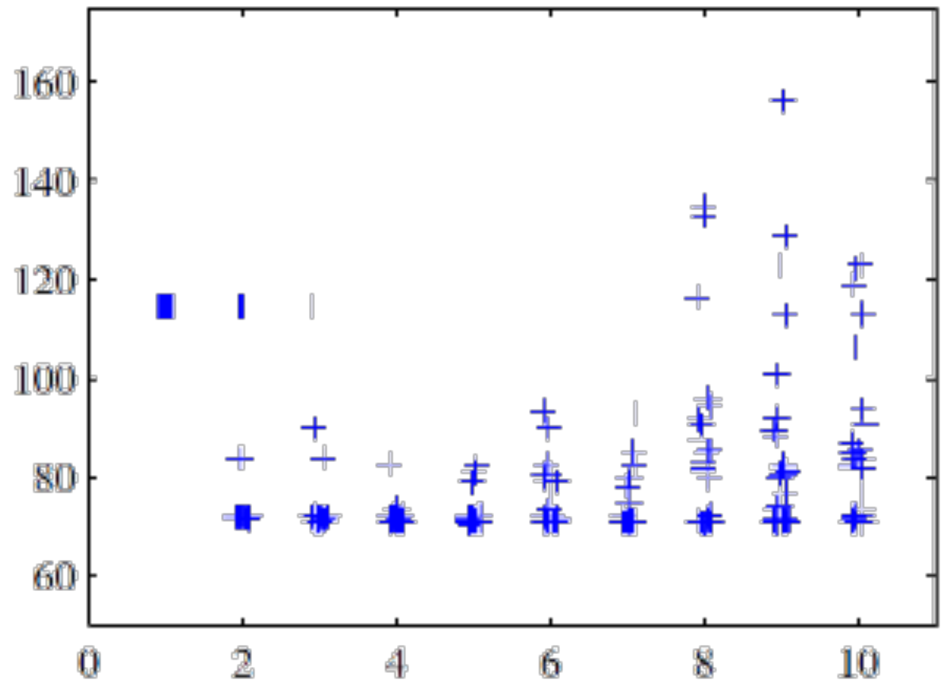
- Effect of number of hidden nodes M



- Note that the model has $(D + 1)M + (M + 1)K$ weights

Regularization in neural networks

- Plot a graph choosing random starts and different numbers of hidden units
- Choose the solution with the smallest generalization error on validation set
- 30 random starts for each M



Regularization in neural networks

- Weights decay regularization

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}$$

- Weight decay has shortcomings:
  - not invariant to scaling and translations

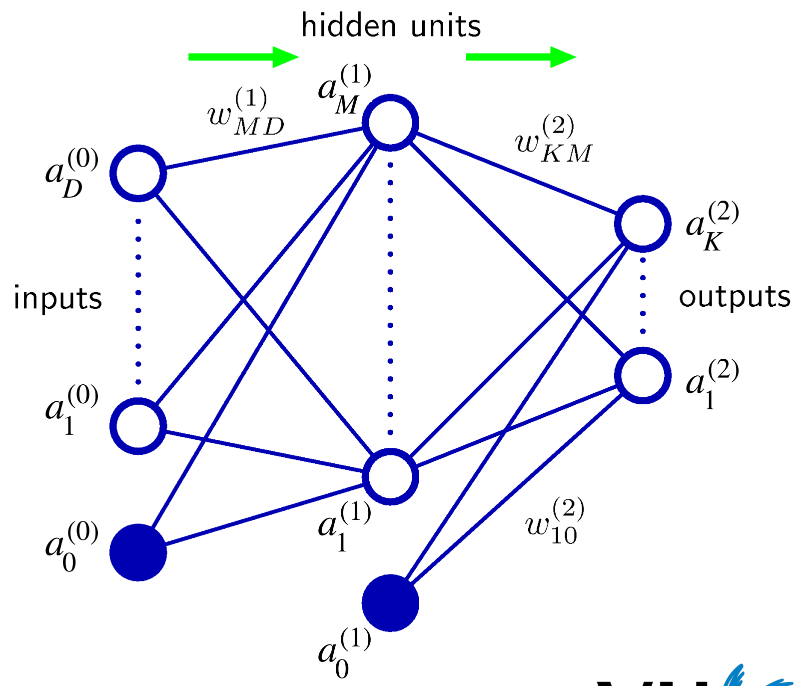
Regularization in neural networks

- Recall that for this network we have for inputs $\{x_i\}$ and outputs $\{y_k\}$:

$$a_j^{(1)} = h\left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right)$$

- and

$$y_k = \sum_j w_{kj}^{(2)} a_j^{(1)} + w_{k0}^{(2)}$$



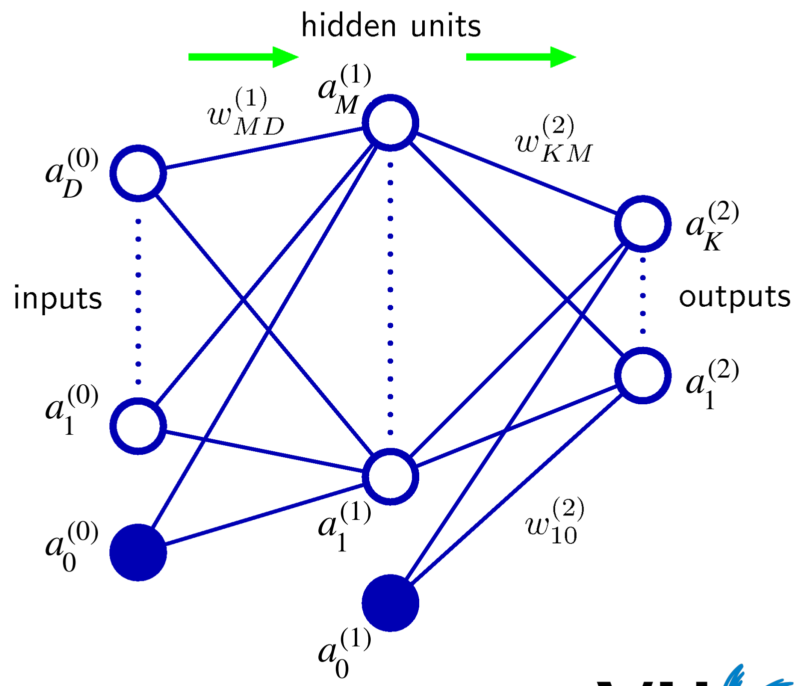
Regularization in neural networks

- Perform a linear transformation $x_i \rightarrow \tilde{x}_i = ax_i + b$
- Then we can make a mapping to have unchanged results:

$$w_{ji}^{(1)} \rightarrow \tilde{w}_{ji}^{(1)} = \frac{1}{a} w_{ji}^{(1)}$$

$$w_{j0}^{(1)} \rightarrow \tilde{w}_{j0}^{(1)} = w_{j0}^{(1)} - \frac{b}{a} \sum_i w_{ji}^{(1)}$$

$$\boxed{a_j^{(1)} = h\left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right)}$$



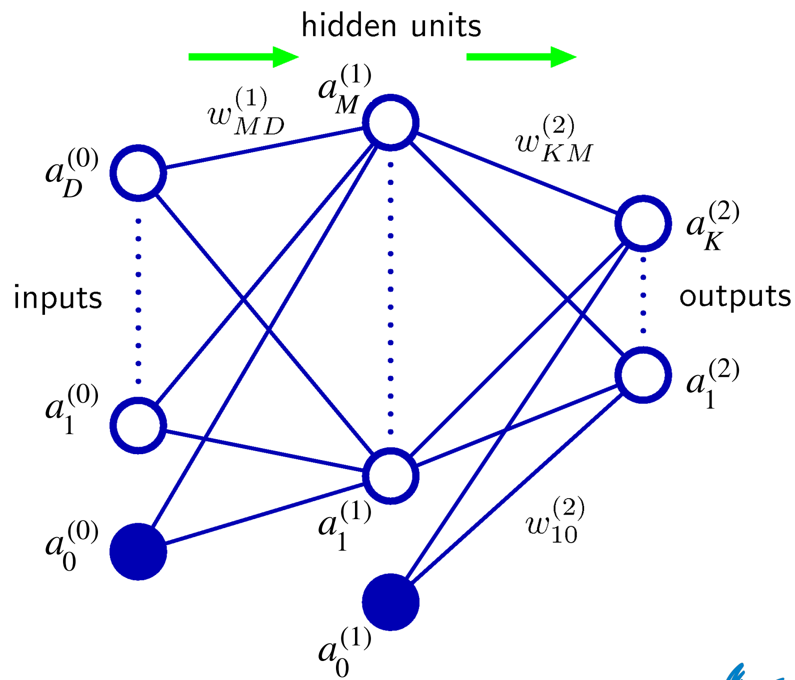
Regularization in neural networks

- Similarly for the output, we have $y_k \rightarrow \tilde{y}_k = c y_k + d$
- Then we can make a mapping to have unchanged results:

$$w_{kj}^{(2)} \rightarrow \tilde{w}_{kj}^{(2)} = c w_{kj}^{(2)}$$

$$w_{k0}^{(2)} \rightarrow \tilde{w}_{k0}^{(2)} = c w_{k0}^{(2)} + d$$

$$\boxed{y_k = \sum_j w_{kj}^{(2)} a_j^{(1)} + w_{k0}^{(2)}}$$



Regularization in neural networks

- Train two networks:
  - First network with $\{x_i\}$ and $\{y_k\}$
  - Second network with $\{\tilde{x}_i\}$ and/or $\{\tilde{y}_k\}$

- Consistency requires that you should obtain equivalent networks that differ only by linear transformation of weights
  - first layer: $w_{ji}^{(1)} \rightarrow \dfrac{1}{a} w_{ji}^{(1)}$
  - second layer: $w_{kj}^{(2)} \rightarrow c w_{kj}^{(2)}$

Regularization in neural networks

- Simple weight decay does not have this property
- The different set of weights should be treated differently
- New regularization term:

$$\frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2$$

Regularization in neural networks

- New regularization term:

$$\frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2$$

- This regularization remains unchanged under the weight transformation provided

$$\lambda_1 \rightarrow a^{1/2} \lambda_1 \qquad \text{and} \qquad \lambda_2 \rightarrow c^{-1/2} \lambda_2$$

- Weight decay equivalent to Gaussian prior
- What is this regularization equivalent to?

Regularization in neural networks

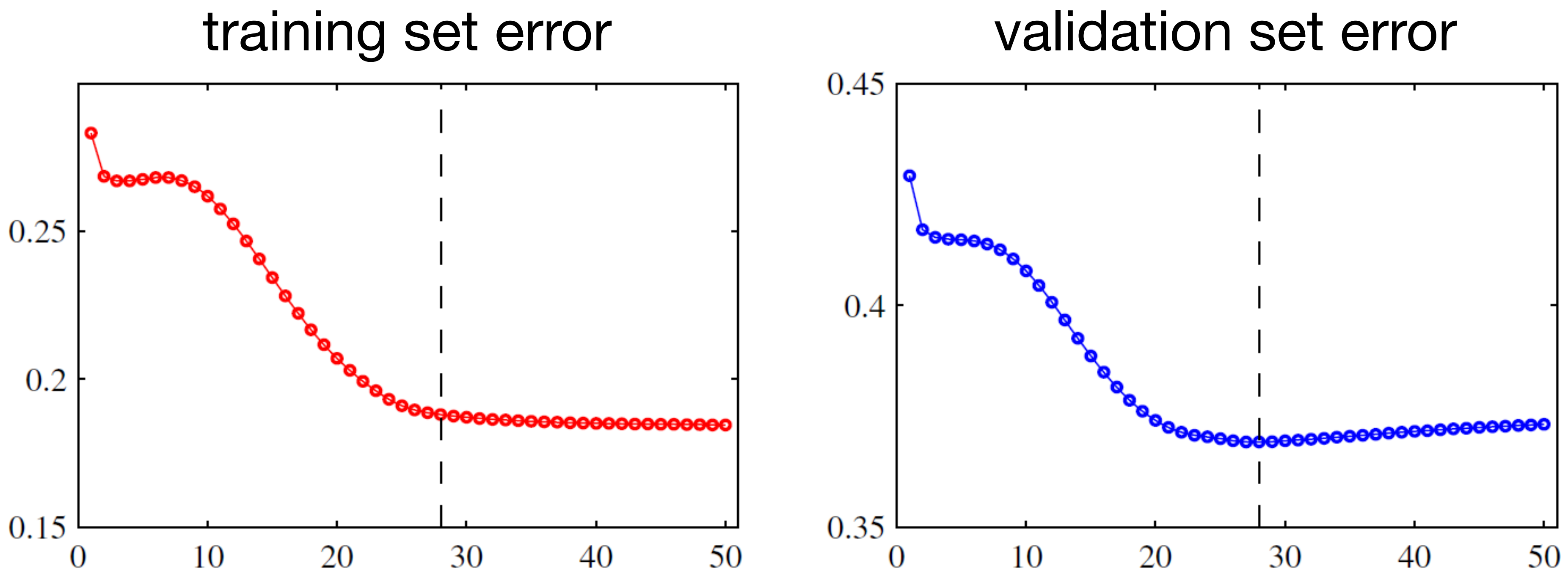
- Prior has the form

$$p(\mathbf{w} \mid \alpha_1, \alpha_2) \propto \exp\left(-\frac{\alpha_1}{2}\sum_{w \in \mathscr{W}_1} w^2 - \frac{\alpha_2}{2}\sum_{w \in \mathscr{W}_2} w^2\right)$$

- This is an improper prior that cannot be normalized
  - > leads to difficulties in selecting regularization coefficients
  - > leads to difficulties in model comparison
  - > include separate priors for biases

Regularization in neural networks

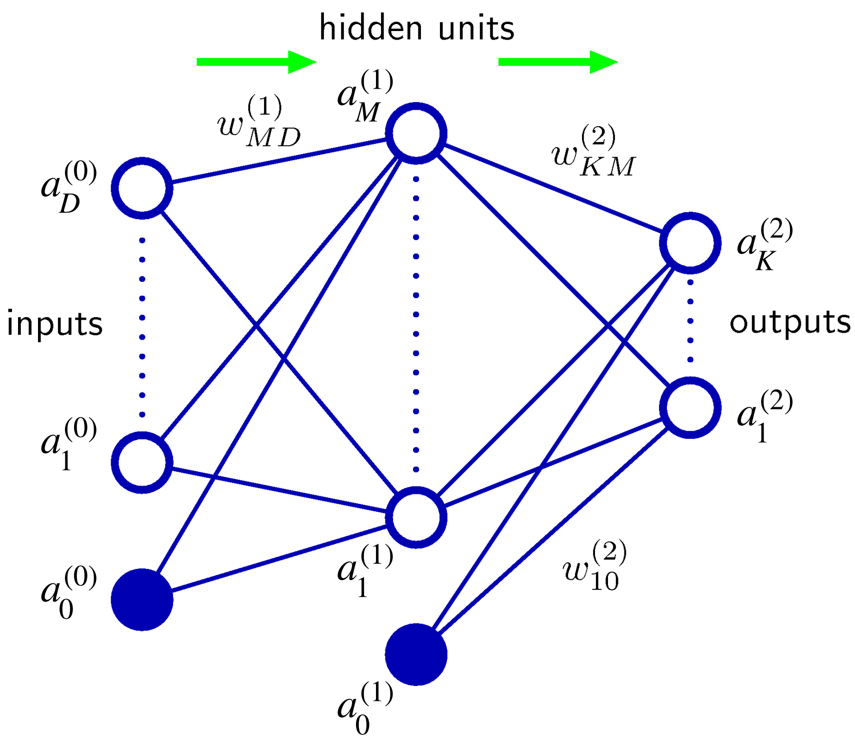
- Alternative to regularization: early stopping
- Stop at smallest error with validation data



Neural networks algebra

- Transform to outputs by activation function σ

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$



$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = h(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

$$\mathscr{L}(a^{[2]}, y)$$

Neural networks algebra

- Transform to outputs by activation function σ

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = h(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

$$\mathcal{L}(A^{[2]}, y)$$

Neural networks algebra

$$z^{[1]} = W^{[1]}x + b^{[1]} \to a^{[1]} = h(z^{[1]}) \to z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \to a^{[2]} = h(z^{[2]}) \to \mathscr{L}(z^{[2]}, y)$$

$$\frac{\partial E}{\partial z_k^{(2)}} = \frac{\partial E}{\partial y_k} = \frac{\partial \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2}{\partial y_k} = y_k - t_k = \delta_k^{(2)}$$

$$\mathrm{d}z^{[2]} = a^{[2]} - y$$

$$\frac{\partial E}{\partial z_l^{(1)}} = \sum_{k=1}^{K} \frac{\partial E}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial a_l^{(1)}} \frac{\partial a_l^{(1)}}{\partial z_l^{(1)}} = [1 - (a_l^{(1)})^2] \sum_{k=1}^{K} \delta_k^{(2)} w_{kl}^{(2)} = \delta_l^{(1)}$$

$$\mathrm{d}z^{[1]} = (W^{[2]})^\top \mathrm{d}z^{[2]} \cdot (h^{[1]})'(z^{[1]})$$

Neural networks algebra

$$z^{[1]} = W^{[1]}x + b^{[1]} \rightarrow a^{[1]} = h(z^{[1]}) \rightarrow z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \rightarrow a^{[2]} = h(z^{[2]}) \rightarrow \mathscr{L}(z^{[2]}, y)$$

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} a_j^{(1)}$$

$$\mathrm{d}W^{[2]} = \mathrm{d}z^{[2]}(a^{[1]})^\top$$

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \frac{\partial E}{\partial z_j^{(1)}}\frac{\partial z_j^{(1)}}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i = \delta_j^{(1)} a_i^{(0)}$$

$$\mathrm{d}W^{[1]} = \mathrm{d}z^{[1]}x^\top = \mathrm{d}z^{[1]}(a^{[0]})^\top$$

Neural networks algebra

$$z^{[1]} = W^{[1]}x + b^{[1]} \to a^{[1]} = h(z^{[1]}) \to z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \to a^{[2]} = h(z^{[2]}) \to \mathscr{L}(z^{[2]}, y)$$

$$\mathrm{d}z^{[2]} = a^{[2]} - y$$

$$\mathrm{d}W^{[2]} = \mathrm{d}z^{[2]}(a^{[1]})^\top$$

$$\mathrm{d}b^{[2]} = \mathrm{d}z^{[2]}$$

$$\mathrm{d}z^{[1]} = (W^{[2]})^\top \mathrm{d}z^{[2]} \cdot (h^{[1]})'(z^{[1]})$$

$$\mathrm{d}W^{[1]} = \mathrm{d}z^{[1]}x^\top$$

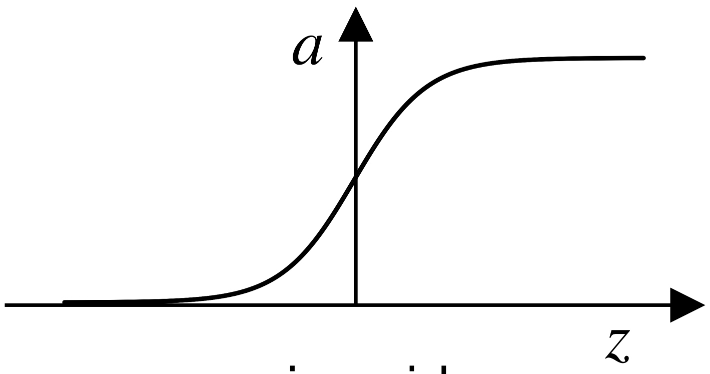
$$\mathrm{d}b^{[1]} = \mathrm{d}z^{[1]}$$

Activation functions

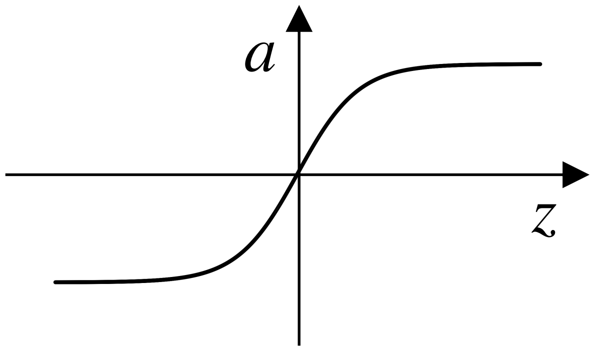
Activation functions

- Sigmoid: $f(z) = \dfrac{1}{1+e^{-z}}$
- Hyperbolic tangent: $\tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$
- Rectified linear unit: $\text{ReLu}(z) = \max\{0,z\}$
- Leaky ReLu: $\text{ReLu}(z) = \max\{0.01z, z\}$

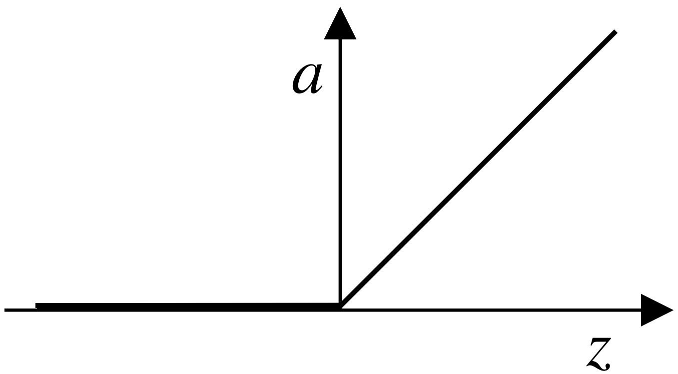
Activation functions



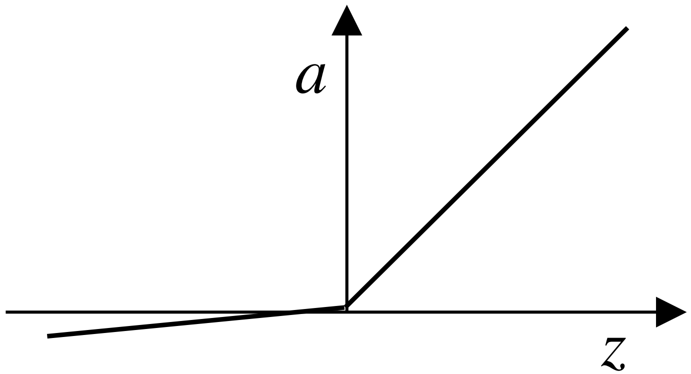
sigmoid



tanh



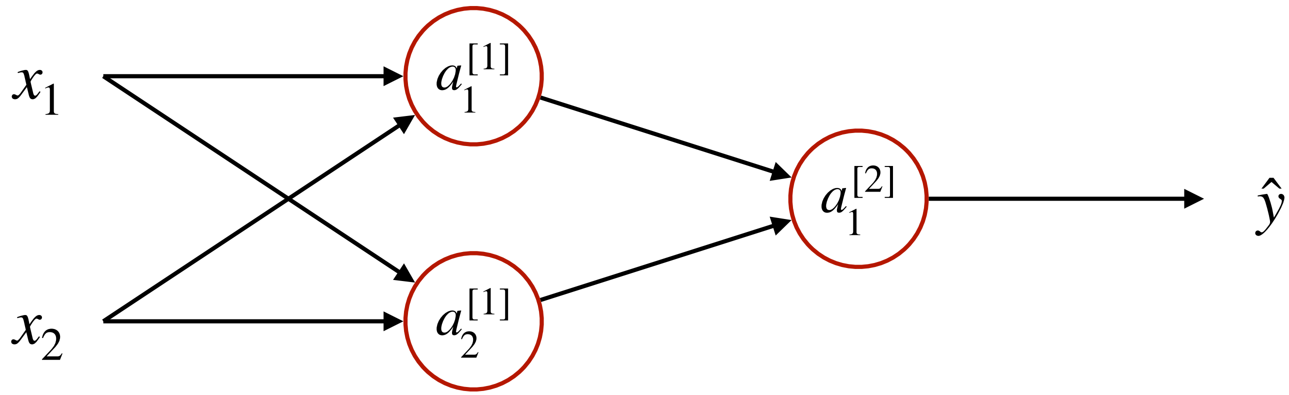
ReLu



leaky ReLu

Initialization

- What happens if you initialize weights to zero?



$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad W^{[2]} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$a_1^{[1]} = a_2^{[1]} \rightarrow \mathrm{d}z_1^{[1]} = \mathrm{d}z_2^{[1]} \rightarrow \mathrm{d}W^{[1]} = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

Invariances

- Quite often in classification problems there is a need for invariance under one or more transformations

  > handwriting: digits should have same classification irrespective of position (translation), size (scale), or pixel intensities of the image

  > speech recognition: invariant to non-linear warping along the time axis that preserve temporal ordering

Invariances

- Large sample set where all transformations are present
  - impractical: number of different samples grows exponentially with number of transformations
- Seek alternative approaches for adaptive models to exhibit required variances

Invariances

Four approaches:

1. Training set is augmented by transforming training patterns according to desired invariances

2. Add regularization terms to error function that penalizes changes in model output when input is transformed

3. Invariance built into pre-processing by extracting features invariant to required transformations

4. Build invariance property into structure of neural network (e.g., convolutional networks)