DD2437 – Artificial Neural Networks and Deep Architectures (annda)

Introduction to Lecture 2b
Multi-layer perceptron and backprop

Pawel Herman

Computational Science and Technology (CST)

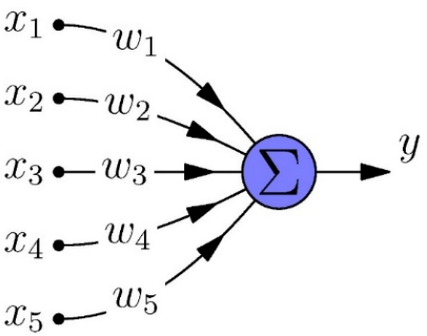
KTH Royal Institute of Technology

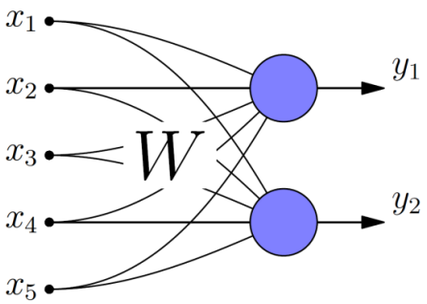
Outline

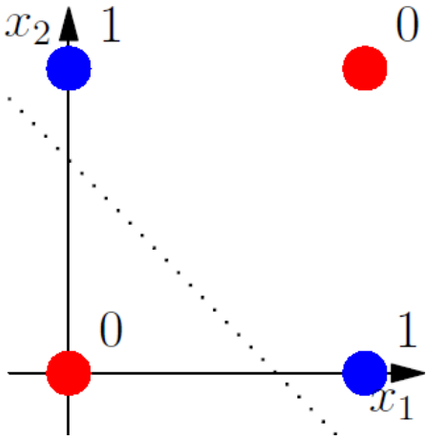
- Beyond a perceptron: multi-layer feedforward networks
- How do we train multi-layer perceptrons?
  - generalised delta rule
  - backprop algorithm
- Practicalities of backprop learning
- Special applications of an MLP
  - system identification
  - compression with an autoencoder

Single-layer perceptron, TLU

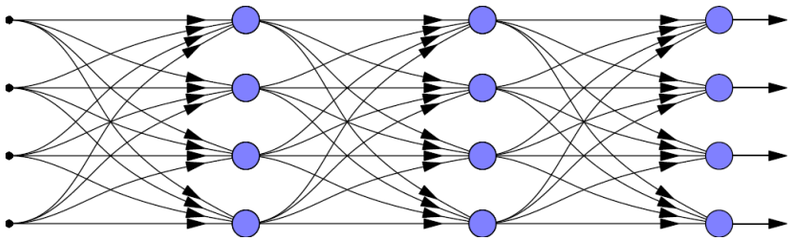
Concerns about the limited functionality of single-layer perceptron





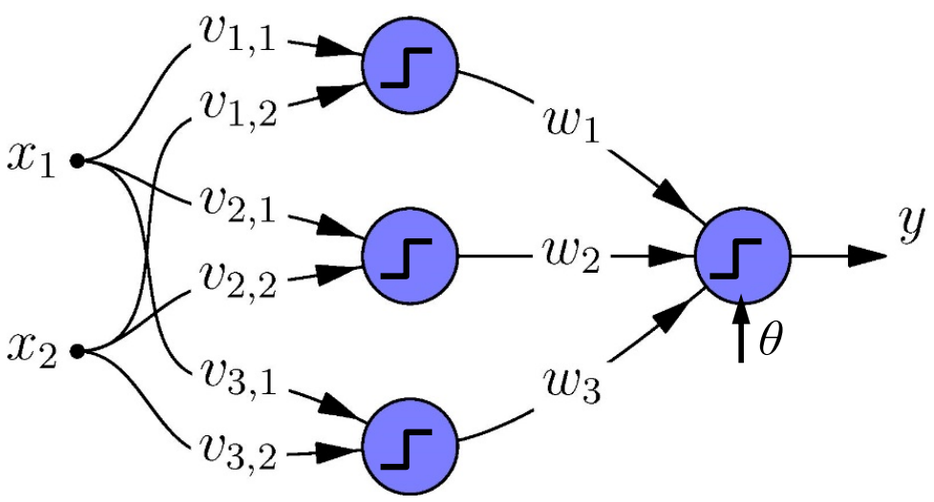


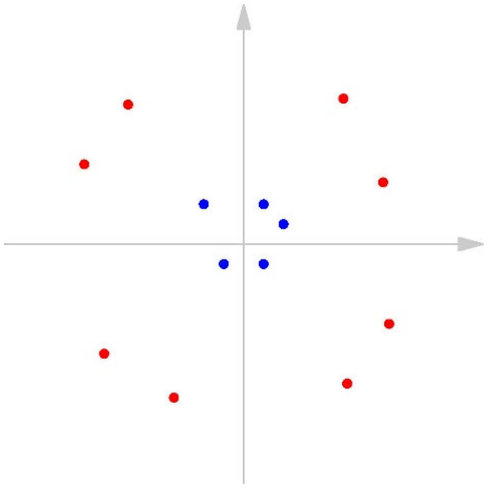
Cascading layers with linear units??

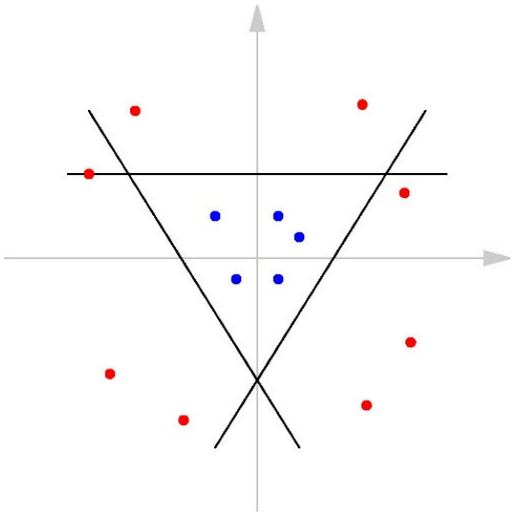


Multi-layer feed-forward networks

What can we compute with two-layer feed-forward networks with thresholded units?



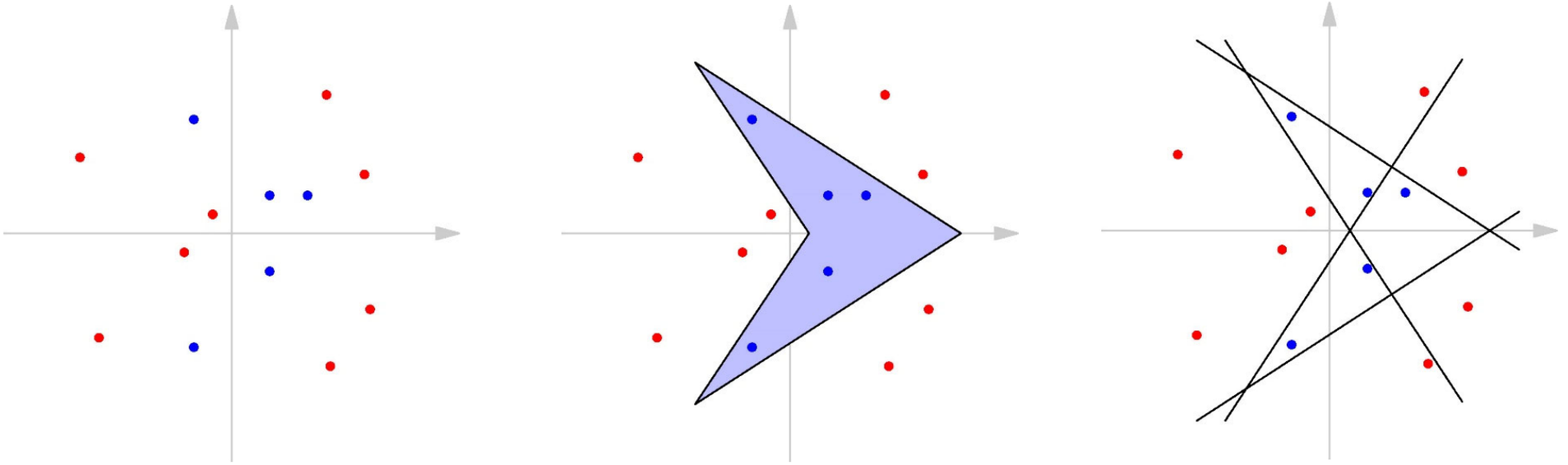




For $w_1 = w_2 = w_3 = 1$ and $\theta = 0.25$, the node in the output layer operates as AND-gate.

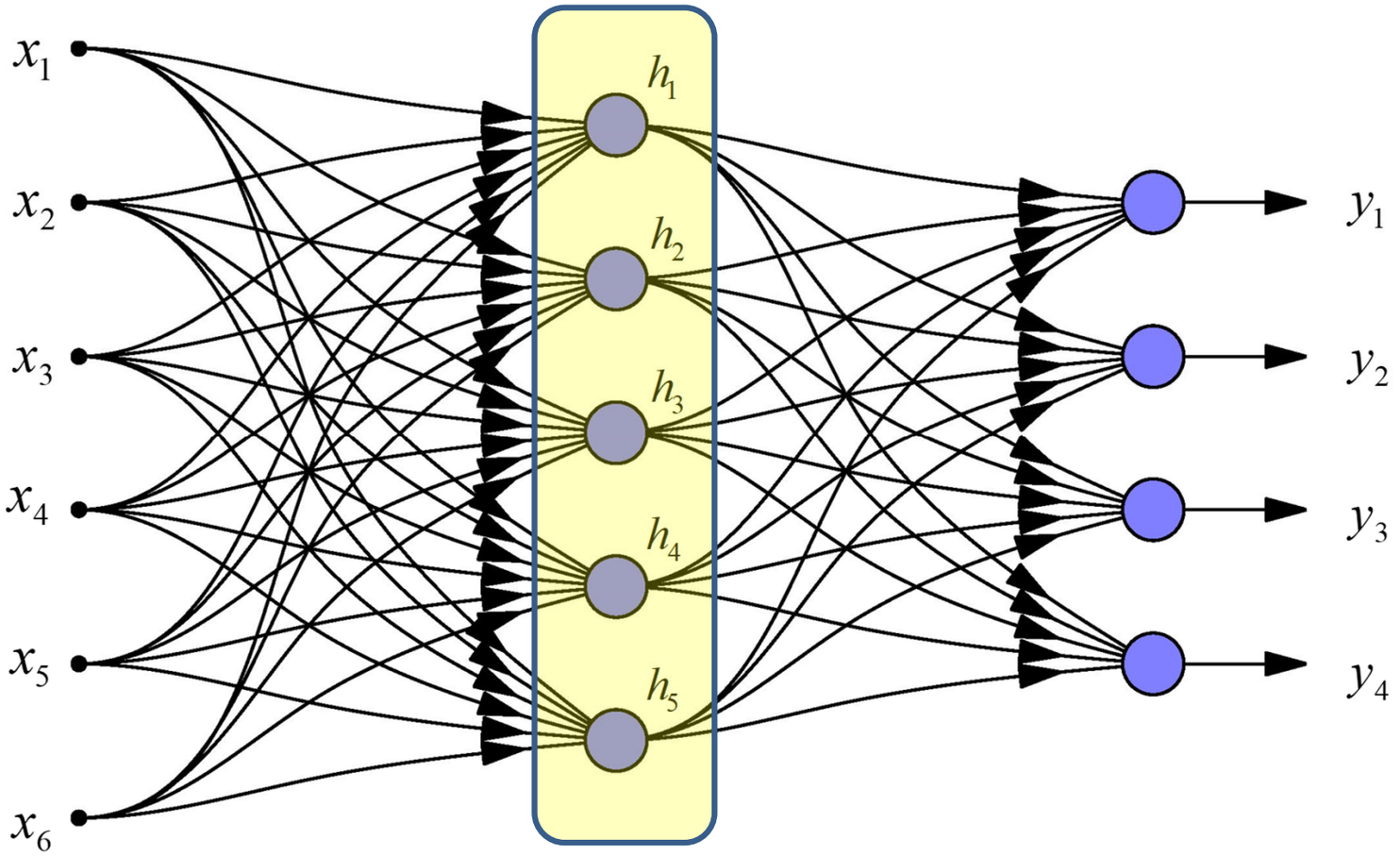
Multi-layer feed-forward networks

What can we compute with two-layer feed-forward networks with thresholded units?



Arbitrary complex areas can be extracted provided that there are enough hidden units

Multi-layer perceptron (MLP)



hidden layer(s)

Training multi-layer networks

But how do we train a multi-layer network?

Let's revisit two options:

- Perceptron learning

- Delta rule

Training multi-layer networks

But how do we train a multi-layer network?

Let's revisit two options:

- Perceptron learning

  We only know how to change weights from the hidden to the output layer based on the error (move weight vector nearer or away from data…… but the data is fed to the input layer) -> Does not work!

- Delta rule

  We need to measure the error before thresholding …. but it only works for the output layer (in hidden layers must be nonlinearity) -> Does not work!

Training multi-layer networks

But how do we train a multi-layer network?

We have a **dilemma** to solve:

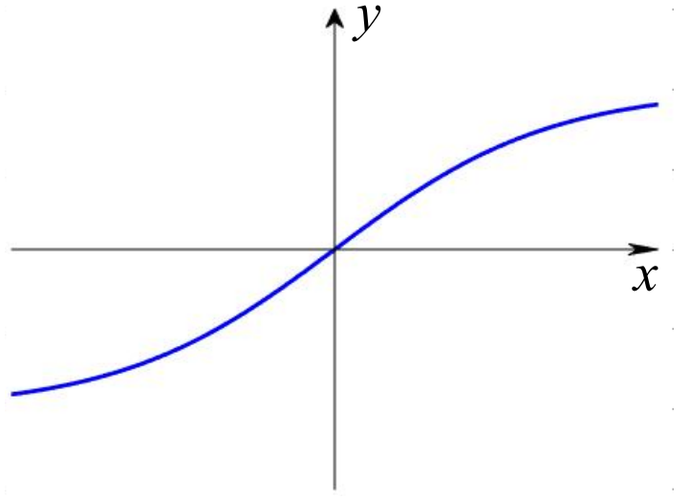
- Thresholding destroys information needed for learning (delta rule)
- Without thresholding we lose the advantage (computational functionality) of multiple layers

Solution

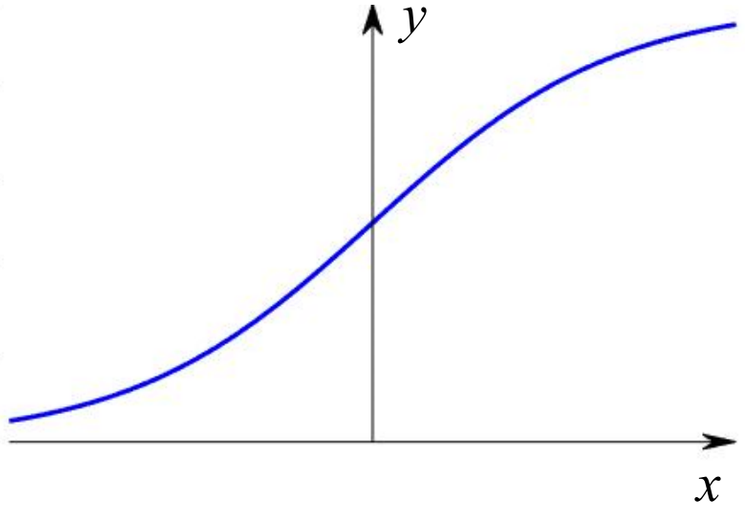
Use threshold-like but differentiable function (for delta)

Training multi-layer networks

Commonly used activation functions



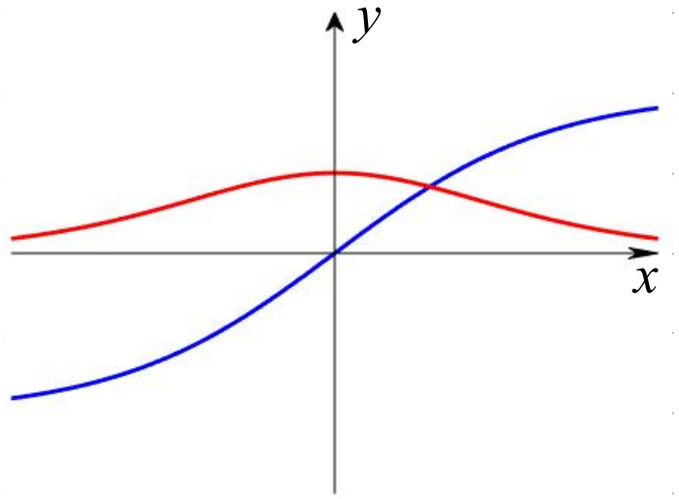
$$\varphi(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$



$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

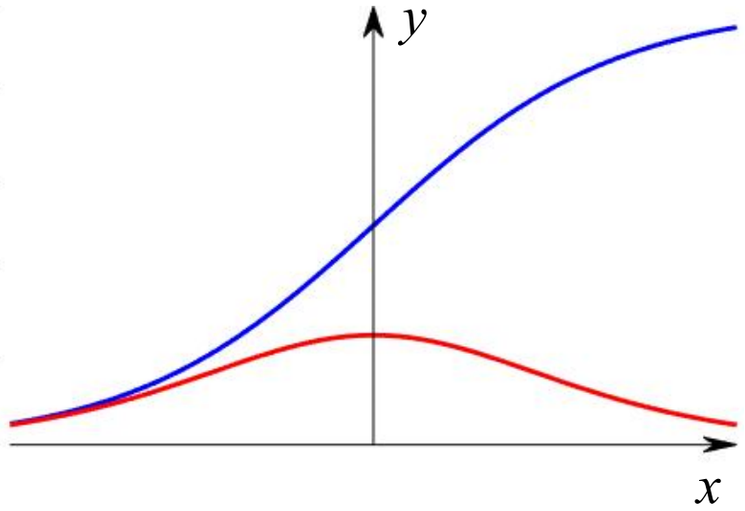
Training multi-layer networks

Commonly used activation functions and their derivatives



$$\varphi(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

$$\varphi'(x) = \frac{1 - \varphi^2}{2}$$

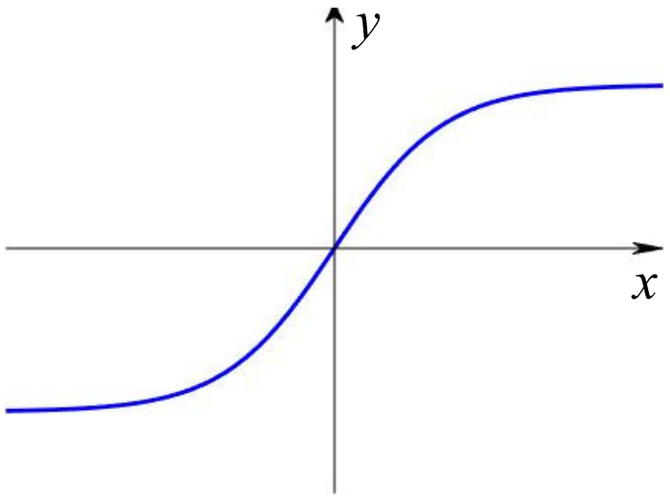


$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

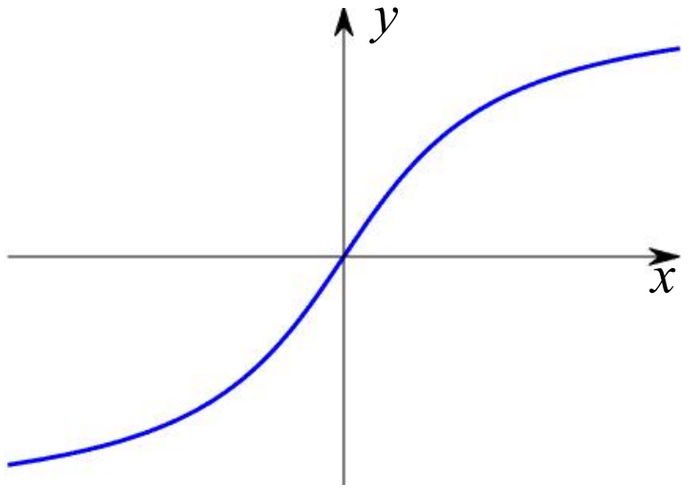
$$\varphi'(x) = \varphi(x)\left(1 - \varphi(x)\right)$$

Training multi-layer networks

Commonly used activation functions



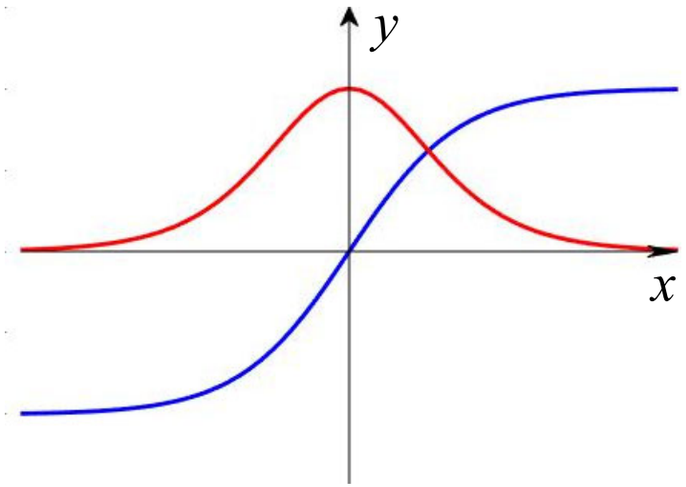
$$\varphi(x) = \tanh(x)$$

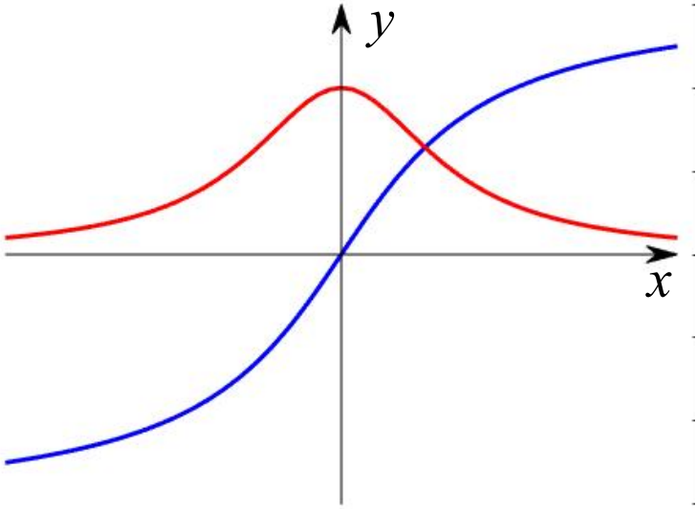


$$\varphi(x) = \arctan(x)$$

Training multi-layer networks

Commonly used activation functions and their derivatives





$$\varphi(x) = \tanh(x)$$

$$\varphi'(x) = 1 - \varphi^2$$

$$\varphi(x) = \arctan(x)$$

$$\varphi'(x) = \frac{1}{1 + x^2}$$

Generalised delta rule

1) Define/choose the cost or error function, ε

2) Minimise it using the principle of steepest descent

Generalised delta rule

1) Define/choose the cost or error function, ε

$$\varepsilon = \frac{1}{2}\left\|\vec{t} - \vec{y}\right\|^2 = \sum_k \left(t_k - y_k\right)^2, \qquad \vec{y} = [y_1, y_2, \ldots, y_k]^{\mathrm{T}}$$

2) Minimise it using the principle of steepest descent

$$\Delta\vec{w} = -\eta \frac{\partial \varepsilon(\vec{x}, \vec{w})}{\partial \vec{w}}$$

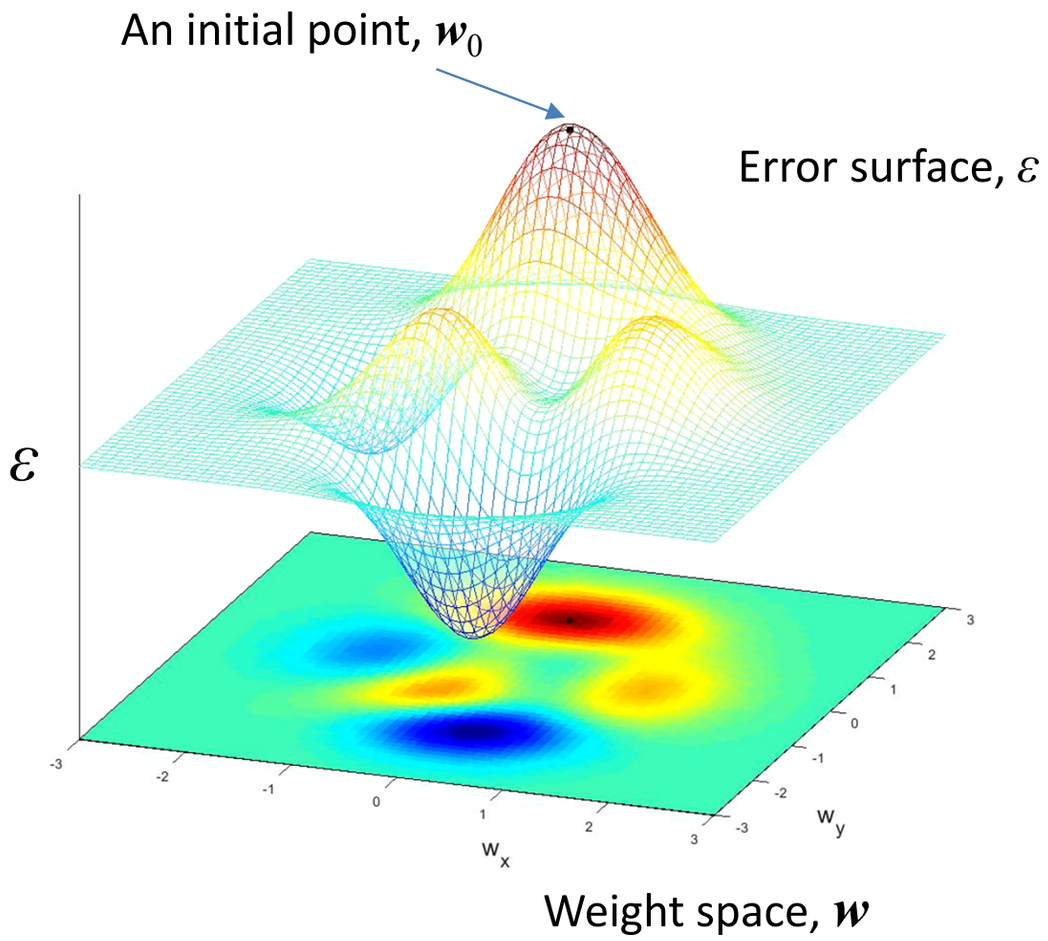
$$\Delta w_{kj} = -\eta \frac{\partial \varepsilon}{\partial w_{kj}} \qquad \Delta v_{ji} = -\eta \frac{\partial \varepsilon}{\partial v_{ji}}$$

Gradient descent

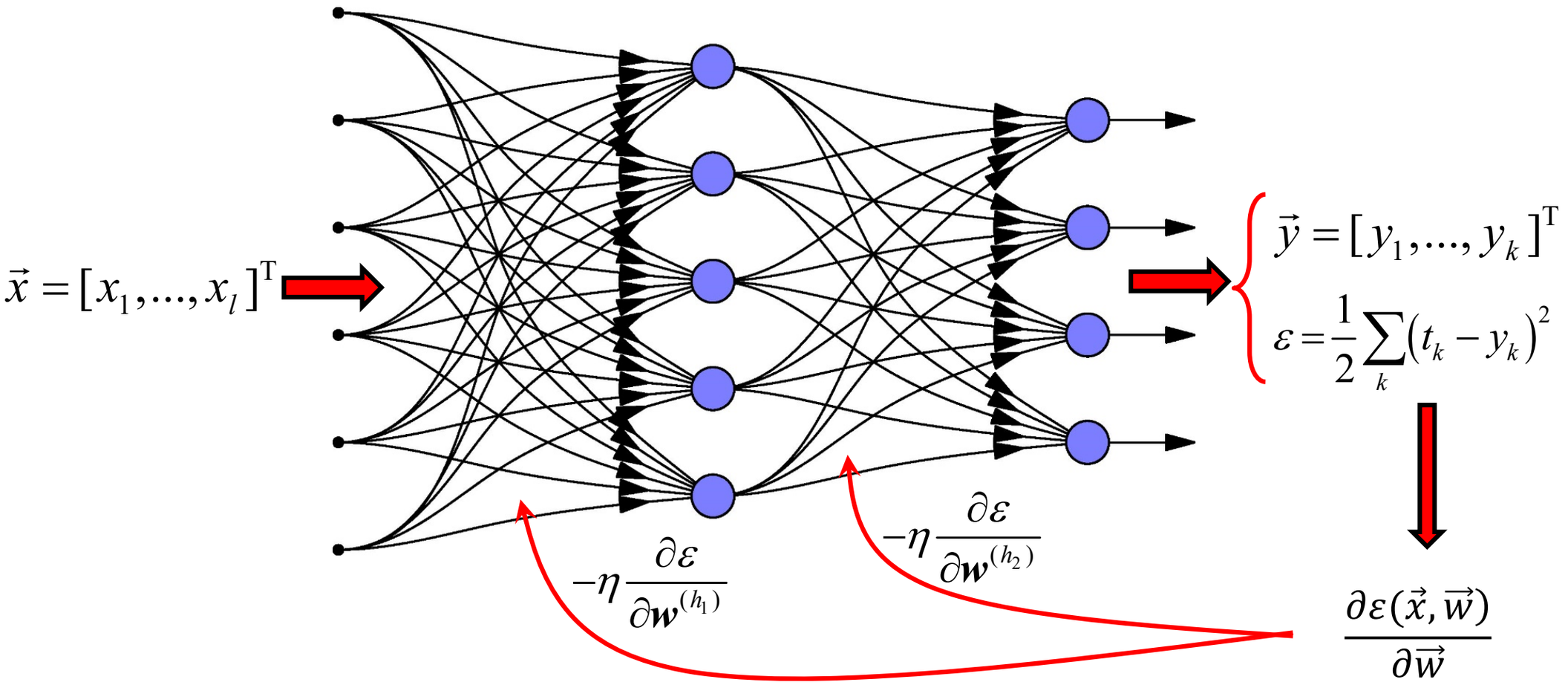
Steepest descent:

following *iteratively* the direction of the negative gradient of the error wrt. the weights (along all axes in the weight space)

$$-\frac{\partial \varepsilon}{\partial \boldsymbol{w}}$$

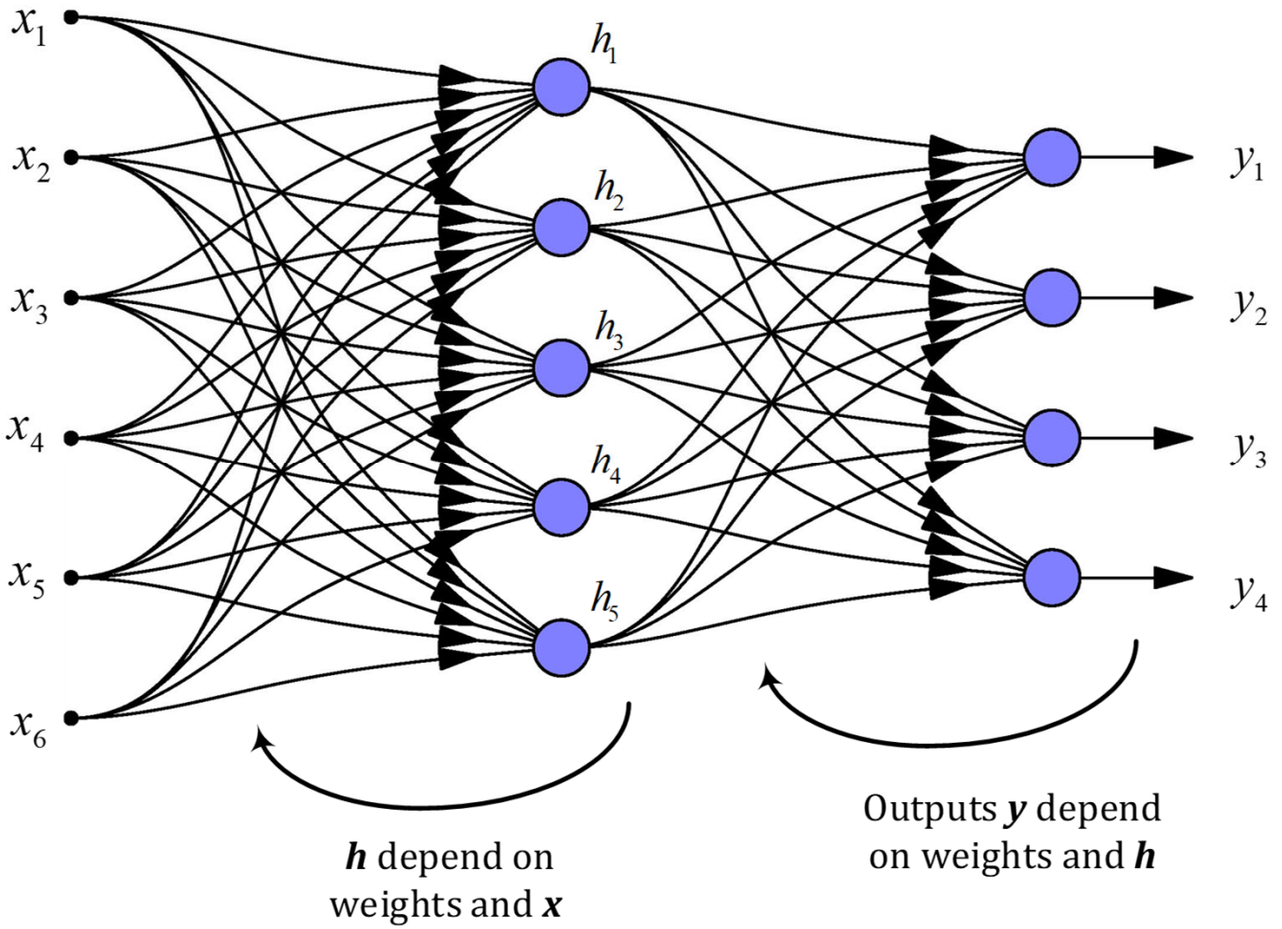


Backpropagation algorithm – overview

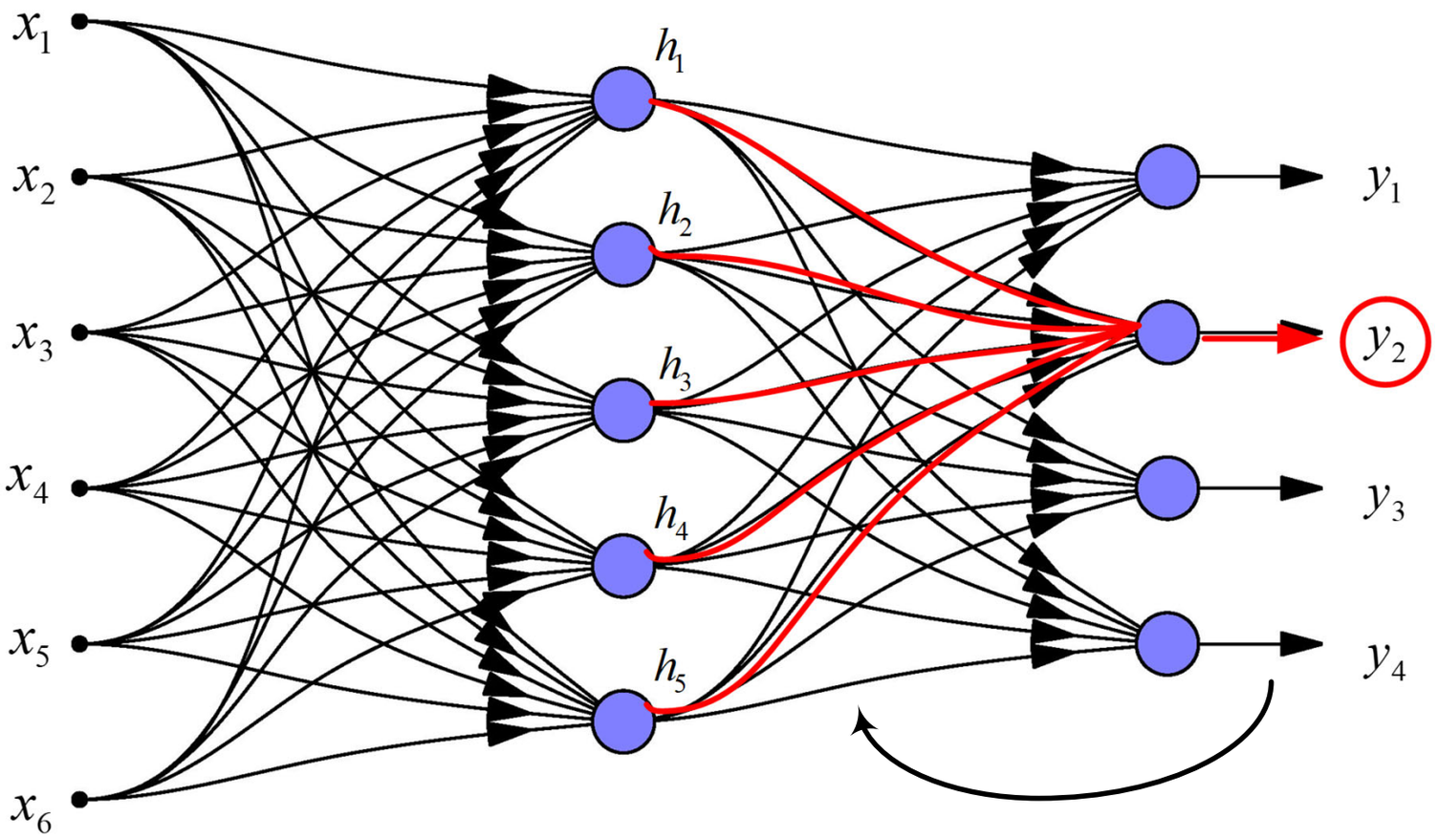


Backpropagation algorithm – overview

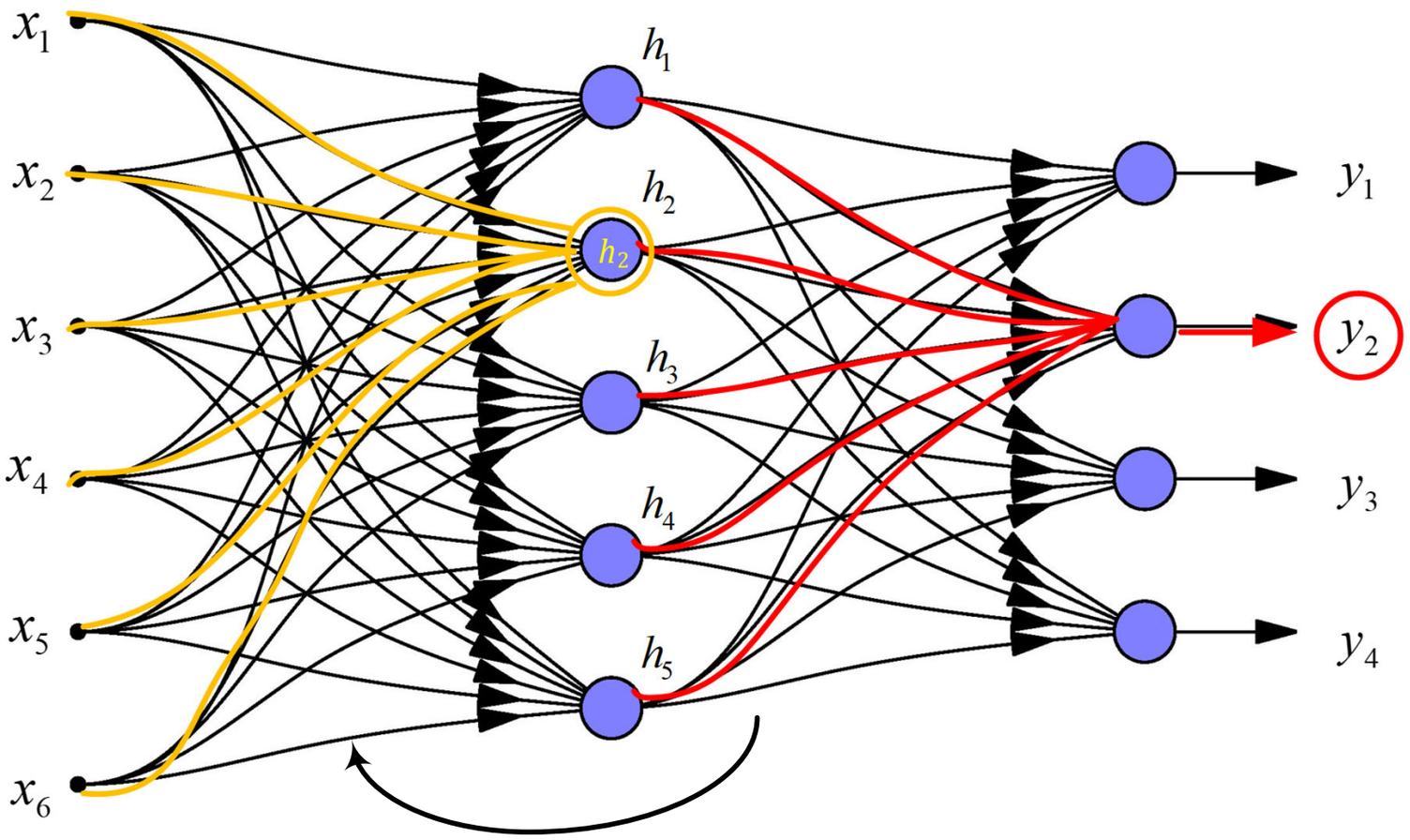
In fact, it is a ***chain*** of dependencies (sensitivity)



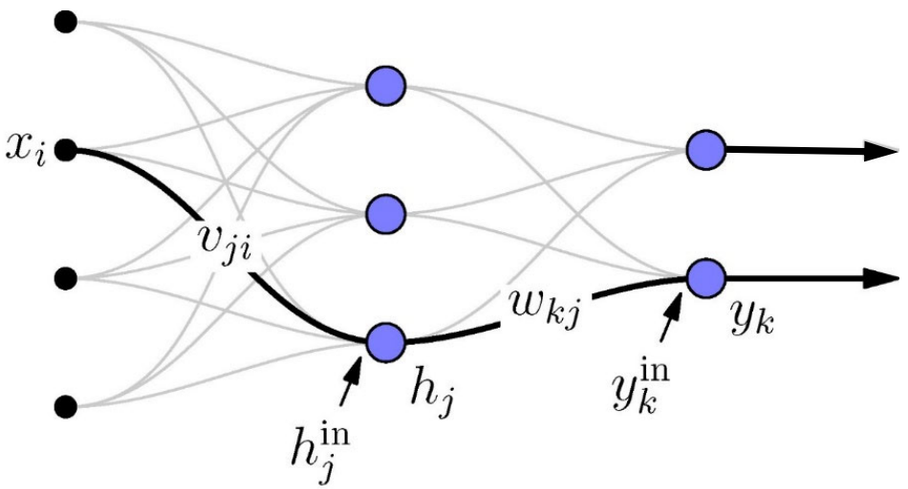
Backpropagation algorithm – overview



Backpropagation algorithm – overview



Backpropagation algorithm – summary



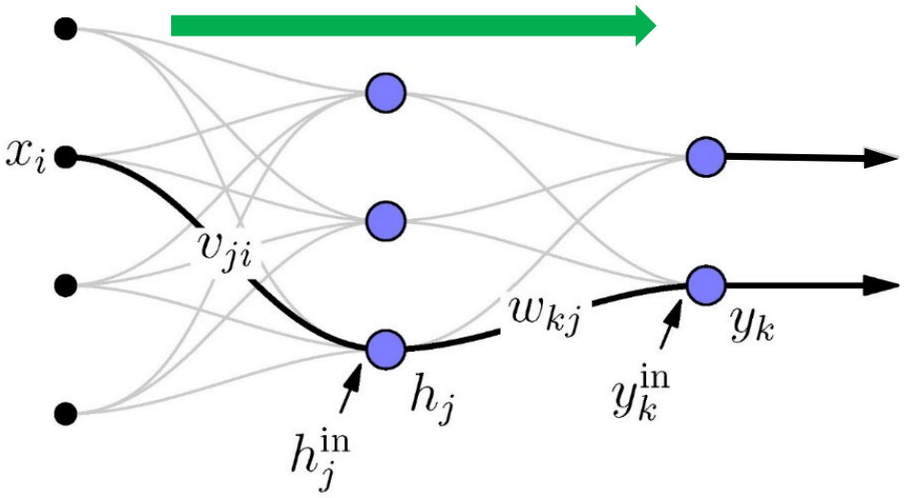
$$\frac{\partial \varepsilon}{\partial w_{kj}} = -\delta_k \, h_j \qquad \frac{\partial \varepsilon}{\partial v_{ji}} = -\delta_j \cdot x_i$$

where $\delta_k = (t_k - y_k) \cdot \varphi'\left(y_k^{\text{in}}\right)$ $\delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'\left(h_j^{\text{in}}\right)$

Backprop weight updates

$$\Delta w_{kj} = \eta \, \delta_k \, h_j \qquad \Delta v_{ji} = \eta \, \delta_j \, x_i$$

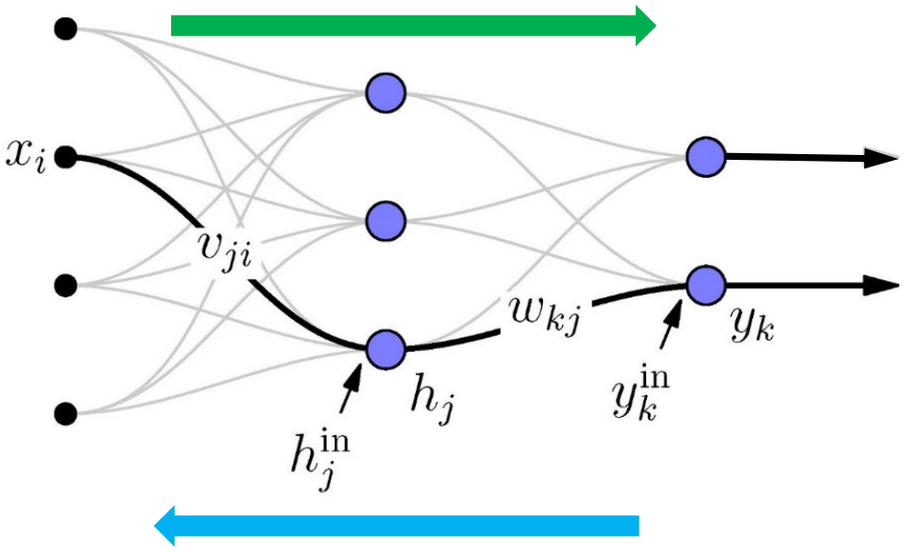
Backpropagation algorithm – summary



1) **Forward pass**: Compute all h_j and y_k

$$h_j = \varphi\left(\sum_i v_{ji}\, x_i\right) \qquad y_k = \varphi\left(\sum_j w_{kj}\, h_j\right)$$

Backpropagation algorithm – summary



1) Forward pass: Compute all h_j and y_k

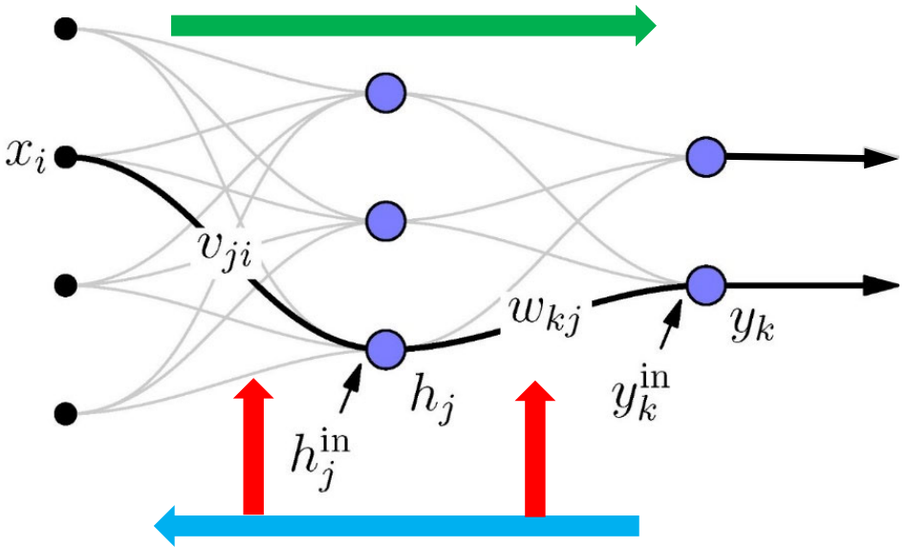
$$h_j = \varphi\left(\sum_i v_{ji}\, x_i\right) \qquad y_k = \varphi\left(\sum_j w_{kj}\, h_j\right)$$

2) **Backward pass**: Compute all δ_k and δ_j

$$\delta_k = (t_k - y_k) \cdot \varphi'\left(y_k^{\text{in}}\right)$$

$$\delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'\left(h_j^{\text{in}}\right)$$

Backpropagation algorithm – summary



1) Forward pass: Compute all h_j and y_k

$$h_j = \varphi\left(\sum_i v_{ji}\, x_i\right) \qquad y_k = \varphi\left(\sum_j w_{kj}\, h_j\right)$$

2) Backward pass: Compute all δ_k and δ_j

$$\delta_k = (t_k - y_k)\cdot\varphi'\left(y_k^{\text{in}}\right)$$

$$\delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'\left(h_j^{\text{in}}\right)$$

3) Updating:

$$\Delta w_{kj} = \eta\, \delta_k\, h_j \qquad \Delta v_{ji} = \eta\, \delta_j\, x_i$$

Backpropagation – problems, concerns

Problems with backprop

- Problems with convergence
  - slow convergence
  - getting stuck in local minima
- Many parameters must be tuned
- Challenging scaling
- Biologically unrealistic

Training MLPs with backprop – practicalities

Practical considerations

- sequential (online) vs batch mode
  - Sequential update is faster (though requires lower learning rate) especially for large data sets with redundancy
  - On-line scheme is less susceptible to getting stuck in local minima (noise)
  - However, sequential update does not implement a true gradient descent, it is referred to as *stochastic* gradient descent (SGD)

Training MLPs with backprop – practicalities

Practical considerations

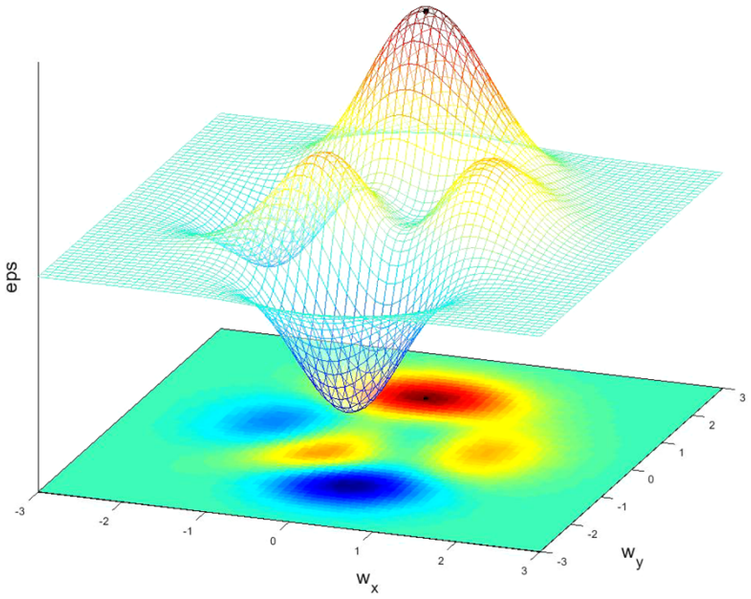
- sequential (online) vs batch mode
- weight initialisation
  - random initialization
  - Gaussian distribution $N(0,\sigma)$
  - uniform distribution with zero mean and $\sigma^2 = \frac{1}{fan\_in}$ (*heuristic*)

Training MLPs with backprop – practicalities

Practical considerations

- sequential (online) vs batch mode
- weight initialisation
- learning rate adaptation
  - too low η makes learning very slow
  - too large η makes the weights overshoot the error minimum
  - adaptive schemes:

    $\eta(t)=\eta(1)/t$ or $\eta(t)=\eta(1)/(1+t/\tau)$
  - other heuristic adjustments for individual learning rates (depending on the sequence of the sign of the gradient)



Training MLPs with backprop – practicalities

Practical considerations

- sequential (online) vs batch mode
- weight initialisation
- learning rate adaptation + momentum

$$\Delta\vec{w}^{(new)} = \beta\,\Delta\vec{w}^{(old)} - (1-\beta)\frac{\partial\mathcal{E}}{\partial\vec{w}}$$

Training MLPs with backprop – practicalities

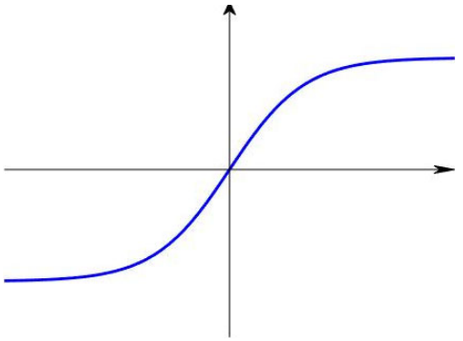
Practical considerations

- sequential (online) vs batch mode
- weight initialisation
- learning rate adaptation + momentum
- when to stop training
  - fixed number of epochs
  - error convergence
  - early stopping and other heuristic schemes

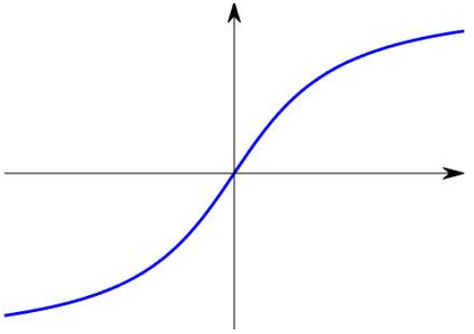
Training MLPs with backprop – practicalities

Practical considerations

- sequential (online) vs batch mode
- weight initialisation
- learning rate adaptation + momentum
- when to stop training
- faster convergence with anti-symmetric activation function, i.e. $f(-x)=-f(x)$



$$\varphi(x) = \tanh(x)$$



$$\varphi(x) = \arctan(x)$$

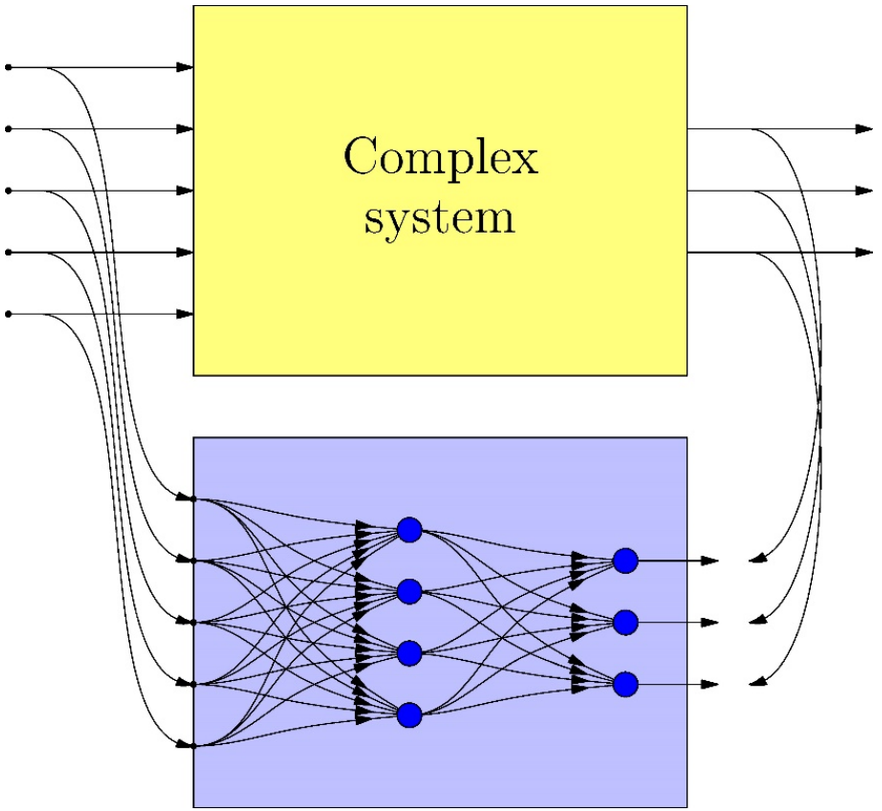
Training MLPs with backprop – practicalities

Practical considerations

- sequential (online) vs batch mode
- weight initialisation
- learning rate adaptation + momentum
- when to stop training
- faster convergence with anti-symmetric activation function
- off-setting target values to minimize the risk of sigmoid saturation
- maximising information content of data samples
- data preprocessing – normalization (e.g. mean removal, decorrelation)
- add noise during training
- hyperparameter selection

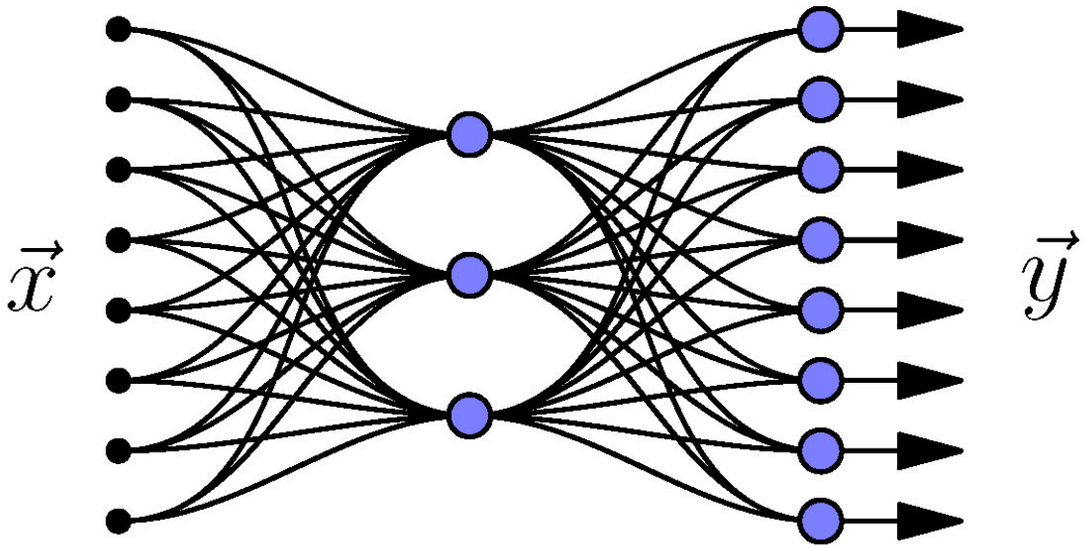
Special applications of MLPs

System identification: “mimic” an existing system



Special applications of MLPs

Autoencoder for data compression



hour-glass form

Train using backprop with $\vec{y} = \vec{x}$ (auto-association); using a supervised learning scheme for “unsupervised” (label free) problems

Forces the network to find a compact encoding scheme for the input patterns.