Neural Networks and Learning Systems
TBMI26 / 732A55
2021

Lecture 3

Supervised learning – Neural networks

Magnus Borga
magnus.borga@liu.se

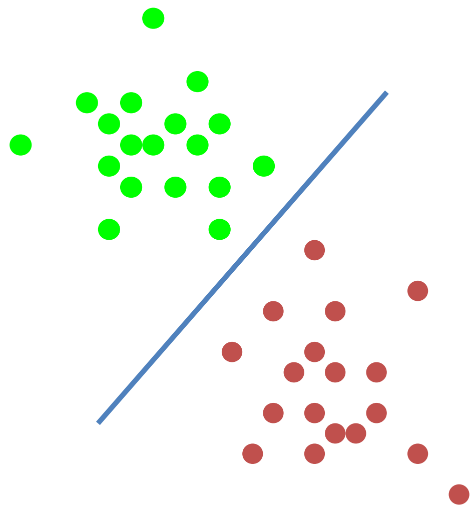
Recap - Supervised learning

- **Task:** Learn to predict/classify new data from labelled examples.
- **Input:** Training data examples $\{\mathbf{x}_i, y_i\}$ i=1...N, where $\mathbf{x}_i$ is a feature vector and y_i is a class label.
- **Output:** A function $f(\mathbf{x}; w_1,\ldots,w_k)$ that can predict the class label of $\mathbf{x}$.

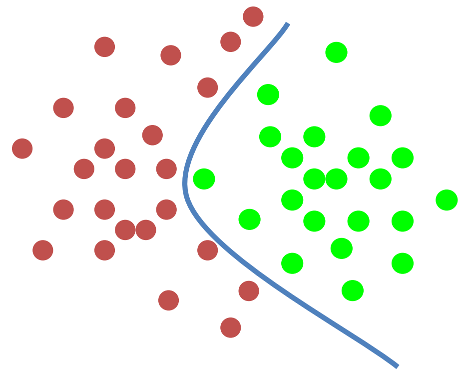
Find a function f and adjust the parameters $w_1,\ldots,w_k$ so that new feature vectors are classified correctly. Generalization!!

Linear separability

Linearly separable

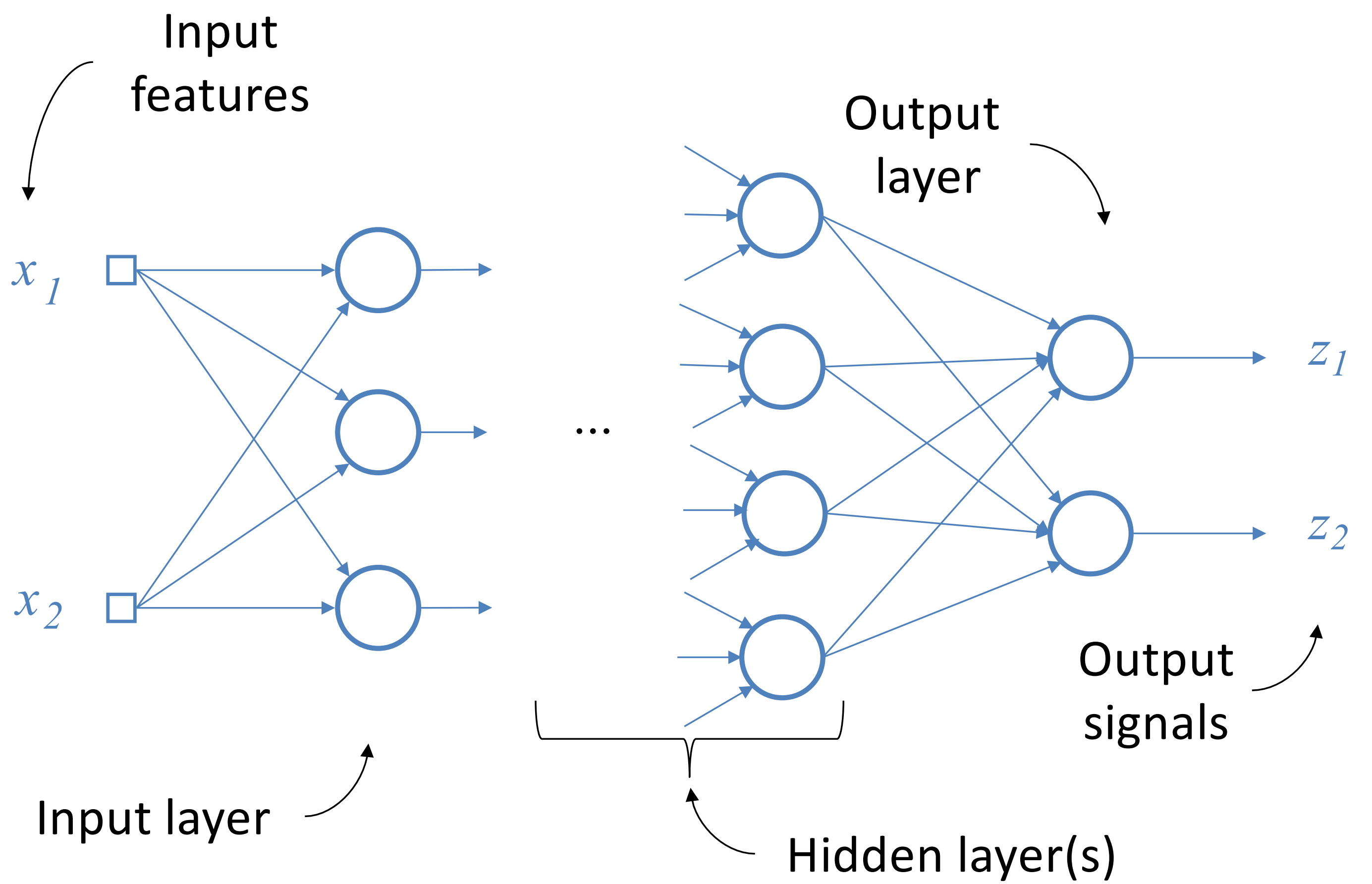


Non-linearly separable

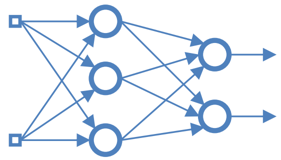


Neural Networks

a.k.a the Multi-layer Perceptron



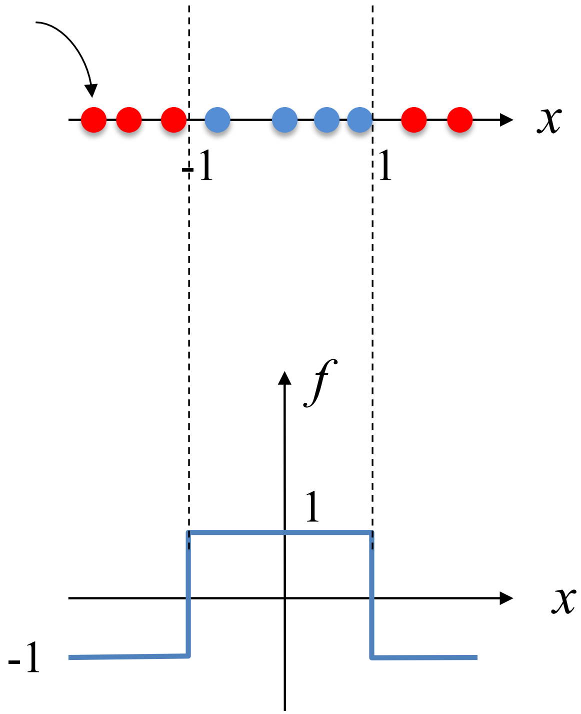
History of neural networks



- 1960's: Large enthusiasm around the perceptron and "connectionism" (Frank Rosenblatt).
- 1969: Limitations of the perceptron made clear in a paper by Minsky & Papert, e.g., the XOR problem.
- "Winter period" – little research
- 1980's: Revival of connectionism and neural networks:
  - Multi-layer perceptrons can solve nonlinear problems (this was known before, but not how to train them!)
  - Back-propagation training algorithm
- 1990's: Reduced interest, other methods seemed more promising
- 2010's: Renewed interest – "Deep learning"

A simple 1D example

Training samples with only one feature value!

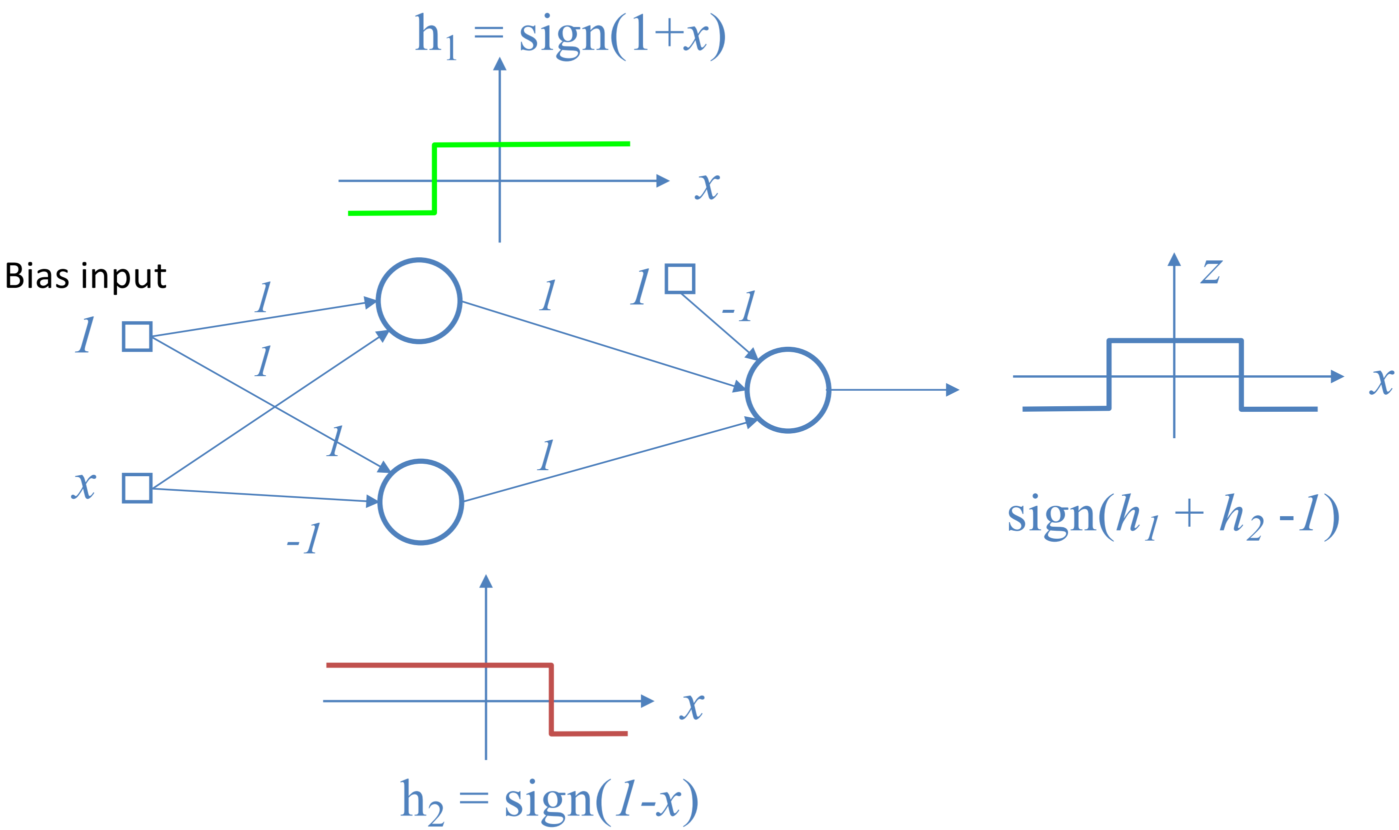


Not separable with a linear function!

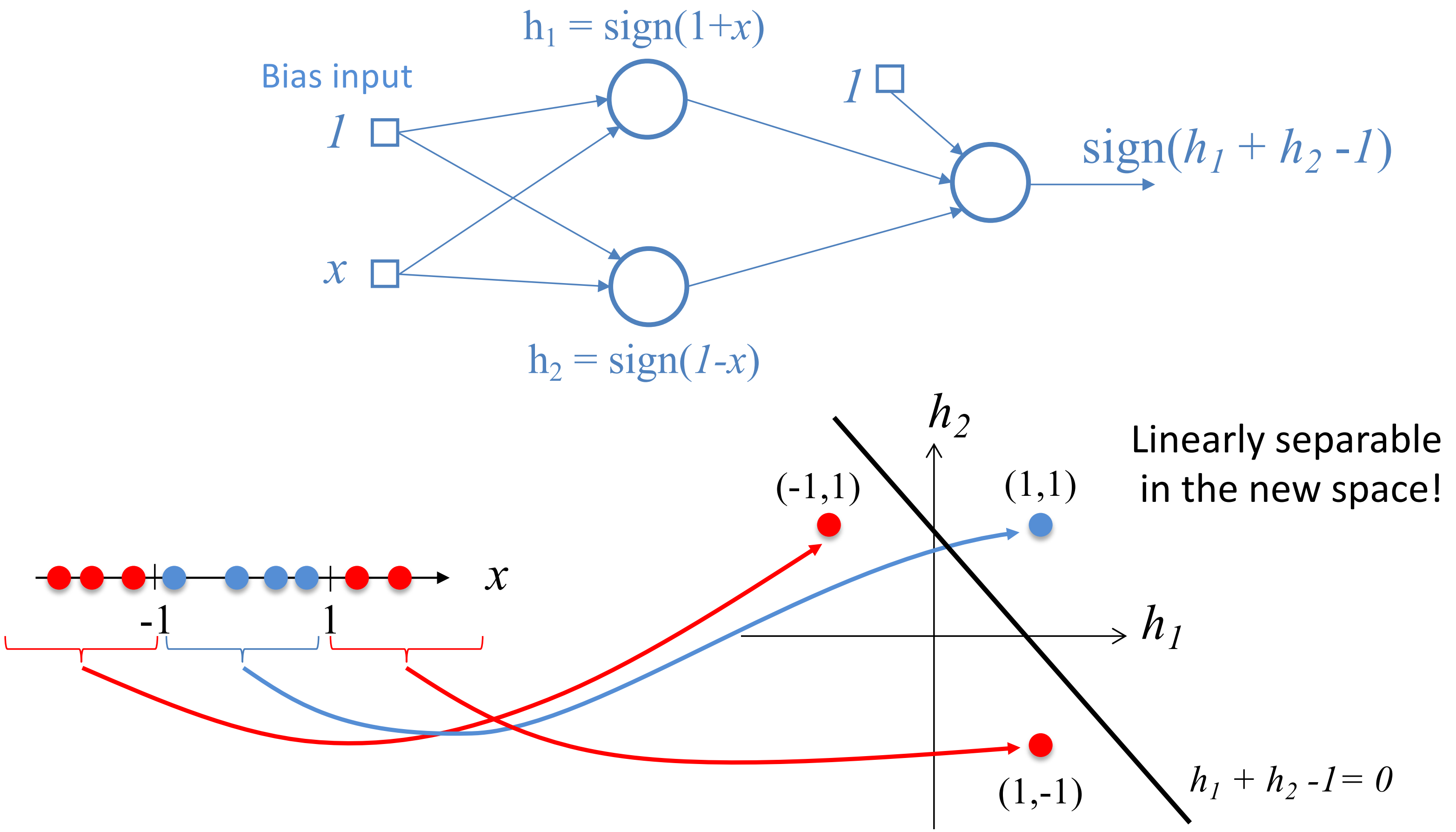
But with a nonlinear function!

$$f(x; w_0, \ldots, w_n) = \begin{cases} -1 & |x| > 1 \\ 1 & |x| < 1 \end{cases}$$

Example solution



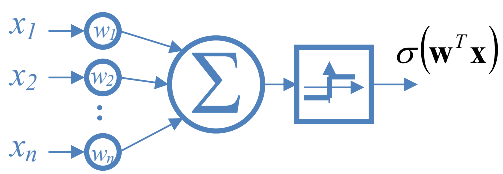
Nonlinear mapping to a new feature space!



Key: The hidden layer(s)

- The output layer requires linear separability. The purpose of the hidden layers is to make the problem linearly separable!
- **Cover's theorem (1965):** The probability that classes are linearly separable increases when the features are nonlinearly mapped to a higher-dimensional feature space.

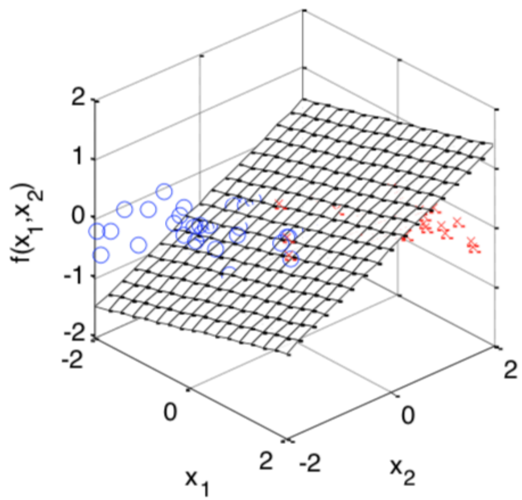
The Perceptron Revisited



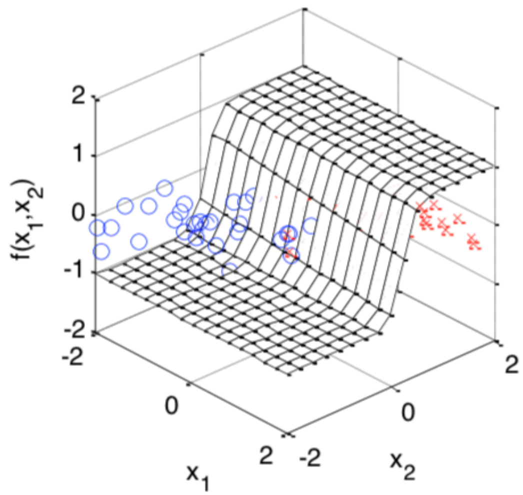
Minimize the following loss function

$$\varepsilon(\mathbf{w}) = \sum_{i=1}^{N} \left(\sigma\left(\mathbf{w}^T\mathbf{x}_i\right) - y_i\right)^2$$

$$\sigma\left(\mathbf{w}^T\mathbf{x}\right) = \mathbf{w}^T\mathbf{x}$$



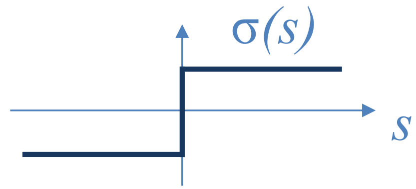
$$\sigma\left(\mathbf{w}^T\mathbf{x}\right) = \tanh\left(\mathbf{w}^T\mathbf{x}\right)$$



Nonlinear activation functions

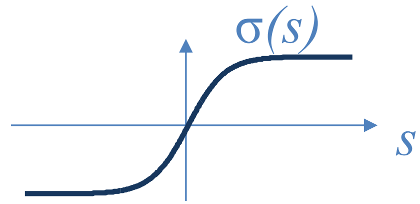
- Step/sign function

Not differentiable – cannot be optimized!
(by gradient search)



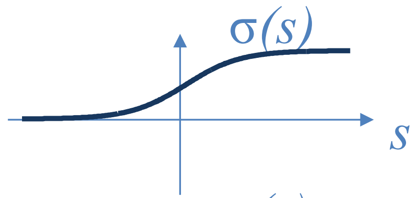
- Hyperbolic tangent

$$\sigma(s) = \tanh(s) \quad \sigma' = 1 - \tanh^2(s) = 1 - \sigma^2$$



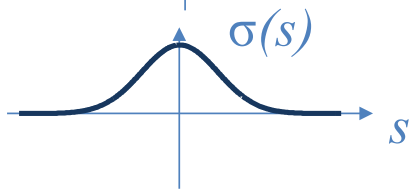
- The Fermi-function

$$\sigma(s) = \frac{1}{1+e^{-s}} \quad \sigma' = \sigma(1-\sigma)$$



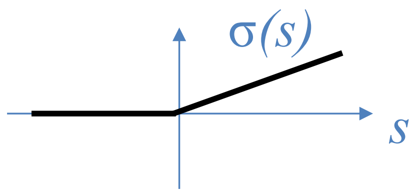
- Gaussian function

$$\sigma(s;\gamma) = e^{-\frac{s^2}{\gamma^2}} \quad \sigma'(s;\gamma) = -\frac{2s}{\gamma}\sigma$$



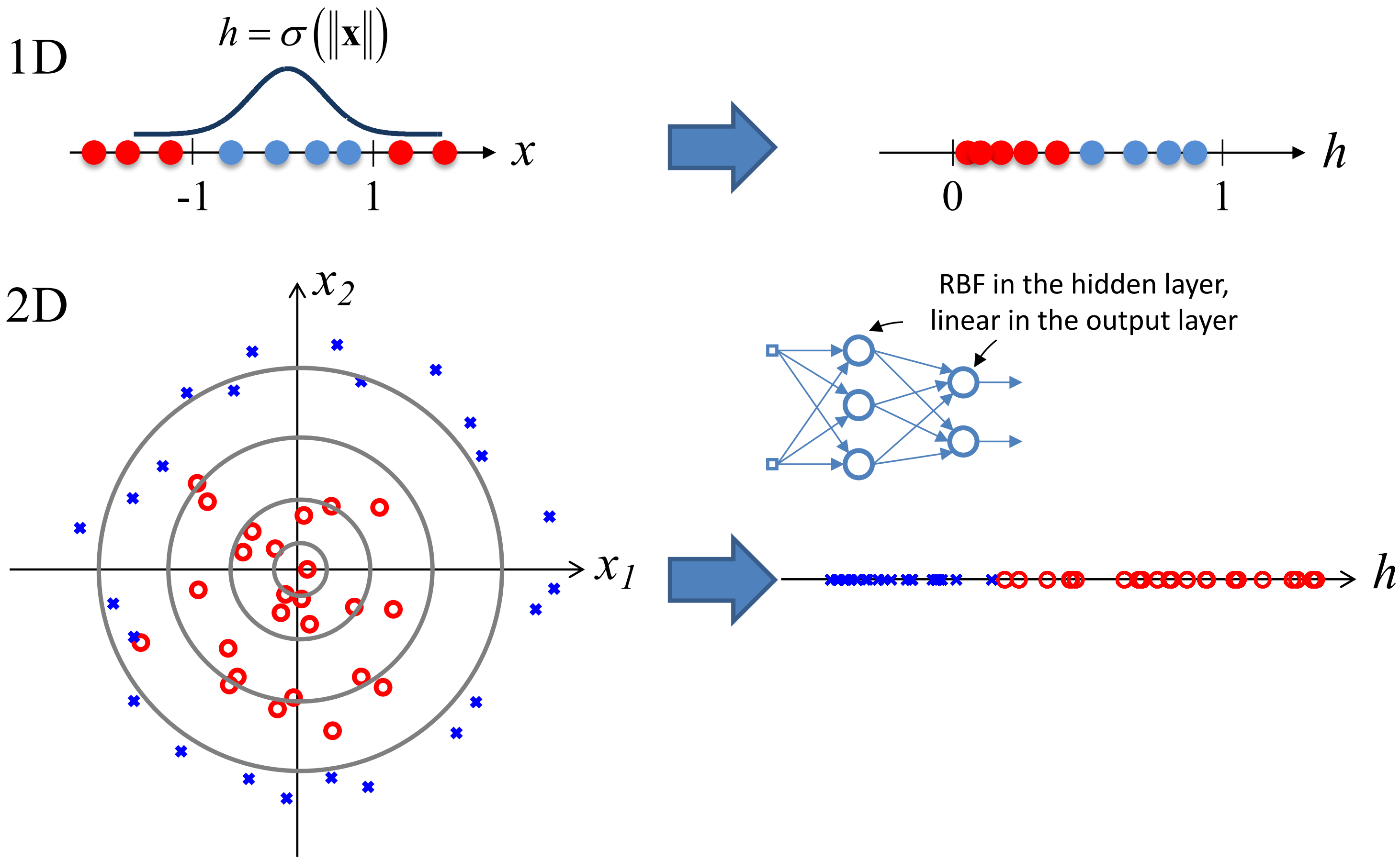
- Rectifying Linear Unit (ReLU)

$$\sigma(s) = \max(0, s) \quad \sigma' = \begin{cases} 0, & s < 0 \\ 1, & s \geq 0 \end{cases}$$



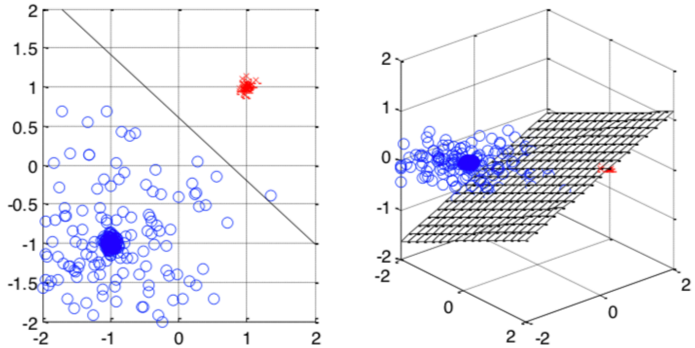
Example - Radial Basis Functions

For example a Gaussian



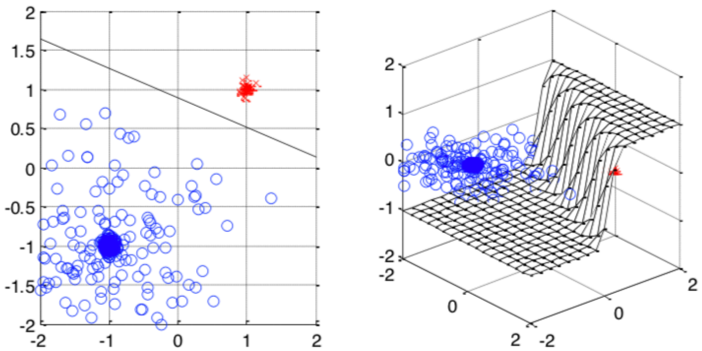
Example - tanh

$$\sigma(s) = s$$



Same as in lecture 2!

$$\sigma(s) = \tanh(s)$$



Updated optimization algorithm

$$\varepsilon(\mathbf{w}) = \sum_{i=1}^{N} \left(\sigma\left(\mathbf{w}^T \mathbf{x}_i\right) - y_i\right)^2$$

$$\frac{\partial \varepsilon}{\partial \mathbf{w}} = 2\sum_{i=1}^{N} \left(\sigma\left(\mathbf{w}^T \mathbf{x}_i\right) - y_i\right)\sigma'\left(\mathbf{w}^T \mathbf{x}_i\right)\mathbf{x}_i$$

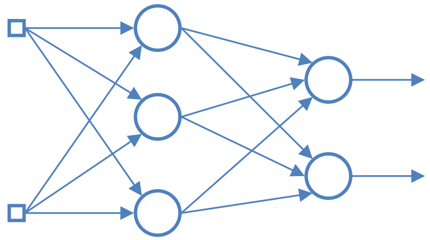
Gradient descent:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{\partial \varepsilon}{\partial \mathbf{w}} \quad (\text{Eq. } 1)$$

Algorithm:

1. Start with a random **w**
2. Iterate Eq. 1 until convergence

Training multi-layer neural networks



Loss function

\# training examples   # output nodes

$$\varepsilon(\mathbf{w}) = \sum_{k=1}^{K}\sum_{m=1}^{M}\left[y_{mk} - z_{mk}(\mathbf{w})\right]^2$$

all weights   desired output   actual output

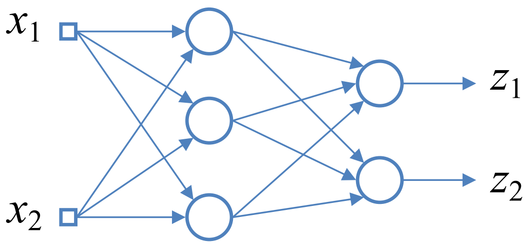
Stochastic gradient descent

Update using one (K=1) training example

$$\varepsilon(\mathbf{w}) = \sum_{m=1}^{M} \left[y_m - z_m(\mathbf{w}) \right]^2$$

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \frac{\partial \varepsilon}{\partial w_{ij}}$$

From node i to node j
in a layer



The chain rule

$$f\left(g\left(x\right)\right) \qquad\qquad f\left(g\left(x\right), h\left(x\right)\right)$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} \qquad\qquad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial x}$$

Example:

$$f(x; w) = \sigma(wx) \rightarrow \frac{\partial f}{\partial w} = \frac{\partial f}{\partial s}\frac{\partial s}{\partial w} = \sigma'(x, w) \cdot x$$

Automatic Differentiation

- Any algorithm that is defined by a sequence of arithmetic operations can be automatically differentiated by repeatedly applying the chain rule!
- This is the basis for error back propagation

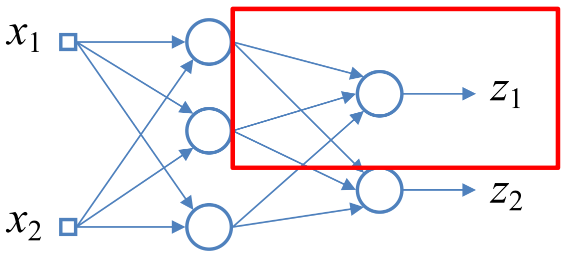
- (Note that this is different from numeric and analytical differentiation)

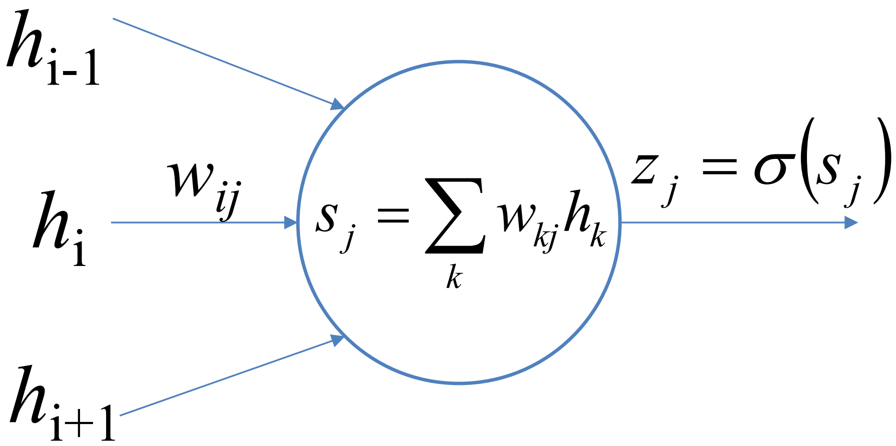
The error back propagation algorithm

- an exercise of the chain rule!

$$\frac{\partial \varepsilon}{\partial w_{ij}} = \frac{\partial \varepsilon}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial \varepsilon}{\partial z_j}\frac{\partial z_j}{\partial s_j}\frac{\partial s_j}{\partial w_{ij}}$$

$$\varepsilon(\mathbf{w}) = \sum_{m=1}^{M}\left[y_m - z_m(\mathbf{w})\right]^2$$





Back propagation, cont.

$$\varepsilon(\mathbf{w}) = \sum_{m=1}^{M} \left[y_m - z_m(\mathbf{w}) \right]^2 \qquad \frac{\partial \varepsilon}{\partial w_{ij}} = \frac{\partial \varepsilon}{\partial z_j} \frac{\partial z_j}{\partial s_j} \frac{\partial s_j}{\partial w_{ij}}$$

$$\frac{\partial \varepsilon}{\partial z_j} = -2\left(y_j - z_j\right)$$

$$\frac{\partial z_j}{\partial s_j} = \sigma'\left(s_j\right) = 1 - \sigma\left(s_j\right)^2 = 1 - z_j^2$$

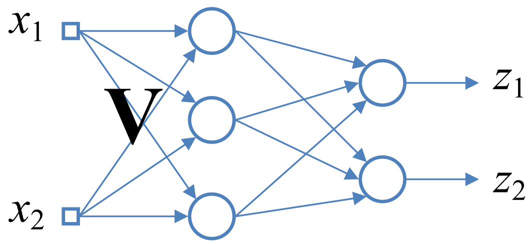
If σ(s) = tanh(s) is used!

$$\frac{\partial s_j}{\partial w_{ij}} = h_i$$

(input *i* to unit *j*)

Updating the hidden layer(s)

$$\frac{\partial \varepsilon}{\partial v_{ij}} = ?$$



$$\varepsilon(\mathbf{v}) = \sum_{m=1}^{M} \left[y_m - z_m(\mathbf{v}) \right]^2$$

A weight in a hidden layer affects **all** output nodes!

$$\varepsilon\left(z_1(\mathbf{v}), \ldots, z_M(\mathbf{v})\right)$$

$$\frac{\partial \varepsilon}{\partial v_{ij}} = \sum_{k=1}^{M} \frac{\partial \varepsilon}{\partial z_k} \frac{\partial z_k}{\partial v_{ij}} = \ldots.$$ Exercise!

Chain rule! Continue expanding!

Back propagation – Summary

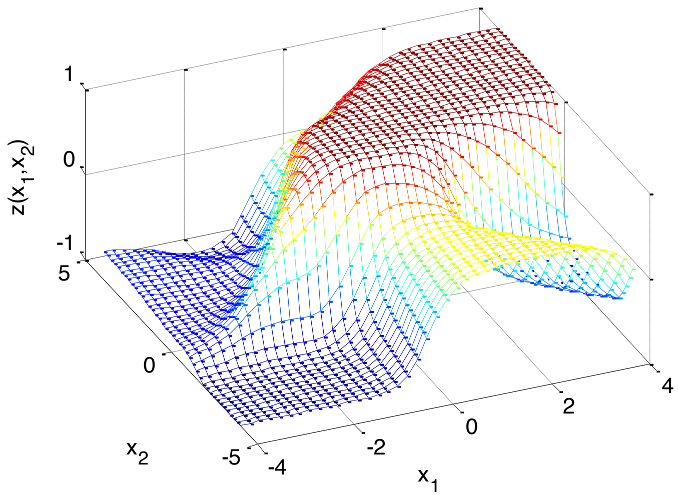
- Two phases:
  - Forward propagation: Propagate a training example through the network
  - Backward propagation: Propagate the error relative to the desired output backwards in the net and update parameter weights.
- Batch update: update after all examples have been presented.

$$\Delta w_{ij} = -\eta \sum_{k=1}^{K} \frac{\partial \varepsilon(k)}{\partial w_{ij}}$$

Decision boundaries

Neural networks can produce
very complex class boundaries!

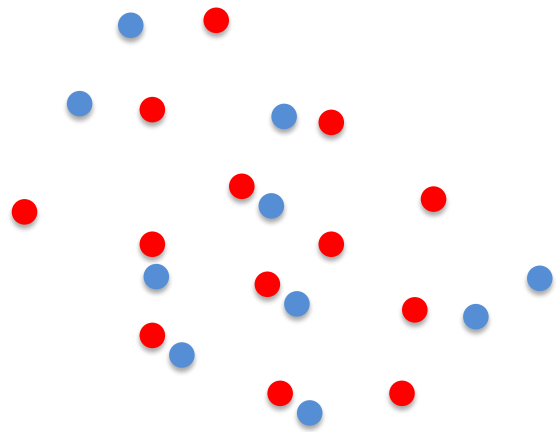
$$z(x_1, x_2)$$



$$f(x_1, x_2) = sign(z(x_1, x_2))$$

Note - Magic is not possible!

No neural network, however complex,
can separate inseparable classes!



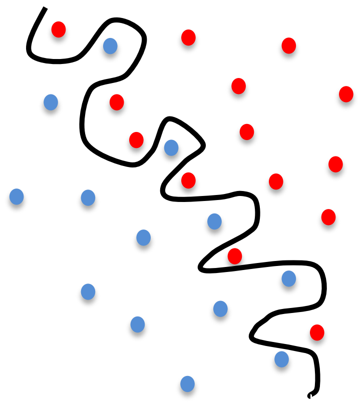
Finding and extracting suitable features are
the critical problems in machine learning!

Pros and cons of neural networks

- A multi-layer neural network can learn any class boundaries.
- The large number of parameters is a problem:
  - Local optima → suboptimal performance
  - Overfitting → poor generalization
  - Slow convergence → long training times

Overfitting

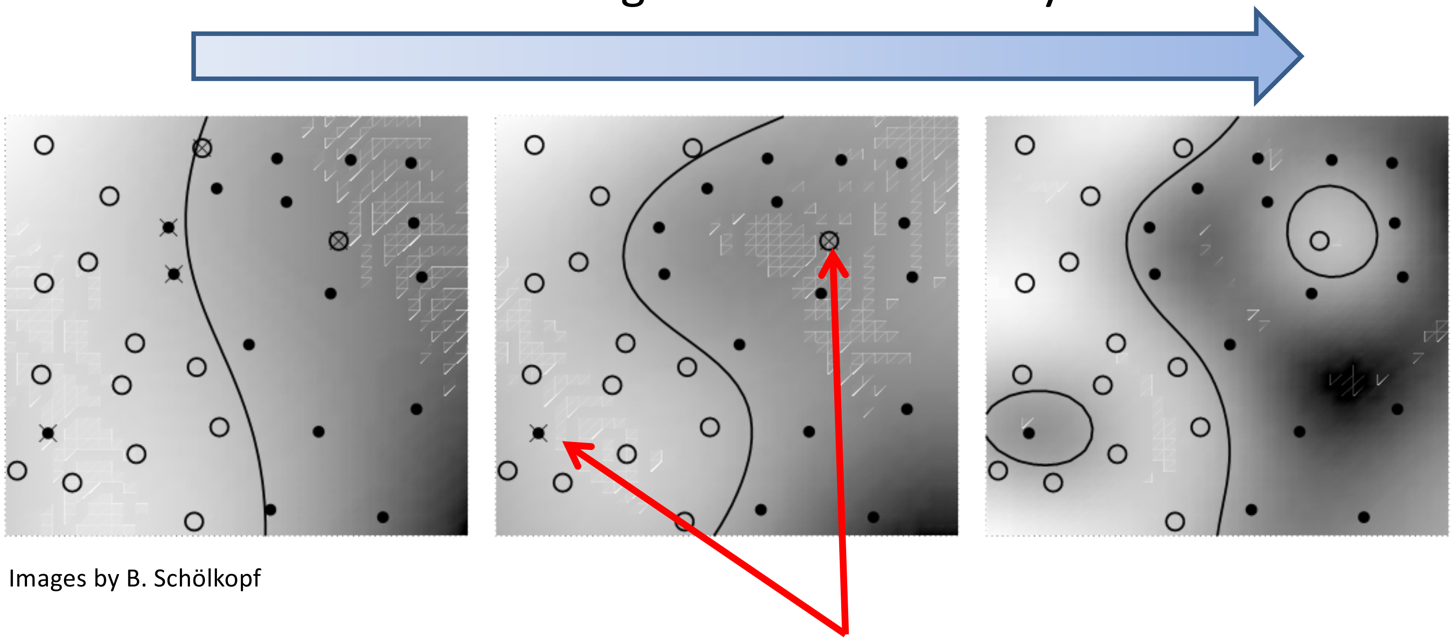
- The large number of parameters makes it possible to produce overly complicated boundaries.



- A too good fit to the training data can perform poorly for new cases, i.e. worse generalization properties!

Overfitting – Example

Increasing classifier flexibility

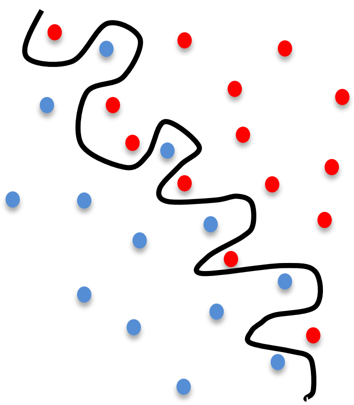


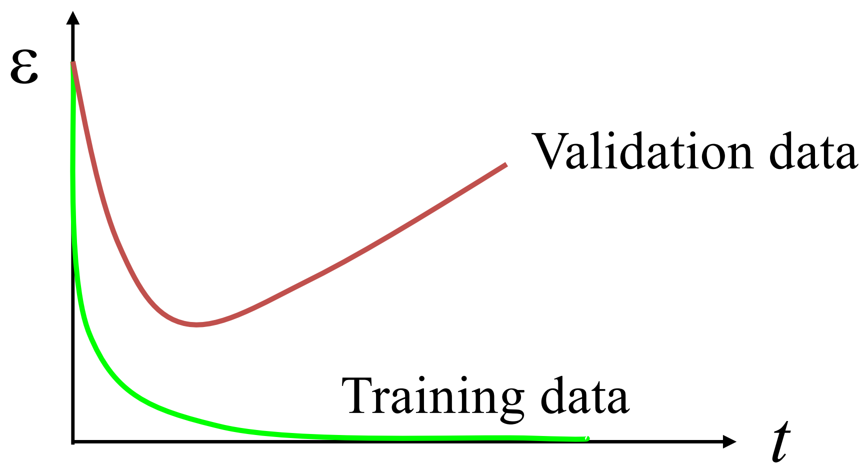
Images by B. Schölkopf

Outliers/incorrect labels?

Over-fitting

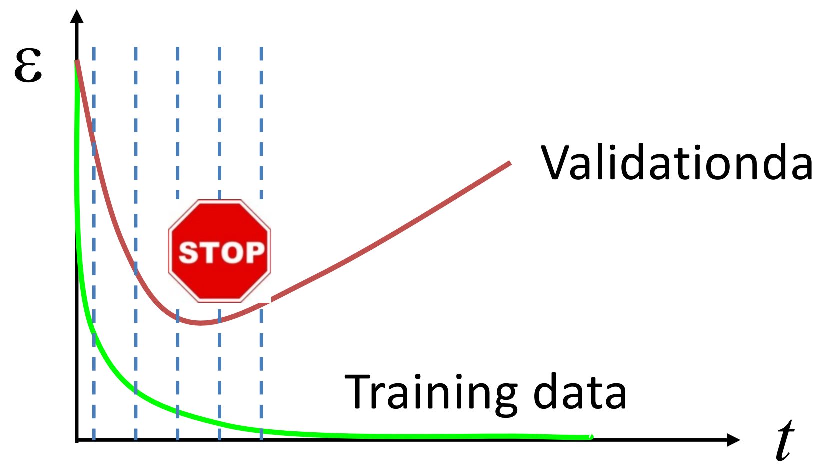
- The error on training data *always* decreases with increased training
- The error on validation data (the generalization error) decreases in the beginning, but can then start to increase if over-fitting occurs!





Preventing overfitting in neural networks

- **Early stopping:**
  Pause training regularly and calculate the performance on the validation data.
- <u>Caution: </u>Validation data becomes training data! Will bias evaluation.
- That's why we need a third dataset for testing – the test data



Training – Validation –Testing