Learning in nonlinear systems

Antanas Verikas
antanas.verikas@hh.se

IDE, Halmstad University

2013

Assumptions

We assume nonlinear process:

$$y(\mathbf{x}) = g(\mathbf{x}) + \varepsilon$$

with IID noise ε

Our model is:

$$\widehat{y}(\mathbf{x}) = f(\mathbf{x}, \mathbf{w})$$

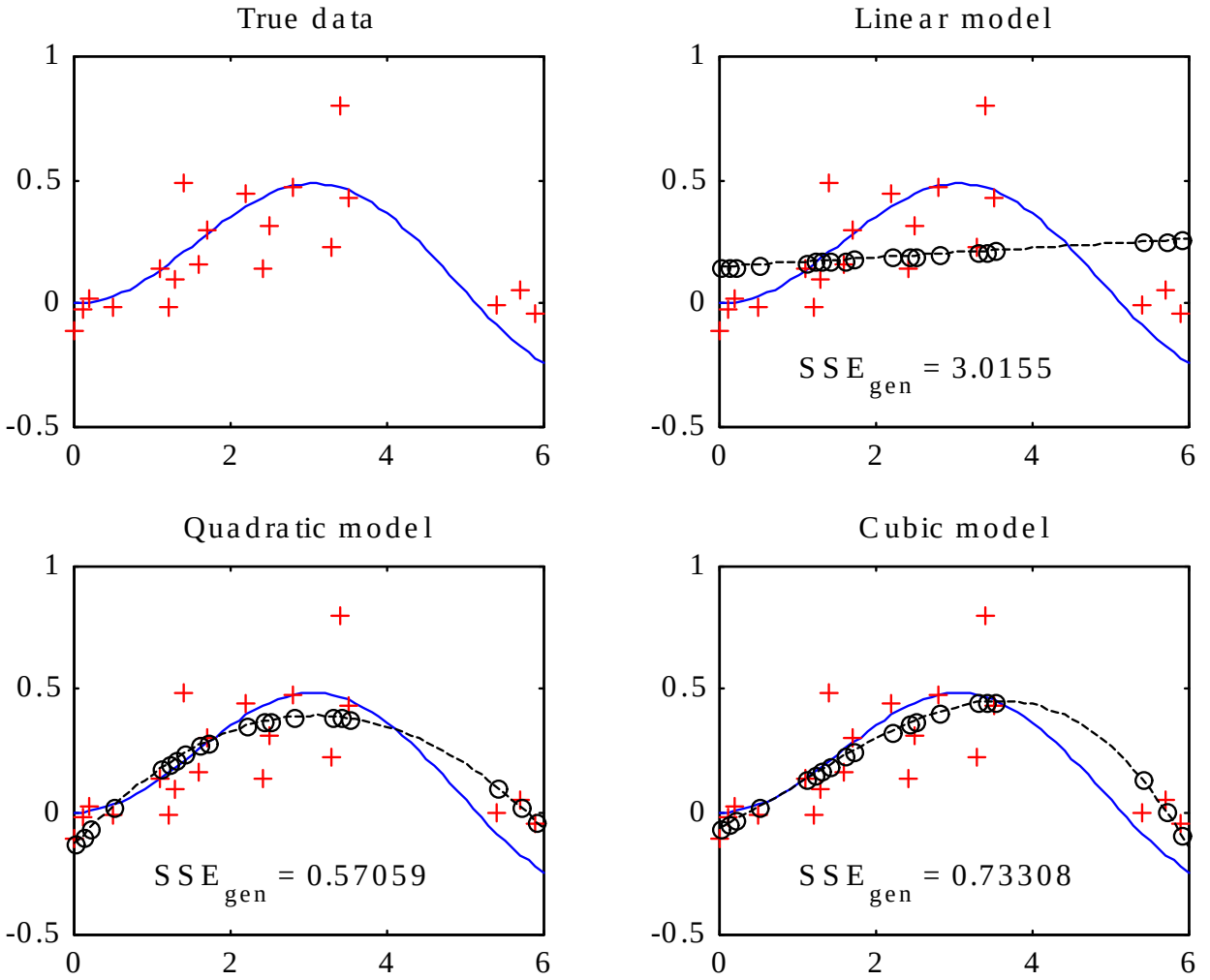
Polynomial model family

$$f(x, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x^2 + ... + w_M x^M$$

The function is nonlinear in x, but linear in $\mathbf{w} \Rightarrow$ reduces to the linear regression case.

Number of terms grows as D^M

Example: Polynomial model



Generalized linear model

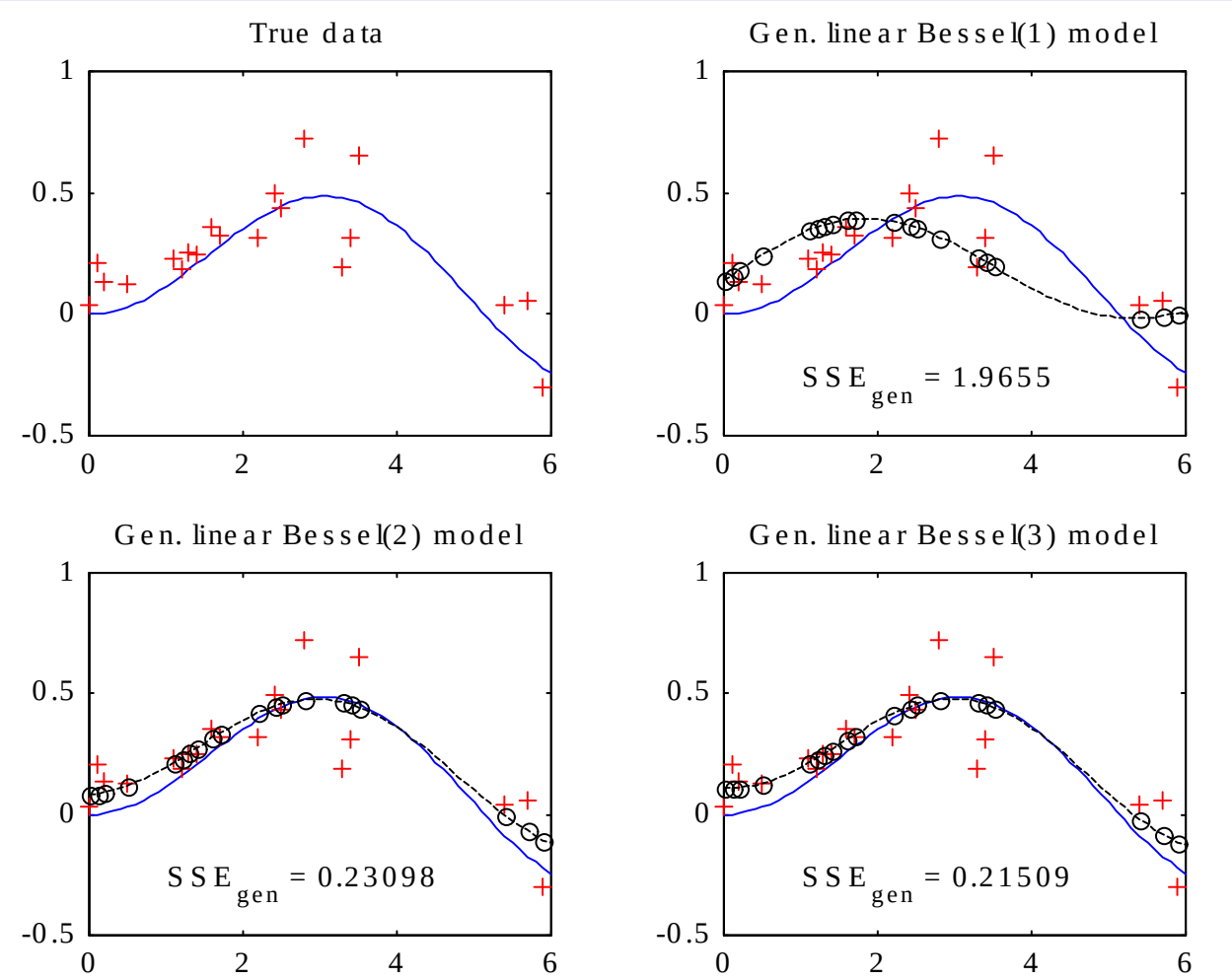
$$f(x, \mathbf{w}) = w_0 + w_1 h_1(\mathbf{x}) + ... + w_M h_M(\mathbf{x})$$

The function is nonlinear in x, but linear in $\mathbf{w} \Rightarrow$ reduces to the linear regression case.

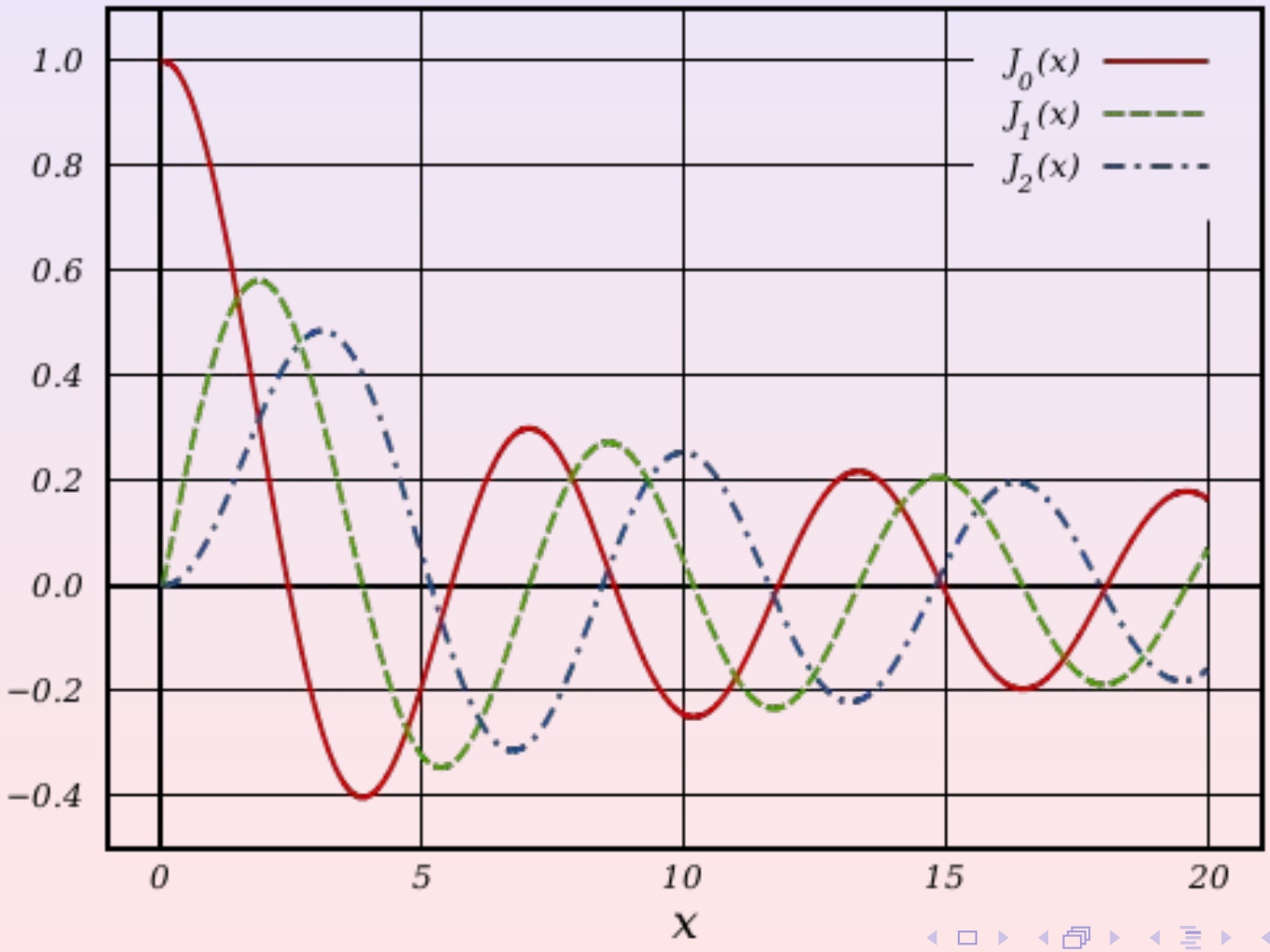
Requires good guess on "basis functions" $h_k(\mathbf{x})$.

Example: Generalized linear model

$$f(x, \mathbf{w}) = w_1 J_1(x) + w_2 J_2(x) + \ldots$$



Bessel functions

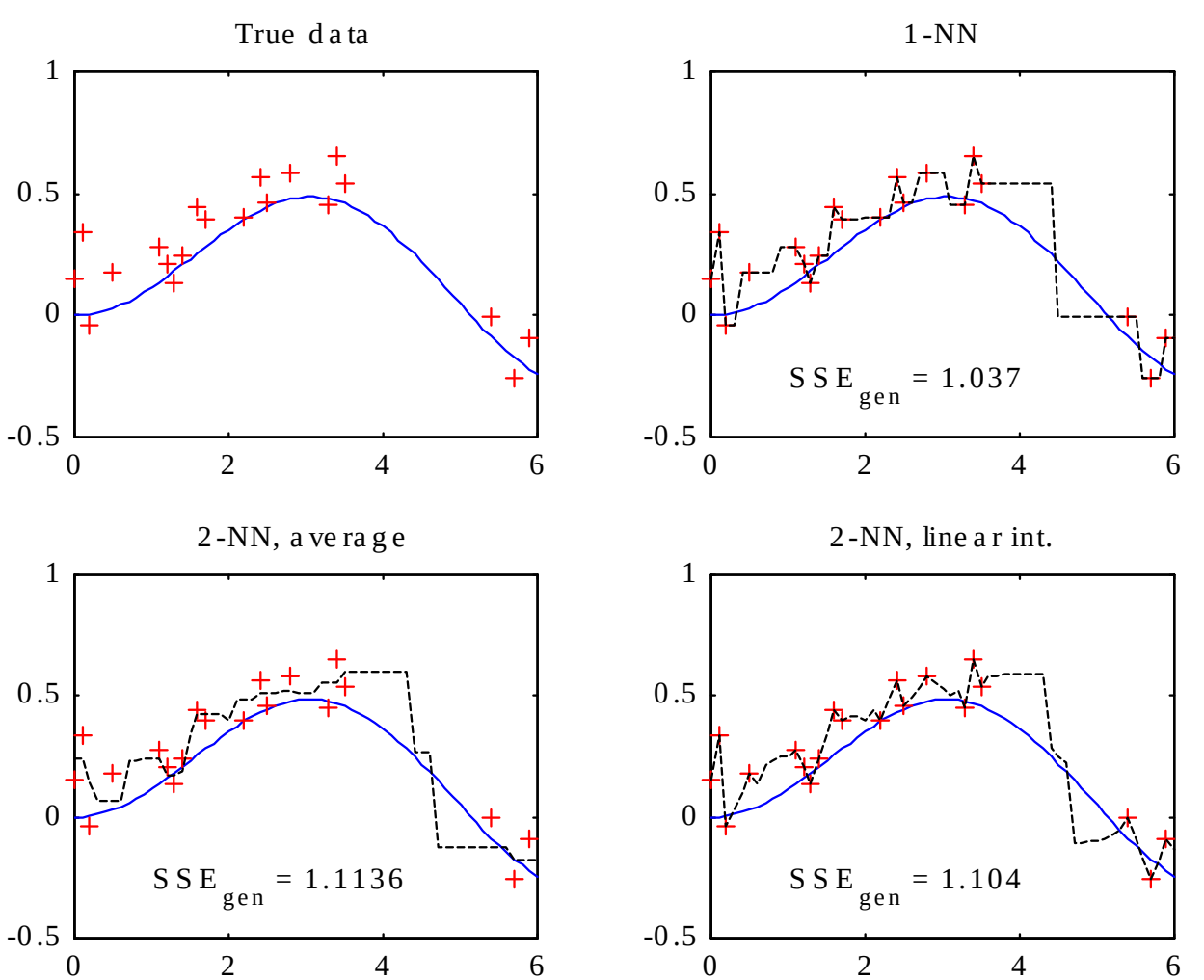


K nearest neighbours regression

- $\widehat{y}$ equals y of nearest neighbour;
- $\widehat{y}$ equals y average of K nearest neighbours;
- $\widehat{y}$ is obtained using y of K nearest neighbours and linear interpolation.

$$\widehat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{K} \left(y_k / r_k\right)}{\sum_{k=1}^{K} \left(1 / r_k\right)}$$

Example: KNN regression



Kernel regression

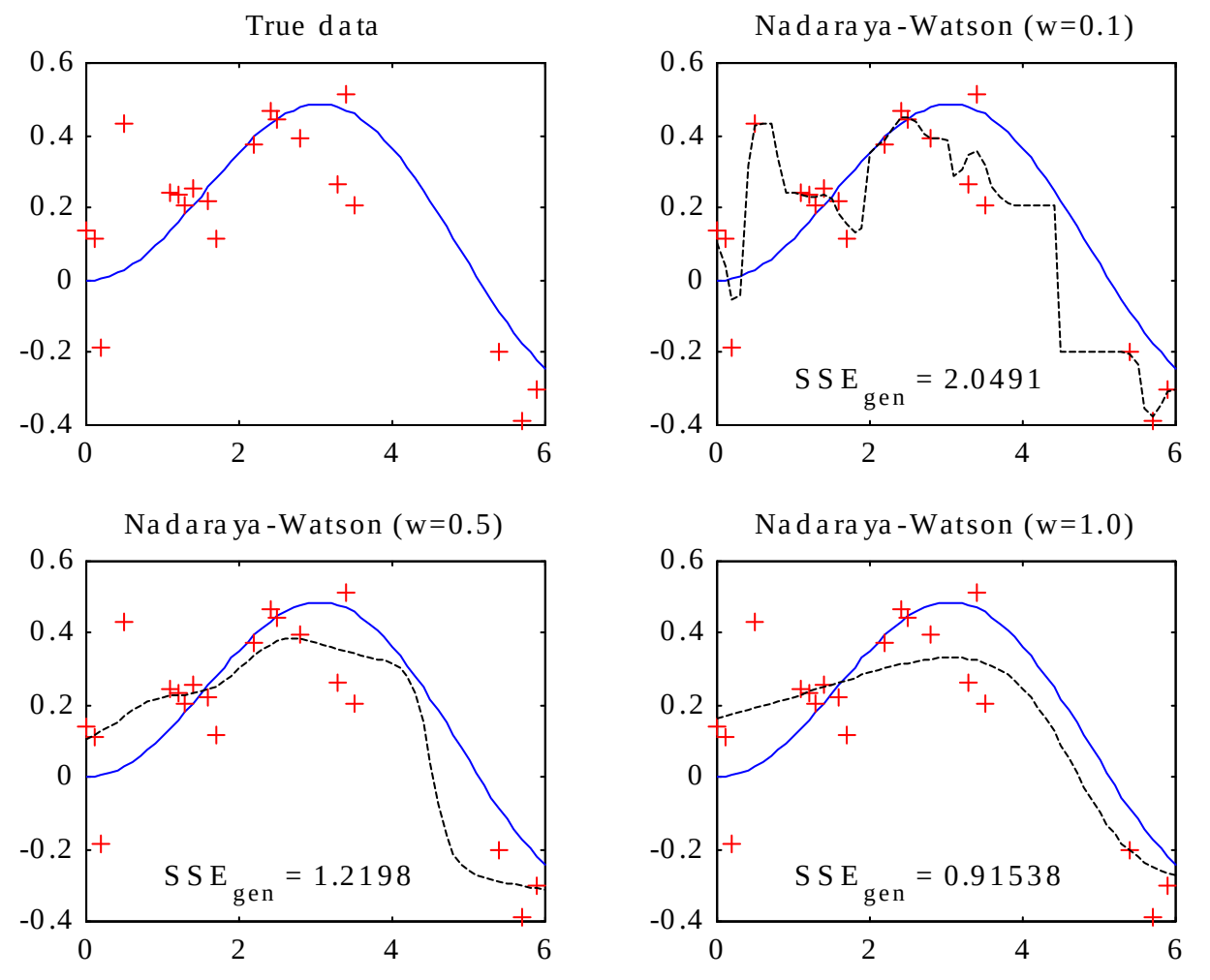
A common kernel regression model is the *Nadaraya-Watson* estimator

$$\widehat{y}(\mathbf{x}) = f(\mathbf{x}, w_0) = \frac{\sum_{n=1}^{N} y(n) \exp[-r_n^2(\mathbf{x})/2w_0^2]}{\sum_{n=1}^{N} \exp[-r_n^2(\mathbf{x})/2w_0^2]}$$

where

$$r_n^2(\mathbf{x}) = ||\mathbf{x} - \mathbf{x}(n)||^2$$

Example: Kernel regression



Different covariance matrices

Assume Gaussian $p(\mathbf{x}|c)$ with different $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$.
A new observation $\mathbf{x}$ is classified by computing the probability.

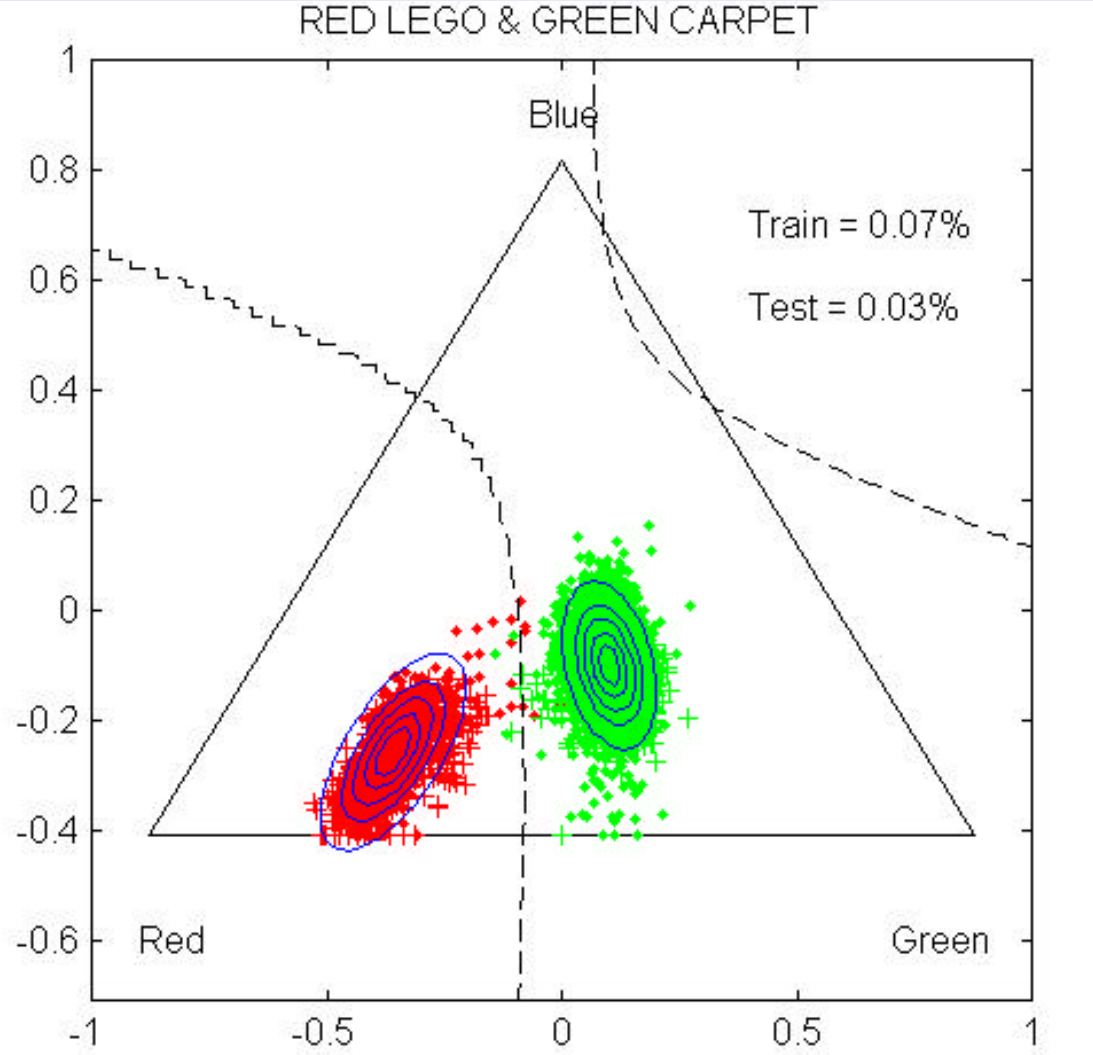
$$p(c_k|\mathbf{x}) = \frac{p(c_k)p(\mathbf{x}|c_k)}{p(\mathbf{x})}$$

The decision boundary between c_k and c_j is given by

$$\begin{aligned} p(c_k|\mathbf{x}) &= p(c_j|\mathbf{x}) \Rightarrow \\ \ln\left[\frac{N_k\sqrt{|\widehat{\boldsymbol{\Sigma}}_j|}}{N_j\sqrt{|\widehat{\boldsymbol{\Sigma}}_k|}}\right] &= \frac{(\mathbf{x}-\widehat{\boldsymbol{\mu}}_k)^T\widehat{\boldsymbol{\Sigma}}_k^{-1}(\mathbf{x}-\widehat{\boldsymbol{\mu}}_k)}{2} - \frac{(\mathbf{x}-\widehat{\boldsymbol{\mu}}_j)^T\widehat{\boldsymbol{\Sigma}}_j^{-1}(\mathbf{x}-\widehat{\boldsymbol{\mu}}_j)}{2} \end{aligned}$$

and defines a quadratic hypersurface.

Example: Quadratic Gaussian classifier



K-NN classifier

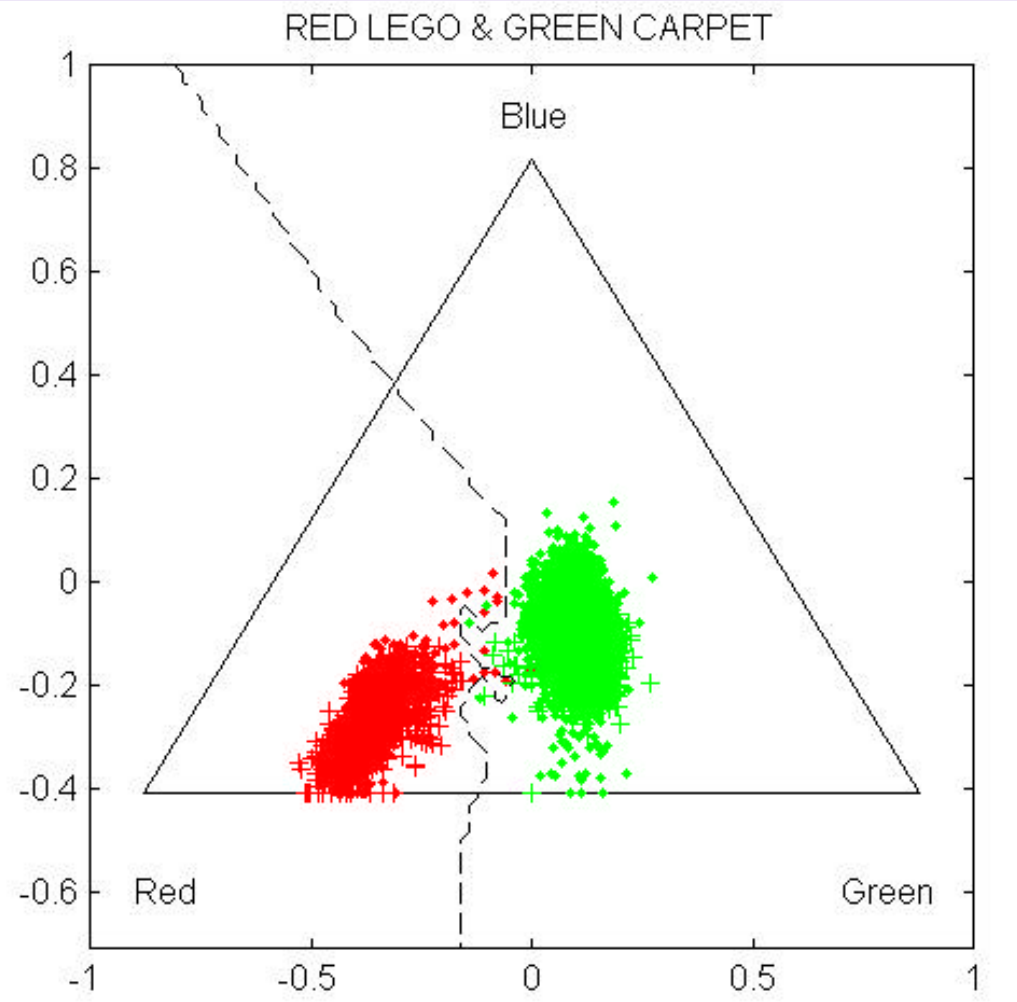
- Classify according to K nearest neighbours

$$\widehat{p}(c_j|\mathbf{x}) = \frac{K_j}{K}$$

- Majority vote

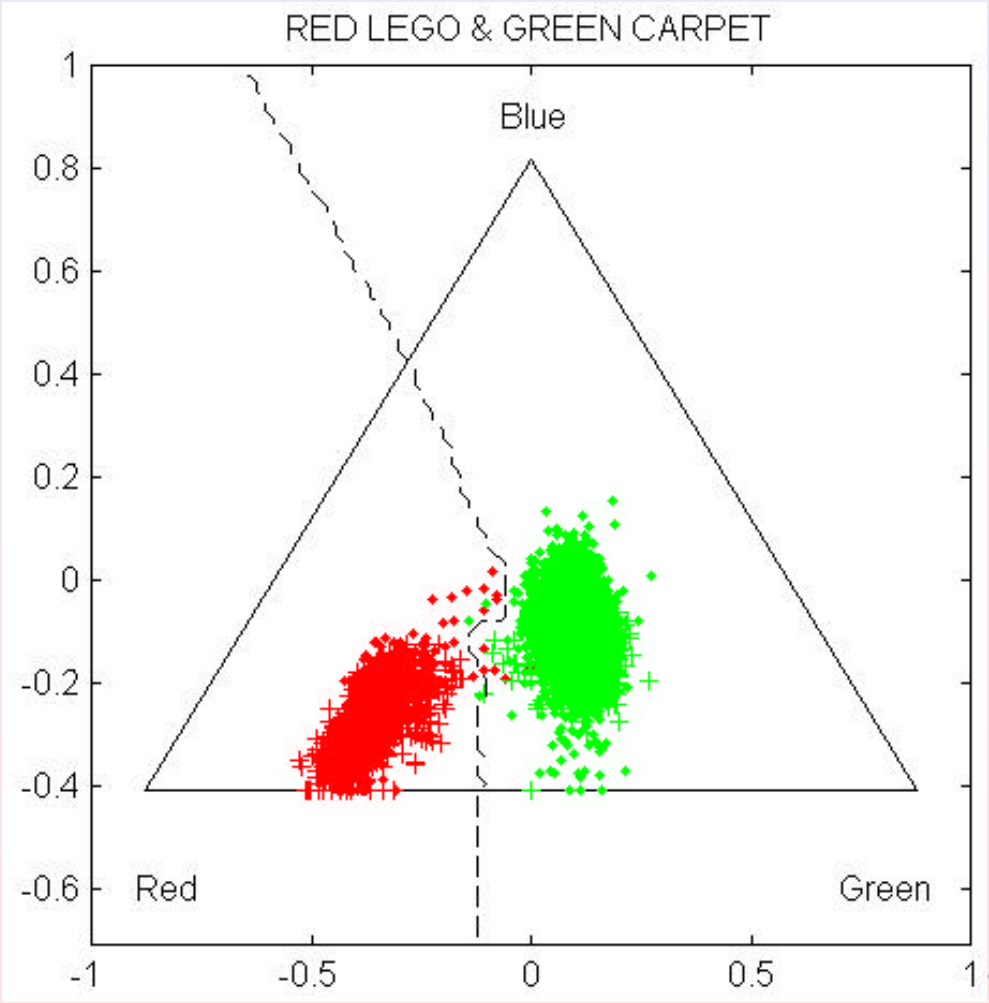
Example: 1-NN classifier

Test error = 0.10%



Example: 5-NN classifier

Test error = 0.14%



Problems with simple perceptron

$$\widehat{y}(\mathbf{x}) = \phi[\mathbf{w}^T\mathbf{x}] = \begin{cases} +1, & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ -1, & \text{if } \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

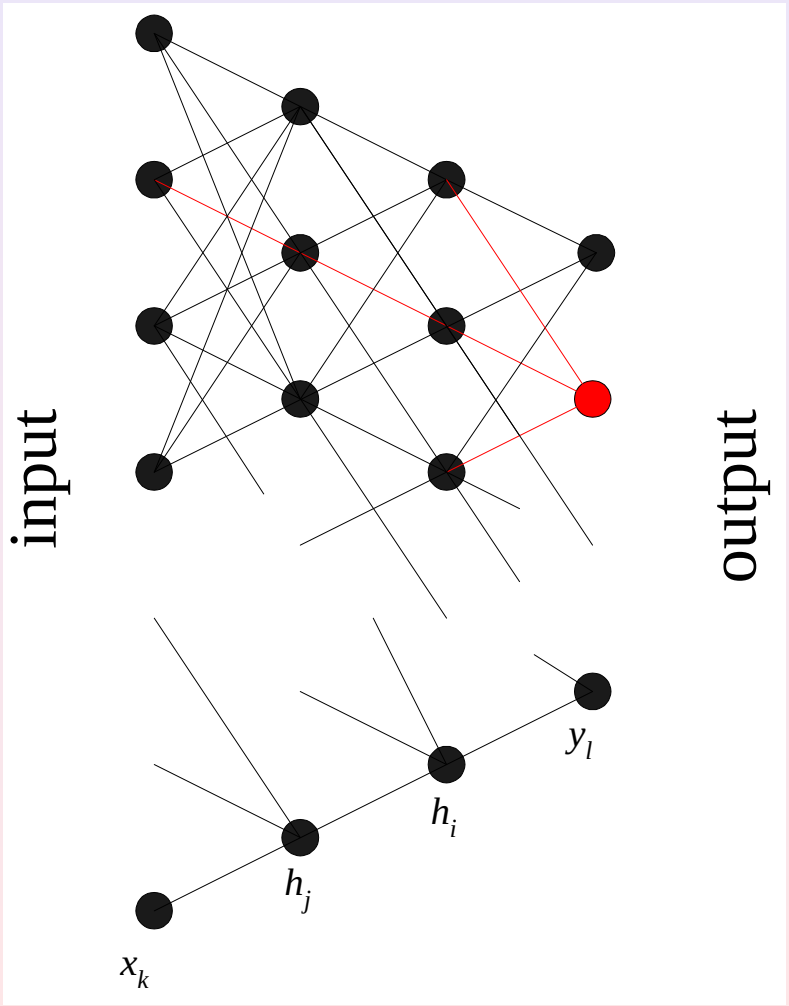
- Single layer = only linear machine.
- Perceptron learning oscillates if data distributions overlap.
- Step functions complicate learning with many perceptrons.

- Use many perceptrons.
- Use other learning algorithm.
- Use smooth functions.

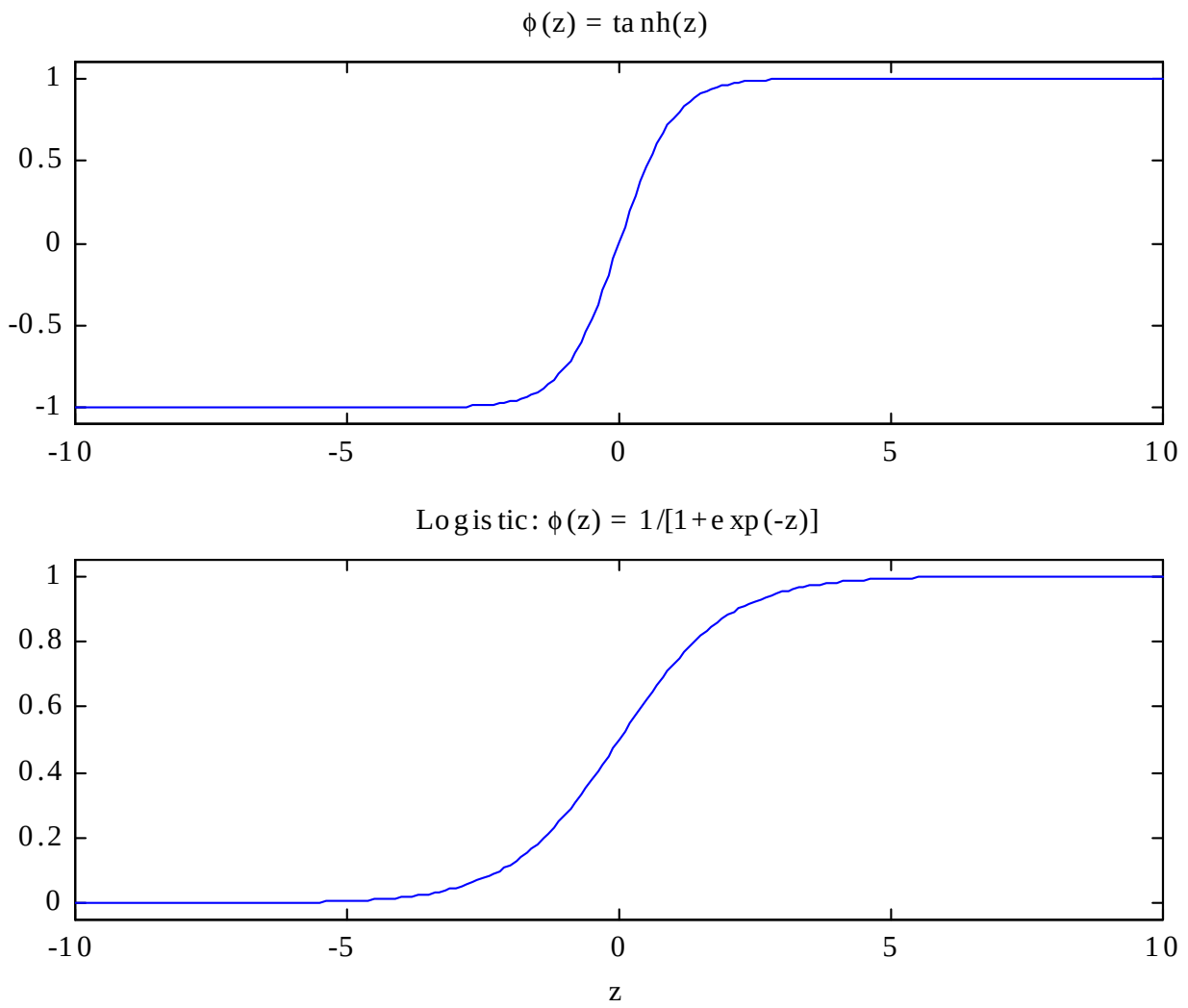
The Multilayer Perceptron (MLP)

- Combine several single layer perceptrons.
- Each single layer perceptron uses a sigmoid function e.g.

$$\phi(z) = \tanh(z)$$
$$\phi(z) = \frac{1}{1+\exp(-z)}$$



Transfer functions $\phi(z)$



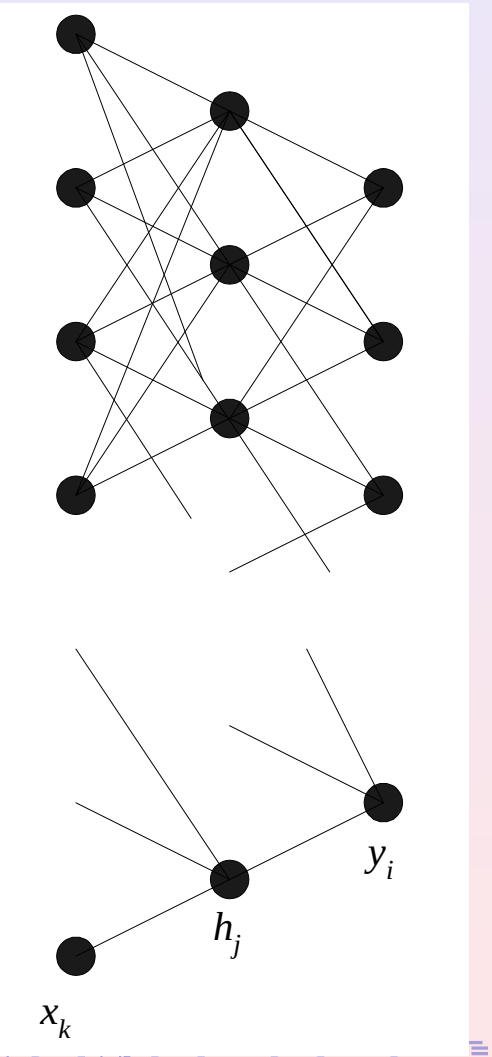
Example: One hidden layer

- Can approximate **any** continuous function $f(x)$.

$$\widehat{y}_i(\mathbf{x}) = \theta\left[v_{i0} + \sum_{j=1}^{J} v_{ij} h_j(\mathbf{x})\right]$$

$$h_j(\mathbf{x}) = \phi\left[w_{j0} + \sum_{k=1}^{D} w_{jk} x_k\right]$$

where:
$\theta(z) =$ sigmoid or linear,
$\phi(z) =$sigmoid.



Training: Backpropagation (gradient descent)

$$\begin{aligned}
\Delta \mathbf{w}(t) &= -\eta \nabla_w E(t) \\
&\Rightarrow \\
\Delta v_{ij}(t) &= \eta \sum_{n=1}^{N} \delta_i(n,t) h_j[\mathbf{x}(n), t] \\
\Delta w_{jk}(t) &= \eta \sum_{n=1}^{N} \delta_j(n,t) x_k(n)
\end{aligned}$$

where

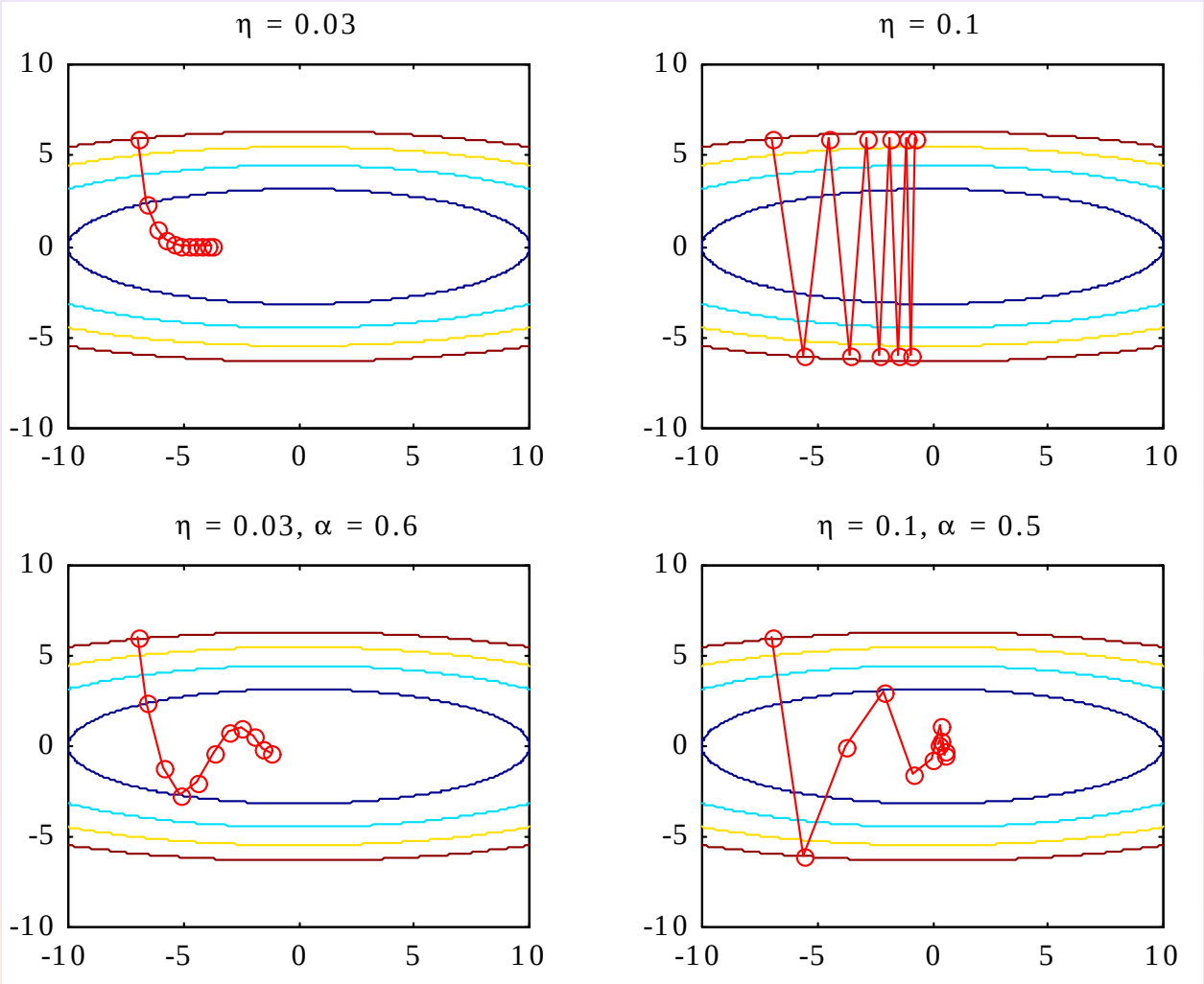
$$\delta_j(n,t) = \sum_i \delta_i(n,t) v_{ij}(t) h'_j[\mathbf{x}(n), t]$$

Backpropagation with momentum

- $$\Delta \mathbf{w}(t) = -\eta \nabla_w E(t) + \alpha \Delta \mathbf{w}(t-1)$$
- Speeds up if several steps are in the same direction.
- Slows down if steps change direction all the time.

Backpropagation with momentum

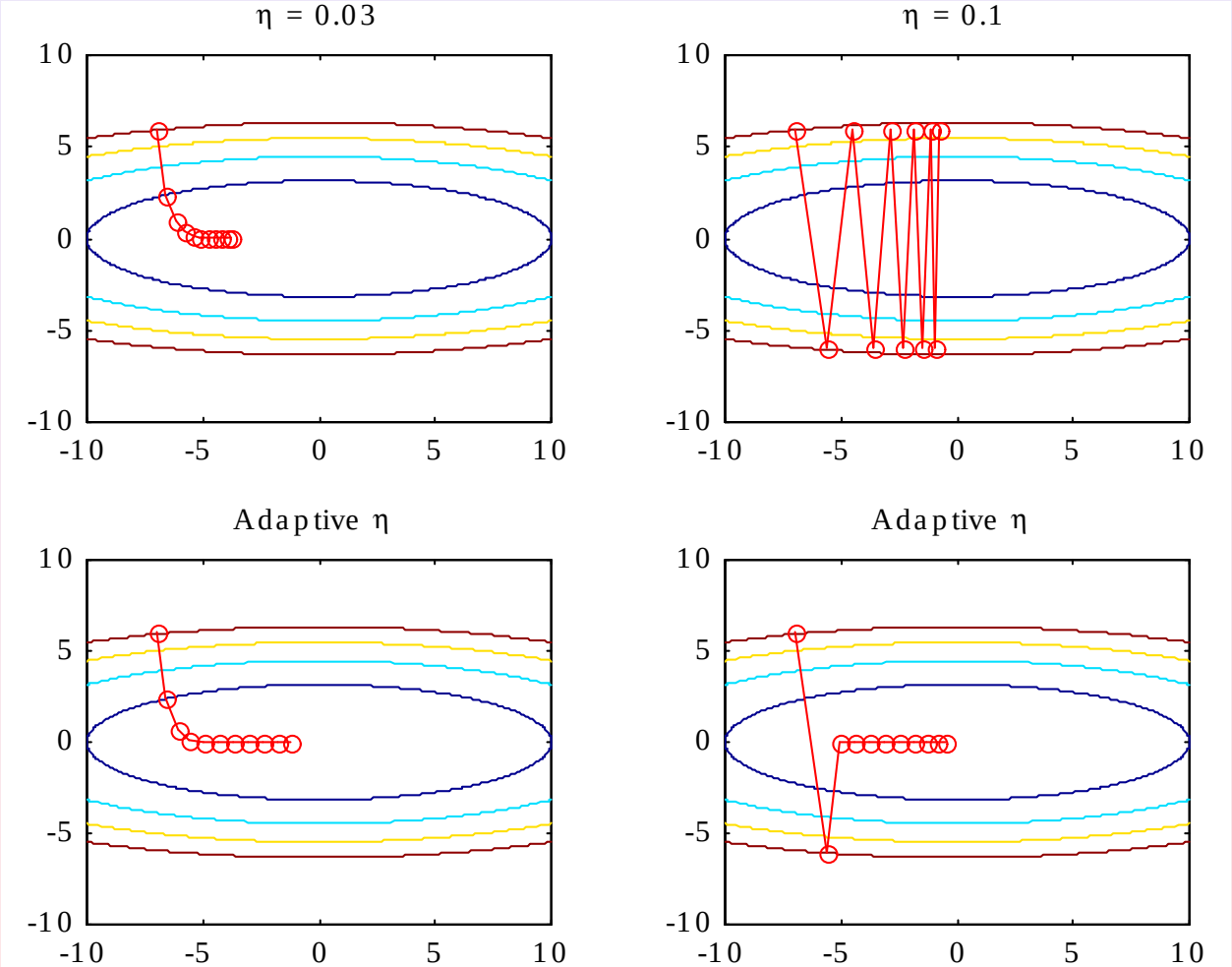
- No momentum:
- With momentum:



Adaptive η ("Bold driver")

- $$\begin{aligned} \Delta \mathbf{w}(t) &= -\eta(t) \nabla_w E(t) \\ \eta(t) &= \begin{cases} 0.5\eta(t-1), & \text{if } E(t) \geq E(t-1) \\ 1.2\eta(t-1), & \text{if } E(t) < E(t-1) \end{cases} \end{aligned}$$
- If things are going well $\Rightarrow$ increase speed.
- If things are going bad $\Rightarrow$ decrease speed.

Example: Adaptive η



RPROP (Resilient PROPagation)

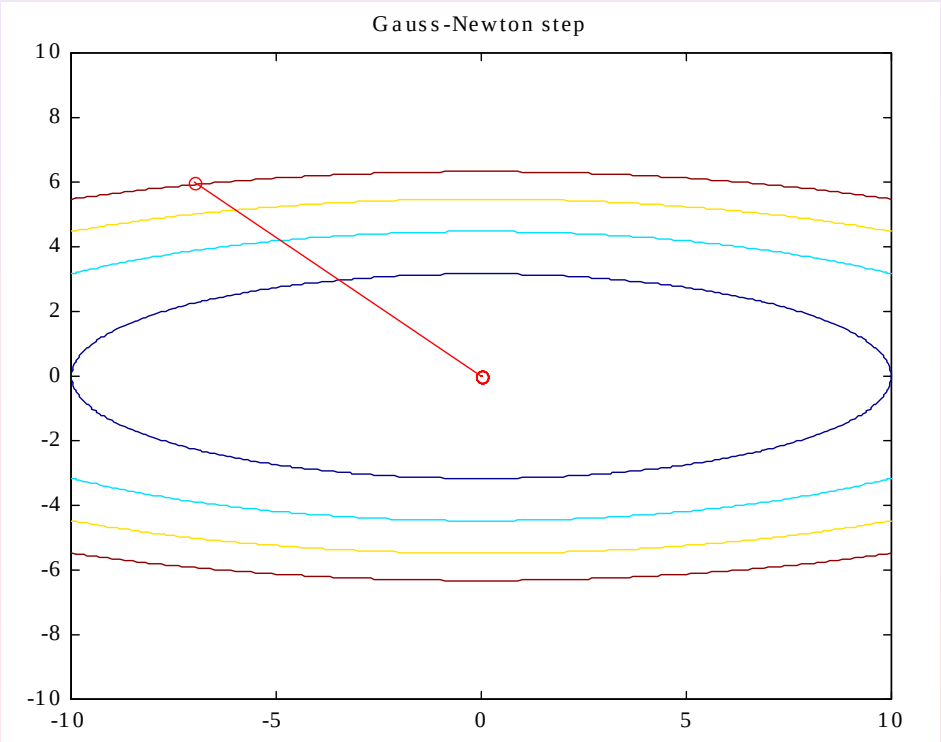
$$\Delta w_{ij}(t) = -\eta_{ij}(t)\mathtt{sign}\left[\frac{\partial E(t)}{\partial w_{ij}}\right]$$

$$\eta_{ij}(t) = \begin{cases} 1.2\eta_{ij}(t-1), & \text{if } \frac{\partial E(t)}{\partial w_{ij}} \cdot \frac{\partial E(t-1)}{\partial w_{ij}} > 0 \\ 0.5\eta_{ij}(t-1), & \text{if } \frac{\partial E(t)}{\partial w_{ij}} \cdot \frac{\partial E(t-1)}{\partial w_{ij}} < 0 \\ \eta_{ij}(t-1) \end{cases}$$

Requires **no** tuning of parameters.

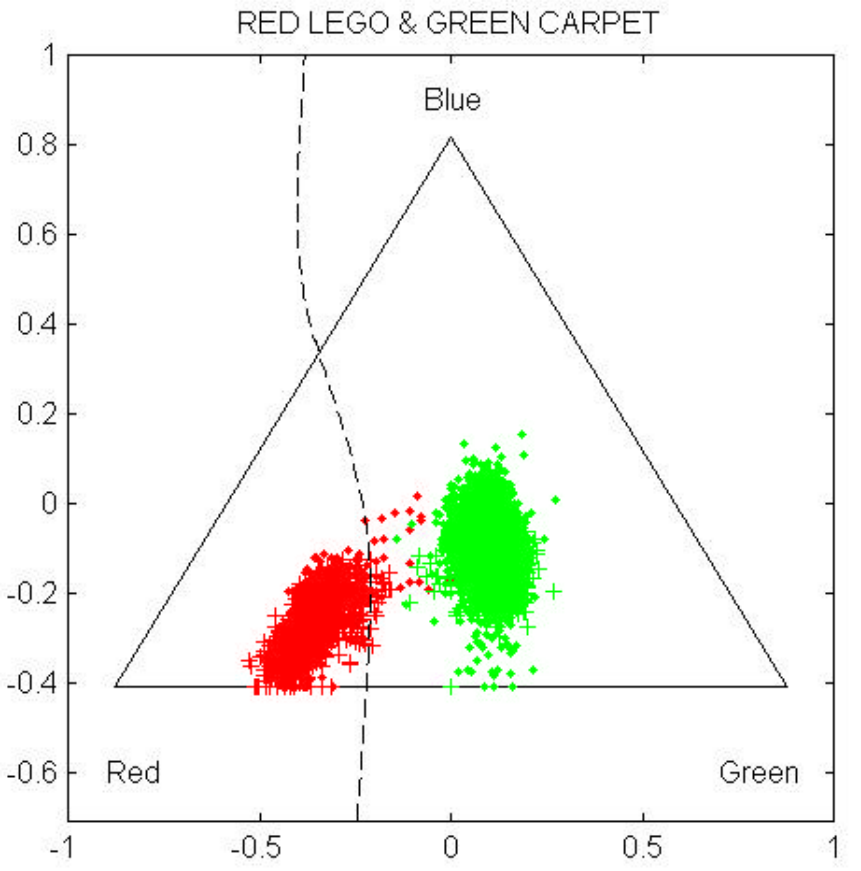
Second order learning algorithms

$$\Delta \mathbf{w}(t) = -\eta \mathbf{H}^{-1}(t) \nabla_w E(t)$$



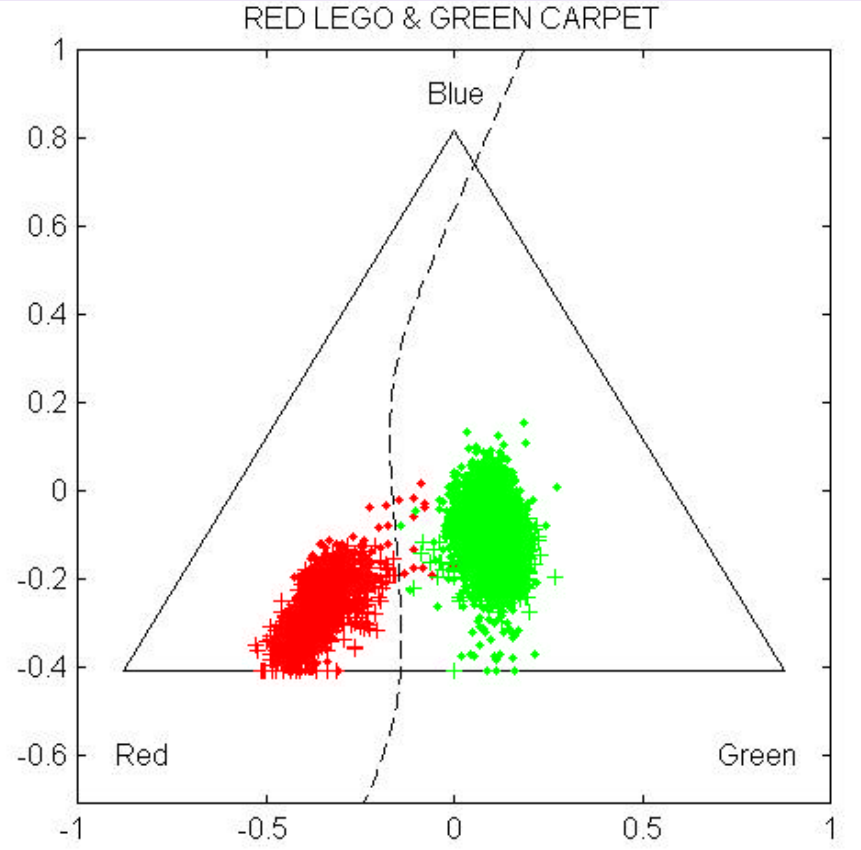
MLP (2-2-1), backpropagation, static η

After 3000 epochs: Training error = 0.20%, Test error = 0.24%



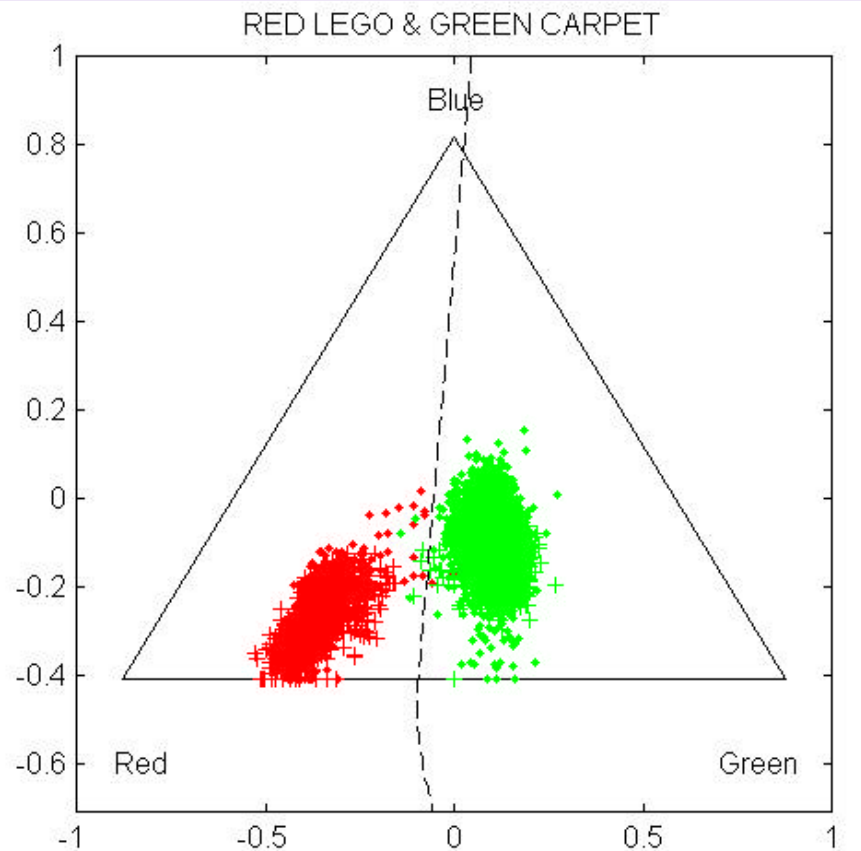
MLP (2-2-1), RPROP

After 10 epochs: Training error = 0.07%, Test error = 0.05%



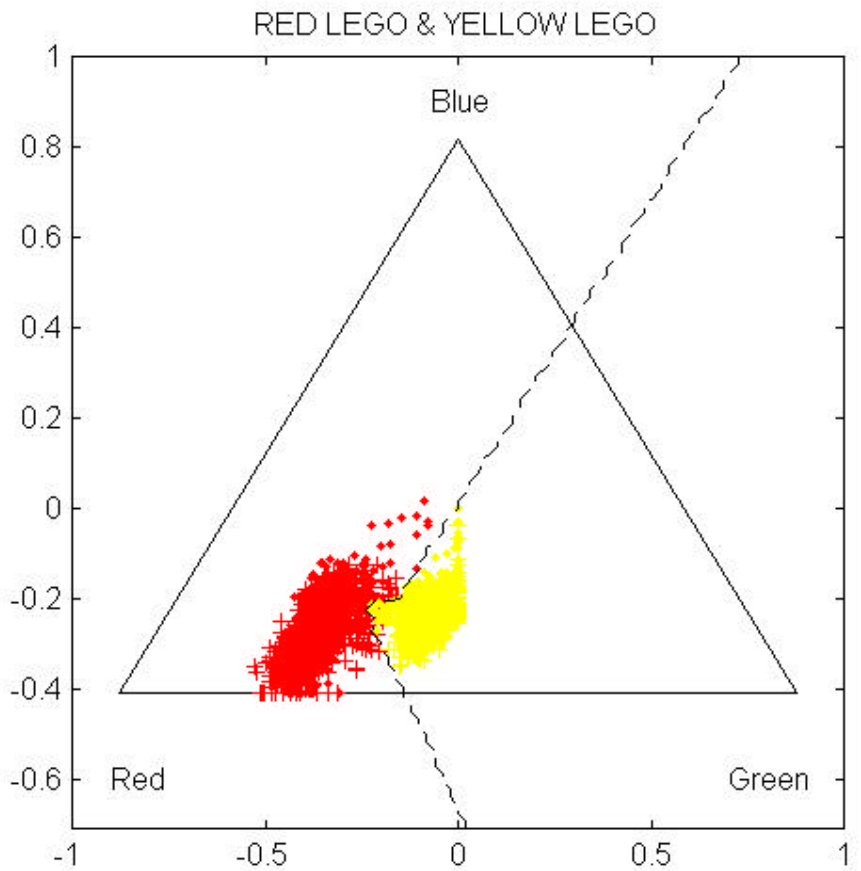
MLP (2-2-1), Levenberg-Marquardt

After 3 epochs: Training error = 0.03%, Test error = 0.07%



MLP (2-3-1), Levenberg-Marquardt

After 150 epochs: Training error = 0.17%, Test error = 0.16%



Nonlinear regression with MLP