Machine Learning Algorithms

Classification Algorithms

Markus Rupp

12.9.2020

Summary

- We were investigating an iterative procedure of the form

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu_k \underline{z}_k ; k = 1,2,...$$

- In order to find approximations of the Wiener solution.

- We found that the cost function

$$\underline{\underline{R}}^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}} \underline{w}_0 = \underline{r}^*_{\underline{\mathbf{x}}\mathbf{d}} \Rightarrow \underline{w}_0 = \left(\underline{\underline{R}}^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}}\right)^{-1} \underline{r}^*_{\underline{\mathbf{x}}\mathbf{d}}$$

$$g_o = \min_{\underline{w}} \mathrm{E}\left[\left|\mathbf{d} - \underline{w}^T \underline{\mathbf{x}}\right|^2\right] = \sigma^2_{\mathbf{d}} - \underline{r}^T_{\mathbf{d}\underline{\mathbf{x}}} R^{-1}_{\underline{\mathbf{x}}\underline{\mathbf{x}}} \underline{r}_{\underline{\mathbf{x}}\mathbf{d}} = \sigma^2_{\mathbf{d}} - \underline{r}^H_{\underline{\mathbf{x}}\mathbf{d}} R^{-1}_{\underline{\mathbf{x}}\underline{\mathbf{x}}} \underline{r}_{\underline{\mathbf{x}}\mathbf{d}}$$

Summary

- decreases

$$g\left(\underline{\hat{w}}_k\right) < g\left(\underline{\hat{w}}_{k-1}\right)$$

- if the direction for updates is of the form

$$\begin{aligned} \underline{z}_k &= -B\nabla g(\underline{\hat{w}}_{k-1})^H \\ &= -B\left(-\underline{r}_{\underline{\mathbf{x}}\mathbf{d}} + R_{\underline{\mathbf{xx}}}\underline{\hat{w}}^*_{k-1}\right)^* \end{aligned}$$

- With B>0
- For B=I, we found the steepest descent algorithm, with convergence condition

$$0 < \mu_k \leq \frac{2}{\lambda_{\max}} \leq \frac{2}{\lambda_i}$$

Resume: Step-size of steepest descent

- As convergence condition for B=I we had
- thus

$$\left|1-\mu_k\lambda_i\right|<1$$

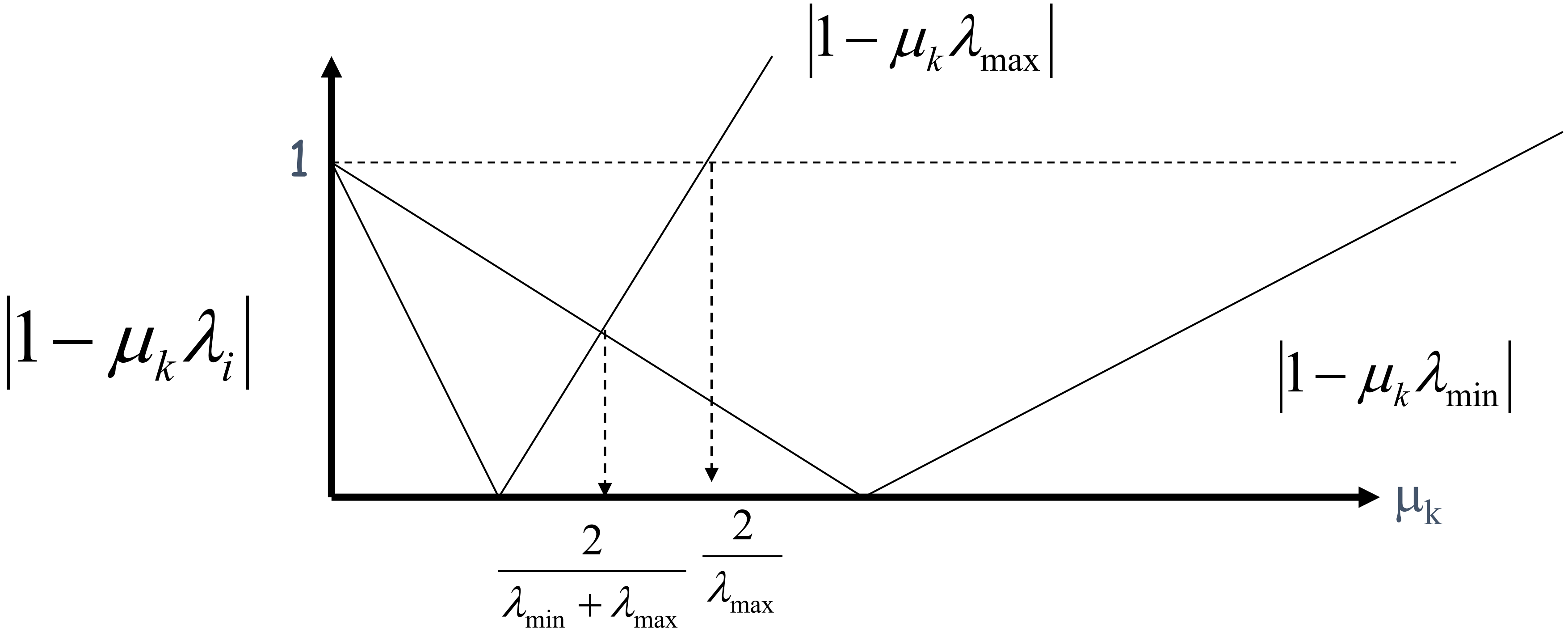
$$-1<1-\mu_k\lambda_i<1$$

$$-2<-\mu_k\lambda_i<0$$

$$0<\mu_k\lambda_i<2 \Rightarrow \mu_k<\frac{2}{\lambda_i}$$

Resume: Optimal Step-size of steepest descent

- Consider the smallest and the largest eigenvalue



Univ.-Prof. Dr.-Ing. Markus Rupp

Resume

- The steepest descent algorithm allows to **iteratively** find the minimum of a (quadratic) cost function.

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu_k \left[r_{\mathbf{d}\underline{\mathbf{x}}} - R^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}} \underline{\hat{w}}_{k-1} \right]; k = 1,2,\ldots$$

- Under some not too restrictive conditions we found that the algorithm converges as long as the step-size is bounded:

$$0 < \mu_k < \frac{2}{\lambda_{\max}} \le \frac{2}{\lambda_i}$$

Resume

- The LMS algorithm allows to **recursively** find the minimum of a (quadratic) cost function.

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu_k \underline{x}_k^* \left[d_k - \underline{x}_k^T \underline{\hat{w}}_{k-1} \right]; k = 1,2,\ldots$$

- For convergence in the mean we find the same result as for the steepest descent algorithm:

$$0 < \mu_k < \frac{2}{\lambda_{\max}} \leq \frac{2}{\lambda_i}$$

Resume

- For convergence in the **mean square sense**, however, we find a much more restricted result:

$$P_k = E\left[\left(\underline{w}-\hat{\underline{w}}_k\right)\left(\underline{w}-\hat{\underline{w}}_k\right)^H\right] = E\left[\underline{\tilde{w}}_k \underline{\tilde{w}}_k^H\right] \rightarrow tr(P_k) \rightarrow \underline{c}_k$$

$$\underline{c}_k = B\underline{c}_{k-1} + \mu^2\sigma_v^2\underline{\lambda} \rightarrow \gamma_l \lambda_{B,l}^k$$

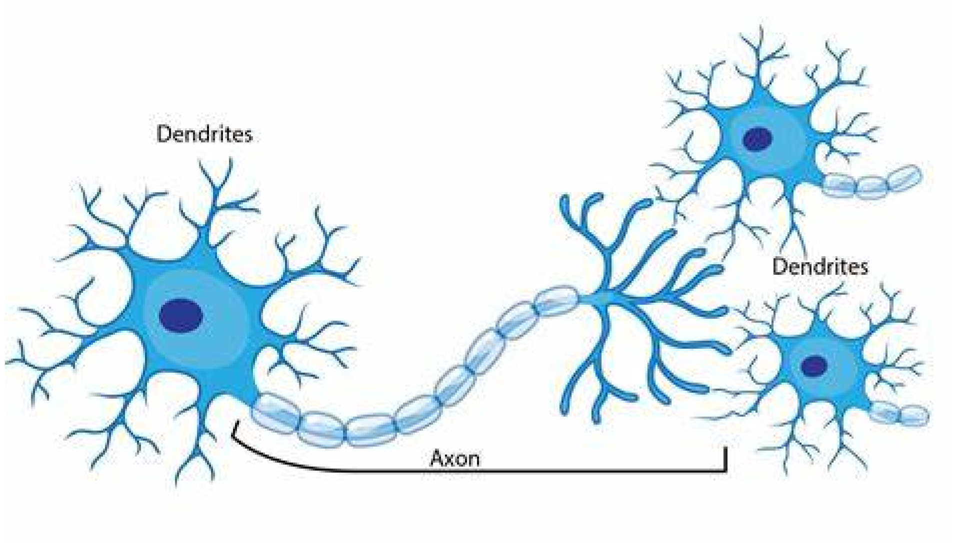
$$B = I - 2\mu\Lambda + \mu^2\gamma\Lambda^2 + \mu^2\underline{\lambda}\underline{\lambda}^T$$

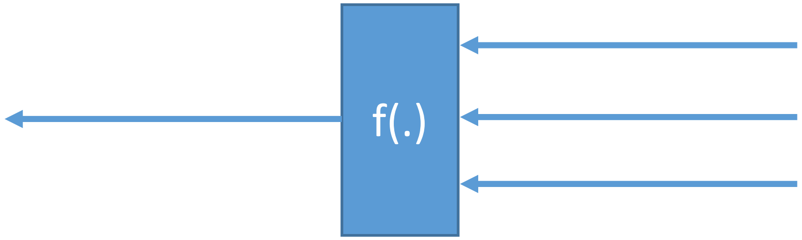
$$0 < \mu \leq \frac{2}{(\gamma+1)\,\mathrm{trace}\left(R_{\underline{\mathbf{uu}}}\right)} \leq \frac{2}{\gamma\lambda_{\max} + \mathrm{trace}\left(R_{\underline{\mathbf{uu}}}\right)}$$

Univ.-Prof. Dr.-Ing. Markus Rupp

How does Nature do?

- Consider Neuron





$$y_i = f\left(c_0 + c_1 x_{i,1} + \ldots + c_M x_{i,M}\right)$$
$$= f\left(\underline{\hat{x}}_i^H \underline{c}\right)$$

Activation Functions

Xavier Glorot, Antoine Bordes, Yoshua Bengio

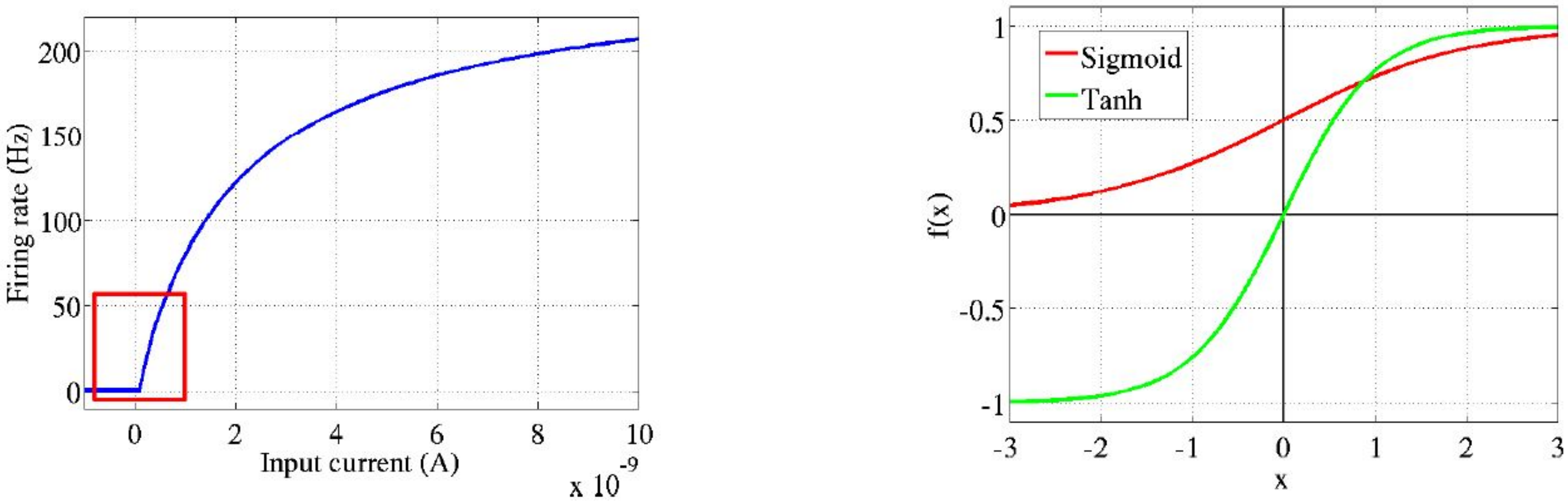


Figure 1: *Left:* **Common neural activation function motivated by biological data.** *Right:* **Commonly used activation functions** in neural networks literature: logistic sigmoid and hyperbolic tangent (*tanh*).

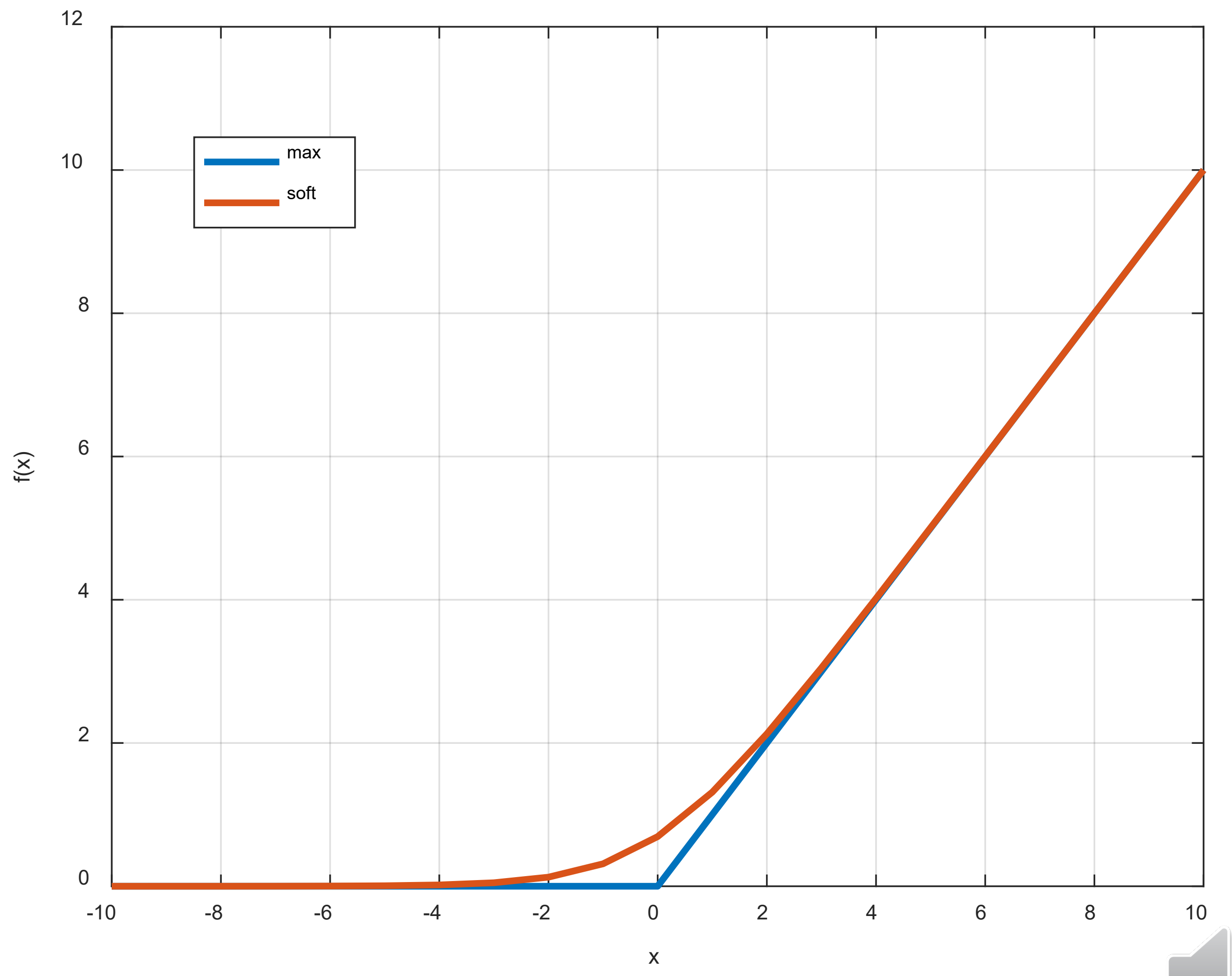
The „Hinge“

- Also known as ReLU:
  Rectifier Linear unit

  max(0,x)
- As the max(0,x) function is not differentiable, it is wise to approximate it by a smooth softdecision function:

$$\mathrm{soft}(x) = \log\left(1+e^{x}\right)$$

$$\frac{\partial}{\partial x}\mathrm{soft}(x) = \frac{1}{1+e^{-x}} = \sigma(x)$$



The „Hinge“

- Let us differentiate this function

$$\frac{\partial}{\partial x}\mathrm{soft}(x)=\frac{\partial}{\partial x}\log\left(1+e^x\right)=\frac{e^x}{1+e^x}=1-\frac{1}{1+e^x}=1-g(x)=g(-x)$$

$$\frac{\partial^2}{\partial x^2}\mathrm{soft}(x)=\frac{\partial^2}{\partial x^2}\log\left(1+e^x\right)=g(x)\left(1-g(x)\right)\geq 0$$

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

- Note that the range of g(x) is in[0,1] and thus the function is convex
- Iterative algorithms can thus guarantee convergence

Affine Transformations

- Note that we define a classification scheme via its potential output set, e.g. {-1,1},{-A,A},{0,1}
- While it does not matter which one we prefer and the classification itself is indifferent, the algorithms for optimization may behave differently.
- We thus have to keep „affine transformations“ in mind as a viable tool to condition the problem into an optimal way
- E.g.: 2{0,1}-1={-1,1} ½ {-1,1}+ ½ ={0,1}

Binary Classification

- Binary classification: y from {-1,1}

$$\underline{\widehat{x}}_i^H\,\underline{\widehat{c}} > 0; \quad if \quad y_i = +1$$

$$\underline{\widehat{x}}_i^H\,\underline{\widehat{c}} < 0; \quad if \quad y_i = -1$$

- or, equivalently, and more compactly

$$y_i\left(\underline{\widehat{x}}_i^H\,\underline{\widehat{c}}\right) > 0,$$

$$-y_i\left(\underline{\widehat{x}}_i^H\,\underline{\widehat{c}}\right) < 0.$$

Binary Classification

- If a hyperplane can correctly separate

$$-y_i\left(\underline{\widehat{x}}_i^H\,\underline{\widehat{c}}\right)<0$$

$$\max\left(0,-y_i\left(\underline{\widehat{x}}_i^H\,\underline{\widehat{c}}\right)\right)\quad=0$$

- If the hyperplane does not do a perfect job, for some value pairs, we obtain

$$\max\left(0,-y_i\left(\underline{\widehat{x}}_i^H\,\underline{\widehat{c}}\right)\right)\quad>0$$

Binary Classification

- We thus have to check for all data points that the individual results are below zero:

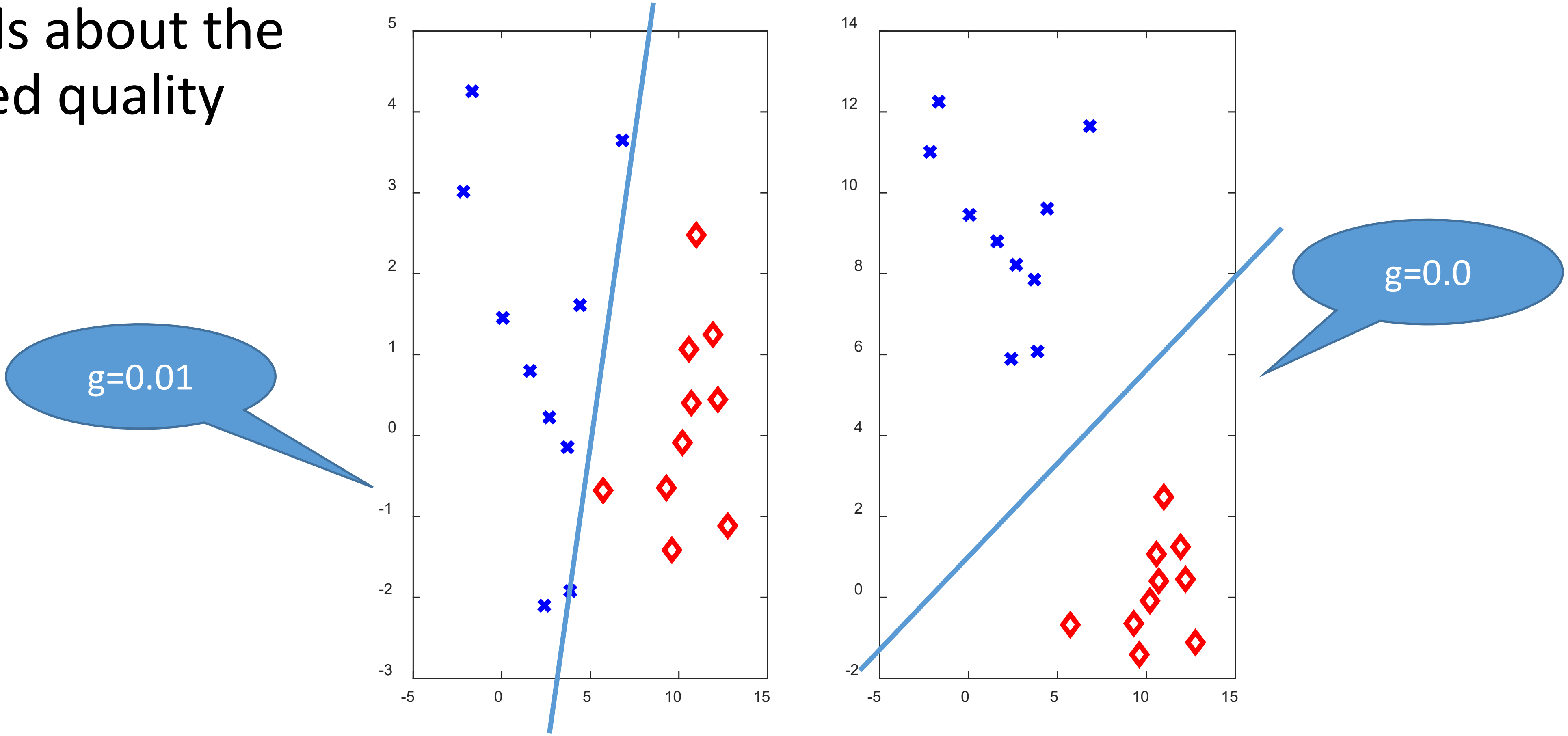
$$g(\underline{\hat{c}}) = \sum_{i=1}^{N} \max\left(0, -y_i\left(\underline{\hat{x}}_i^H \underline{\hat{c}}\right)\right)$$

$$\underline{\hat{c}}_{opt} = \arg\min_{\underline{c}} g(\underline{\hat{c}})$$

- If the so found optimal $\underline{c}$ results in a positive cost function g($\underline{c}$), we were not successful with separating the data.
- Even so, it is nevertheless not necessarily as poor solution

Example of almost separable

- g(c) tells about the achieved quality

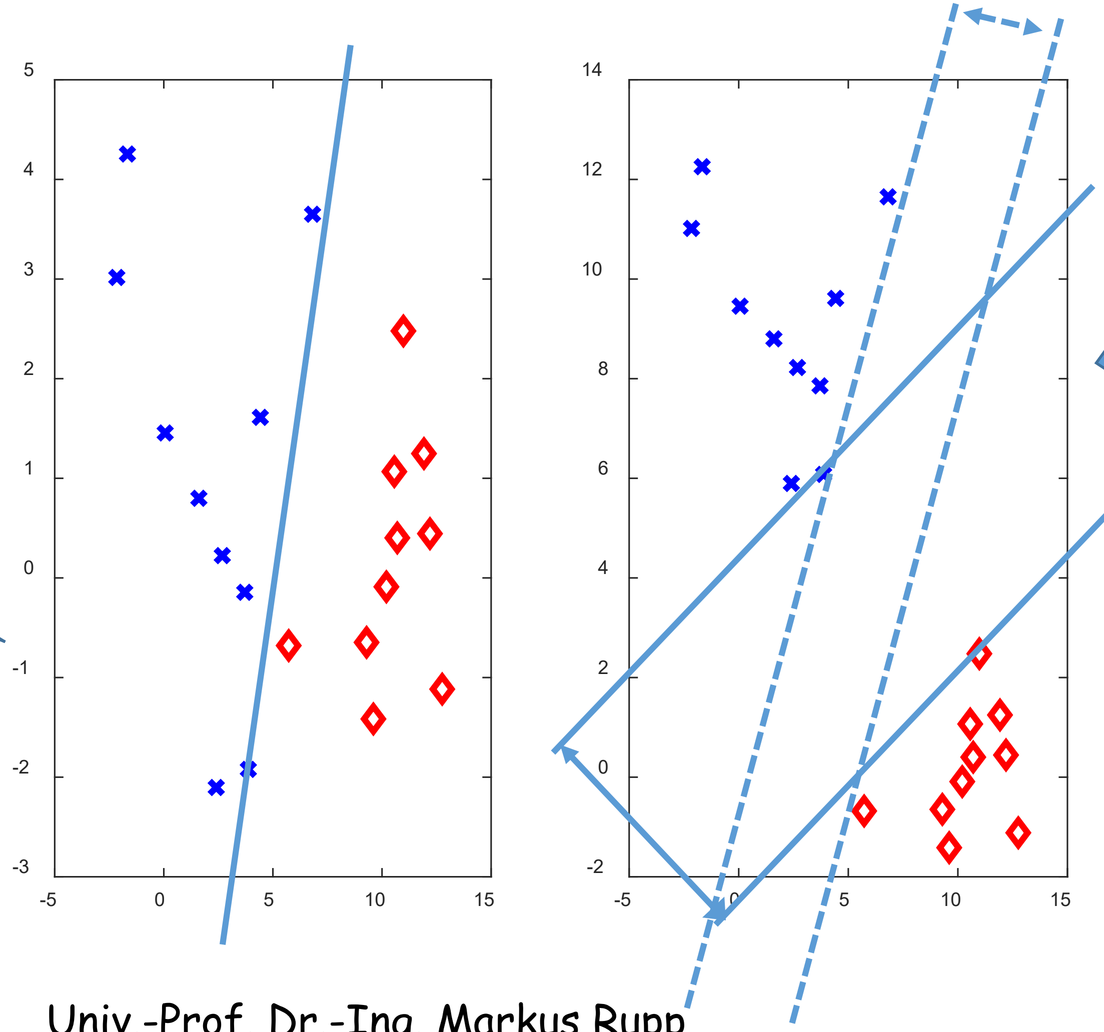


Univ.-Prof. Dr.-Ing. Markus Rupp

Example of almost separable

- g(c) tells about the achieved quality

g=0.01



But here we would like to learn about the distance of both lines and how this can be maximum

Margin Perceptron

- By „Percepton“ we denote an algorithm that finds a hyperplane that separates for a binary classification.
- By „Margin Perceptron“ we denote an algorithm that finds the maximum margins in terms of two parallel hyperplanes.

- We select the most outer points that touch the margin hyperplane by „-1“ and „+1“, respectively.
- This is not a restriction as the solution can be scaled arbitrarily.

Margin Perceptron

- Binary classification: y from {-1,1}

$$
\begin{aligned}
\underline{\widehat{x}}_i^H \underline{\widehat{c}} &\geq +1; \quad if \quad y_i = +1 \\
\underline{\widehat{x}}_i^H \underline{\widehat{c}} &\leq -1; \quad if \quad y_i = -1
\end{aligned}
$$

- or, equivalently, and more compactly

$$
\begin{aligned}
y_i\left(\underline{\widehat{x}}_i^H \underline{\widehat{c}}\right) &\geq 1, \\
-y_i\left(\underline{\widehat{x}}_i^H \underline{\widehat{c}}\right) &\leq -1, \\
1 - y_i\left(\underline{\widehat{x}}_i^H \underline{\widehat{c}}\right) &\leq 0.
\end{aligned}
$$

Margin Perceptron

- Applying the same concepts as before, we readily find:

$$g(\hat{\underline{c}}) = \sum_{i=1}^{N} \max\left(0, 1 - y_i \left(\hat{\underline{x}}_i^H \hat{\underline{c}}\right)\right)$$

$$\hat{\underline{c}}_{\text{opt,margin}} = \arg\min_{\underline{c}} g(\hat{\underline{c}})$$

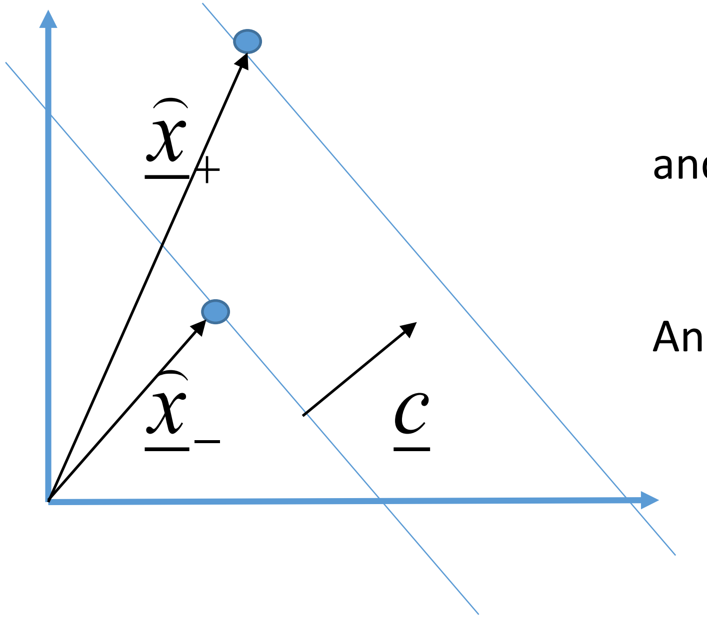
- And similarly with a differentiable cost function:

$$g(\hat{\underline{c}}) = \sum_{i=1}^{N} \text{soft}\left(1 - y_i \left(\hat{\underline{x}}_i^H \hat{\underline{c}}\right)\right)$$

$$\hat{\underline{c}}_{\text{opt, margin}} = \arg\min_{\underline{c}} g(\hat{\underline{c}})$$

Maximum Margin Hyperplane

- We like to maximize the distance of the two margins



Have two feature vectors on the margins

$$\underline{\hat{x}}_-, \quad \underline{\hat{x}}_+$$

and one vector perpendicular to the hyperplane

$$\underline{c}$$

And obtain for the width of the „street“

$$\left(\underline{\hat{x}}_+ - \underline{\hat{x}}_-\right)^H \frac{\underline{c}}{\|\underline{c}\|} = \frac{1-(-1)}{\|\underline{c}\|} = \frac{2}{\|\underline{c}\|}$$

Note that this vector does not contain the bias term

Hard Margin VSM

- Maximizing the width is equivalent to

$$
\begin{aligned}
&\min \|\underline{c}\| \\
&\text{subject to} \\
&\max\left(0, 1 - y_i\left(\underline{\widehat{x}}_i^H \underline{\widehat{c}}\right)\right) = 0 \quad ; i = 1, 2, \ldots, N
\end{aligned}
$$

- As we needed the support of the margin vectors, the method is called Vector Support Machine (SVM)

Modifications

- Applying the previous techniques, we can easily modify the formulation using soft cost functions and slightly different constraints.
- Original formulation only works if perfectly linearly separable.
- Problem: find the formulation that can be solved easiest and still provides a practically useful result