

# Machine Learning – Lecture 7

## Linear Discriminants II

31.10.2019

Bastian Leibe

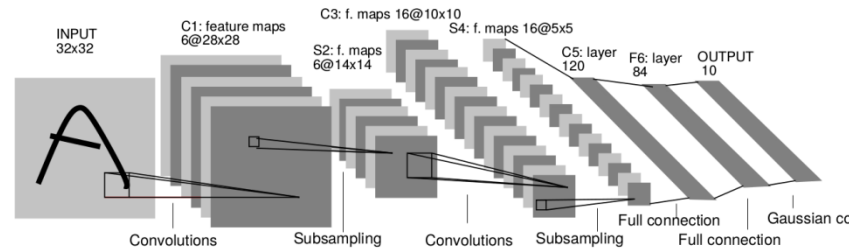
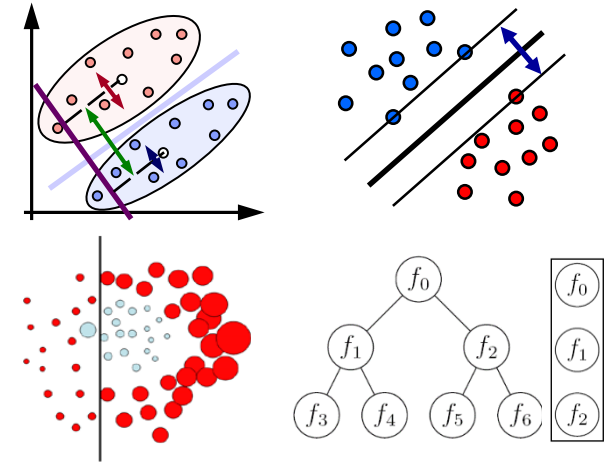
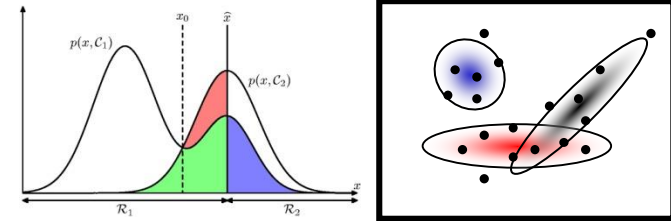
RWTH Aachen

<http://www.vision.rwth-aachen.de>

leibe@vision.rwth-aachen.de

# Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Randomized Trees, Forests & Ferns
  
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks



# Recap: Generalized Linear Models

- Generalized linear model

$$y(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x} + w_0)$$

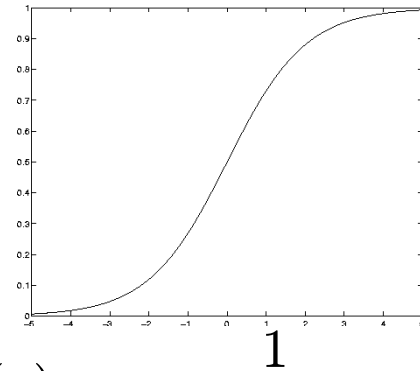
- $g(\cdot)$  is called an **activation function** and may be nonlinear.
- The decision surfaces correspond to

$$y(\mathbf{x}) = \text{const.} \quad \Leftrightarrow \quad \mathbf{w}^T \mathbf{x} + w_0 = \text{const.}$$

- If  $g$  is monotonous (which is typically the case), the resulting decision boundaries are still linear functions of  $\mathbf{x}$ .

- Advantages of the non-linearity

- Can be used to bound the influence of outliers and “too correct” data points.
- When using a sigmoid for  $g(\cdot)$ , we can interpret the  $y(\mathbf{x})$  as posterior probabilities.



$$g(a) \equiv \frac{1}{1 + \exp(-a)}$$

# Recap: Extension to Nonlinear Basis Fcts.

- Generalization

- Transform vector  $\mathbf{x}$  with  $M$  nonlinear basis functions  $\phi_j(\mathbf{x})$ :

$$y_k(\mathbf{x}) = \sum_{j=1}^M w_{kj} \phi_j(\mathbf{x}) + w_{k0}$$

- Advantages

- Transformation allows non-linear decision boundaries.
- By choosing the right  $\phi_j$ , every continuous function can (in principle) be approximated with arbitrary accuracy.

- Disadvantage

- The error function can in general no longer be minimized in closed form.

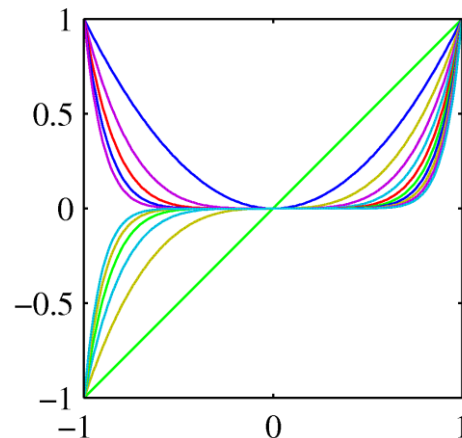
⇒ Minimization with Gradient Descent

# Recap: Basis Functions

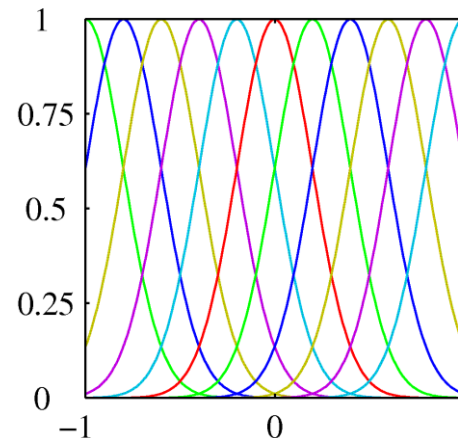
- Generally, we consider models of the following form

$$y_k(\mathbf{x}) = \sum_{j=0}^M w_{kj} \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

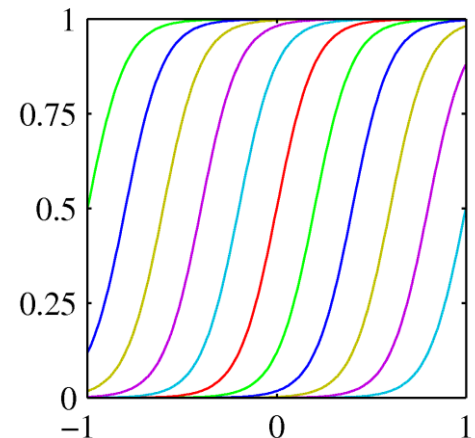
- ▶ where  $\phi_j(\mathbf{x})$  are known as *basis functions*.
  - ▶ In the simplest case, we use linear basis functions:  $\phi_d(\mathbf{x}) = x_d$ .
- Other popular basis functions



Polynomial



Gaussian



Sigmoid

# Recap: Iterative Methods for Estimation

- Gradient Descent (1<sup>st</sup> order)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w})|_{\mathbf{w}^{(\tau)}}$$

- Simple and general
- Relatively slow to converge, has problems with some functions

- Newton-Raphson (2<sup>nd</sup> order)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \mathbf{H}^{-1} \nabla E(\mathbf{w})|_{\mathbf{w}^{(\tau)}}$$

where  $\mathbf{H} = \nabla \nabla E(\mathbf{w})$  is the Hessian matrix, i.e. the matrix of second derivatives.

- Local quadratic approximation to the target function
- Faster convergence

# Recap: Gradient Descent

- Iterative minimization

- Start with an initial guess for the parameter values  $w_{kj}^{(0)}$ .
- Move towards a (local) minimum by following the gradient.

- Basic strategies

- “Batch learning”

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- “Sequential updating”

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

where

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w})$$

# Recap: Gradient Descent

- Example: Quadratic error function

$$E(\mathbf{w}) = \sum_{n=1}^N (y(\mathbf{x}_n; \mathbf{w}) - \mathbf{t}_n)^2$$

- Sequential updating leads to **delta rule** (=LMS rule)

$$\begin{aligned} w_{kj}^{(\tau+1)} &= w_{kj}^{(\tau)} - \eta (y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}) \phi_j(\mathbf{x}_n) \\ &= w_{kj}^{(\tau)} - \eta \delta_{kn} \phi_j(\mathbf{x}_n) \end{aligned}$$

- ▶ where

$$\delta_{kn} = y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}$$

- ⇒ Simply feed back the input data point, weighted by the classification error.



# Recap: Gradient Descent

- Cases with differentiable, non-linear activation function

$$y_k(\mathbf{x}) = g(a_k) = g \left( \sum_{j=0}^M w_{kj} \phi_j(\mathbf{x}_n) \right)$$

- Gradient descent (again with quadratic error function)

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} = \frac{\partial g(a_k)}{\partial w_{kj}} (y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}) \phi_j(\mathbf{x}_n)$$

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \delta_{kn} \phi_j(\mathbf{x}_n)$$

$$\delta_{kn} = \frac{\partial g(a_k)}{\partial w_{kj}} (y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn})$$

# Recap: Probabilistic Discriminative Models

- Consider models of the form

$$p(\mathcal{C}_1|\phi) = y(\phi) = \sigma(\mathbf{w}^T \phi)$$

with 
$$p(\mathcal{C}_2|\phi) = 1 - p(\mathcal{C}_1|\phi)$$

- This model is called **logistic regression**.
- Properties
  - Probabilistic interpretation
  - But discriminative method: only focus on decision hyperplane
  - Advantageous for high-dimensional spaces, requires less parameters than explicitly modeling  $p(\phi|\mathcal{C}_k)$  and  $p(\mathcal{C}_k)$ .

# Recap: Logistic Regression

- Let's consider a data set  $\{\phi_n, t_n\}$  with  $n = 1, \dots, N$ , where  $\phi_n = \phi(\mathbf{x}_n)$  and  $t_n \in \{0, 1\}$ ,  $\mathbf{t} = (t_1, \dots, t_N)^T$ .

- With  $y_n = p(\mathcal{C}_1 | \phi_n)$ , we can write the likelihood as

$$p(\mathbf{t} | \mathbf{w}) = \prod_{n=1}^N y_n^{t_n} \{1 - y_n\}^{1-t_n}$$

- Define the error function as the negative log-likelihood

$$\begin{aligned} E(\mathbf{w}) &= -\ln p(\mathbf{t} | \mathbf{w}) \\ &= -\sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \end{aligned}$$

- This is the so-called **cross-entropy error function**.

# Gradient of the Error Function

$$y_n = \sigma(\mathbf{w}^T \phi_n)$$

$$\frac{dy_n}{d\mathbf{w}} = y_n(1 - y_n)\phi_n$$

- Error function

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

- Gradient

$$\begin{aligned} \nabla E(\mathbf{w}) &= - \sum_{n=1}^N \left\{ t_n \frac{\frac{d}{d\mathbf{w}} y_n}{y_n} + (1 - t_n) \frac{\frac{d}{d\mathbf{w}} (1 - y_n)}{(1 - y_n)} \right\} \\ &= - \sum_{n=1}^N \left\{ t_n \frac{\cancel{y_n} (1 - y_n)}{\cancel{y_n}} \phi_n - (1 - t_n) \frac{\cancel{y_n} (1 - y_n)}{\cancel{(1 - y_n)}} \phi_n \right\} \\ &= - \sum_{n=1}^N \{ (t_n - \cancel{t_n y_n} - y_n + \cancel{t_n y_n}) \phi_n \} \\ &= \sum_{n=1}^N (y_n - t_n) \phi_n \end{aligned}$$

# Gradient of the Error Function

- Gradient for logistic regression

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (y_n - t_n) \phi_n$$

- Does this look familiar to you?
- This is the same result as for the Delta (=LMS) rule

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta (y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}) \phi_j(\mathbf{x}_n)$$

- We can use this to derive a sequential estimation algorithm.
  - However, this will be quite slow...

# Newton-Raphson for Least-Squares Estimation

- Let's first apply Newton-Raphson to the least-squares error function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (\mathbf{w}^T \phi_n - t_n)^2$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (\mathbf{w}^T \phi_n - t_n) \phi_n = \Phi^T \Phi \mathbf{w} - \Phi^T \mathbf{t}$$

$$\mathbf{H} = \nabla \nabla E(\mathbf{w}) = \sum_{n=1}^N \phi_n \phi_n^T = \Phi^T \Phi \quad \text{where } \Phi = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_N^T \end{bmatrix}$$

- Resulting update scheme:

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - (\Phi^T \Phi)^{-1} (\Phi^T \Phi \mathbf{w}^{(\tau)} - \Phi^T \mathbf{t}) \\ &= (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t} \end{aligned}$$

Closed-form solution!

# Newton-Raphson for Logistic Regression

- Now, let's try Newton-Raphson on the cross-entropy error function:

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

$$\frac{dy_n}{d\mathbf{w}} = y_n(1 - y_n)\phi_n$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (y_n - t_n)\phi_n = \Phi^T (\mathbf{y} - \mathbf{t})$$

$$\mathbf{H} = \nabla \nabla E(\mathbf{w}) = \sum_{n=1}^N y_n(1 - y_n)\phi_n \phi_n^T = \Phi^T \mathbf{R} \Phi$$

where  $\mathbf{R}$  is an  $N \times N$  diagonal matrix with  $R_{nn} = y_n(1 - y_n)$ .

$\Rightarrow$  The Hessian is no longer constant, but depends on  $\mathbf{w}$  through the weighting matrix  $\mathbf{R}$ .

# Iteratively Reweighted Least Squares

- Update equations

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - (\Phi^T \mathbf{R} \Phi)^{-1} \Phi^T (\mathbf{y} - \mathbf{t}) \\ &= (\Phi^T \mathbf{R} \Phi)^{-1} \left\{ \Phi^T \mathbf{R} \Phi \mathbf{w}^{(\tau)} - \Phi^T (\mathbf{y} - \mathbf{t}) \right\} \\ &= (\Phi^T \mathbf{R} \Phi)^{-1} \Phi^T \mathbf{R} \mathbf{z}\end{aligned}$$

$$\text{with } \mathbf{z} = \Phi \mathbf{w}^{(\tau)} - \mathbf{R}^{-1} (\mathbf{y} - \mathbf{t})$$

- Again very similar form (normal equations)
  - But now with non-constant weighing matrix  $\mathbf{R}$  (depends on  $\mathbf{w}$ ).
  - Need to apply normal equations iteratively.

⇒ Iteratively Reweighted Least-Squares (IRLS)



# Summary: Logistic Regression

- Properties

- Directly represent posterior distribution  $p(\phi|\mathcal{C}_k)$
- Requires fewer parameters than modeling the likelihood + prior.
- Very often used in statistics.
- It can be shown that the cross-entropy error function is concave
  - Optimization leads to unique minimum
  - But no closed-form solution exists
  - Iterative optimization (IRLS)
- Both online and batch optimizations exist

- Caveat

- Logistic regression tends to systematically overestimate odds ratios when the sample size is less than  $\sim 500$ .

# Topics of This Lecture

- **Softmax Regression**
  - Multi-class generalization
  - Gradient descent solution
- **Note on Error Functions**
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
- **Linear Support Vector Machines**
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

# Softmax Regression

- Multi-class generalization of logistic regression
  - In logistic regression, we assumed binary labels  $t_n \in \{0, 1\}$ .
  - Softmax generalizes this to  $K$  values in 1-of- $K$  notation.

$$\mathbf{y}(\mathbf{x}; \mathbf{w}) = \begin{bmatrix} P(y = 1 | \mathbf{x}; \mathbf{w}) \\ P(y = 2 | \mathbf{x}; \mathbf{w}) \\ \vdots \\ P(y = K | \mathbf{x}; \mathbf{w}) \end{bmatrix} = \frac{1}{\sum_{j=1}^K \exp(\mathbf{w}_j^\top \mathbf{x})} \begin{bmatrix} \exp(\mathbf{w}_1^\top \mathbf{x}) \\ \exp(\mathbf{w}_2^\top \mathbf{x}) \\ \vdots \\ \exp(\mathbf{w}_K^\top \mathbf{x}) \end{bmatrix}$$

- This uses the **softmax** function

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

- Note: the resulting distribution is normalized.

# Softmax Regression Cost Function

- Logistic regression

- Alternative way of writing the cost function with indicator function  $\mathbb{I}(\cdot)$

$$\begin{aligned} E(\mathbf{w}) &= - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \\ &= - \sum_{n=1}^N \sum_{k=0}^1 \{ \mathbb{I}(t_n = k) \ln P(y_n = k | \mathbf{x}_n; \mathbf{w}) \} \end{aligned}$$

- Softmax regression

- Generalization to K classes using indicator functions.

$$E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^K \exp(\mathbf{w}_j^\top \mathbf{x})} \right\}$$

# Optimization

- Again, no closed-form solution is available
  - Resort again to Gradient Descent
  - Gradient

$$\nabla_{\mathbf{w}_k} E(\mathbf{w}) = - \sum_{n=1}^N [\mathbb{I}(t_n = k) \ln P(y_n = k | \mathbf{x}_n; \mathbf{w})]$$

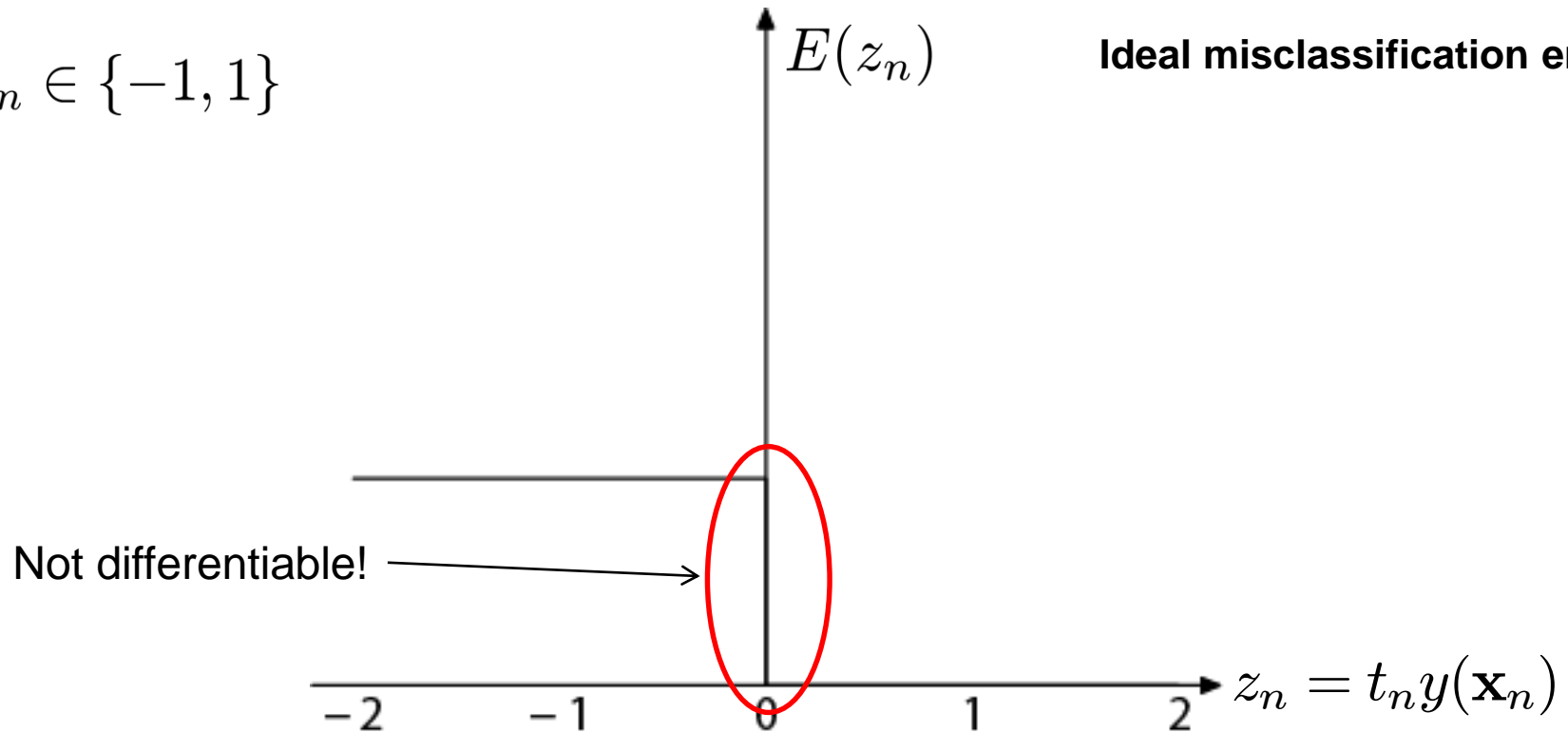
- Note
  - $\nabla_{\mathbf{w}_k} E(\mathbf{w})$  is itself a vector of partial derivatives for the different components of  $\mathbf{w}_k$ .
  - We can now plug this into a standard optimization package.

# Topics of This Lecture

- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- **Note on Error Functions**
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
- Linear Support Vector Machines
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

# Note on Error Functions

$$t_n \in \{-1, 1\}$$



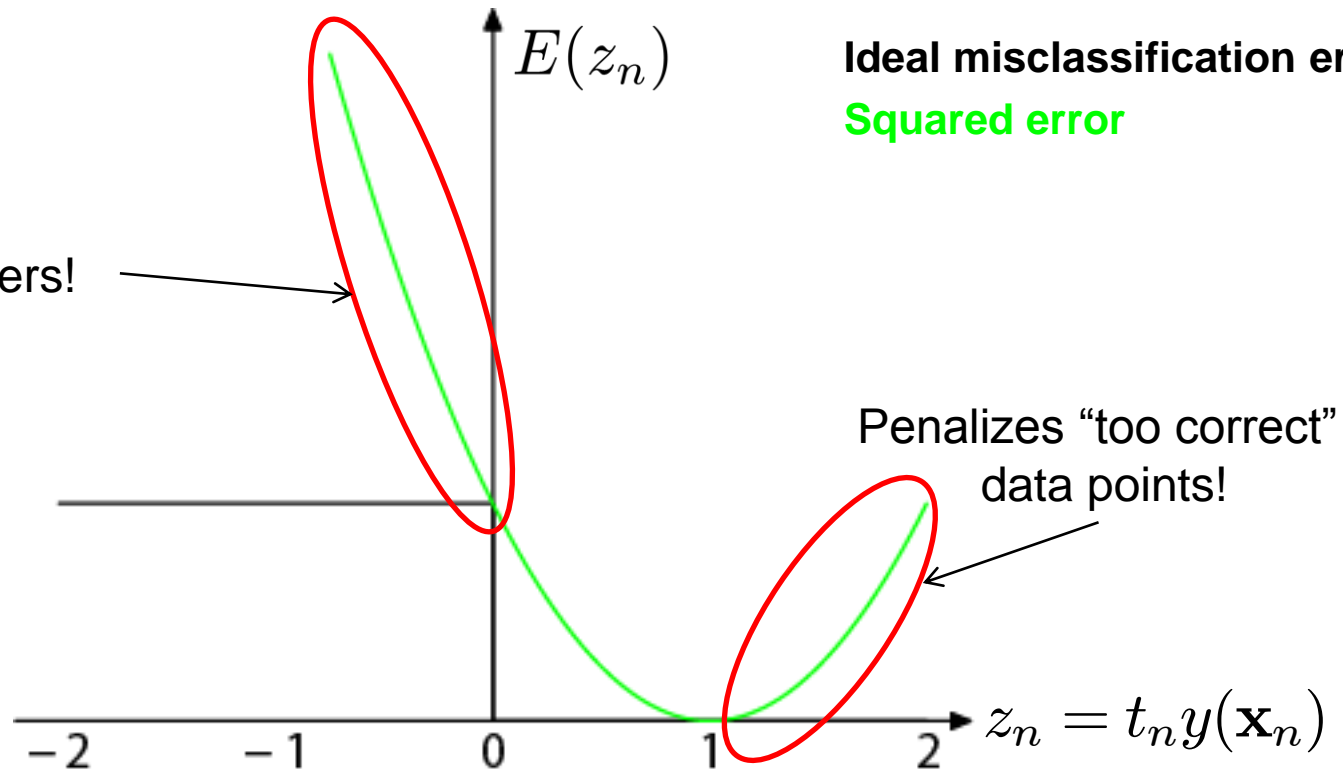
- Ideal misclassification error function (black)
  - This is what we want to approximate (error = #misclassifications)
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.

⇒ We cannot minimize it by gradient descent.

# Note on Error Functions

$$t_n \in \{-1, 1\}$$

Sensitive to outliers!

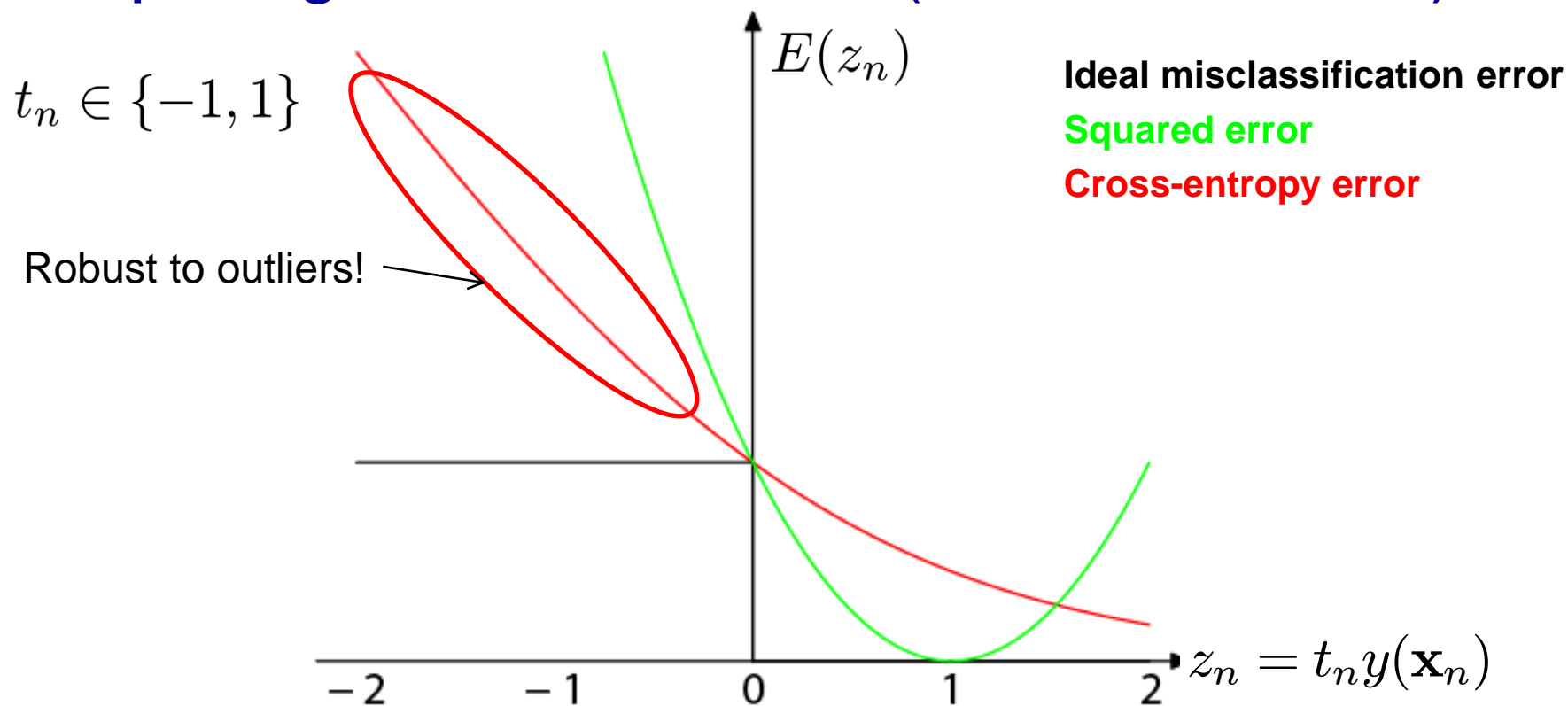


- Squared error used in Least-Squares Classification

- Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes “too correct” data points
- ⇒ Generally does not lead to good classifiers.



# Comparing Error Functions (Loss Functions)

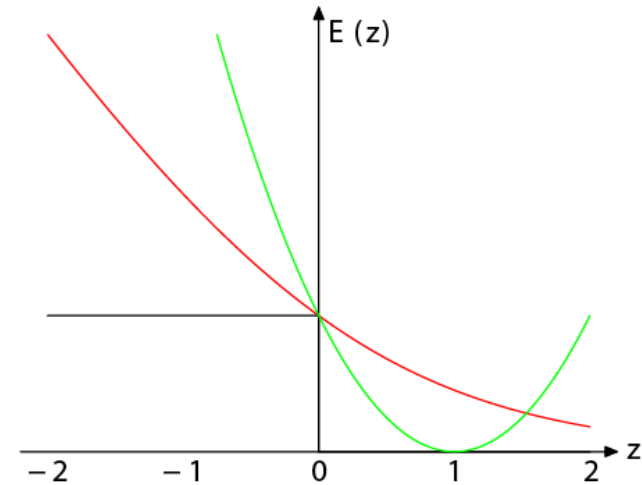


- **Cross-Entropy Error**

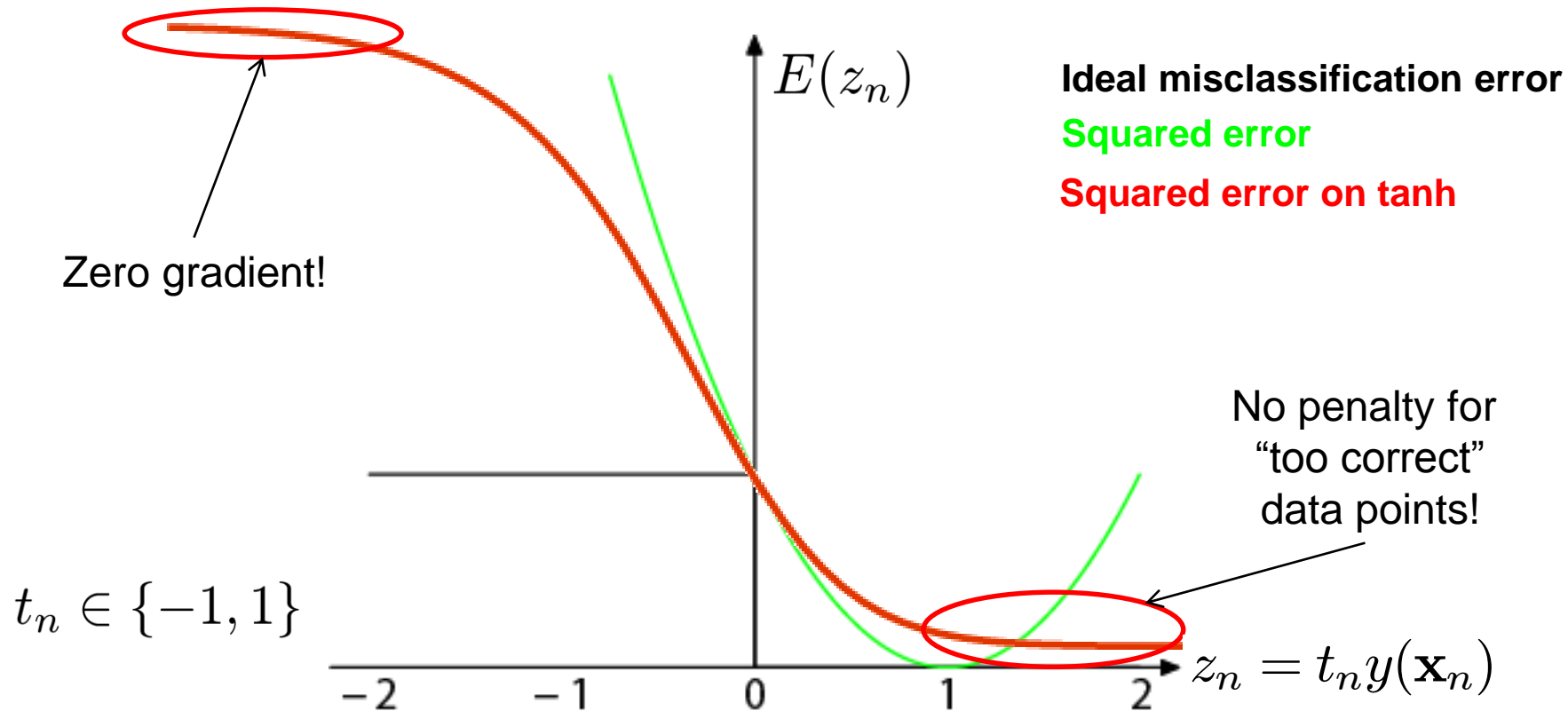
- Minimizer of this error is given by posterior class probabilities.
- Concave error function, unique minimum exists.
- Robust to outliers, error increases only roughly linearly
- But no closed-form solution, requires iterative estimation.

# Overview: Error Functions

- **Ideal Misclassification Error**
    - This is what we would like to optimize.
    - But cannot compute gradients here.
  - **Quadratic Error**
    - Easy to optimize, closed-form solutions exist.
    - But not robust to outliers.
  - **Cross-Entropy Error**
    - Minimizer of this error is given by posterior class probabilities.
    - Concave error function, unique minimum exists.
    - But no closed-form solution, requires iterative estimation.
- ⇒ *Looking at the error function this way gives us an analysis tool to compare the properties of classification approaches.*



## Let's Put This To Practice...



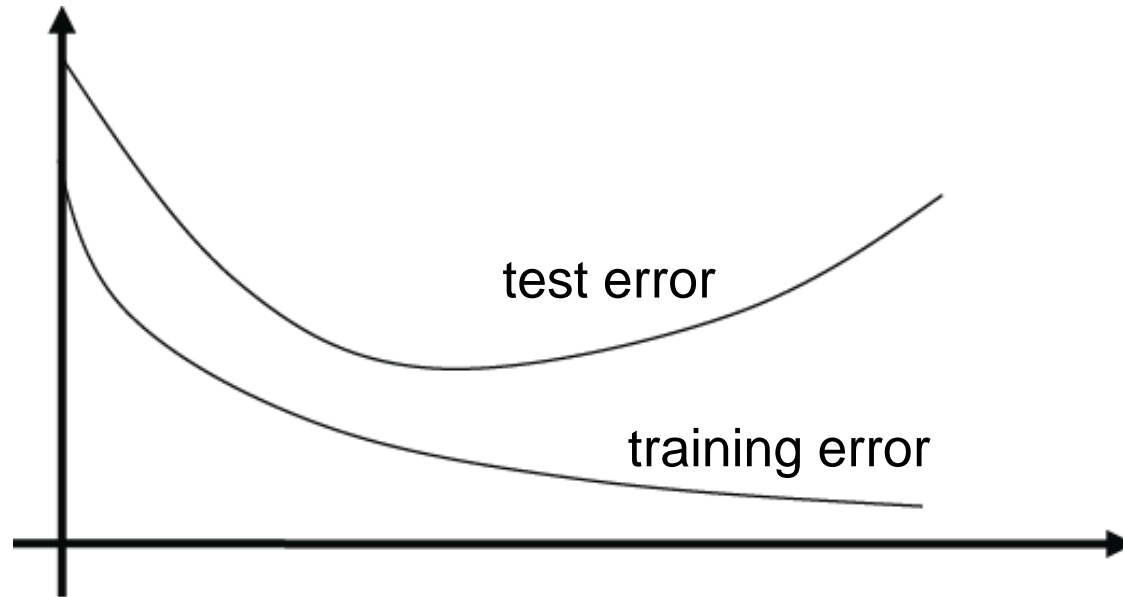
- Squared error on sigmoid/tanh output function

- Avoids penalizing "too correct" data points.
- But: zero gradient for confidently incorrect classifications!
- ⇒ Do not use  $L_2$  loss with sigmoid outputs (instead: cross-entropy)!

# Topics of This Lecture

- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
- **Linear Support Vector Machines**
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

# Generalization and Overfitting

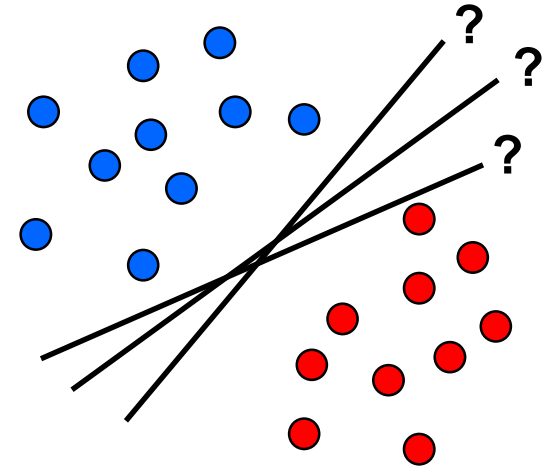


- Goal: predict class labels of new observations
  - Train classification model on limited training set.
  - The further we optimize the model parameters, the more the **training error** will decrease.
  - However, at some point the **test error** will go up again.

⇒ *Overfitting to the training set!*

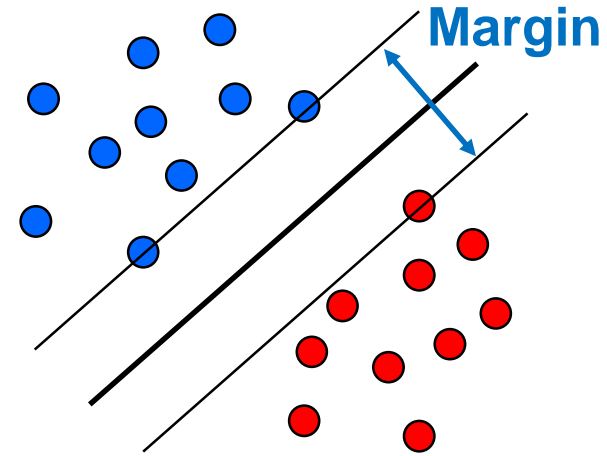
# Example: Linearly Separable Data

- Overfitting is often a problem with linearly separable data
  - Which of the many possible decision boundaries is correct?
  - All of them have zero error on the training set...
  - However, they will most likely result in different predictions on novel test data.  
⇒ Different generalization performance
- How to select the classifier with the best generalization performance?



# Revisiting Our Previous Example...

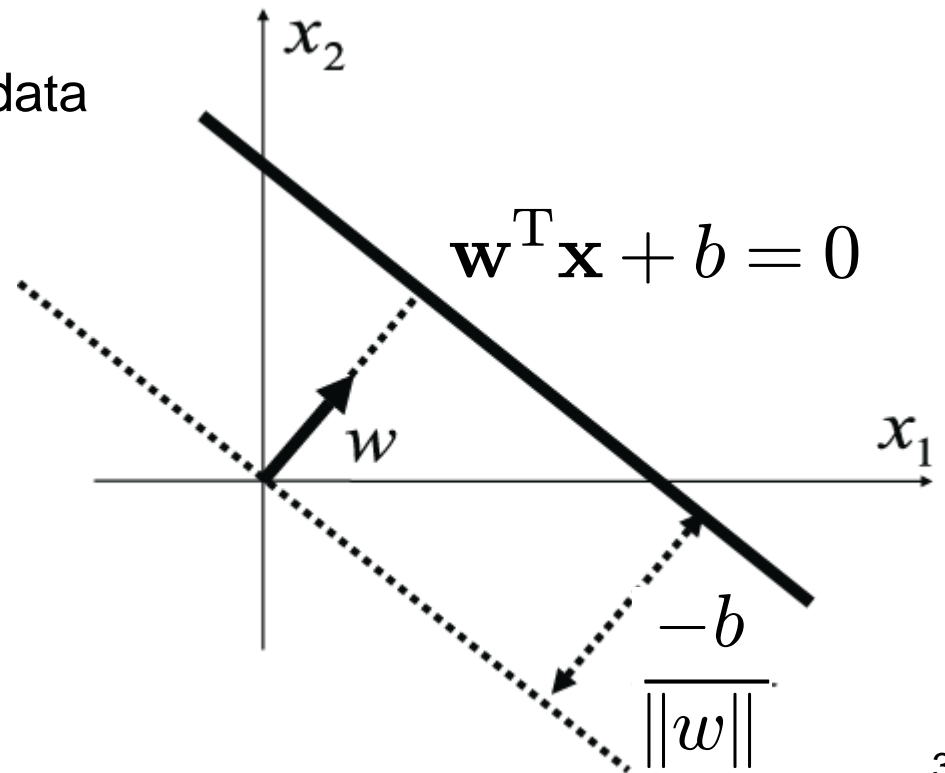
- How to select the classifier with the best generalization performance?
  - Intuitively, we would like to select the classifier which leaves maximal “safety room” for future data points.
  - This can be obtained by maximizing the **margin** between positive and negative data points.
  - It can be shown that the larger the margin, the lower the corresponding classifier’s VC dimension (capacity for overfitting).
- The SVM takes up this idea
  - It searches for the classifier with maximum margin.
  - Formulation as a convex optimization problem  
⇒ Possible to find the globally optimal solution!



# Support Vector Machine (SVM)

- Let's first consider linearly separable data

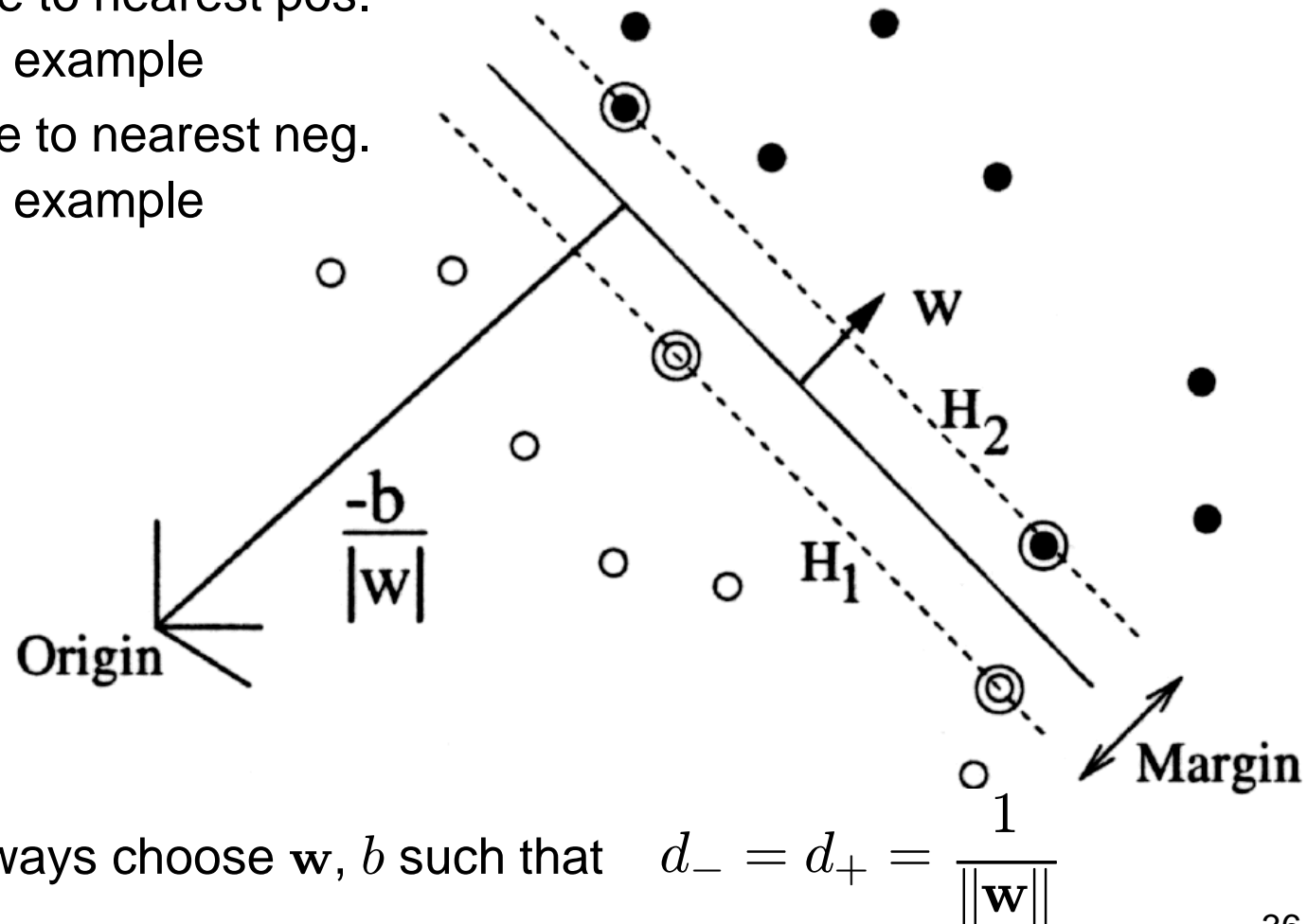
- $N$  training data points  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$   $\mathbf{x}_i \in \mathbb{R}^d$
- Target values  $t_i \in \{-1, 1\}$
- Hyperplane separating the data





# Support Vector Machine (SVM)

- Margin of the hyperplane:  $d_- + d_+$ 
  - $d_+$ : distance to nearest pos. training example
  - $d_-$ : distance to nearest neg. training example



- We can always choose  $w, b$  such that  $d_- = d_+ = \frac{1}{\|w\|}$

# Support Vector Machine (SVM)

- Since the data is linearly separable, there exists a hyperplane with

$$\mathbf{w}^T \mathbf{x}_n + b \geq +1 \quad \text{for } t_n = +1$$

$$\mathbf{w}^T \mathbf{x}_n + b \leq -1 \quad \text{for } t_n = -1$$

- Combined in one equation, this can be written as

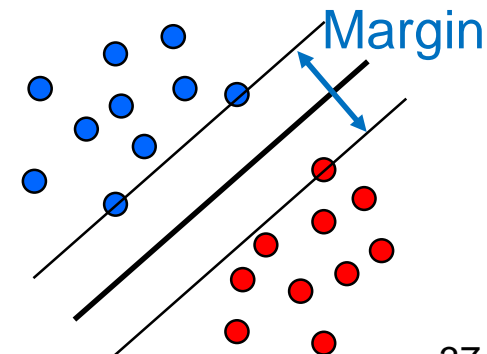
$$t_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \quad \forall n$$

⇒ Canonical representation of the decision hyperplane.

- The equation will hold exactly for the points on the margin

$$t_n(\mathbf{w}^T \mathbf{x}_n + b) = 1$$

- By definition, there will always be at least one such point.



# Support Vector Machine (SVM)

- We can choose  $w$  such that

$$\mathbf{w}^T \mathbf{x}_n + b = +1 \quad \text{for one } t_n = +1$$

$$\mathbf{w}^T \mathbf{x}_n + b = -1 \quad \text{for one } t_n = -1$$

- The distance between those two hyperplanes is then the margin

$$d_- = d_+ = \frac{1}{\|\mathbf{w}\|}$$

$$d_- + d_+ = \frac{2}{\|\mathbf{w}\|}$$

⇒ We can find the hyperplane with maximal margin by minimizing  $\|\mathbf{w}\|^2$

# Support Vector Machine (SVM)

- Optimization problem

- Find the hyperplane satisfying

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

under the constraints

$$t_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \quad \forall n$$

- Quadratic programming problem with linear constraints.
- Can be formulated using Lagrange multipliers.

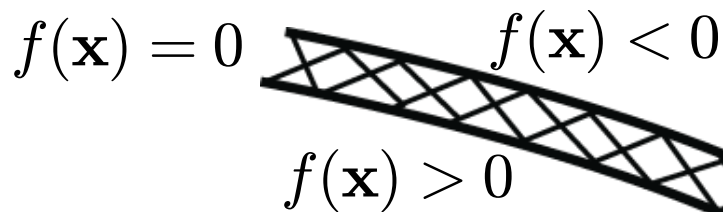
- *Who is already familiar with Lagrange multipliers?*

- Let's look at a real-life example...

# Recap: Lagrange Multipliers

- Problem

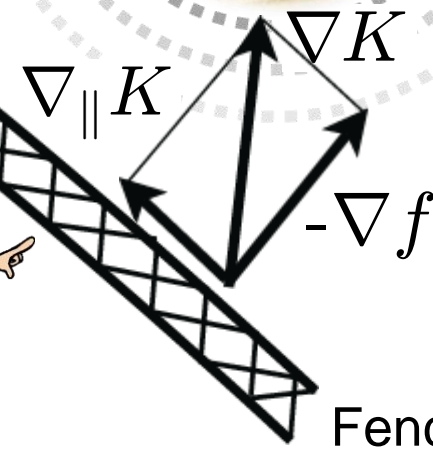
- We want to maximize  $K(\mathbf{x})$  subject to constraints  $f(\mathbf{x}) = 0$ .
- Example: we want to get as close as possible, **but there is a fence.**
- How should we move?



- We want to maximize  $\nabla K$
- But we can only move parallel to the fence, i.e. along

$$\nabla_{\parallel} K = \nabla K + \lambda \nabla f$$

with  $\lambda \neq 0$ .



# Recap: Lagrange Multipliers

- Problem

- We want to maximize  $K(\mathbf{x})$  subject to constraints  $f(\mathbf{x}) = 0$ .
- Example: we want to get as close as possible, but there is a fence.
- How should we move?

$f(\mathbf{x}) = 0$        $f(\mathbf{x}) < 0$

⇒ Optimize

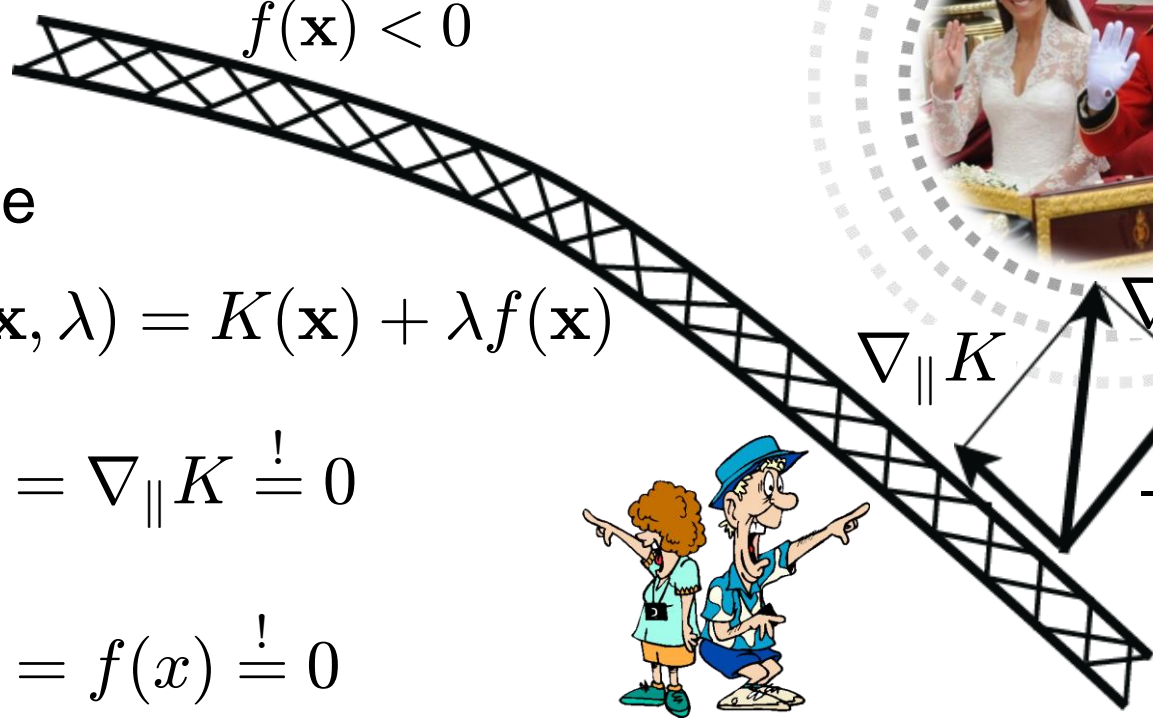
$$\max_{\mathbf{x}, \lambda} L(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$$

$$\frac{\partial L}{\partial \mathbf{x}} = \nabla_{\parallel} K \stackrel{!}{=} 0$$

$$\frac{\partial L}{\partial \lambda} = f(\mathbf{x}) \stackrel{!}{=} 0$$



$K(\mathbf{x})$



Fence  $f$



# Recap: Lagrange Multipliers

- Problem

- Now let's look at constraints of the form  $f(\mathbf{x}) \geq 0$ .
- Example: There might be a hill from which we can see better...
- Optimize  $\max_{\mathbf{x}, \lambda} L(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$

$$f(\mathbf{x}) = 0 \qquad f(\mathbf{x}) < 0$$

- Two cases

- Solution lies on boundary  
 $\Rightarrow f(\mathbf{x}) = 0$  for some  $\lambda > 0$
- Solution lies inside  $f(\mathbf{x}) > 0$   
 $\Rightarrow$  Constraint inactive:  $\lambda = 0$
- In both cases  
 $\Rightarrow \lambda f(\mathbf{x}) = 0$



Fence  $f$

# Recap: Lagrange Multipliers

- Problem

- Now let's look at constraints of the form  $f(\mathbf{x}) \geq 0$ .
- Example: There might be a hill from which we can see better...
- Optimize  $\max_{\mathbf{x}, \lambda} L(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$

$f(\mathbf{x}) = 0$

Karush-Kuhn-Tucker (KKT)

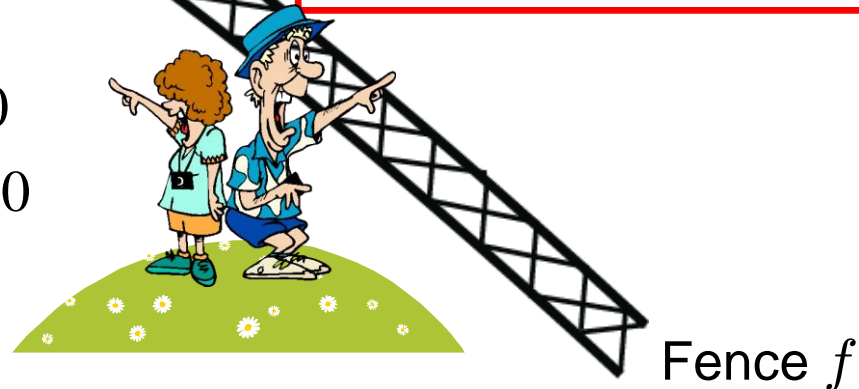
conditions:  $\lambda \geq 0$

$f(\mathbf{x}) \geq 0$

$\lambda f(\mathbf{x}) = 0$

- Two cases

- Solution lies on boundary  
 $\Rightarrow f(\mathbf{x}) = 0$  for some  $\lambda > 0$
- Solution lies inside  $f(\mathbf{x}) > 0$   
 $\Rightarrow$  Constraint inactive:  $\lambda = 0$
- In both cases  
 $\Rightarrow \lambda f(\mathbf{x}) = 0$



B. Leibe



# SVM – Lagrangian Formulation

- Find hyperplane minimizing  $\|\mathbf{w}\|^2$  under the constraints

$$t_n(\mathbf{w}^T \mathbf{x}_n + b) - 1 \geq 0 \quad \forall n$$

- Lagrangian formulation

➤ Introduce positive Lagrange multipliers:  $a_n \geq 0 \quad \forall n$

➤ Minimize Lagrangian (“**primal form**”)

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n \{t_n(\mathbf{w}^T \mathbf{x}_n + b) - 1\}$$

➤ I.e., find  $\mathbf{w}$ ,  $b$ , and  $\mathbf{a}$  such that

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^N a_n t_n = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^N a_n t_n \mathbf{x}_n$$

# SVM – Lagrangian Formulation

- Lagrangian primal form

$$\begin{aligned}
 L_p &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n \{t_n(\mathbf{w}^T \mathbf{x}_n + b) - 1\} \\
 &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n \{t_n y(\mathbf{x}_n) - 1\}
 \end{aligned}$$

- The solution of  $L_p$  needs to fulfill the KKT conditions
  - Necessary and sufficient conditions

$$\begin{aligned}
 a_n &\geq 0 \\
 t_n y(\mathbf{x}_n) - 1 &\geq 0 \\
 a_n \{t_n y(\mathbf{x}_n) - 1\} &= 0
 \end{aligned}$$

**KKT:**

$$\begin{aligned}
 \lambda &\geq 0 \\
 f(\mathbf{x}) &\geq 0 \\
 \lambda f(\mathbf{x}) &= 0
 \end{aligned}$$

# SVM – Solution (Part 1)

- Solution for the hyperplane
  - Computed as a linear combination of the training examples

$$\mathbf{w} = \sum_{n=1}^N a_n t_n \mathbf{x}_n$$

- Because of the KKT conditions, the following must also hold

$$a_n (t_n (\mathbf{w}^T \mathbf{x}_n + b) - 1) = 0$$

$$\text{KKT:} \\ \lambda f(\mathbf{x}) = 0$$

- This implies that  $a_n > 0$  only for training data points for which

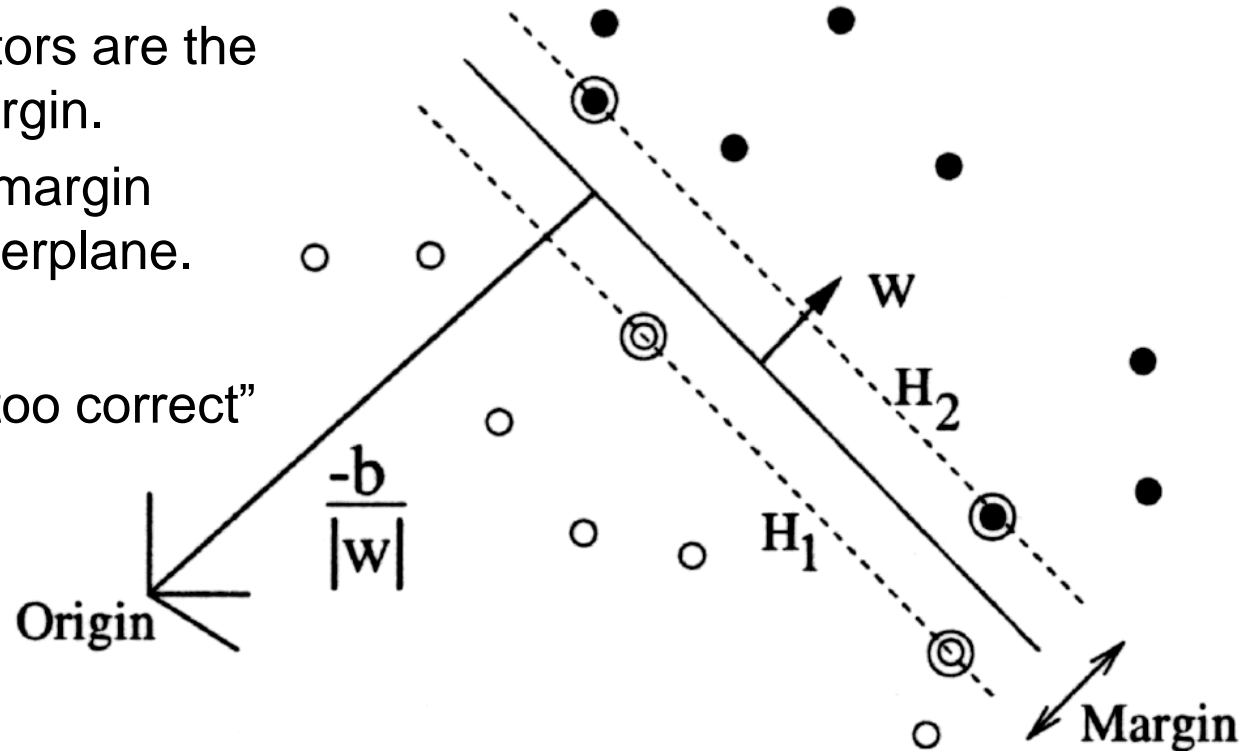
$$(t_n (\mathbf{w}^T \mathbf{x}_n + b) - 1) = 0$$

⇒ *Only some of the data points actually influence the decision boundary!*

# SVM – Support Vectors

- The training points for which  $a_n > 0$  are called “support vectors”.
- Graphical interpretation:
  - The support vectors are the points on the margin.
  - They *define* the margin and thus the hyperplane.

⇒ Robustness to “too correct” points!



# SVM – Solution (Part 2)

- Solution for the hyperplane
  - To define the decision boundary, we still need to know  $b$ .
  - Observation: any support vector  $\mathbf{x}_n$  satisfies

$$t_n y(\mathbf{x}_n) = t_n \left( \sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^T \mathbf{x}_n + b \right) = 1$$

KKT:

 $f(\mathbf{x}) \geq 0$

- Using  $t_n^2 = 1$  we can derive:  $b = t_n - \sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^T \mathbf{x}_n$
- In practice, it is more robust to average over all support vectors:

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left( t_n - \sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^T \mathbf{x}_n \right)$$

# SVM – Discussion (Part 1)

- Linear SVM
  - Linear classifier
  - SVMs have a “guaranteed” generalization capability.
  - Formulation as convex optimization problem.  
⇒ Globally optimal solution!
- Primal form formulation
  - Solution to quadratic prog. problem in  $M$  variables is in  $\mathcal{O}(M^3)$ .
  - Here:  $D$  variables  $\Rightarrow \mathcal{O}(D^3)$
  - Problem: scaling with high-dim. data (“curse of dimensionality”)

# SVM – Dual Formulation

- Improving the scaling behavior: rewrite  $L_p$  in a dual form

$$\begin{aligned}
 L_p &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n \{t_n (\mathbf{w}^T \mathbf{x}_n + b) - 1\} \\
 &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n t_n \mathbf{w}^T \mathbf{x}_n - b \sum_{n=1}^N a_n t_n + \sum_{n=1}^N a_n
 \end{aligned}$$

- Using the constraint  $\sum_{n=1}^N a_n t_n = 0$  we obtain

$$L_p = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n t_n \mathbf{w}^T \mathbf{x}_n + \sum_{n=1}^N a_n$$

$$\frac{\partial L_p}{\partial b} = 0$$

## SVM – Dual Formulation

$$L_p = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n t_n \mathbf{w}^T \mathbf{x}_n + \sum_{n=1}^N a_n$$

- ▶ Using the constraint  $\mathbf{w} = \sum_{n=1}^N a_n t_n \mathbf{x}_n$  we obtain

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0$$

$$\begin{aligned} L_p &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n t_n \sum_{m=1}^N a_m t_m \mathbf{x}_m^T \mathbf{x}_n + \sum_{n=1}^N a_n \\ &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m (\mathbf{x}_m^T \mathbf{x}_n) + \sum_{n=1}^N a_n \end{aligned}$$



# SVM – Dual Formulation

$$L = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m (\mathbf{x}_m^T \mathbf{x}_n) + \sum_{n=1}^N a_n$$

- ▶ Applying  $\frac{1}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \mathbf{w}^T \mathbf{w}$  and again using  $\mathbf{w} = \sum_{n=1}^N a_n t_n \mathbf{x}_n$

$$\frac{1}{2} \mathbf{w}^T \mathbf{w} = \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m (\mathbf{x}_m^T \mathbf{x}_n)$$

- ▶ Inserting this, we get the **Wolfe dual**

$$L_d(\mathbf{a}) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m (\mathbf{x}_m^T \mathbf{x}_n)$$

# SVM – Dual Formulation

- Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m (\mathbf{x}_m^T \mathbf{x}_n)$$

under the conditions

$$a_n \geq 0 \quad \forall n$$

$$\sum_{n=1}^N a_n t_n = 0$$

- The hyperplane is given by the  $N_S$  support vectors:

$$\mathbf{w} = \sum_{n=1}^{N_S} a_n t_n \mathbf{x}_n$$

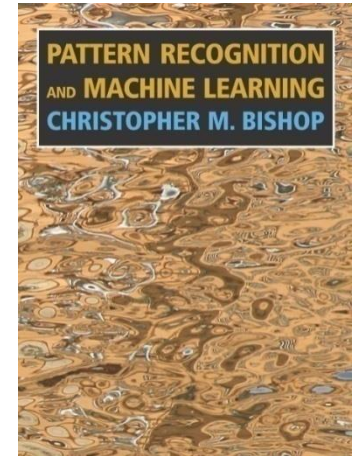
# SVM – Discussion (Part 2)

- Dual form formulation
  - In going to the dual, we now have a problem in  $N$  variables ( $a_n$ ).
  - Isn't this worse??? We penalize large training sets!
- However...
  1. SVMs have sparse solutions:  $a_n \neq 0$  only for support vectors!  
⇒ This makes it possible to construct efficient algorithms
    - e.g. Sequential Minimal Optimization (SMO)
    - Effective runtime between  $\mathcal{O}(N)$  and  $\mathcal{O}(N^2)$ .
  2. We have avoided the dependency on the dimensionality.  
⇒ This makes it possible to work with infinite-dimensional feature spaces by using suitable basis functions  $\phi(\mathbf{x})$ .  
⇒ We'll see that in the next lecture...

# References and Further Reading

- More information on SVMs can be found in Chapter 7.1 of Bishop's book.

Christopher M. Bishop  
Pattern Recognition and Machine Learning  
Springer, 2006



- Additional information about Statistical Learning Theory and a more in-depth introduction to SVMs are available in the following tutorial:
  - C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](#), Data Mining and Knowledge Discovery, Vol. 2(2), pp. 121-167 1998.