Machine Learning – Lecture 6

Linear Discriminants II

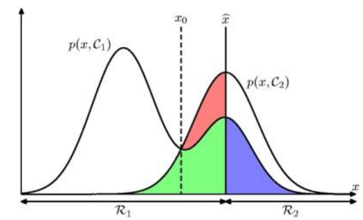
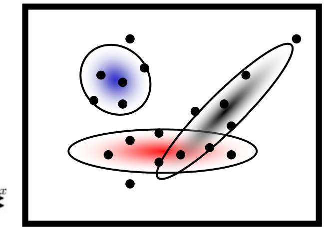
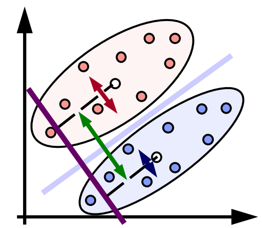
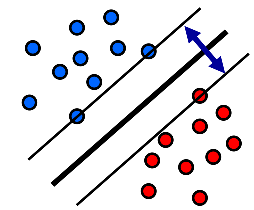
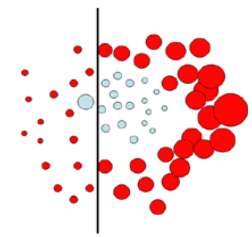
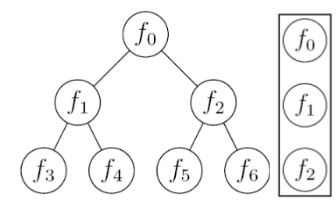
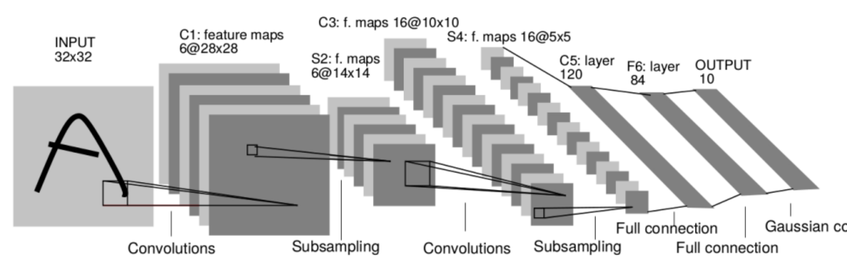
30.10.2019

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

Course Outline

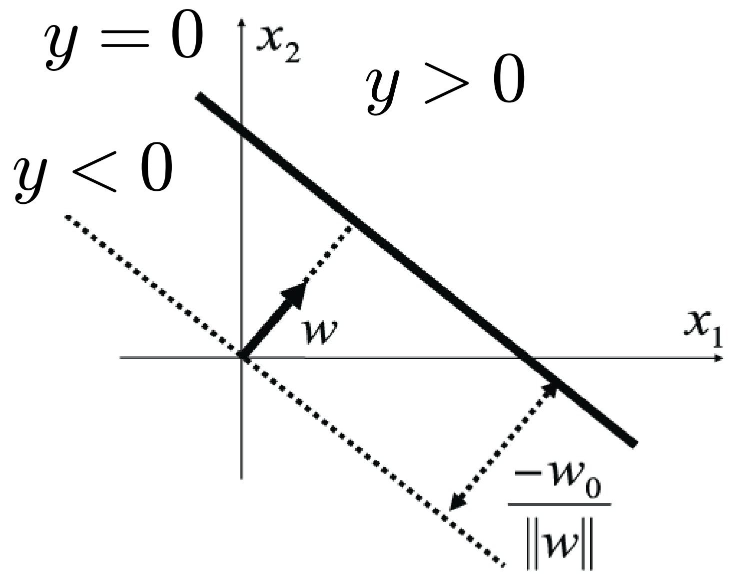
- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Randomized Trees, Forests & Ferns
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent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

Recap: Linear Discriminant Functions

- Basic idea
  - Directly encode decision boundary
  - Minimize misclassification probability directly.

- Linear discriminant functions



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

weight vector

"bias"
(= threshold)

  - $\mathbf{w}$, w_0 define a hyperplane in $\mathbb{R}^D$.
  - If a data set can be perfectly classified by a linear discriminant, then we call it linearly separable.

Recap: Least-Squares Classification

- Simplest approach
  - Directly try to minimize the sum-of-squares error

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

$$E_D(\widetilde{\mathbf{W}}) = \frac{1}{2}\mathrm{Tr}\left\{(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})^{\mathrm{T}}(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})\right\}$$

  - Setting the derivative to zero yields

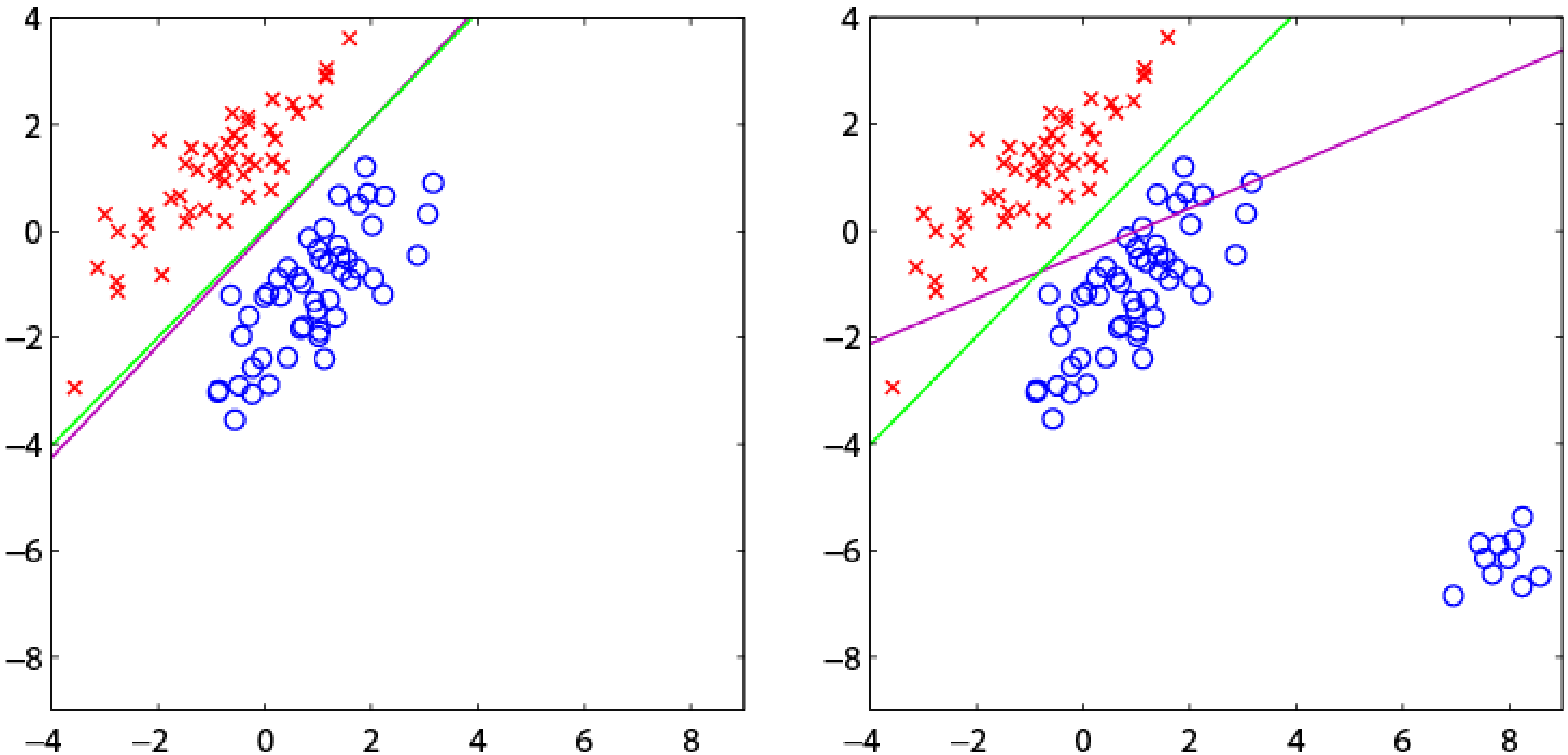
$$\widetilde{\mathbf{W}} \;=\; (\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{T} = \widetilde{\mathbf{X}}^{\dagger}\mathbf{T} = (\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{T}$$

  - We then obtain the discriminant function as

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{x}} = \mathbf{T}^{\mathrm{T}}\left(\widetilde{\mathbf{X}}^{\dagger}\right)^{\mathrm{T}}\widetilde{\mathbf{x}}$$

$\Rightarrow$ Exact, closed-form solution for the discriminant function parameters.

Recap: Problems with Least Squares



- Least-squares is very sensitive to outliers!
  - The error function penalizes predictions that are "too correct".

Image source: C.M. Bishop, 2006

Recap: Generalized Linear Models

- Generalized linear model

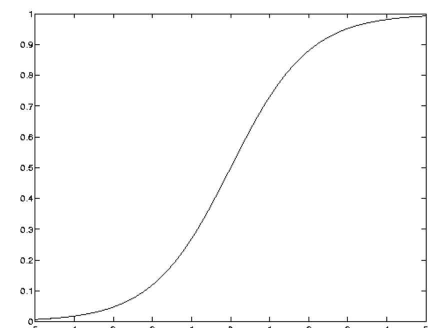
$$y(\mathbf{x}) = g(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0)$$

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

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

  - 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.

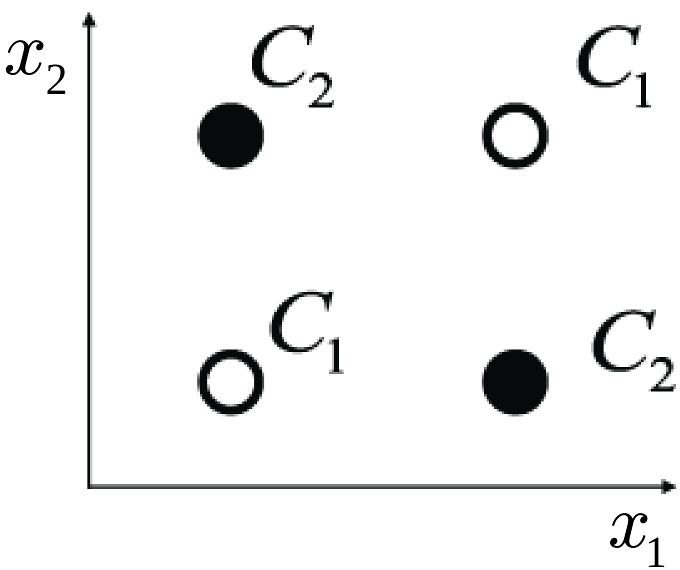


$$g(a) \equiv \frac{1}{1+\exp(-a)}$$

Linear Separability

- Up to now: restrictive assumption
  - Only consider linear decision boundaries

- Classical counterexample: XOR



Generalized Linear Discriminants

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

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

  - Purpose of $\phi_j(\mathbf{x})$: basis functions
  - Allow non-linear decision boundaries.
  - By choosing the right ϕ_j, every continuous function can (in principle) be approximated with arbitrary accuracy.

- Notation

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}) \qquad \text{with} \quad \phi_0(\mathbf{x}) = 1$$

Linear Basis Function Models

- Generalized Linear Discriminant Model

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

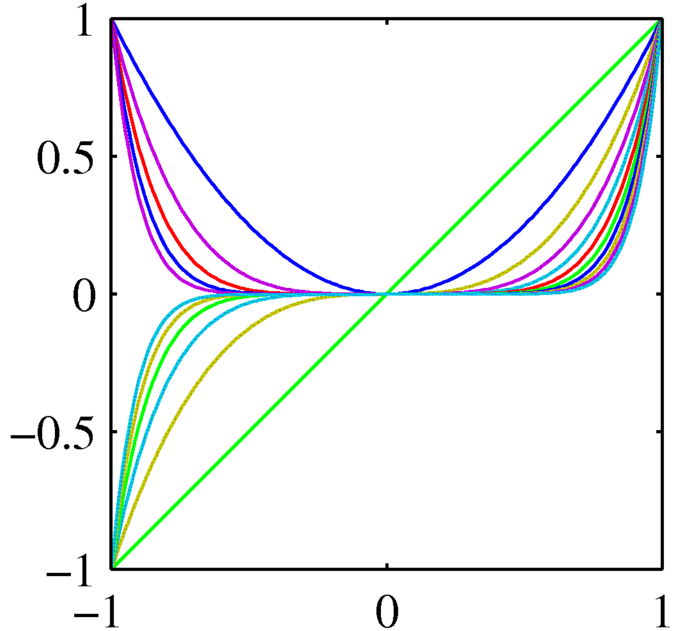
  - where $\phi_j(\mathbf{x})$ are known as *basis functions*.
  - Typically, $\phi_0(\mathbf{x}) = 1$, so that w_0 acts as a bias.
  - In the simplest case, we use linear basis functions: $\phi_d(\mathbf{x}) = x_d$.

- *Let's take a look at some other possible basis functions...*

Linear Basis Function Models (2)

- Polynomial basis functions

$$\phi_j(x) = x^j.$$



- Properties
  - Global

  $\Rightarrow$ A small change in x affects all basis functions.

- Result
  - If we use polynomial basis functions, the decision boundary will be a polynomial function of x.

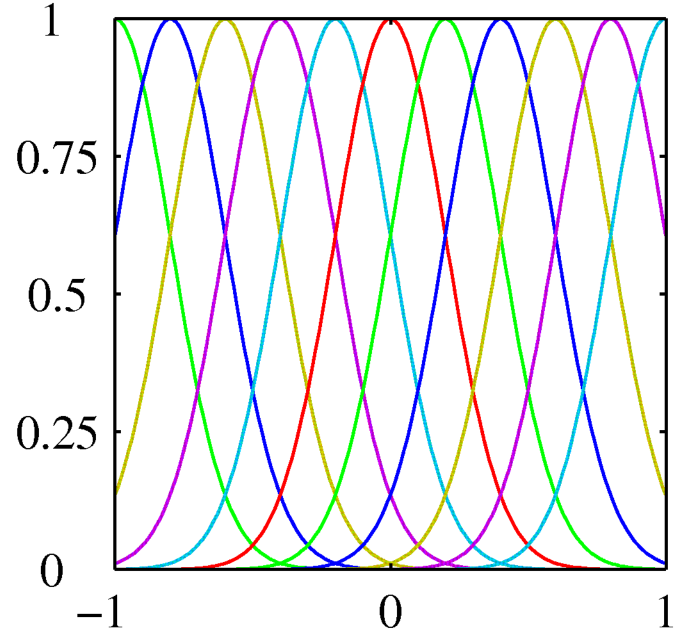
  $\Rightarrow$ Nonlinear decision boundaries

  $\Rightarrow$ However, we still solve a linear problem in $\phi(x)$.

Linear Basis Function Models (3)

- Gaussian basis functions

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$



- Properties
  - Local

  $\Rightarrow$ A small change in x affects only nearby basis functions.
  - μ_j and s control location and scale (width).

Image source: C.M. Bishop, 2006

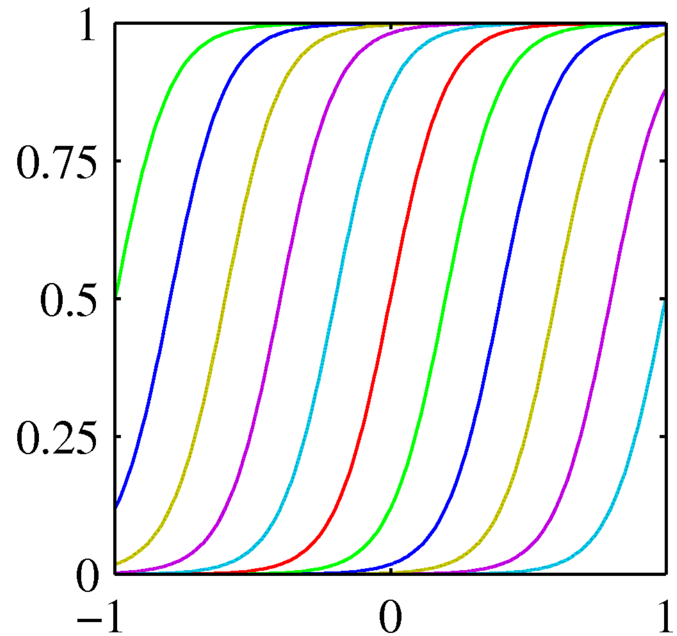
Linear Basis Function Models (4)

- Sigmoid basis functions

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

  - where

$$\sigma(a) = \frac{1}{1+\exp(-a)}.$$



- Properties
  - Local

$\Rightarrow$ A small change in x affects only nearby basis functions.

  - μ_j and s control location and scale (slope).

Topics of This Lecture

- Gradient Descent
- Logistic Regression
  - Probabilistic discriminative models
  - Logistic sigmoid (logit function)
  - Cross-entropy error
  - Iteratively Reweighted Least Squares
- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error

Gradient Descent

- Learning the weights $\mathbf{w}$:
  - N training data points: $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
  - K outputs of decision functions: $y_k(\mathbf{x}_n; \mathbf{w})$
  - Target vector for each data point: $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$

  - Error function (least-squares error) of linear model

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}\right)^2$$

$$= \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(\sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}_n) - t_{kn}\right)^2$$

Slide credit: Bernt Schiele

Gradient Descent

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

- Idea (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.

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

η: Learning rate

  - This simple scheme corresponds to a 1st-order Taylor expansion (There are more complex procedures available).

Gradient Descent – Basic Strategies

- “Batch learning”

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

η: Learning rate

  - Compute the gradient based on all training data:

$$\frac{\partial E(\mathbf{w})}{\partial w_{kj}}$$

Gradient Descent – Basic Strategies

- "Sequential updating"

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

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

η: Learning rate

  - Compute the gradient based on a single data point at a time:

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{kj}}$$

Gradient Descent

- Error function

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \;=\; \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(\sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}_n) - t_{kn}\right)^2$$

$$E_n(\mathbf{w}) \;=\; \frac{1}{2}\sum_{k=1}^{K}\left(\sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}_n) - t_{kn}\right)^2$$

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} \;=\; \left(\sum_{\tilde{j}=1}^{M} w_{k\tilde{j}}\phi_{\tilde{j}}(\mathbf{x}_n) - t_{kn}\right)\phi_j(\mathbf{x}_n)$$

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

Slide credit: Bernt Schiele

Gradient Descent

- Delta rule (=LMS rule)

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

  - where

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

$\Rightarrow$ Simply feed back the input data point, weighted by the classification error.

Gradient Descent

- Cases with differentiable, non-linear activation function

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

- Gradient descent

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

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

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

Slide credit: Bernt Schiele

Summary: Generalized Linear Discriminants

- Properties
  - General class of decision functions.
  - Nonlinearity $g(\cdot)$ and basis functions ϕ_j allow us to address linearly non-separable problems.
  - Shown simple sequential learning approach for parameter estimation using gradient descent.
  - Better 2^{nd} order gradient descent approaches are available (e.g. Newton-Raphson), but they are more expensive to compute.
- Limitations / Caveats
  - Flexibility of model is limited by curse of dimensionality
    - $g(\cdot)$ and ϕ_j often introduce additional parameters.
    - Models are either limited to lower-dimensional input space or need to share parameters.
  - Linearly separable case often leads to overfitting.
    - Several possible parameter choices minimize training error.

Topics of This Lecture

- Gradient Descent
- Logistic Regression
  - Probabilistic discriminative models
  - Logistic sigmoid (logit function)
  - Cross-entropy error
  - Iteratively Reweighted Least Squares
- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error

Probabilistic Discriminative Models

- We have seen that we can write

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(a) = \frac{1}{1+\exp(-a)}$$

logistic sigmoid function

- We can obtain the familiar probabilistic model by setting

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

- Or we can use generalized linear discriminant models

$$a = \mathbf{w}^T\mathbf{x}$$

or $$a = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x})$$

Probabilistic Discriminative Models

- In the following, we will consider models of the form

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

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

- This model is called logistic regression.

- Why should we do this? What advantage does such a model have compared to modeling the probabilities?

$$p(\mathcal{C}_1|\phi) = \frac{p(\phi|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\phi|\mathcal{C}_1)p(\mathcal{C}_1) + p(\phi|\mathcal{C}_2)p(\mathcal{C}_2)}$$

- Any ideas?

Comparison

- Let's look at the number of parameters…
  - Assume we have an M-dimensional feature space ϕ.
  - And assume we represent $p(\phi|\mathcal{C}_k)$ and $p(\mathcal{C}_k)$ by Gaussians.
  - How many parameters do we need?
    - For the means: $2M$
    - For the covariances: $M(M+1)/2$
    - Together with the class priors, this gives $M(M+5)/2+1$ parameters!
  - How many parameters do we need for logistic regression?

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

    - Just the values of $\mathbf{w} \Rightarrow M$ parameters.

$\Rightarrow$ *For large M, logistic regression has clear advantages!*

Logistic Sigmoid

- Properties
  - Definition: $$\sigma(a) = \frac{1}{1+\exp(-a)}$$
  - Inverse: $$a = \ln\left(\frac{\sigma}{1-\sigma}\right)$$ "logit" function
  - Symmetry property: $$\sigma(-a) = 1 - \sigma(a)$$
  - Derivative: $$\frac{d\sigma}{da} = \sigma(1-\sigma)$$

Logistic Regression

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

- 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

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

Gradient of the Error Function

- Gradient for logistic regression

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

- 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…

A More Efficient Iterative Method…

- Second-order Newton-Raphson gradient descent scheme

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

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

- Properties
  - Local quadratic approximation to the log-likelihood.
  - Faster convergence.

Newton-Raphson for Least-Squares Estimation

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

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

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

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

where $\mathbf{\Phi} = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_N^T \end{bmatrix}$

- Resulting update scheme:

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

Closed-form solution!

Newton-Raphson for Logistic Regression

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

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

$$\boxed{\frac{dy_n}{d\mathbf{w}} = y_n(1-y_n)\boldsymbol{\phi}_n}$$

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

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

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

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

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

- 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.
  - $\Rightarrow$ 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 ~500.

Topics of This Lecture

- Gradient Descent
- Logistic Regression
  - Probabilistic discriminative models
  - Logistic sigmoid (logit function)
  - Cross-entropy error
  - Iteratively Reweighted Least Squares
- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error

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

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

- Softmax regression
  - Generalization to K classes using indicator functions.

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

Optimization

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

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

- 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.

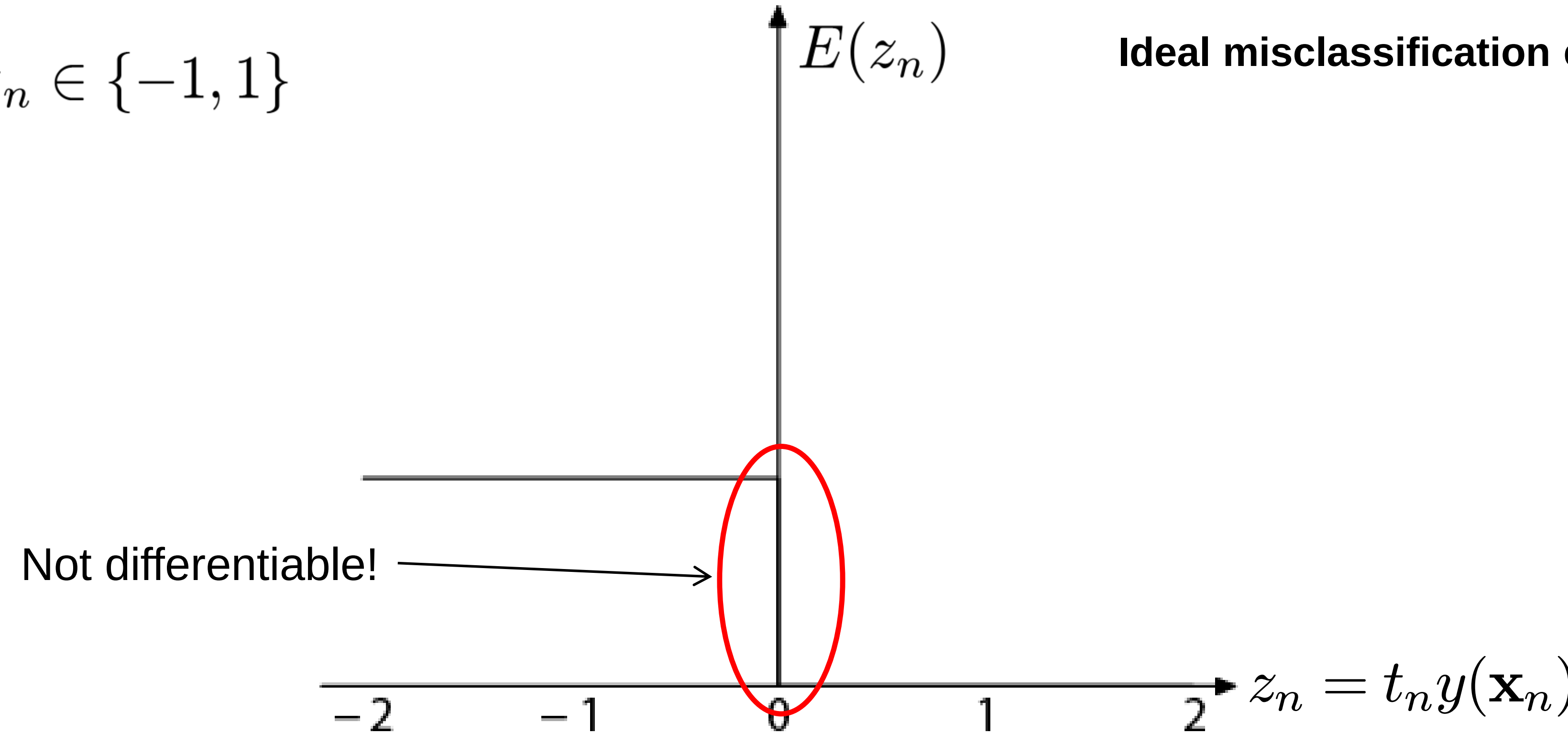
Topics of This Lecture

- Gradient Descent
- Logistic Regression
  - Probabilistic discriminative models
  - Logistic sigmoid (logit function)
  - Cross-entropy error
  - Iteratively Reweighted Least Squares
- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error

Note on Error Functions

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

Ideal misclassification error

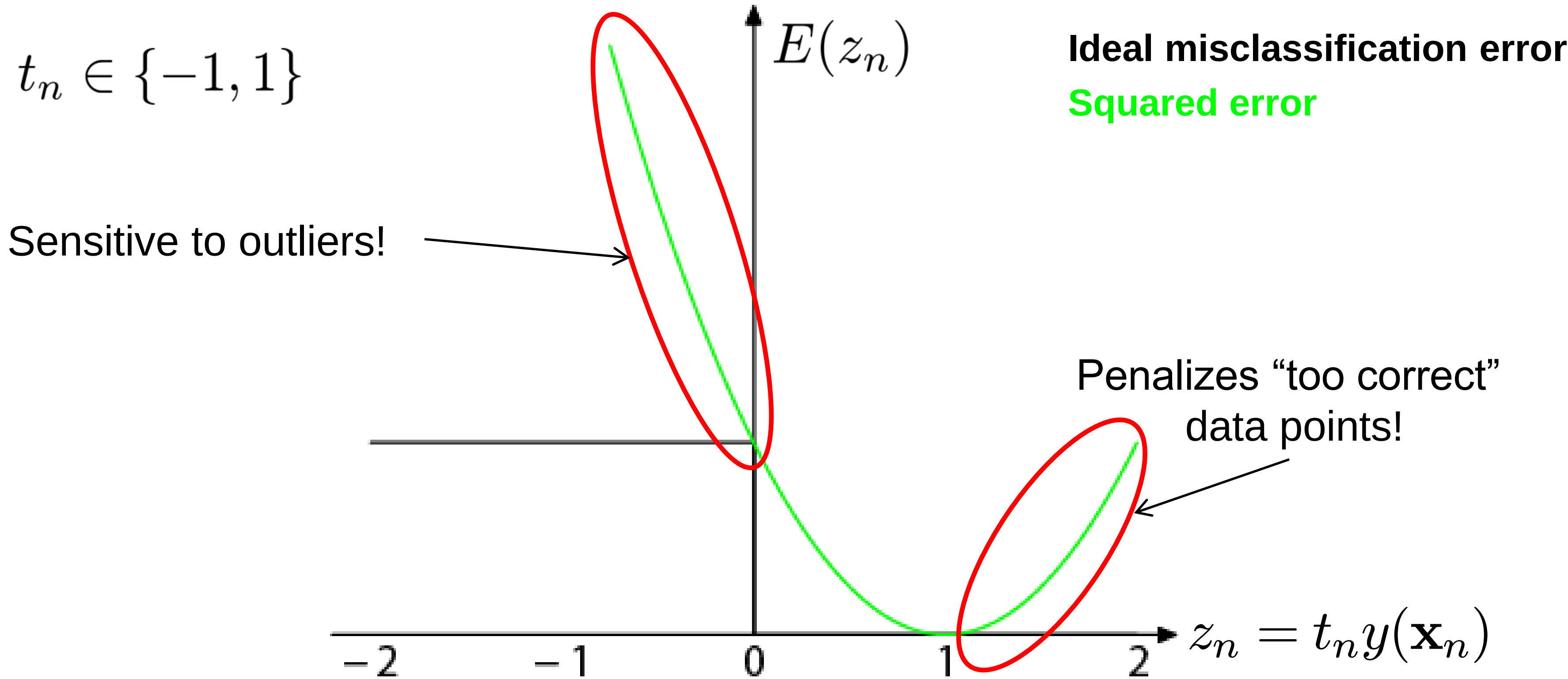


- Ideal misclassification error function (black)
  - This is what we want to approximate (error = #misclassifications)
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.

  $\Rightarrow$ We cannot minimize it by gradient descent.

Image source: Bishop, 2006

Note on Error Functions

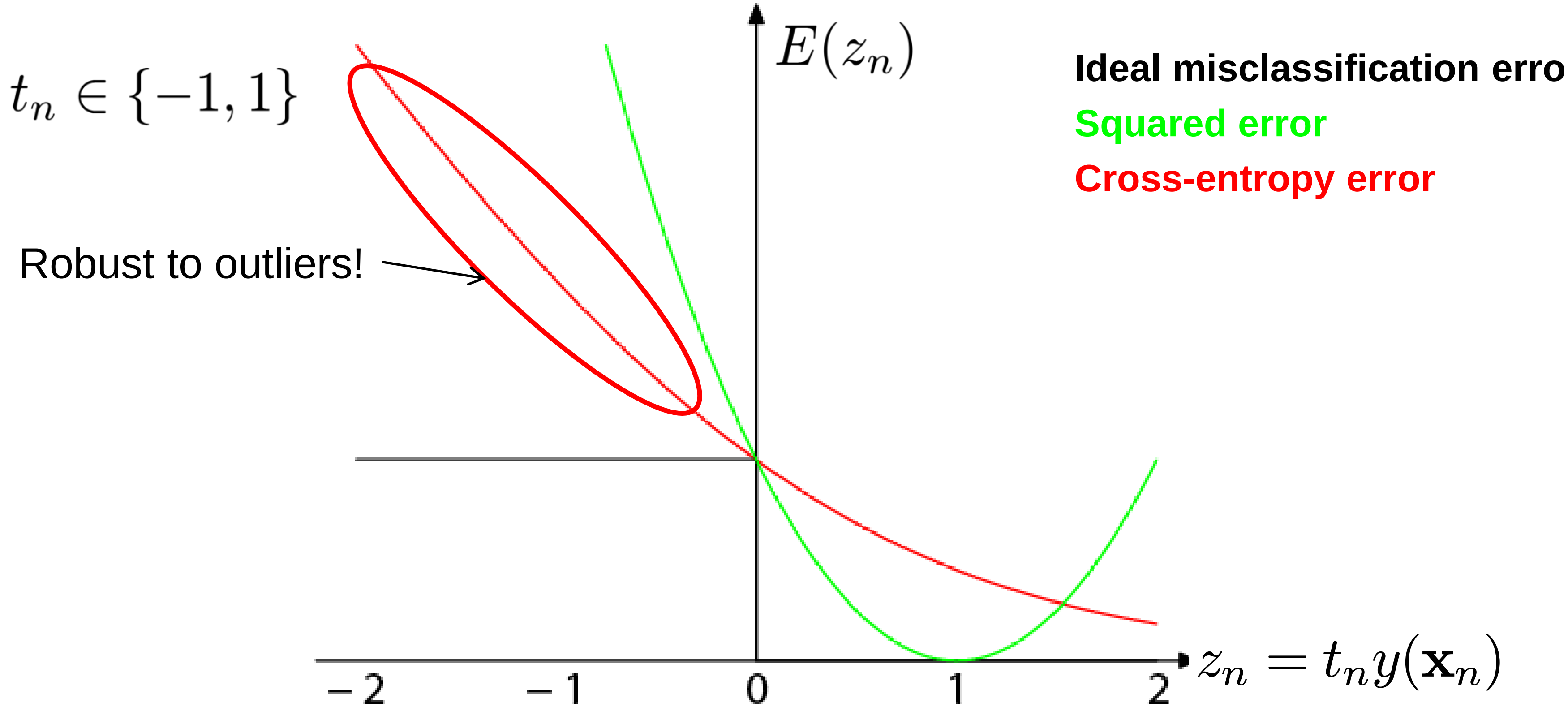


- Squared error used in Least-Squares Classification
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes "too correct" data points

  $\Rightarrow$ Generally does not lead to good classifiers.

Image source: Bishop, 2006

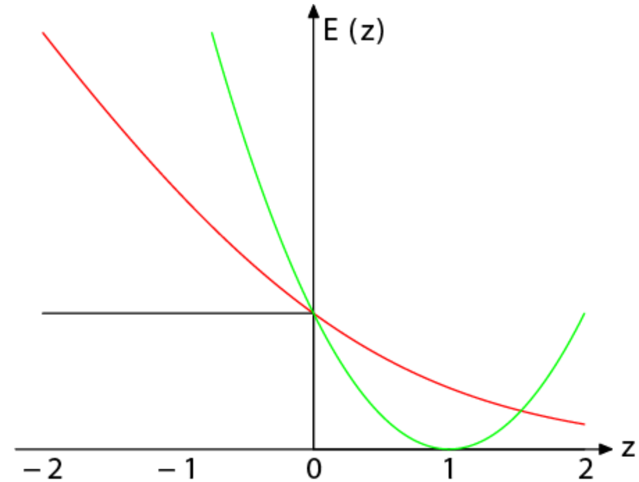
Comparing Error Functions (Loss Functions)



- 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.

Image source: Bishop, 2006

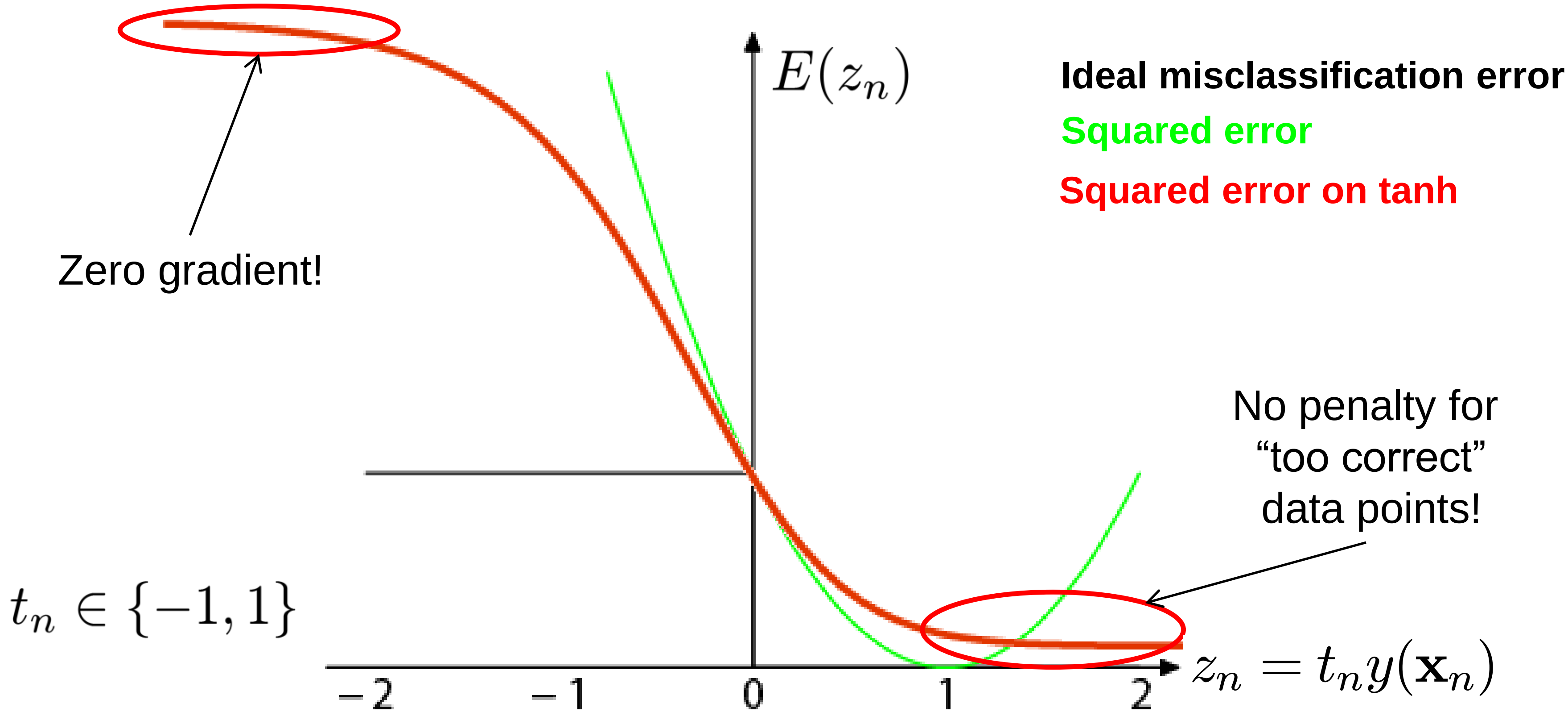
Overview: Error Functions



- 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.

$\Rightarrow$ *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!

  $\Rightarrow$ Do not use L_2 loss with sigmoid outputs (instead: cross-entropy)!

Image source: Bishop, 2006

References and Further Reading

- More information on Linear Discriminant Functions can be found in Chapter 4 of Bishop’s book (in particular Chapter 4.1 - 4.3).

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