Machine Intelligence I

Prof. Dr. Klaus Obermayer

June 24, 2015

Contents

1 Artificial Neural Networks & Empirical Risk Minimization

1.1 General Comments

Observation: Brains are good at solving problems which are difficult for (current) machines (and vice versa) in many different areas of pattern recognition and learning.

Research area of Neuroinformatics: Extract, analyse, and use principles of neural information processing.

Properties of Artificial Neural Networks (ANN)

- brain inspired model architectures
- allow for model selection through inductive learning (e.g. finding the best model parameters via learning from examples)

Architecture of ANNs

- simple elements
- massively parallel systems
- low precision (individual elements) & robustness (system)
- distributed representation of information
- no separation between data and program

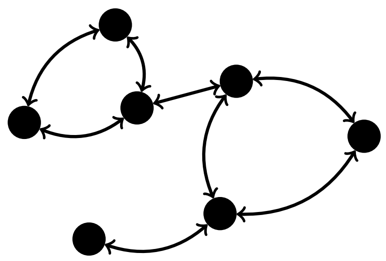


Figure 1: Graph structure with multiple connected computational units: rich dynamics and powerful pattern recognition capabilities can result from the interaction of many simple nonlinear units.

Inductive learning (Learning from observations)

- data driven, adaptive systems
- learning & self-organization vs. deduction & programming
- often seen as a plus: biologically inspired learning rules

Learning paradigms for ANNs

Given: series of observations: $\underline{\mathbf{x}}^{(1)}, \underline{\mathbf{x}}^{(2)}, \dots, \underline{\mathbf{x}}^{(p)}$:

(1) **Supervised Learning:** "learning with a teacher"

Additional Information: Additional training artefacts ("labels"): $y^{(1)}, y^{(2)}, \dots, y^{(p)}$. Training is based on pairs $\{(\underline{\mathbf{x}}^{(\alpha)}, y^{(\alpha)})\}_{\alpha=1,2,\dots,p}$

Goal: Predict correct output for new (previously unseen) examples.

Typical Problems: classification and regression

(2) **Unsupervised Learning:** "self-organization"

Additional Information: No additional information

Goal: Detect statistical regularities, find a new representation of $\underline{\mathbf{x}}$ useful for reasoning, decision making, prediction (e.g. efficient data storage)

Typical Problems: clustering, categorization, source separation

(3) **Reinforcement Learning:**

Additional Information: additional ratings $r^{(1)}, r^{(2)}, \dots, r^{(p)}$

Goal: Selection of the right action y for a given observation $\underline{\mathbf{x}}$

Typical Problems: learning association, strategy learning

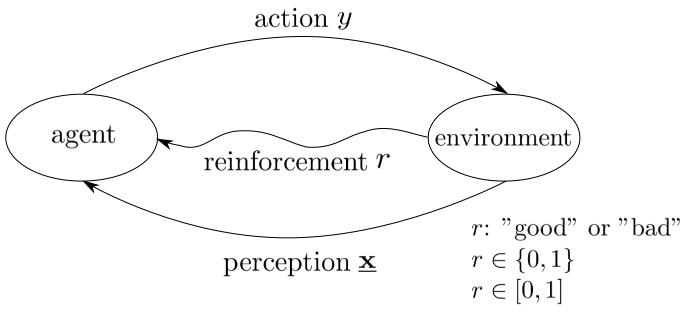


Figure 2: Schematic illustration of reinforcement learning

Comments:

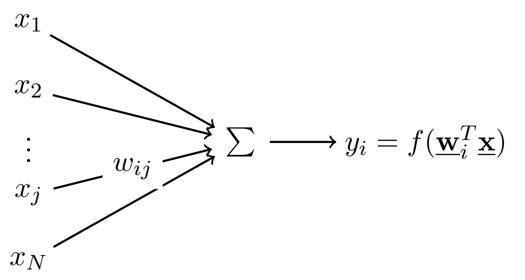
- (1)-(3) is a *phenomenological* characterization of learning paradigms
- <u>not</u> based on mathematical principles (e.g. same inductive learning approaches for "supervised" and "unsupervised" problems)

1.2 Connectionist Neurons

The components of an ANN are typically modeled as a simple input-output function. In principle one could also use more complex components.

1.2.1 Input-Output Relationship

Simple example: Connectionist neurons can be modeled as a *linear filter* (see below) with a static non-linearity, i.e. as a sequence of linear summation and a specific nonlinear function:


x_1
x_2
$\vdots$
w_{ij}
x_j
x_N
$\sum$
$y_i = f(\underline{\mathbf{w}}_i^T \underline{\mathbf{x}})$

$$\text{i.e. } y_i = f\Big(\underbrace{\sum_j \mathrm{w}_{ij}\mathrm{x}_j - \theta_i}_{h_i}\Big)$$

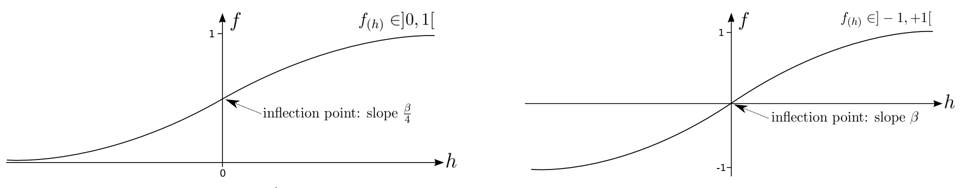
Figure 3: Input-output function for the connectionist neuron.(*cf.:* rate neurons, mean-field approximation, receptive field models ...).

Nomenclature:

$\underline{\mathbf{x}}$:	input vector with components x_j
y_i :	scalar output of neuron i
$\underline{\mathbf{w}}_i$:	weight vector of neuron i with components $\mathrm{w}\underbrace{ij}_{out\leftarrow in}$
θ_i :	threshold of neuron i
h_i :	total input of neuron i
f :	transfer function

Typical transfer functions: In ANN applications the *hyperbolic tangent* and the *logistic function* are commonly used. Depending on interpretation (e.g. as probabilities when used at output units), one or the other might seem more intuitive.

logistic function hyperbolic tangent



$f_{(h)} = \frac{1}{1+\exp(-\beta h)}$ $f_{(h)} = \tanh \beta h$

Figure 4: Typical transfer functions

Note: Both transfer functions are computationally equivalent

$$\begin{aligned} \frac{1}{1+e^{-x}} &= \frac{e^{\frac{x}{2}}}{e^{\frac{x}{2}}+e^{-\frac{x}{2}}} \\ &= \frac{1}{2}\left\{ \underbrace{\frac{e^{\frac{x}{2}}-e^{-\frac{x}{2}}}{e^{\frac{x}{2}}+e^{-\frac{x}{2}}}}_{\tanh \frac{x}{2}} + \underbrace{\frac{e^{\frac{x}{2}}+e^{-\frac{x}{2}}}{e^{\frac{x}{2}}+e^{-\frac{x}{2}}}}_{1} \right\} \qquad (1.1) \\ &= \frac{1}{2}\big(\tanh\tfrac{x}{2}+1\big) \end{aligned}$$

$\rightsquigarrow$ scale input weights w_{ij} or slope parameter β by 2 (reparametrization)

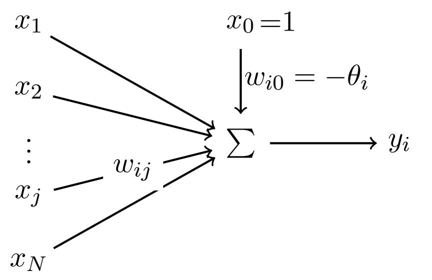
$$\left.\begin{aligned} &\rightsquigarrow \text{shift output by -1} \\ &\rightsquigarrow \text{scale output by 2} \end{aligned}\right\} \text{change in units only}$$

$\Rightarrow$ $\tanh x$

Shortcut notation for neurons with thresholds: The effect of a threshold θ_i can be accounted for by extending the input vector $\underline{\mathbf{x}}$ by a constant entry $x_0 = 1$ with weight w_{i0}:

Note: With slight abuse of notation, in the following

$$\underline{\mathbf{w}} \text{ will be used for } \begin{pmatrix} \mathrm{w}_1 \\ \vdots \\ \mathrm{w}_N \end{pmatrix} \text{ as well as for } \begin{pmatrix} \mathrm{w}_0 \\ \mathrm{w}_1 \\ \vdots \\ \mathrm{w}_N \end{pmatrix}$$



i.e. $y_i = f_{\left(\sum_j \mathrm{w}_{ij}\mathrm{x}_j\right)} = f_{(\underline{\mathbf{w}}_i^T \underline{\mathbf{x}})}$

Figure 5: Shortcut notation for neurons with threshold

$$\underline{\mathbf{x}} \text{ will be used for } \begin{pmatrix} \mathrm{x}_1 \\ \vdots \\ \mathrm{x}_N \end{pmatrix} \text{ as well as for } \begin{pmatrix} \mathrm{x}_0 \\ \mathrm{x}_1 \\ \vdots \\ \mathrm{x}_N \end{pmatrix}$$

1.2.2 Feature Detection and Evaluation

- The weights w_i can be interpreted as a “linear filter”
- Linear filters can be used for feature detection (cmp. “receptive field”)

Example: Filters for points and edges:

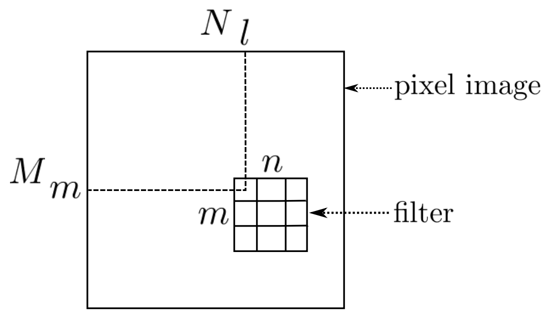


Figure 6: Illustration: point and edge filters

$$\underbrace{y_{kl}}_{\text{strength of feature}} = \sum_{i,j=0}^{m,n} \underbrace{\mathrm{w}_{ij}}_{\text{filter coefficients}} \mathrm{x}_{\underbrace{(k+i)(l+j)}_{\text{pixel value}}} \tag{1.2}$$

- $\underline{\mathbf{w}}$ describing input summation in a sensory neuron is often called its ”*receptive field*”

-1	-1	-1
-1	+8	-1
-1	-1	-1

point filter

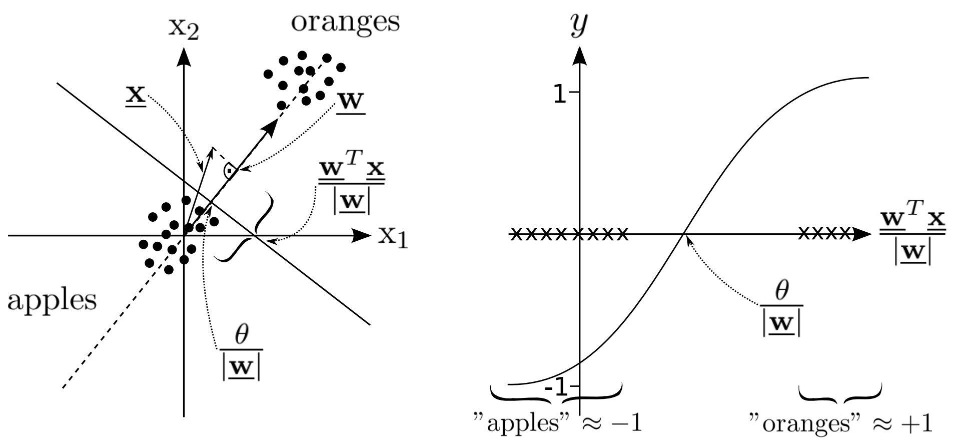
-1	0	+1
−2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	−2	-1

edge filter (Sobel filter)

Figure 7: Linear coefficients for image filters

Evaluation of filter output: Assume a sample of data (black points) corresponding to size (x_1) and weight (x_2) of apples and oranges (left plot, "data space"). Complex features are combinations $\underline{\mathbf{w}}$ of such elementary features and can be evaluated by projecting the datapoints onto the weight vector ($\rightarrow$ right plot, "feature space").



x_2
oranges
$\underline{\mathbf{x}}$
$\underline{\mathbf{w}}$
$\frac{\underline{\mathbf{w}}^T\underline{\mathbf{x}}}{|\underline{\mathbf{w}}|}$
x_1
apples
$\frac{\theta}{|\underline{\mathbf{w}}|}$
y
1
-1
$\frac{\underline{\mathbf{w}}^T\underline{\mathbf{x}}}{|\underline{\mathbf{w}}|}$
$\frac{\theta}{|\underline{\mathbf{w}}|}$
"apples" ≈ -1
"oranges" $\approx +1$

$$\underbrace{\underline{\mathbf{x}} \longrightarrow \underline{\mathbf{w}}^T}_{\text{feature detection}}\underbrace{\underline{\mathbf{x}} \longrightarrow f_{(\underline{\mathbf{w}}^T\underline{\mathbf{x}}-\theta)}}_{\text{evaluation}}$$

Figure 8: Feature detection and evaluation

$\Rightarrow$ computing the match between feature and stimulus (feature detection) and evaluating this match partitions the feature space into two half-spaces

1.2.3 Special Transfer Functions

Linear neuron

$$f_{(h)} = \beta h \tag{1.3}$$

$\Rightarrow$ extraction and detection of complex linear features

Binary neuron

$$f_{(h)} = \text{sign}(h) \tag{1.4}$$

$\Rightarrow$ feature extraction and classification $\rightarrow$ perception

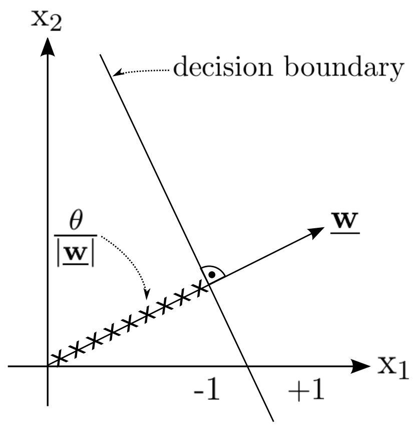


Figure 9: The binary input-output function $y = f_{(h)} \in \{-1, +1\}$ partitions the feature space into two half-spaces.

Stochastic binary neuron In addition to the previous 2 deterministic transfer functions, stochastic rules can be used to describe the response behavior of a connectionist neurons (see MacKay, 2003, ch. 39.1), e.g.

$$P_{(y \rightarrow -y)} = \frac{1}{1 + \exp(\beta y h)} \tag{1.5}$$

β: noise parameter

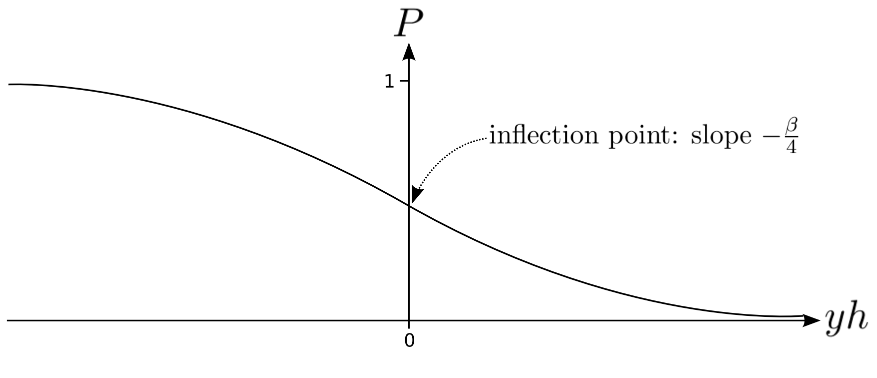


Figure 10: Stochastic binary neuron: The state of the stochastic binary neuron is not deterministic, i.e. for a given input, its state can vary from trial to trial.

1.3 Multilayer Perceptron

1.3.1 Classes of Neural Networks

The big variety of different ANNs can be classified according to their structure and the corresponding network graph

neural network	$\widehat{=}$	directed graph
(connectionist) neuron	$\widehat{=}$	node of the graph
connection between neurons	$\widehat{=}$	weighted edge

Recurrent networks $\widehat{=}$ directed graphs containing cycles

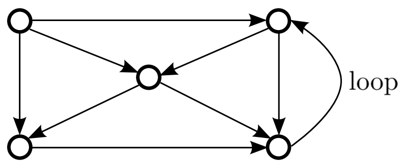


Figure 11: Recurrent neural networks are represented by graphs with cycles

Fields of application:

- models of dynamic systems
- spatio-temporal pattern analysis
- sequence processing
- associative memory and pattern completion

Examples: Hopfield networks, Boltzmann machines, infinite impulse response (IIR) networks, long short-term memory networks (LSTM, S. Hochreiter and Schmidhuber, 1997)

Feedforward networks $\widehat{=}$ directed acyclic graphs

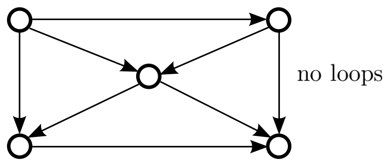


Figure 12: Feedforward neural networks are represented by graphs without cycles

Fields of application:

- association between variables
- prediction of attributes

Examples: Multilayer-Perceptron (MLP), Radial Basis Function (RBF) network, Support Vector Machines (SVM)

Prediction of attributes: Many computational tasks involve the prediction of attributes. Depending on the type of predicted attribute, one distinguishes between *regression* and *classification*:

Real-valued attributes: → *regression* problems

$$\begin{array}{lll} f : \mathbb{R}^N \to \mathbb{R} & \text{(or subsets)} & \text{one target value} \\ f : \mathbb{R}^N \to \mathbb{R}^M & \text{(or subsets)} & \text{multivariate attributes} \end{array} \tag{1.6}$$

Ordinal attributes: → *classification* problems
let S be the set of attributes $\{a_1, a_2, \dots, a_M\}$

$$f : \mathbb{R}^N \to S$$

special case: two class problems, e.g. $f : \mathbb{R}^N \to \{-1, +1\}$

→ predicting probabilities

These approaches can be generalized for mappings from structured input to structured output

1.3.2 The Multilayer-Perceptron for Regression

Simplifications used in this lecture:

- $\underline{\mathbf{x}} \in \mathbb{R}^N, y_i \in \mathbb{R}$, i.e. scalar output, layered architecture
- connections between subsequent layers only (except for bias node)
- similar transfer functions for all neurons

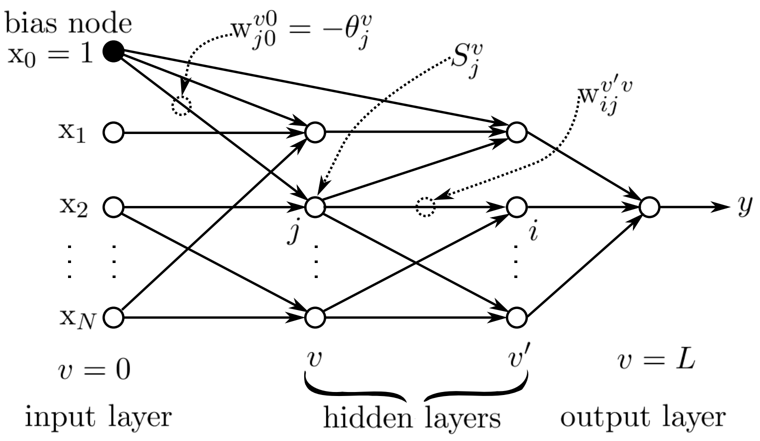


Figure 13: Architecture of the MultiLayer Perceptron

Nomenclature:

S_i^v	activity of neuron $\overbrace{(v,i)}^{\text{layer, unit}} := f(h_i^v) = f(\sum_j w_{ij}^{v,v-1} S_j^{v-1} - \theta_i^v)$
$S_0^0 = x_0 = 1$	activity of bias neuron
$S_i^0 = x_i$	input to MLP, i^{th} component
$S_i^L = y_i$	output of MLP ($i = 1$ in figure 13)
$w_{ij}^{v'v}$	connection weight between neurons (v, j) and (v', i)
$w_{j0}^{v0} = \theta_j^v$	connection weight between bias node and neuron (v, j)
$h_i^v = \sum_j w_{ij}^{vv-1} S_j^{v-1}$	total input of neuron (v, i)
f	transfer function (sigmoid)

MLP parameters

$$\left.\begin{array}{l}\rightarrow \text{ number of layers} \\ \rightarrow \text{ number of neurons per layer}\end{array}\right\} \text{"architecture"}$$

$\rightarrow$ weights (including thresholds)} "model parameters"

$\Rightarrow$ **both** sets have to be chosen during *model selection*!

Visualization of weights and thresholds **Note:** Non-linear transfer functions are essential – a 3 layered network of units with linear transfer

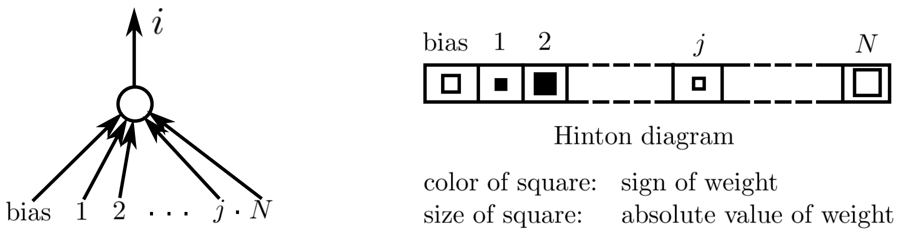

i
bias 1 2 · · · j · N
bias 1 2 j N
Hinton diagram
color of square: sign of weight
size of square: absolute value of weight

Figure 14: Visualisation of weights and thresholds

functions is equivalent to a single connectionist neuron:

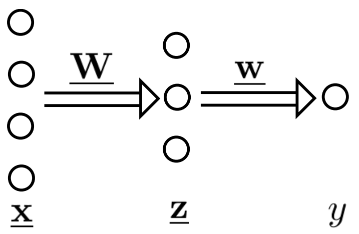


Figure 15: Importance of nonlinear transfer functions

$$y = \underline{\mathbf{w}}^T \underline{\mathbf{z}} = \underline{\mathbf{w}}^T \underline{\mathbf{W}} \underline{\mathbf{x}} = \underline{\widehat{\mathbf{W}}} \underline{\mathbf{x}} \quad \widehat{=} \quad \text{connectionist neuron}$$

MLPs are universal function approximators: Even with simple non-linear functions, MLPs provide a model class with powerful computational capabilities: For details, see e.g. Funahashi (1989) and Hornik, Stinchcombe, and White (1989).

Theorem: Let $y^*_{(\underline{\mathbf{x}})}$ be a continuous, real valued function over a compact interval K. Let

$$\hat{y}_{(\underline{\mathbf{x}})} = \sum_{i=1}^{M} \mathrm{w}_i^{21} f_{\left(\sum_{j=1}^{N} \mathrm{w}_{ij}^{10} \mathrm{x}_j - \theta_i \right)}$$

be a three-layered MLP with a non-constant, bounded, monotonously increasing and continuous function $f : \mathbb{R} \to \mathbb{R}$.

Then there exists a set of parameters $M, N \in \mathbb{N}$ and $\mathrm{w}_i^{21}, \mathrm{w}_{ij}^{10}, \theta_i \in \mathbb{R}$ such that for every $\varepsilon > 0$:

$$\max_{\underline{\mathbf{x}} \in K} \left| \hat{y}_{(\underline{\mathbf{x}})} - y^*_{(\underline{\mathbf{x}})} \right| \leq \varepsilon$$

1.3.3 Performance Measures and Model Selection

Remark: For both regression and classification problems, the goal is to find a model that predicts the observed outputs (and potential future observations) as good as possible. This goodness of fit can be quantified via *error* or *cost functions*. Below, we describe error functions for *regression* problems, for classification problems, see 1.4.7.

Prediction of attributes

$$\underbrace{\underline{\mathbf{x}} \in \overbrace{\mathbb{R}^N}^{\substack{\text{data}\\\text{point}}}}_{\substack{\text{feature}\\\text{vector}}} \longrightarrow \underbrace{y \in \overbrace{\mathbb{R}}^{\text{label}}}_{\text{attribute}}$$

y_T :	true value of attribute
$y_{(\underline{\mathbf{x}})}$:	predicted value of attribute (e.g. by MLP)

Error measure (individual cost / individual loss): The error measure quantifies the cost of a wrong prediction (e.g. loss in $$, ...) and determines the "goodness" of a solution.

Example error functions

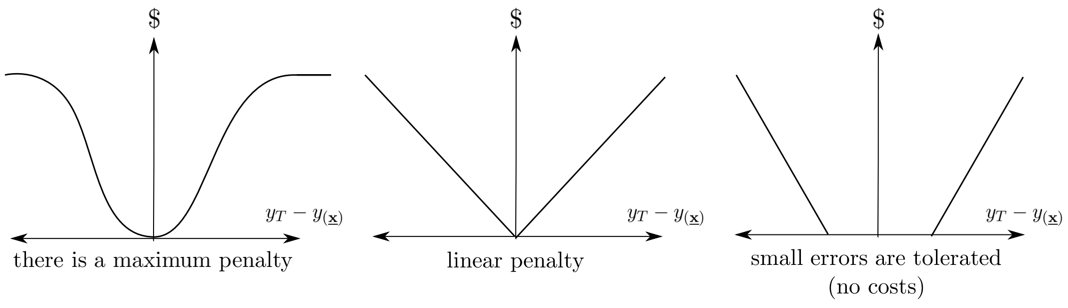


Figure 16: Example error functions: choice depends on task setting & assumptions regarding noise.

- several choices possible ⇒ one must be made
- predictor will depend on error measure!

Most common error measure:

$$e_{(y_T,\underline{\mathbf{x}})} = \frac{1}{2}\big(y_T - y_{(\underline{\mathbf{x}})}\big)^2 \qquad \text{(quadratic error)}$$

Performance measure: How many $$ per prediction do I have to spend on average?

$\Rightarrow$ mathematical expectation

$$E^G = \underbrace{< e >_{y_T, \underline{\mathbf{x}}}}_{\substack{\text{mathematical}\\ \text{expectation}\\ \text{w.r.t } y_T \text{ and } \underline{\mathbf{x}}}} = \int d\underline{\mathbf{x}} dy_T P_{(y_T, \underline{\mathbf{x}})} e_{(y_T, \underline{\mathbf{x}})} \qquad \text{(generalization error)}$$

$P_{(y_T, \underline{\mathbf{x}})}$: joint probability density function (pdf) of observations

$$\left.\begin{array}{ll} \text{good predictor:} & \text{low value for } E^G \\ \text{bad predictor:} & \text{high value for } E^G \end{array}\right\} E^G \overset{!}{=} \min$$

but:

- $P_{(y_T, \underline{\mathbf{x}})}$ is - in general - not known.
- If we knew it, there would be no need for learning!

$\rightarrow$ E^G cannot be minimized directly.

Inductive learning through empirical risk minimization (ERM)

General idea

mathematical expectation $\rightarrow$ empirical average over a "training set"

training set of observations: $\left\{ \left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)} \right) \right\}, \alpha = 1, \ldots, p$

$$E^T = \frac{1}{p} \sum_{\alpha=1}^{p} e^{(\alpha)} \qquad \text{(training error, empirical risk } E^T\text{)}$$

Model selection: Find model (parameters) such that: $E^T \overset{!}{=} \min$

Consequences

- *Validation:* Is the selected model indeed a good predictor?
- *Mathematical analysis:* When does "$E^T \overset{!}{=} \min$" imply "E^G is small (enough)"? $\Rightarrow$ statistical learning theory, ch. 2.

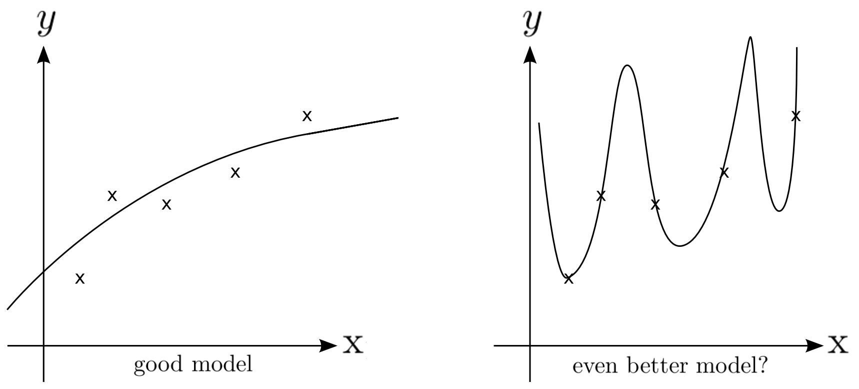


Figure 17: model selection and overfitting

1.3.4 Optimization of Model Parameters: Gradient Descent

architecture	vs.	model parameters
(number of layers and nodes)		(weights, thresholds)

Learning can affect both aspects; here we focus on learning the model parameters.

training set: $\left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha = 1, \ldots, p$

training error E^T for the given training set: $E^T_{[\underline{\mathbf{w}}]} = \frac{1}{p} \sum_{\alpha=1}^{p} e^{(\alpha)}_{[\underline{\mathbf{w}}]}$

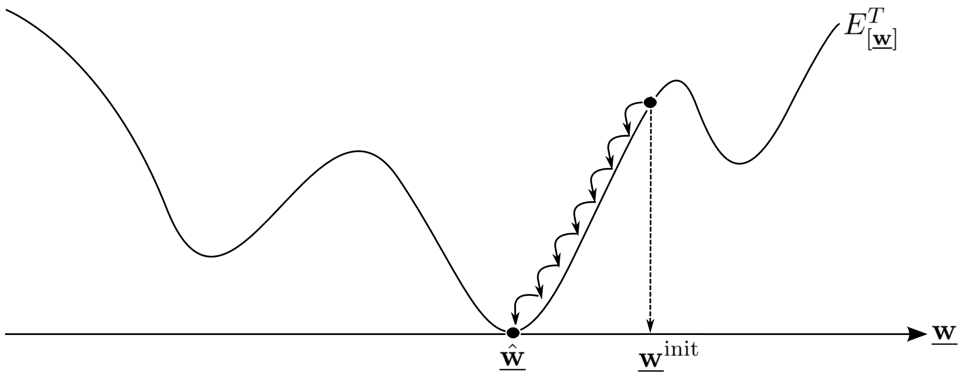

$E^T_{[\underline{\mathbf{w}}]}$
$\underline{\mathbf{w}}$
$\hat{\underline{\mathbf{w}}}$
$\underline{\mathbf{w}}^{\text{init}}$

Figure 18: illustration of gradient descent

Gradient Descent

$$\Delta \mathrm{w}_{ij}^{v'v} = \underbrace{-\hat{\eta}}_{\substack{\text{learning} \\ \text{step}}} \underbrace{\frac{\partial E_{[\underline{\mathbf{w}}]}^{T}}{\partial \mathrm{w}_{ij}^{v'v}}}_{\substack{\text{gradient vector:} \\ \text{direction of} \\ \text{steepest ascent}}} \tag{1.7}$$

$$= \underbrace{-\eta}_{\overset{!}{=} \frac{\hat{\eta}}{p}} \sum_{\alpha=1}^{p} \frac{\partial e_{[\underline{\mathbf{w}}]}^{(\alpha)}}{\partial \mathrm{w}_{ij}^{v'v}}$$

Calculation of the gradient:

$$\frac{\partial e_{[\underline{\mathbf{w}}]}^{(\alpha)}}{\partial \mathrm{w}_{ij}^{v'v}} = \underbrace{\frac{\partial e_{[\underline{\mathbf{w}}]}^{(\alpha)}}{\partial y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}}}_{\substack{\text{factor depending} \\ \text{on cost function}}} \cdot \underbrace{\frac{\partial y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}}{\partial \mathrm{w}_{ij}^{v'v}}}_{\substack{\text{factor depending on} \\ \text{model class (e.g. MLP)}}} \tag{1.8}$$

for the quadratic error we obtain:

$$\frac{\partial e_{[\underline{\mathbf{w}}]}^{(\alpha)}}{\partial y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}} = \frac{\partial}{\partial y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}} \frac{1}{2} \left(y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})} - y_T \right)^2 = y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})} - y_T \tag{1.9}$$

Problems of the gradient descent method

- convergence to local minima
  (problem of all gradient-based, local optimization methods)
- choice of η may be critical (slow convergence vs. oscillations)

1.3.5 The Backpropagation Method

Backpropagation of errors is a computationally efficient method for calculating the derivatives

$$\frac{\partial y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}}{\partial \mathrm{w}_{ij}^{v'v}} \tag{1.10}$$

required to determine the parameters of Multilayer-Perceptrons (MLPs) via gradient descent (see eq. 1.8).

$\Rightarrow$ can be extended to other feedforward networks

$\Rightarrow$ efficient way to do message passing in directed acyclic graphs

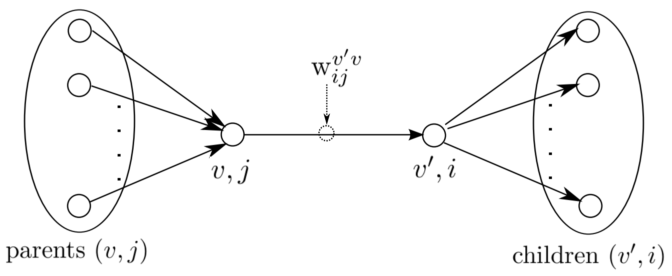


Figure 19: schema backpropagation

Smart application of the chain rule exploiting $h_i = \underline{\mathbf{w}}_i^T \underline{\mathbf{s}} - \theta_i$:

$$\frac{\partial y}{\partial \mathrm{w}_{ij}^{v'v}} = \underbrace{\frac{\partial y}{\partial h_i^{v'}}}_{\substack{:=\delta_i^{v'} \\ \text{"local error"} \\ \text{at neuron} \\ (v',i)}} \cdot \underbrace{\frac{\partial h_i^{v'}}{\partial \mathrm{w}_{ij}^{v'v}}}_{\substack{=S_j^v \\ \text{activity} \\ \text{of neuron} \\ (v,j)}} \tag{1.11}$$

Note: this definition $\delta_i^{v'} := \partial y / \partial h_i^{v'}$ of a units local effect on y is slightly different from the definition $\delta_i^{v'} := \partial e / \partial h_i^{v'}$ of "local error" given in e.g. Bishop (2006).

Forward propagation step: calculation of *activities*

$$S_j^0 = \mathrm{x}_j^{(\alpha)} \qquad \text{(initialization: inputs)}$$

$$S_j^v = f\Big(\sum_{(\gamma,h)\in \text{ parents } (v,j)} \mathrm{w}_{jh}^{v\gamma} S_h^\gamma \Big) \qquad \text{(propagation)}$$

Backpropagation step: calculation of *"local errors"*

$$\delta_1^v = f'_{(h_1^v)} \qquad \text{(initialization: outputs)}$$

$$\delta_i^{v'} = \sum_{(\beta,l)\in \text{ children } (v',i)} \frac{\partial y}{\partial h_l^\beta} \cdot \frac{\partial h_l^\beta}{\partial h_i^{v'}} \tag{1.12}$$

$$= \sum_{(\beta,l)\in \text{ children } (v',i)} \delta_l^\beta \mathrm{w}_{li}^{\beta v'} f'_{(h_i^{v'})} \tag{1.13}$$

$$= f'_{(h_i^{v'})} \sum_{(\beta,l)\in \text{ children } (v',i)} \delta_l^\beta \mathrm{w}_{li}^{\beta v'} \qquad \text{(propagation)}$$

Computational complexity: $\mathcal{O}(n)$, n: number of weights & thresholds

1.3.6 Summary of the Gradient Descent Method

Initialization: random numbers, such that h_i^v approx. $\mathcal{O}(1)$???

- numbers too large: transfer function saturates
  $\rightsquigarrow$ gradients become too small
- numbers too small: neurons operate in the linear regime of f
  $\rightsquigarrow$ MLP becomes equivalent to one connectionist neuron

Stopping criterion

- fixed number of iterations
- fixed CPU-time
- E^T falls below a predefined value
- $\frac{\Delta E^T}{E^T}$ falls below a predefined value
- validation criterion fulfilled

1.3.7 Validation of Model Selection Results

Assessment of prediction quality $\Rightarrow$ estimation of E^G

Test Set Method:

$$\text{observations} \begin{cases} \text{training data } \left\{ \left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)} \right) \right\}, \alpha = 1, \ldots, p \\ \rightarrow \text{for selection of model parameters} \\ \\ \text{test data } \left\{ \left(\underline{\mathbf{x}}^{(\beta)}, y_T^{(\beta)} \right) \right\}, \beta = 1, \ldots, q \\ \rightarrow \text{for estimation of the generalization error} \end{cases}$$

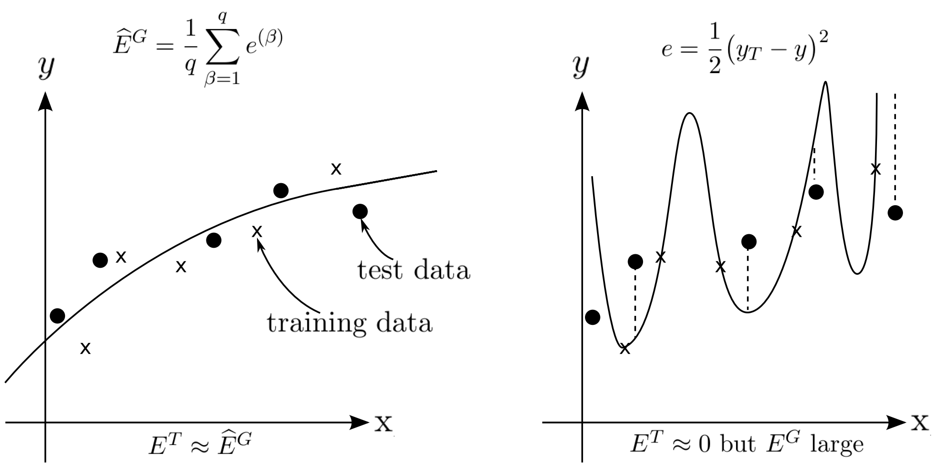
Notes:

- Test data must not be used for selection of model parameters.
  Therefore, the method is problematic if only few data available
- Estimate for the generalization performance of a *specific* model.

Resampling methods: N-Fold Cross-Validation

(1) The set of all observations D is partitioned into n disjunct subsets D_j such that $\bigcup\limits_{j=1}^{n} D_j = D$

(2) On each of the training datasets $D \,/\, D_j$, train a network N_j and compute average prediction error $\hat{E}_i^G$ on the test set D_j.



$\widehat{E}^G = \frac{1}{q}\sum_{\beta=1}^{q} e^{(\beta)}$
y
x
test data
training data
$E^T \approx \widehat{E}^G$
$e = \frac{1}{2}(y_T - y)^2$
y
x
$E^T \approx 0$ but E^G large

Figure 20: symptoms of overfitting: training vs. generalization error

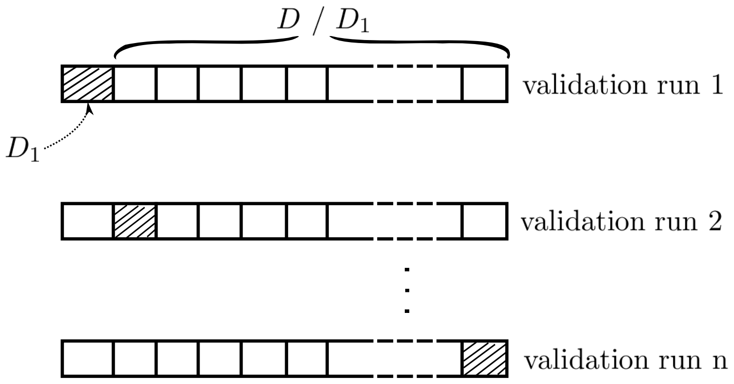


Figure 21: Crossvalidation

Estimation of E^G:

$$\widehat{E}^G = \frac{1}{N}\sum_{i=1}^{N}\hat{E}_i^G = \frac{1}{p}\sum_j \sum_{\alpha\in D_j} e^{(\alpha)} \tag{1.14}$$

- Typical choice: n $\approx$ 5 ... 10
- "leave-one-out cross-validation": $n = p$

Variance of the cross-validation estimator

$$\operatorname{var}\left(\widehat{E}^G\right) = \frac{n-1}{n}\sum_{j=1}^{n}\Big(\underbrace{\widehat{E}_j^G}_{\substack{\text{test error}\\ \text{on } D_j}} - \widehat{E}^G\Big)^2 \tag{1.15}$$

Notes:

- n-fold cross-validation is only used for estimating E^G
- the resampling procedure yields n different sets of parameters
- for selecting the final model parameters all data are used!
- estimates average prediction performance of the *combination* (architecture + training method) ($\leftrightarrow$ test-set-method)

1.4 Additional Topics

1.4.1 Stochastic Approximation and On-line Learning

Gradient descent (MLP example):

$$\begin{aligned}\Delta \mathrm{w}_{ij}^{v'v} &= -p\eta \frac{\partial E^T_{[\underline{\mathbf{w}}]}}{\partial \mathrm{w}_{ij}^{v'v}} \\ &= -\eta \sum_{\alpha=1}^{p} \frac{\partial e^{(\alpha)}_{[\underline{\mathbf{w}}]}}{\partial \mathrm{w}_{ij}^{v'v}}\end{aligned} \tag{1.16}$$

Batch learning: All observations are used for every weight update.

On-line learning: Sequential processing of observations

- one weight update per data point (”learning while doing”)

$\rightsquigarrow$ learning and adaption in *non-stationary* environments

Online learning for MLPs can easily be implemented:

Algorithm 1: On-line learning of MLP-weights

```
$t \leftarrow 1$
begin
    $\eta_t \leftarrow \frac{\eta_0}{t}$        (learning schedule, depends on learning goal)
    select next data point: $\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)$
    change weights: $\left(\Delta \mathrm{w}_{ij}^{v'v}\right)^{t+1} = -\eta_t \frac{\partial e^{(\alpha)}_{[\underline{\mathbf{w}}^{(t)}]}}{\partial \mathrm{w}_{ij}^{v'v}}$
    $t \leftarrow t + 1$
end
```

In practice, online learning has proven to be *robust*:

- convergence to (good) minima of E^T
- less affected by the ”local optimum” problem (stochastic update)

But: no general proof of convergence yet

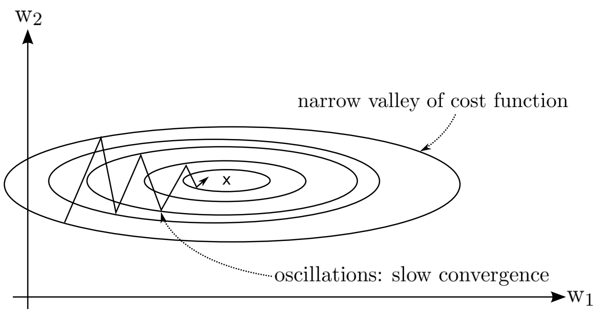
(1) Theorem by Robbins & Monro (1951): ”stochastic approximation”
Holds only for convex optimization problems.[1]

(2) Theorem by Bottou (1998): Almost sure convergence only to extrema of E^T (not necessarily minima)
$\Rightarrow$ conditions not necessarily fulfilled for MLPs with sigmoid transfer functions.[2]

[1]For theorem statement, see lecture slides, for proof, see supplementary material
[2]For theorem statement, see lecture slides, for proof, see supplementary material

1.4.2 Improved Gradient Descent Optimization

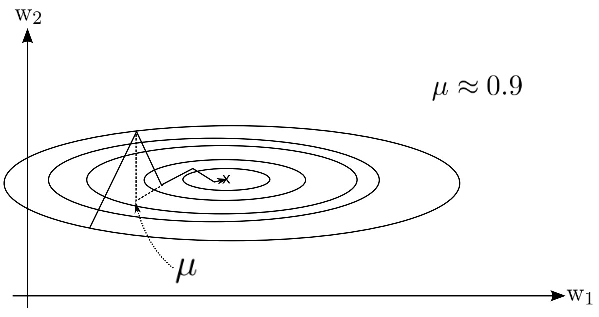
(a) **Impulse terms / Momentum**



w2
narrow valley of cost function
oscillations: slow convergence
w1

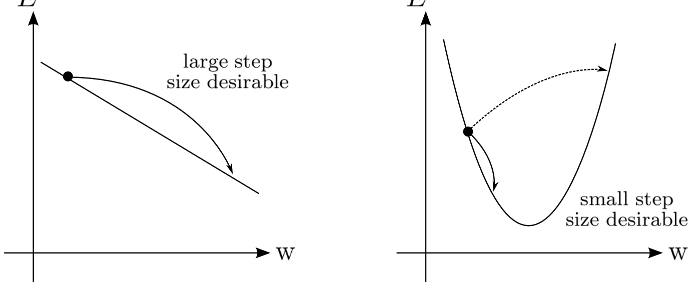
Oscillations can be reduced by introducting "momentum" or impulse terms into the weight-update

$$\Delta\underline{\mathbf{w}}_{t+1} = -\hat{\eta}\frac{\partial E^T}{\partial\underline{\mathbf{w}}}\bigg|_{\underline{\mathbf{w}}_t} + \underbrace{\mu\Delta\underline{\mathbf{w}}_t}_{\substack{\text{impulse}\\ \text{term}}} \tag{1.17}$$



Impulse terms can be interpreted as smoothing the weight updates with an exponentially weighted running average.

(b) **Adaptive step size**



$$\eta_{t+1} = \begin{cases} \rho\,\eta_t, & \text{if } \Delta E^T < 0, \quad \text{increase step size, if } E^T \downarrow \\ \delta\,\eta_t, & \text{if } \Delta E^T > 0, \quad \text{decrease step size, if } E^T \uparrow \end{cases}$$

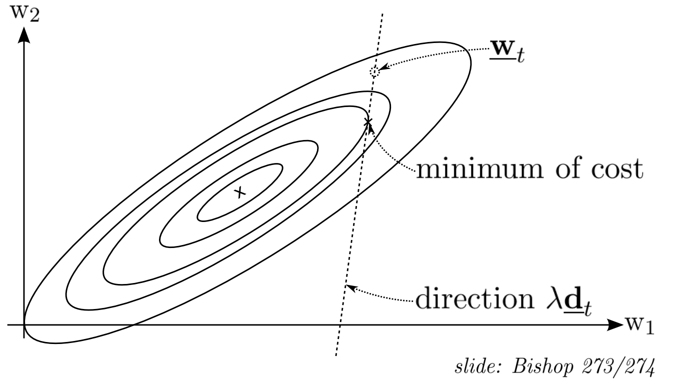
typical values: $\rho = 1.1, \delta = 0.5$

1.4.3 The Conjugate Gradient Method

$$\text{local optimization} \begin{cases} \text{choice of direction} \Rightarrow \text{conjugate direction} \\ \text{choice of step size} \Rightarrow \text{line search method} \end{cases}$$

for details & implementation see Numerical Recipes, 2nd edition chapter 10.2 & 10.6

Line search method
Search for the minimum of the cost *along a given direction* in parameter space. There are different alternatives to find this minimum, one of them is



w2
$\underline{\mathbf{w}}_t$
minimum of cost
direction $\lambda\underline{\mathbf{d}}_t$
w1
slide: Bishop 273/274

Figure 22: linesearch

parabolic interpolation:

The conjugate direction
Problem with standard gradient direction: Minimum $\underline{\mathbf{w}}_t$ after line search: new gradient $\perp$ old direction $\rightsquigarrow$ oscillations. ☐Oscillations

Definition of conjugate direction $\underline{\mathbf{d}}_{t+1}$:
The component of the gradient parallel to the old direction ($\underline{\mathbf{d}}_t$) should remain zero along the new direction ($\underline{\mathbf{d}}_{t+1}$). It can be computed as: ☐Conjugate Direction

$$\underline{\mathbf{g}}_{t+1} := \left.\frac{\partial E^T}{\partial \underline{\mathbf{w}}}\right|_{\underline{\mathbf{w}}_{t+1}} \qquad \text{(gradient)}$$

$$\underline{\mathbf{d}}_{t+1} = -\underline{\mathbf{g}}_{t+1} + \beta_t \underline{\mathbf{d}}_t \qquad (1.18)$$

Algorithm 2: parabolic interpolation. Note: the last step effectively selects the best 3 points for the next iteration. Problem: works only if close enough to local minimum.

```
Initialization: $a_0, b_0, c_0 (\text{on } \lambda \underline{\mathbf{d}}_t); E^T_{(a_0)}, E^T_{(b_0)} > E^T_{(c_0)}$
begin
    Fit a parabola through the three points $a_t, b_t, c_t$
    Calculate location $d_t$ of its minimum
    if stopping criterion fulfilled then STOP
    Set $c_{t+1} = d_t, b_{t+1} = c_t, a_{t+1} = \begin{cases} a_t, & E^T_{(a_t)} < E^T_{(b_t)} \\ b_t, & \text{else} \end{cases}$
end
```

$$\beta_t = \underbrace{\frac{\underline{\mathbf{g}}^T_{t+1}\left(\underline{\mathbf{g}}_{t+1} - \underline{\mathbf{g}}_t\right)}{\underline{\mathbf{g}}^T_t \underline{\mathbf{g}}_t}}_{\text{Pollach-Ribiere rule}} \tag{1.19}$$

The conjugate gradient method therefore contains the following steps

Algorithm 3: Conjugate gradient method

```
Initialization: $\underline{\mathbf{w}}, \underline{\mathbf{d}} = -\underline{\mathbf{g}}$
begin
    Minimize $E^T$ along $\underline{\mathbf{d}}$ using line-search $\rightarrow$ new $\underline{\mathbf{w}}$
    if stopping criterion fulfilled then STOP
    Calculate the new conjugate direction $\rightarrow$ new $\underline{\mathbf{d}}$
end
```

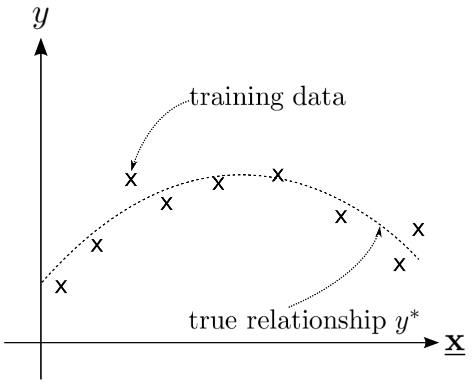
Remark: Conjugate Gradient realizes efficient gradient descent with adaptive step size and impulse term

- step size: calculated via line search
- impulse parameter: given by β_t

$\Rightarrow$ preferred gradient-based method for unconstrained optimization

1.4.4 Overfitting and Underfitting

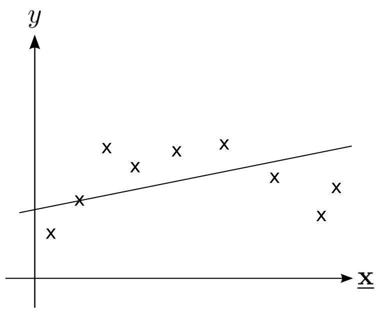
Typical task: Given a limited set of data (x_T, y_T), find a model to describe the relation between x and y and make predictions $y^*(x_N)$ for new data x_N.



$$y_T = y^*_{(\underline{\mathbf{x}})} + \underbrace{\eta}_{\text{noise}}$$

Three cases:

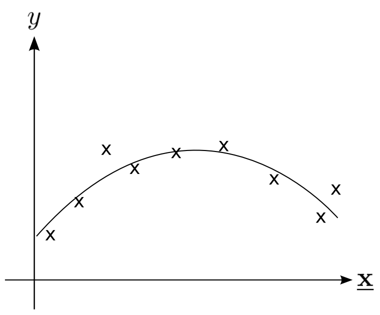
I. **Model is too simple** (e.g. MLP is too small)
$\rightsquigarrow y^*$ cannot be approximated well



$$\left.\begin{matrix} E^T \text{ large} \\ E^G \text{ large} \end{matrix}\right\} E^T \text{ approx. } E^G$$

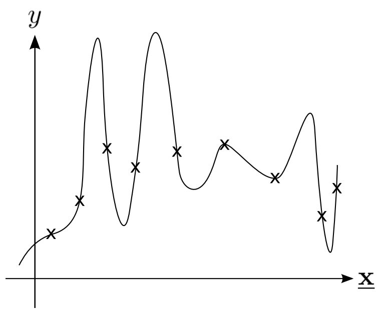
"underfitting"

II. **Model complexity matches** (e.g. MLP has the right size)



$$\left.\begin{matrix} E^T \text{ small} \\ E^G \text{ small} \end{matrix}\right\} E^T \text{ approx. } E^G$$

III. **Model is overly complex** (e.g. MLP is too large)



$$\left.\begin{array}{l} E^T \text{ small} \\ E^G \text{ large} \end{array}\right\} E^T \ll E^G$$

"overfitting"

⇒ often happens if E^T does not vary strongly over a large volume of parameter space

⇒ using a different sample of observations (from the same underlying distribution) may lead to a large shift of the minimum of E^T in parameter space. □Overfitting

1.4.5 Bias and Variance

Complexity of a model class influences how well it can describe different data sets but also how much parameter estimates are affected by random fluctuations in the training data. The bias and variance of an estimator quantify these two statistical aspects and together determine its generalization performance.[3]

Example scenario: Data $y_T = \underbrace{y^*_{(\underline{\mathbf{x}})}}_{\substack{\text{true} \\ \text{relationship}}} + \underbrace{\eta}_{\text{noise}}, \qquad \underline{\mathbf{x}} \in \mathbb{R}^N, y_T \in \mathbb{R}$

η :	zero mean additive noise
$P_{(\underline{\mathbf{x}})}$:	pdf of data points
$P_{(y\vert\underline{\mathbf{x}})}$:	conditional pdf of attributes

Assessing generalization performance:

1. sample many iid. datasets of equal length from $P_{(y_T,\underline{\mathbf{x}})} = P_{(y_T|\underline{\mathbf{x}})} P_{(\underline{\mathbf{x}})}$
2. fit one MLP to every dataset using squared error as individual cost.

Remark: $\underline{\mathbf{x}}, y_T$ are random variables

[3]The total generalization error can always be separated into the three terms squared bias, variance, and the error variance.

$\rightarrow$ $\underline{\mathbf{w}}$ (model parameters) are random variables

$\rightarrow$ $y_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}$ (predicted values) are random variables

Generalization performance is based on the mathematical expectation

$$\Big\langle \Big(\underbrace{y_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}}_{y} - \underbrace{y^*_{(\underline{\mathbf{x}})}}_{y^*} \Big)^2 \Big\rangle_{\text{all datasets}} \qquad \text{("ensemble average")}$$

With

$$\begin{aligned} (y-y^*)^2 &= (y- <y> + <y> -y^*)^2 \\ &= (y- <y>)^2 + (<y> -y^*)^2 + 2\underbrace{(y- <y>)}_{\substack{=0 \\ \text{when averaged}}}(<y> -y^*) \end{aligned} \tag{1.20}$$

one obtains the following decomposition of the generalization error into bias and variance:

$$\big\langle (y-y^*)^2 \big\rangle = \underbrace{(<y> -y^*)^2}_{\text{bias}^2} + \underbrace{\big\langle (y- <y>)^2 \big\rangle}_{\text{variance}} \tag{1.21}$$

- useful for scenarios, where a deviation measure makes sense
- not useful however for classification problems (other ansatz needed)

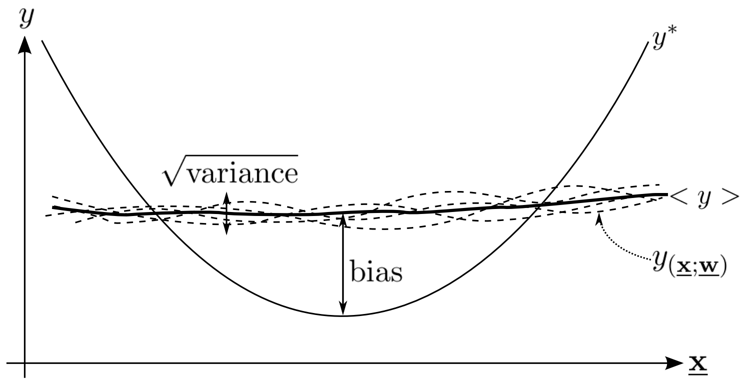


Figure 23: Illustration of the bias-variance decomposition: different datasets result in different estimates ($y = y_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}$, dashed lines). The bias is given by the mean squared deviation between their average ($\langle y \rangle$, thick line) and the true values (y^*, thin line). The variance quantifies the variability of the individual sample-based estimates y around their mean $\langle y \rangle$.

Bias-variance trade-off: While *variance* describes the amount by which our estimate is expected to vary if we compute it using a different training dataset, its *bias* describes the systematic error due to our approximation with a (too) simple (e.g. linear) model. As a result, more flexible (complex) statistical methods typically have lower bias but higher variance. This bias-variance trade-off applies to many inductive learning problems.

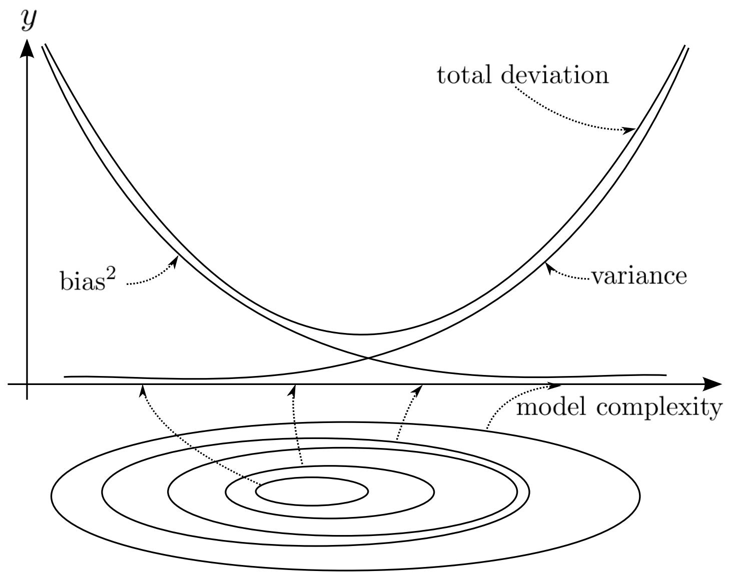


Figure 24: Illustration of the bias-variance trade-off. Underfitting typically occurs when bias dominates the total deviation (too simple models), overfitting when variance dominates (too complex models).

Examples for model classes of different complexity

→ MLP with increasing number of layers/units

→ polynomials of increasing degree ☐Duda&Hart p.467

Relation to generalization performance (for above example scenario)

$$E^G = \langle e^G \rangle_{P_{(\underline{\mathbf{x}})}}, \text{ where } e^G = \left\langle (y_{(\underline{\mathbf{x}};\underline{\mathbf{w}})} - y_T)^2 \right\rangle_{P_{(y_T|\underline{\mathbf{x}})}} \tag{1.22}$$

$$\begin{aligned} e^G &= \left\langle (y - y_T)^2 \right\rangle \\ &= \left\langle (y - y^* + y^* - y_T)^2 \right\rangle \\ &= (y - y^*)^2 + \left\langle (y^* - y_T)^2 \right\rangle + 2(y - y^*) \underbrace{\left\langle (y^* - y_T) \right\rangle}_{=0} \\ &= (y - y^*)^2 + \sigma_\eta^2 \end{aligned} \tag{1.23}$$

$$\langle e^G \rangle_{\text{ensemble}} = \langle \underbrace{(y - y^*)^2}_{\text{bias}^2+\text{variance}} \rangle_{\text{ensemble}} + \sigma_\eta^2 \tag{1.24}$$

- for classification example, see *cf. slide: Duda & Hart 470*
- relationship bias-variance usually not additive and can be non-linear

1.4.6 Regularization

Complex model but few observations → danger of overfitting

→ include additional (prior) knowledge about the problem

→ bias the selection procedure towards a certain model class

$$R_{[\underline{\mathbf{w}}]} = \underbrace{E^T_{[\underline{\mathbf{w}}]}}_{\substack{\text{training}\\ \text{error}}} + \underbrace{\lambda E^R_{[\underline{\mathbf{w}}]}}_{\substack{\text{regularization}\\ \text{term}}} \overset{!}{=} \min \qquad \text{(risk function)}$$

E^R :	penalizes certain models ⇝ "soft" restrictions on model space
λ :	regularization parameter; trade-off between observations and prior knowledge

Example I: inclusion unspecific knowledge: weight decay

$$E^R_{[\underline{\mathbf{w}}]} = \frac{1}{2} \sum_{i,j,v',v} \left(\mathrm{w}_{ij}^{vv'}\right)^2 \tag{1.25}$$

→ penalty for models with many weights of large absolute value

→ implicit smoothness assumption (e.g. MLPs smaller weights ∼ smoother input-output functions)

Minimization of the risk R through gradient descent:

$$\Delta \mathrm{w}_{ij}^{vv'} \sim -\frac{\partial R}{\partial \mathrm{w}_{ij}^{vv'}} = -\left(\frac{\partial E^T}{\partial \mathrm{w}_{ij}^{vv'}} + \frac{\partial E^R}{\partial \mathrm{w}_{ij}^{vv'}}\right) = - \underbrace{\frac{\partial E^T}{\partial \mathrm{w}_{ij}^{vv'}}}_{\substack{\text{e.g. via}\\ \text{backprop}}} - \underbrace{\lambda \mathrm{w}_{ij}^{vv'}}_{\substack{\text{decay}\\ \text{term}}} \tag{1.26}$$

⇝ Parameters which are not supported by data (can be estimated less reliably) decay to zero.

⇝ Different components of the weight vector can be estimated more or less reliably from the data. They are affected differently by weight decay and therefore differ more or less from the unregularized estimate $\underline{\mathbf{w}}^*$ minimizing training Error E^T

w_1:	"badly" supported parameter	→ shrinks more
w_2:	"well" supported parameter	→ shrinks less

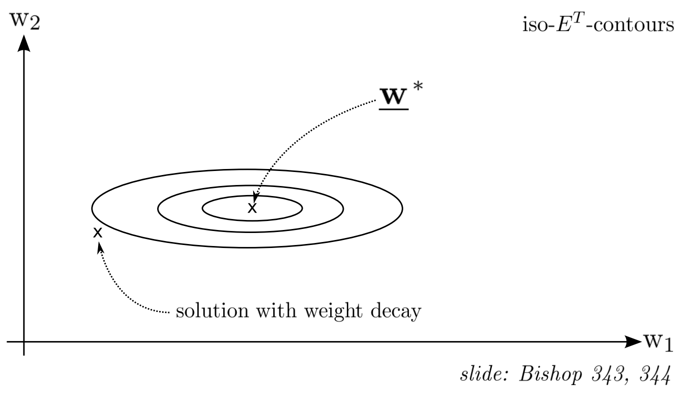


Figure 25: weight decay

Example II: inclusion of specific knowledge: symmetries

(a) Odd vs. even function

$$E^R_{[\mathrm{w}]} = \frac{1}{2p} \sum_{\alpha=1}^{p} \Big(y_{(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \pm y_{(-\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \Big)^2 \tag{1.27}$$

(b) Invariance under shift $\underline{\mathbf{t}}$:

$$E^R_{[\mathrm{w}]} = \frac{1}{2p} \sum_{\alpha=1}^{p} \Big(y_{(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} - y_{(\underline{\mathbf{x}}^{(\alpha)}-\underline{\mathbf{t}};\underline{\mathbf{w}})} \Big)^2 \tag{1.28}$$

(c) Monotony:

$$E^R_{[\mathrm{w}]} = \frac{1}{n_p} \sum_{\underline{\mathbf{x}}^{(\alpha)} > \underline{\mathbf{x}}^{(\beta)}} \begin{cases} \Big(y_{(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} - y_{(\underline{\mathbf{x}}^{(\beta)};\underline{\mathbf{w}})} \Big)^2, & \text{if } y_{(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} < y_{(\underline{\mathbf{x}}^{(\beta)};\underline{\mathbf{w}})} \\ 0, & \text{else} \end{cases} \tag{1.29}$$

Choice of regularization parameter

→ perform model selection for different values of λ
(on training data - still weighted)

→ select value of λ, which provides best prediction results
(on test data - still biased)

→ validation of the selected model
(on validation data - test set method)

⇒ To find the optimal value of λ, one can use n-fold crossvalidation:

Algorithm 4: Choosing the hyperparameter via n-fold cross-validation

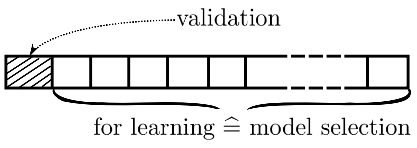
```
for $\lambda = \lambda_1$ TO $\lambda_n$ do
    perform n-fold cross-validation on data
end
Pick optimal $\lambda^{\text{opt}}$ with minimum $\widehat{E}^G$

Train network on all data with $\lambda^{\text{opt}}$
```

Nested n-fold cross-validation
If model fitting includes crossvalidation to find hyperparameters (e.g. regularization parameter), one can use *nested* n-fold crossvalidation (CV) to estimate the *generalization performance* of this model (model class + learning procedure).

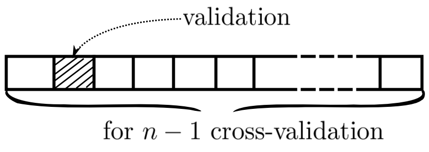
Using nested-CV to estimate generalization performance

(1)



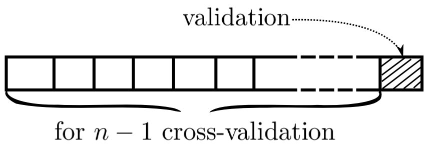
- do $n-1$ CV for all values of λ
- pick best result $\rightarrow \lambda_1$
- validate with $\lambda = \lambda_1 \rightarrow \hat{E}^G_{(1)}$

(2)



- do $n-1$ CV for all values of λ
- pick best result $\rightarrow \lambda_2$
- validate with $\lambda = \lambda_2 \rightarrow \hat{E}^G_{(2)}$

(3)



- do $n-1$ CV for all values of λ
- pick best result $\rightarrow \lambda_N$
- validate with $\lambda = \lambda_N \rightarrow \hat{E}^G_{(n)}$

Generalization performance $\widehat{E}^G$ of this training procedure is then computed as the average over all validation errors $\hat{E}^G_{(1)}, \hat{E}^G_{(2)}, ..., \hat{E}^G_{(n)}$.

Remarks:

- never use test data (for validation purposes) for selection
- always embed the whole selection procedure (including hyperparameter search) within an n-fold cross-validation run
- nested x-val is used here to estimate $\hat{E}^G$ of a specific procedure – for prediction: find best λ via simple crossvalidation and finally train on the whole dataset

1.4.7 Classification Problems

Remark: In the simplest case, the classification problem is binary such that the target class can be predicted by a single output unit (e.g. representing the probability of belonging to class 1 vs. 2). The use of a 1-out-of-c-code (see below) generalizes this idea and provides an approach allowing to deal with multi-class problems, too.

Observations: $\left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha = 1, \ldots, p$ from c classes $C_k, k = 1, \ldots, c$

① **Prediction of class labels**

- ordinal (or binary) attributes y

$\rightarrow$ gradient-based optimization methods not applicable

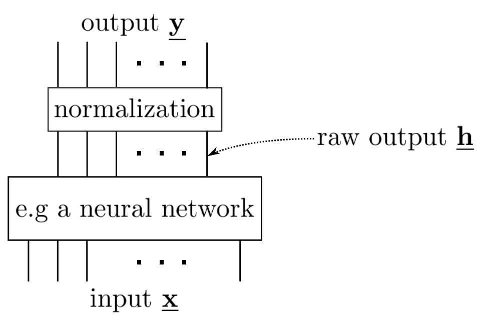
② **Prediction of class probabilities**

- real-valued attributes

$\rightarrow$ most of previous machinery can be transferred

Parametrized ansatz for class probabilities

$$y_{k(\underline{\mathbf{x}};\underline{\mathbf{w}})} := P_{(C_k|\underline{\mathbf{x}};\underline{\mathbf{w}})}$$


output $\underline{\mathbf{y}}$
normalization
raw output $\underline{\mathbf{h}}$
e.g a neural network
input $\underline{\mathbf{x}}$

probabilistic interpretation of network output

normalization:

$$0 \leq y_{k(\underline{\mathbf{x}};\underline{\mathbf{w}})} \leq 1$$

$$\sum_{k=1}^{c} y_{k(\underline{\mathbf{x}};\underline{\mathbf{w}})} = 1$$

Transfer function for the ouput layer:

$$y_{k(\underline{\mathbf{x}};\underline{\mathbf{w}})} = \frac{\exp h_{k(\underline{\mathbf{x}};\underline{\mathbf{w}})}}{\sum_l \exp h_{l(\underline{\mathbf{x}};\underline{\mathbf{w}})}} \qquad \text{(softmax function)}$$

1-out-of-c-code:
Observations: $\left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha = 1, \ldots, p$

$$y_{Tk} = \begin{cases} 0, & \underline{\mathbf{x}} \notin C_k \quad \Rightarrow \text{binary vector, one non-zero element} \\ 1, & \underline{\mathbf{x}} \in C_k \end{cases} \tag{1.30}$$

Limiting case of probabilities, assumption: true labels are known.
Cost function
True probability: $P_{(C|\underline{\mathbf{x}})}$
Model prediction: $P_{(C|\underline{\mathbf{x}};\underline{\mathbf{w}})}$
Kullbach-Leibler-Divergence D_{KL}:

$$D_{KL} = \sum_C \int d\underline{\mathbf{x}} P_{(\underline{\mathbf{x}})} P_{(C|\underline{\mathbf{x}})} \ln \frac{P_{(C|\underline{\mathbf{x}})}}{P_{(C|\underline{\mathbf{x}};\underline{\mathbf{w}})}} \cdot \frac{P_{(\underline{\mathbf{x}})}}{P_{(\underline{\mathbf{x}})}} \tag{1.31}$$

$\Rightarrow$ distance measure between probability distributions and densities

$\Rightarrow$ always non-negative

$\Rightarrow$ $D_{KL} = 0$ iff $P_{(C|\underline{\mathbf{x}})} \equiv P_{(C|\underline{\mathbf{x}};\underline{\mathbf{w}})}$ (both distributions / densities equal)

$$D_{KL} = -\sum_C \int d\underline{\mathbf{x}} P_{(\underline{\mathbf{x}})} P_{(C|\underline{\mathbf{x}})} \ln P_{(C|\underline{\mathbf{x}};\underline{\mathbf{w}})} + \underbrace{\sum_C \int d\underline{\mathbf{x}} P_{(\underline{\mathbf{x}})} P_{(C|\underline{\mathbf{x}})} \ln P_{(C|\underline{\mathbf{x}})}}_{\substack{\text{independent of} \\ \text{model parameters}}} \tag{1.32}$$

Measure of prediction performance: "cross entropy"

$$E^G = -\sum_C \int d\underline{\mathbf{x}} \underbrace{P_{(\underline{\mathbf{x}})} P_{(C|\underline{\mathbf{x}})}}_{\text{unknown!}} \ln P_{(C|\underline{\mathbf{x}};\underline{\mathbf{w}})} \tag{1.33}$$

Mathematical expectation $\rightarrow$ empirical average over training set:

$$E^T = -\frac{1}{p} \sum_{\alpha=1}^{p} \sum_{k=1}^{c} y_{Tk}^{(\alpha)} \ln y_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \tag{1.34}$$

Optimization via gradient descent (on-line):

$$e^{(\alpha)} = -\sum_{k=1}^{c} y_{Tk}^{(\alpha)} \ln y_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \tag{1.35}$$

$$\begin{aligned}
\frac{\partial e^{(\alpha)}}{\partial \underline{\mathbf{w}}} &= -\sum_{k=1}^{c} \frac{y_{Tk}^{(\alpha)}}{y_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}} \cdot \frac{\partial y_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}}{\partial \underline{\mathbf{w}}} \\
&= -\sum_{k=1}^{c} \frac{y_{Tk}^{(\alpha)}}{y_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}} \Bigg\{ y_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \frac{\partial h_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}}{\partial \underline{\mathbf{w}}} \\
&\quad - \frac{\exp h_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}}{\left(\sum_{l=1}^{c} \exp h_{l(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \right)^2} \sum_{l=1}^{c} \exp h_{l(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \frac{\partial h_{l(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}}{\partial \underline{\mathbf{w}}} \Bigg\} \\
&= -\sum_{k=1}^{c} y_{Tk}^{(\alpha)} \frac{\partial h_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}}{\partial \underline{\mathbf{w}}} + \underbrace{\left(\sum_{k=1}^{c} y_{Tk}^{(\alpha)} \right)}_{=1} \left(\sum_{l=1}^{c} y_{l(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \frac{\partial h_{l(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}}{\partial \underline{\mathbf{w}}} \right) \\
&= \sum_{k=1}^{c} \left(y_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} - y_{Tk}^{(\alpha)} \right) \underbrace{\frac{\partial h_{k(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}}{\partial \underline{\mathbf{w}}}}_{\substack{\text{using, e.g.} \\ \text{backpropagation}}}
\end{aligned} \tag{1.36}$$

Validation

test-set method or n-fold cross-validation

but: using the cross-entropy measure (*cf. (1.30)!*)

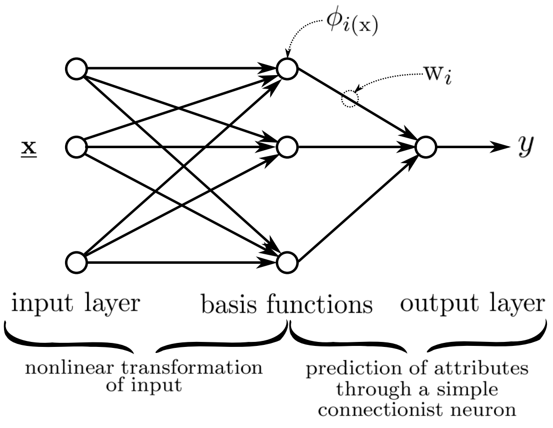
Comment: To find the "best" predictions of actual class labels, additional considerations are required:

- What is the *cost* of a misclassification error? $\rightarrow$ minimize average cost
- If all errors are equally costly: Choose class of highest probability

1.5 Radial Basis Function Networks

1.5.1 Network Architecture

Two layered network:



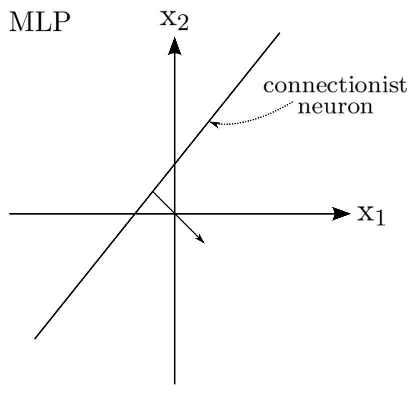
$$y_{(\underline{\mathbf{x}})} = \sum_i \mathrm{w}_i \phi_{i(\underline{\mathbf{x}})}$$

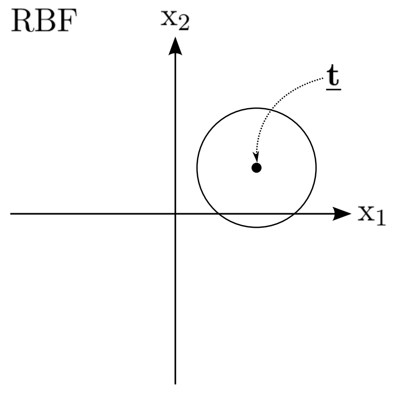
General principle: Expansion into basis functions, e.g. sine-waves (Fourier), polynomials (Taylor), sigmoidal functions (MLP)
r̲adial b̲asis f̲unction (RBF-) networks

$$\begin{aligned} \phi_{i(\underline{\mathbf{x}})} &= \widetilde{\phi}_{i(|\underline{\mathbf{x}}-\underline{\mathbf{t}}_i|)} && \Rightarrow \text{distance dependent output} \\ \phi_{i(\underline{\mathbf{x}})} &= \exp\left\{ -\frac{(\underline{\mathbf{x}}-\underline{\mathbf{t}}_i)^2}{2\sigma_i^2} \right\} && \text{most common choice: Gaussian functions} \end{aligned} \tag{1.37}$$

Differences MLP $\leftrightarrow$ RBF

(1) lines of equal output (basis functions)

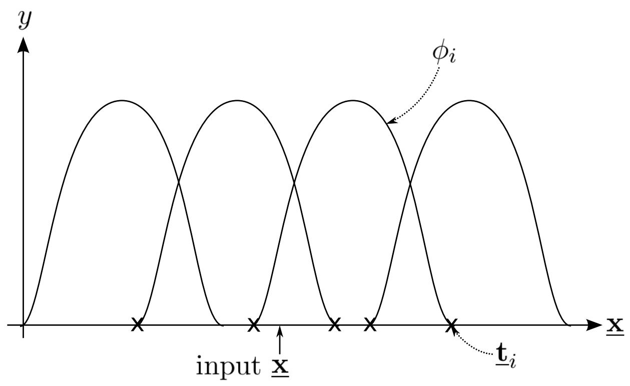




hyperplanes $\rightarrow$ half spaces
$\Rightarrow$ "discriminative features

hyperspheres $\rightarrow$ cluster
$\Rightarrow$ "categories/classes"
$\Rightarrow$ prototype-based analysis

(2) local approximation through RBF-networks



(3) advantages and disadvantages
Advantages: RBF-networks show fast convergence during learning

- few parameters have to be changed per training point
- "credit assignment" is simple

Disadvantages: RBF-networks are prone to "curse of dimensionality".
For good coverage of input space $\sim n^d$ basis functions required[4]
$\Rightarrow$ for $d = 20, n = 10 \rightsquigarrow 10^{20}$ basis functions

$\Rightarrow$ RBF-networks should be used for

- low dimensional data or
- datasets with a pronounced cluster structure

1.5.2 Model Selection - Learning

We assume real-valued attributes:

$$\left\{ \left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)} \right) \right\}, \alpha = 1, \ldots, p$$

$$\underline{\mathbf{x}} \in \mathbb{R}^d, y_T \in \mathbb{R}$$

(classification problems: modifications similar to chapter 1.4.7)

[4] d: dimension, n: no. of basis functions along one dimension

Model class

$$y_{(\underline{\mathbf{x}})} = \sum_{i=1}^{M} \mathrm{w}_i \phi_{i(\underline{\mathbf{x}})} = \sum_{i=1}^{M} \mathrm{w}_i \exp\left\{ -\frac{(\underline{\mathbf{x}} - \underline{\mathbf{t}}_i)^2}{2\sigma_i^2} \right\} \tag{1.38}$$

Parameters to be determined:

		two-step procedure	gradient-based methods
$\underline{\mathbf{t}}_i$	centroids of basis functions	unsupervised	supervised
σ_i	range of basis functions		
w_i	weights of the second layer	supervised	

Remark: The parameters of this architecture can be optimized using previously described methods e.g. gradient descent on the quadratic cost

$$E^T = \frac{1}{2P} \sum_{\alpha=1}^{P} [y(\underline{\mathbf{x}}^{(\alpha)} | \{\underline{\mathbf{t}}_i, \sigma_i\}) - y_T^{(\alpha)}]^2$$

While the problem is convex for the output weights w_i, this does not hold regarding the centroid locations t_i and ranges σ_i. Therefore, the heuristic approach described below is commonly used. Although it does not strictly minimze the training error E^T, this two-step procedure...

... is fast & robust

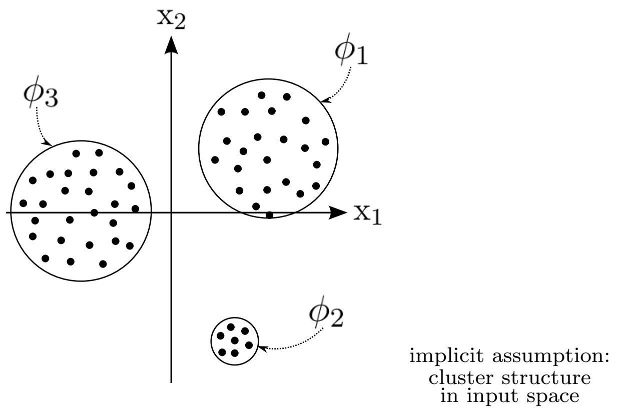
... usually yields equal performance to gradient-based methods

... can use unlabeled data to determine the centroids. This can be useful when obtaining labels y_T is costly but obtaining $\underline{\mathbf{x}}$ is cheap.

Learning via the two-step procedure: The two RBF-parameters $(\underline{\mathbf{t}}_i, \sigma_i)$ and the weights (w_i) can be efficiently determined in 2 subsequent steps.

(1) **Determination of RBF-parameters: Locations and Range**

a.) K-means clustering of centroids $\underline{\mathbf{t}}_i$ using algorithm (5)[5]



[5]for details, see MI2

Algorithm 5: K-means clustering

```
Initialization: $\underline{\mathbf{t}}_i$
begin
    Choose data point $\underline{\mathbf{x}}^{(\alpha)}$
    Determine the closest centroid $\underline{\mathbf{t}}_q$ : $q = \underset{r}{\operatorname{argmin}} \left|\underline{\mathbf{t}}_r - \underline{\mathbf{x}}^{(\alpha)}\right|$
    Change $\underline{\mathbf{t}}_q$ according to: $\Delta\underline{\mathbf{t}}_q = \eta\big(\underline{\mathbf{x}}^{(\alpha)} - \underline{\mathbf{t}}_q\big)$, $\eta$   (learning step)
end
```

Remarks regarding k-means clustering:

- For On-line learning, one typically starts with an initial phase of constant learning rate η and then decreases it slowly, e.g. $\eta \sim \frac{1}{\mathrm{t}}$
- For a fixed number of centroids $\underline{\mathbf{t}}_i$, K-means clustering minimizes the average quadratic distance between the data points and the closest centroid (i.e. the average size of the clusters).[6]

b.) Choice of width parameter: σ_i
Goal: Sufficient (but not too strong) overlap between neighboring basis functions:

$$\sigma_i = \lambda \min_{j \neq i} \left|\underline{\mathbf{t}}_i - \underline{\mathbf{t}}_j\right|, \lambda \approx 2 \tag{1.39}$$

(2) **Selection of output weights**

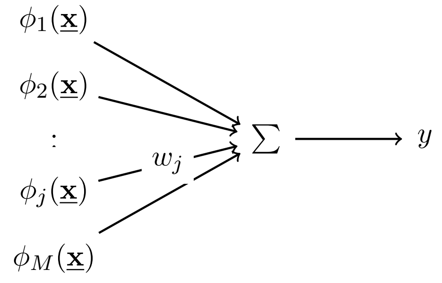

$\phi_1(\underline{\mathbf{x}})$
$\phi_2(\underline{\mathbf{x}})$
:
w_j
$\phi_j(\underline{\mathbf{x}})$
$\phi_M(\underline{\mathbf{x}})$
$\sum$
y

Figure 26: schema: input-output function for RBF networks

Cost function *(cf. chapter 1.3.3)*: quadratic error

$$E^T = \frac{1}{2p} \sum_{\alpha=1}^{p} \left(y_T^{(\alpha)} - \sum_{j=1}^{M} \mathrm{w}_j \underbrace{\phi_{j(\underline{\mathbf{x}}^{(\alpha)})}}_{:=\phi_j^{(\alpha)}} \right)^2 \tag{1.40}$$

[6]for details, see MI 2

Search for the minimum:

$$\frac{\partial E^T}{\partial \mathrm{w}_k} = -\frac{1}{p} \sum_{\alpha=1}^{p} \left(y_T^{(\alpha)} - \sum_{j=1}^{M} \mathrm{w}_j \phi_j^{(\alpha))} \right) \phi_k^{(\alpha)} \overset{!}{=} 0 \tag{1.41}$$

$$\sum_{j=1}^{M} \left(\sum_{\alpha=1}^{p} \phi_k^{(\alpha)} \phi_j^{(\alpha)} \right) \mathrm{w}_j = \sum_{\alpha=1}^{p} \phi_k^{(\alpha)} y_T^{(\alpha)} \tag{1.42}$$

In matrix notation:

$$\underbrace{\left(\underline{\phi}^T \underline{\phi}\right)}_{\text{known}} \underline{\mathbf{w}} = \underbrace{\underline{\phi}^T \underline{\mathbf{y}}_T}_{\text{known}} \tag{1.43}$$

$$\begin{array}{lll} \underline{\phi} & = \left\{\phi_k^{(\alpha)}\right\} & p \text{ x } M \text{ matrix} \\ \underline{\mathbf{w}} & = \left\{\mathrm{w}_j\right\} & M \text{ vector} \\ \underline{\mathbf{y}}_T & = \left\{y_T^{(\alpha)}\right\} & p \text{ vector} \end{array}$$

Remarks

Determination of output weights

- The system of linear equations in eq. (1.43) is known as the "normal equation" and central to least squares fitting (e.g. linear regression).
- $\underline{\phi}^T \underline{\phi}$ is often close to being a singular matrix
  $\rightsquigarrow$ use singular value decomposition[7]

Choice of number of basis functions

- too many basis functions $\Rightarrow$ overfitting
- too few basis functions $\Rightarrow$ underfitting

$\Rightarrow$ selection of number of basis functions via validation set is necessary *(cf. procedure in chapter 1.4.6)*

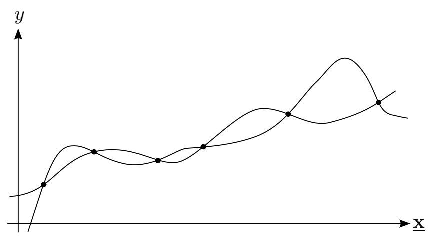
1.5.3 RBF-networks and Function Regularization

Regularization theory addresses the following, more general learning problem:

observations:	$\left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha = 1, \ldots, p$
model class:	all continuous and differentiable functions $y_{(\underline{\mathbf{x}})}$(!)
cost function:	$E^T = \frac{1}{2p} \sum_{\alpha=1}^{p} \left(y_{(\underline{\mathbf{x}}^{(\alpha)})} - y_T^{(\alpha)} \right)^2$

[7]cf. Numerical Recipes, 2nd edition, chapter 2.6

$\Rightarrow$ ill-posed learning problem $\rightarrow$ danger of overfitting $\rightarrow$ regularization



many functions are consistent with the data

Regularization: making the learning problem well-posed
Consider the following representation of a function $y_{(x)}$ in the frequency domain, i.e. parametrised via the amplitudes $\tilde{y}_{(\underline{\mathbf{k}})}$

$$y_{(\underline{\mathbf{x}})} = \int d\underline{\mathbf{k}} e^{(i\underline{\mathbf{k}}^T \underline{\mathbf{x}})} \tilde{y}_{(\underline{\mathbf{k}})} \qquad \text{(Fourier transform)}$$

Regularization term:

$$E^R = \frac{1}{2} \int d\underline{\mathbf{k}} \frac{\left|\tilde{y}_{(\underline{\mathbf{k}})}\right|^2}{\widetilde{G}_{(\underline{\mathbf{k}})}} \tag{1.44}$$

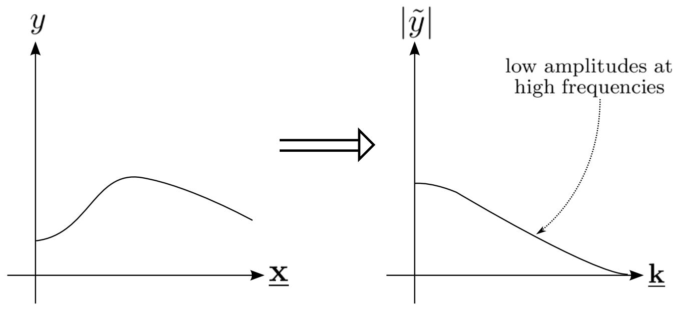
$\Rightarrow$ Filter $\widetilde{G}_{(\underline{\mathbf{k}})}$ imposes (soft-)constraints on admissible functions.
New cost function:

$$R = E^T + \lambda E^R \tag{1.45}$$

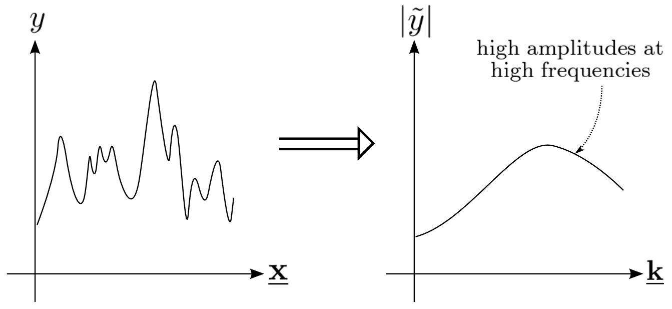
Note: Functions have NOT been parametrized but the problem is now well-posed
Interpretation of E^R:

$$\text{Fourier-transform} \begin{cases} \text{expansion into sine-waves} \\ \text{amplitude and phase information} \ \leftarrow \tilde{y} \end{cases}$$

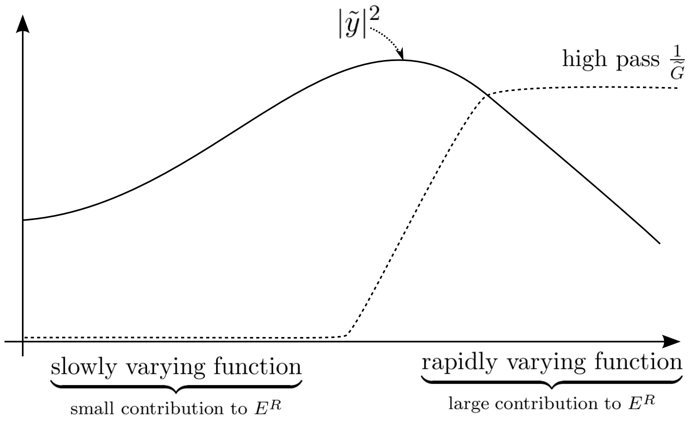
Smooth function:



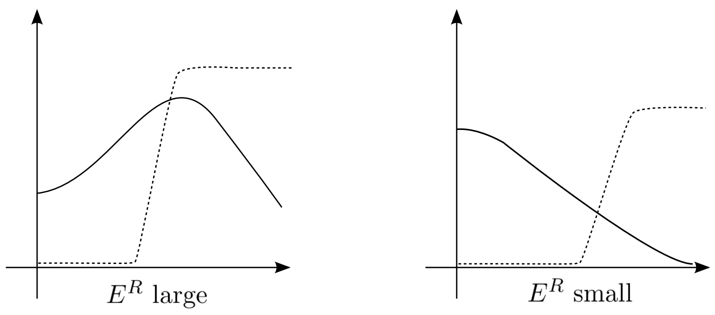
Rough function:



Role of filter $\widetilde{G}_{(\underline{\mathbf{k}})}$:


$|\tilde{y}|^2$
high pass $\frac{1}{\widetilde{G}}$
slowly varying function
small contribution to E^R
rapidly varying function
large contribution to E^R

Minimization of E^R:



high pass $\widetilde{G}^{-1} \Rightarrow$ implicit smoothness constraint
Result of minimization:

$$y_{(\underline{\mathbf{x}})} = \sum_{\alpha=1}^{p} \mathrm{w}_\alpha G_{(\underline{\mathbf{x}} - \underline{\mathbf{x}}^{(\alpha)})} \qquad \text{(RBF-network depending on filter)}$$

with

$$G_{(\underline{\mathbf{x}})} = \int d\underline{\mathbf{k}} e^{(i\underline{\mathbf{k}}^T\underline{\mathbf{x}})} \widetilde{G}_{(\underline{\mathbf{k}})} \qquad \text{(Fourier-transform of filter)}$$

Interpretation

- prior knowledge affects shape of basis functions
- location of data points determines location of centroids (unsupervised)
- labels (& prior knowledge) then determine output weights (supervised)

$$\underline{\mathbf{w}} = \frac{1}{\lambda p} \underline{\mathbf{G}}^{-1} \left(\underline{\mathbf{G}}^{-1} + \frac{1}{\lambda p} \underline{\mathbf{1}} \right)^{-1} \underline{\mathbf{y}}_T, \qquad \text{where } \underline{\mathbf{G}} = G_{(\underline{\mathbf{x}}^{(\alpha)} - \underline{\mathbf{x}}^{(\beta)})} \tag{1.46}$$

** check formula for weights: see Haykin 7.57 bzw. 5.76 **

Example: Gaussian functions

$$E^R = \int d\underline{\mathbf{k}} \underbrace{e^{\sigma^2 \underline{\mathbf{k}}^2}}_{\text{high pass}} \left| \tilde{y}_{(\underline{\mathbf{k}})} \right|^2 \quad \rightarrow \quad G_{(\underline{\mathbf{x}})} \sim \exp\left\{ -\frac{\underline{\mathbf{x}}^2}{\sigma^2} \right\} \tag{1.47}$$

$\Rightarrow$ close connection between RBF-networks and regularization

$\Rightarrow$ yet another model selection procedure for RBF-networks

$\Rightarrow$ but: no. of basis functions = no. of data points (large!)

$\rightsquigarrow$ sparse expansion desirable
$\rightsquigarrow$ support vector machines

2 Learning Theory and Support Vector Machines

Statistical learning theory (SLT) provides a general mathematical framework to describe and analyze learning problems. It allows to identify conditions under which inductive learning is possible. It provides bounds quantifying data requirements and limits to the performance of a learning algorithm. Support Vector Machines (see ch. 2.2) have been developed in the framework of SLT and represent a powerful learning algorithm based on the principle of Structural Risk Minimization (SRM).

2.1 Elements of Statistical Learning Theory

2.1.1 Formulation of the Problem

Learning can be understood as model selection with the goal of finding a model of optimal (generalization) performance. The previous chapter described inductive learning via *empirical risk minimization* (ERM, see 1.3.3) – SLT discusses when the ERM principle can be expected to work. □ learning as model selection

$$E^G_{[\underline{\mathbf{w}}]} = \int d\underline{\mathbf{x}} dy_T P_{(y_T,\underline{\mathbf{x}})} e_{(y_T,\underline{\mathbf{x}};\underline{\mathbf{w}})} \overset{!}{=} \min \qquad \text{(generalization error)}$$

generalization error (mathematical expectation)

$$\Downarrow \textbf{ ERM } \Downarrow \qquad \leftarrow \text{ When does this work?}$$

training error (empirical average)

$$E^T_{[\underline{\mathbf{w}}]} = \frac{1}{p}\sum_{\alpha=1}^{p} e_{(y_T^{(\alpha)},\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})} \overset{!}{=} \min \qquad \text{(training error)}$$

Change in Nomenclature

(to be consistent with Vapnik, 1998)

$$E^G_{[\underline{\mathbf{w}}]} \to R_{[\underline{\mathbf{w}}]} \qquad \text{(risk)}$$

$$E^T_{[\underline{\mathbf{w}}]} \to R_{\text{emp}[\underline{\mathbf{w}}]} \qquad \text{(empirical risk)}$$

$$e_{(y_T,\underline{\mathbf{x}};\underline{\mathbf{w}})} \to \underbrace{Q_{[\underline{\mathbf{z}};\underline{\mathbf{w}}]}}_{\text{data point; model}} \qquad \text{(individual cost)}$$

The learning problem

- p observations $\underline{\mathbf{z}}^{(\alpha)}, \alpha = 1, \ldots, p$ drawn iid from unknown distribution
- *stationarity:* training and test distributions are identical

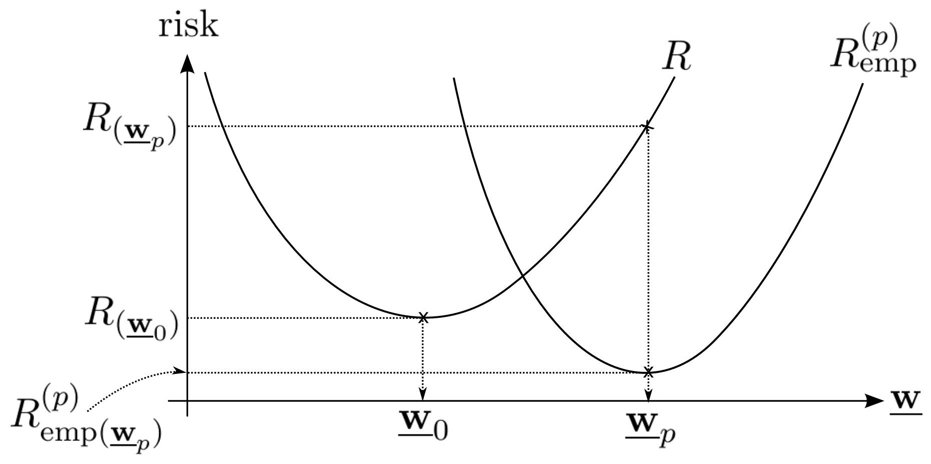


Figure 27: the learning problem

When does inductive learning through empirical risk minimization work?

Goal: Find conditions for which:

$$\lim_{p\to\infty} P\Big\{\big|R_{(\underline{\mathbf{w}}_p)} - R_{(\underline{\mathbf{w}}_0)}\big| \geq \eta\Big\} = 0 \text{ for all } \eta > 0 \tag{2.1}$$

Requirement: The ERM procedure should converge to the optimal predictor in the limit of infinite number of observations.

How strongly does $R_{(\underline{\mathbf{w}}_p)}$ differ from $R_{(\underline{\mathbf{w}}_O)}$ for a finite sample?
For a given confidence ϵ, find η such that

$$P\Big\{\big|R_{(\underline{\mathbf{w}}_0)} - R_{(\underline{\mathbf{w}}_p)}\big| \geq \eta\Big\} < \epsilon \tag{2.2}$$

2.1.2 The Key Theorem of Learning Theory

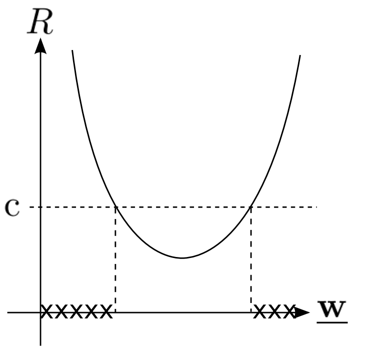
$$\begin{aligned} \Lambda = \{\underline{\mathbf{w}}\} :& \quad \text{set of all predictors} \\ \Lambda_{(c)} \subset \Lambda :& \quad \text{set of all "bad" predictors} \end{aligned}$$

Definition of *strict consistency*:

$$\lim_{p\to\infty} P\Big\{\Big|\inf_{\underline{\mathbf{w}}\in\Lambda_{(c)}} R_{\mathrm{emp}(\underline{\mathbf{w}})}^{(p)} - \inf_{\underline{\mathbf{w}}\in\Lambda_{(c)}} R_{(\underline{\mathbf{w}})}\Big| \geq \eta\Big\} = 0 \tag{2.3}$$

$$\text{strict consistency} \;\underset{\not\leftarrow}{\rightarrow}\; \begin{array}{c}\text{inductive learning via ERM} \\ \text{(eq. (2.1))}\end{array}$$

proof: supplementary material



$\times \times \times : \Lambda_{(c)} = \{\underline{\mathbf{w}} : R_{(\underline{\mathbf{w}})} \geq c\}$

Figure 28: key theorem

Key theorem
Let $a \leq R_{(\underline{\mathbf{w}})} \leq A$ for all $\underline{\mathbf{w}} \in \Lambda$ and $P_{(\underline{\mathbf{z}})} \in \Pi$, then:

$$\begin{gathered}\text{The ERM procedure is strictly consistent for } \Lambda \text{ and } \Pi \\ \Updownarrow \\ \lim_{p \to \infty} P\left\{ \sup_{\underline{\mathbf{w}} \in \Lambda} \left(R_{(\underline{\mathbf{w}})} - R^{(p)}_{\text{emp}(\underline{\mathbf{w}})} \right) > \eta \right\} = 0 \text{ for all } P_{(\underline{\mathbf{z}})} \in \Pi\end{gathered} \tag{2.4}$$

proof: supplementary material

Learning by induction through ERM is linked to the uniform (one-sided) convergence of the empirical averages $R^{(p)}_{\text{emp}(\underline{\mathbf{w}})}$ to their mathematical expectations $R_{(\underline{\mathbf{w}})}$.

Law of large numbers
"The sequence of means converges to the expectation of a random variable (if it exists) as the number p of samples increases."

$\Rightarrow$ can only be applied if set of predictors has a finite number of elements (use Hoeffding's inequality for bound on deviations)

$\Rightarrow$ extension to infinite sets (e.g. MLPs) necessary

$\Rightarrow$ characterization of sets of predictors by "capacity measures" (since number of models is infinite)

Why does this matter? The key theorem of SLT provides the conditions under which one can expect to learn from examples. It gives bounds telling how good we can expect our predictions to be.

2.1.3 An important example: Linear classifiers

Cover (1965) computes bounds on the pattern-separating capacity of hyperplanes and nonlinear decision surfaces. It demonstrates that the probability of ambiguous generalization is large unless the number of training patterns exceeds the capacity of the classifier.

$\Rightarrow$ link between the capacity of a classifier (linear classifier) and its generalisation performance (probability of ambiguous generalization). A related calculation of the capacity of linear connectionist neurons can be found in (MacKay, 2003, ch. 40.3).

Set of classifiers: binary connectionist neurons with $y = \text{sign}(\underline{\mathbf{w}}^T \underline{\mathbf{x}})$

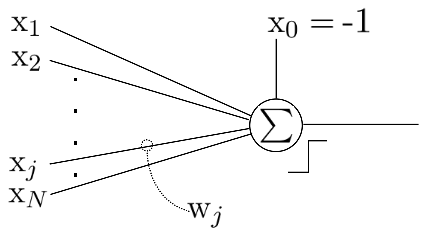
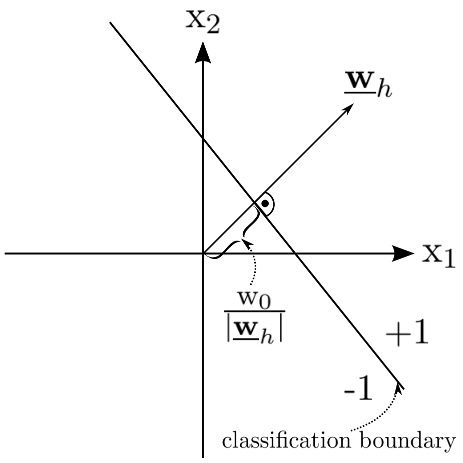


Figure 29: binary connectionist neuron

Classification boundary: $\underline{\mathbf{w}}^T \underline{\mathbf{x}} = 0 \rightsquigarrow$ hyperplane



x_2
x_1
$\underline{\mathbf{w}}_h$
$\frac{w_0}{|\underline{\mathbf{w}}_h|}$
+1
-1
classification boundary

$$\underline{\mathbf{w}} = \begin{pmatrix} w_0 \\ \underline{\mathbf{w}}_h \end{pmatrix}$$

Figure 30: classification boundary induced by weight vector $\underline{\mathbf{w}}_h$

Linear separability

$$\left.\begin{array}{l} \rightsquigarrow \text{classes do not overlap} \\ \rightsquigarrow \text{classes can be separated by a hyperplane} \end{array}\right\} \begin{array}{c} \text{all data points can be correctly} \\ \text{classified by at least one classifier} \\ \text{from the set} \end{array}$$

Statistics of linear separability
Data: observations $\underline{\mathbf{x}}^{(\alpha)} \in \mathbb{R}^N, \alpha = 1, \dots, p$, in general location
$\rightarrow$ (each subset of N points should be linearly independent)

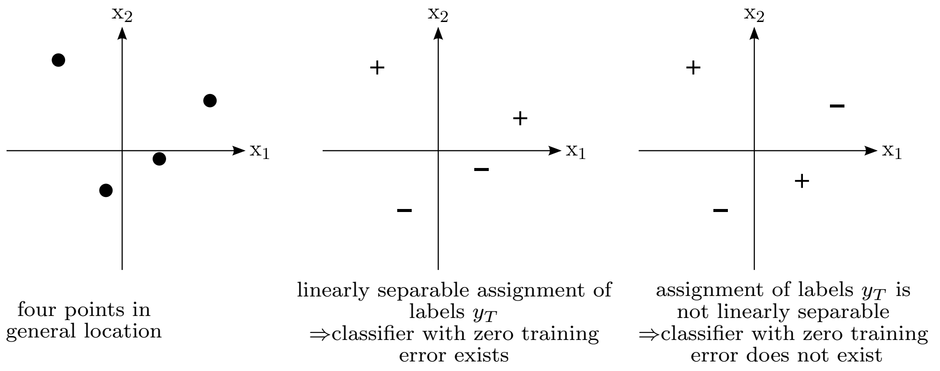

x2
x1
four points in general location
linearly separable assignment of labels y_T ⇒classifier with zero training error exists
assignment of labels y_T is not linearly separable ⇒classifier with zero training error does not exist

Figure 31: Separability of randomly drawn points

(1) **Number** $C_{(P,N)}$ of linearly separable assignments $y_T : \underline{\mathbf{x}} \rightarrow y_T$:

$$C_{(P,N)} = 2 \sum_{k=0}^{N-1} \binom{P-1}{k} \quad \text{using} \quad \overbrace{\binom{r}{q}}^{\text{binomial coefficient}} = \begin{cases} \frac{r!}{q!(r-q)!}, & r \geq q \\ 0, & \text{else} \end{cases} \tag{2.5}$$

proof: supplementary material
Note: N refers to the number dimensions in the homogeneous case ($w_0 = 0$) or number of dimensions+1 for ($w_0 \neq 0$).

(2) **Fraction** $\Pi_{(P,N)}$ of linearly separable assignments:

$$\Pi_{(P,N)} = \underbrace{\frac{C_{(P,N)}}{2^P}}_{\text{number of all possible assignment}} = \frac{\sum_{k=0}^{N-1} \binom{P-1}{k}}{2^{P-1}} \tag{2.6}$$

(3) **Difference** of linearly separable assignments, when one data point is added to the training set:

$$\Delta\Pi_{(P)} := \Pi_{(P,N)} - \Pi_{(P+1,N)} = \frac{C_{(P,N)}}{2^P} - \frac{1}{2}\frac{C_{(P+1,N)}}{2^P} \tag{2.7}$$

with $\begin{pmatrix} r \\ q \end{pmatrix} = \begin{pmatrix} r-1 \\ q \end{pmatrix} + \begin{pmatrix} r-1 \\ q-1 \end{pmatrix}$ we obtain (Pascal's triangle):

$$\begin{aligned}
\Delta\Pi_{(P)} &= \tfrac{1}{2^P}\Big(C_{(P,N)} - \tfrac{1}{2}C_{(P,N)} - \tfrac{1}{2}C_{(P,N-1)}\Big) \\
&= \tfrac{1}{2} \cdot \tfrac{1}{2^P} \begin{pmatrix} P-1 \\ N-1 \end{pmatrix} \\
\Delta\Pi_{(P)} &\approx k \begin{pmatrix} P-1 \\ N-1 \end{pmatrix} \left(\tfrac{1}{2}\right)^{N-1} \left(\tfrac{1}{2}\right)^{(P-1)-(N-1)} \leftarrow \text{ binomial distribution}
\end{aligned} \tag{2.8}$$

(4) **Limit** $N, P \to \infty, \frac{P}{N} = \text{const}$: binomial distrib. $\to$ **normal distrib.**

$$\Delta\Pi_{(N)} \approx N_{(P/2, \sqrt{P}/2)} \qquad \text{(asymptotical distribution)}$$

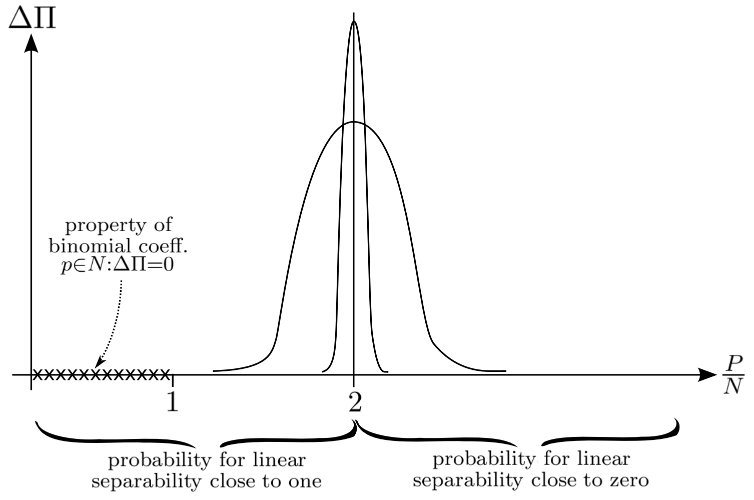


Figure 32: Classification capacity: all 2^P label configurations are realizable below $\frac{P}{N} = 2$. This number therefore provides a capacity measure. Compare with the storage capacity of Hopfield networks under the perceptron learning rule.

Capacity of the set of classifiers Optimal predictors and generalization performance

- Number of ambiguous assignments: $C_{(P,N-1)}$
- for P data points and linearly separable assignments on $P-1$ data points, there are still two choices left for the remaining data point. *proof: supplementary material*

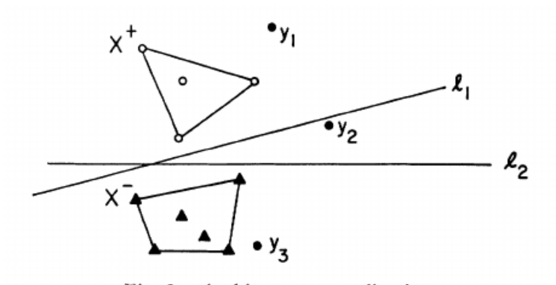


Fig. 2. Ambiguous generalization.

Figure 33: Illustration of ambiguous assignments (from Cover, 1965)

⇒ **Fraction of ambiguous, linearly separable assignments**

$$g_{(P,N)} := \frac{C_{(P,N-1)}}{\underbrace{C_{(P,N)}}_{\substack{\text{all linearly} \\ \text{separable assignments}}}} \tag{2.9}$$

$$g^*_{(\frac{P}{N})} = \underbrace{\lim_{P,N\to\infty}}_{\frac{P}{N}=\text{const}} g_{(P,N)} = \begin{cases} 1, & \text{for } 0 \le \frac{P}{N} \le 2 \\ \frac{1}{\frac{P}{N}-1}, & \text{for } \frac{P}{N} > 2 \end{cases} \tag{2.10}$$

proof: supplementary material

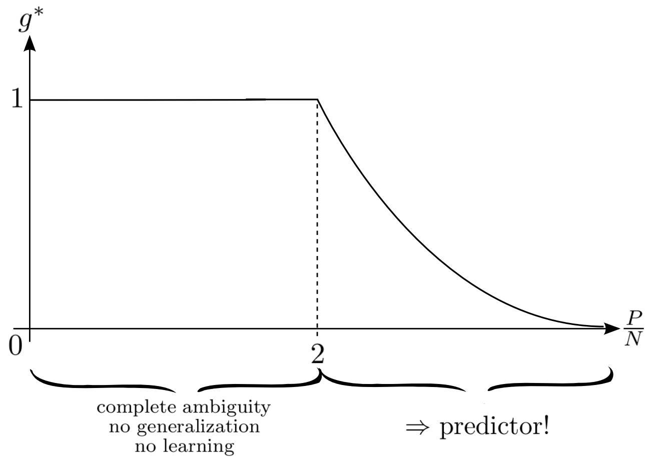


Figure 34: fraction of ambiguous assignments

Interpretation: This means that for $P/N > 2$ learning is possible, i.e. a learned classifier can make useful predictions.

2.1.4 Conditions for Successful Learning with ERM

Under which conditions one can learn with a specific model ($\rightarrow$ classifier) clearly depends on both the model and the availability of data. The *growth function* is an important tool to describe this relation.

Capacity measures quantitatively characterize a classifier and we describe the *VC-Dimension* which will further allow us to derive bounds on the growth function.

General classification problems
Observations: $\left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha = 1, \ldots, p \qquad \underline{\mathbf{x}} \in \mathbb{R}^N, y_T \in \{-1, +1\}$

Set of classifiers: $y_{(\underline{\mathbf{x}};\underline{\mathbf{w}})} \qquad y \in \{-1, +1\}$

Individual cost:

$$Q_{(\underline{\mathbf{z}};\underline{\mathbf{w}})} = 1 - \delta_{y_T y} = \begin{cases} 0, & \text{if } y_{(\underline{\mathbf{x}};\underline{\mathbf{w}})} = y_T \\ \\ 1, & \text{else} \end{cases} \qquad \Rightarrow 0 - 1 \text{ loss} \tag{2.11}$$

Model selection via ERM:

$$R^{(p)}_{\text{emp}[\underline{\mathbf{w}}]} = \frac{1}{p} \sum_{\alpha=1}^{p} Q_{(\underline{\mathbf{z}}^{(\alpha)};\underline{\mathbf{w}})} \overset{!}{=} \min \tag{2.12}$$

Capacity measures for the set of classifiers
Binary cost vector: $q_{(\underline{\mathbf{w}})} = \left(Q_{(\underline{\mathbf{z}}^{(1)},\underline{\mathbf{w}})}, Q_{(\underline{\mathbf{z}}^{(2)},\underline{\mathbf{w}})}, \ldots, Q_{(\underline{\mathbf{z}}^{(p)},\underline{\mathbf{w}})}\right)$
$\Rightarrow$ different classifiers can induce the same cost vector on the training set

Number of different vectors $q_{(\underline{\mathbf{w}})}$ induced by the whole set Λ of classifiers:

$$N^{\overbrace{\Lambda}^{\text{model class}}}_{\underbrace{(\underline{\mathbf{z}}^{(1)}, \ldots, \underline{\mathbf{z}}^{(p)})}_{\text{training set}}} \tag{2.13}$$

$\Rightarrow$ equivalent to the number of labelings (classification) induced by the set of classifiers on a given sample $\left(\underline{\mathbf{x}}^{(1)}, \ldots, \underline{\mathbf{x}}^{(p)}\right)$

$\Rightarrow$ $N^{\Lambda}_{(\underline{\mathbf{z}}^{(1)},\ldots,\underline{\mathbf{z}}^{(p)})} \leq 2^p$

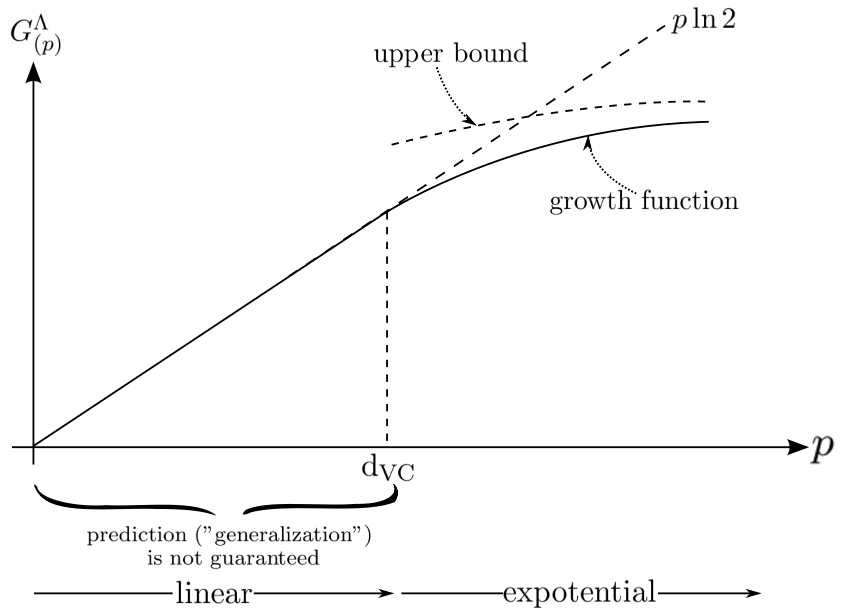
Note: General statements about the model class should not depend on the specific sample observed $\Rightarrow$ Need for a sample and distribution free quantity

$$G^{\Lambda}_{(p)} = \underbrace{\ln\left(\sup_{\underline{\mathbf{z}}^{(1)},\ldots\underline{\mathbf{z}}^{(p)}} N^{\Lambda}_{(\underline{\mathbf{z}}^{(1)},\ldots,\underline{\mathbf{z}}^{(p)})}\right)}_{\text{worst case}} \qquad \text{(growth function)}$$

bound on the growth function

$$G^{\Lambda}_{(p)} \begin{cases} = p \ln 2 & \text{for } p \leq \mathrm{d_{VC}} \\ \leq \mathrm{d_{VC}} \Big(1 + \ln \frac{p}{\mathrm{d_{VC}}} \Big) & \text{for } p > \mathrm{d_{VC}} \end{cases} \tag{2.14}$$

proof: Vapnik (1998, ch. 4.10)



Note: While the number of potential labelings grows as 2^p, the growth function quantifies in how many different ways the data could be labeled by the classifier. The higher the capacity of the classifier, the more different labelings are possible.

$\mathrm{d_{VC}}$: VC-dimension (Vapnik, Chernovenkis)

- capacity measure for a given set of models
- largest number of data points on which (at least for one set) 2^p different loss vectors (i.e. label assignments) can be induced

$\Rightarrow$ These results provide answers to the questions posed in chapter 2.1.1:

(1) **Learnability**

$$P\Big\{ \sup_{\underline{\mathbf{w}} \in \Lambda} \Big| R_{(\underline{\mathbf{w}})} - \underbrace{R^{(p)}_{\text{emp}(\underline{\mathbf{w}})}}_{\frac{1}{p} G^{\Lambda}_{(p)} \xrightarrow[p \to \infty]{} 0} \Big| > \eta \Big\} \xrightarrow[p \to \infty]{\text{a.s.}} 0 \tag{2.15}$$

proof: Vapnik (1998, ch. 14)

Note:

$$\frac{1}{p} H^{\Lambda}_{(p)} \xrightarrow[p\to\infty]{} 0 \qquad \text{(original proof)}$$

and

$$H^{\Lambda}_{(p)} = \Big\langle \ln N^{\Lambda}_{(\underline{\mathbf{z}}^{(1)},\ldots,\underline{\mathbf{z}}^{(p)})} \Big\rangle_{\underline{\mathbf{z}}} \qquad \text{('entropy')}$$

Inductive learning using the ERM method is only guaranteed (for large enough sample size) if d_{VC} is finite.

□ example of an un-learnable problem

(2) **Finite samples: Bound on the generalization error**

$$P\Big\{ \sup_{\underline{\mathbf{w}}\in\Lambda} \Big| R_{(\underline{\mathbf{w}})} - R^{(p)}_{\mathrm{emp}(\underline{\mathbf{w}})} \Big| > \eta \Big\} < 4 \exp \Big\{ G^{\Lambda}_{(2p)} - p\Big(\eta - \frac{1}{p}\Big)^2 \Big\} \tag{2.16}$$

proof: supplementary material

Note:

$$G^{\Lambda}_{(2p)} \to \underbrace{H^{\Lambda}_{\mathrm{ann}(2p)}}_{\text{annealed entropy}} = \ln \Big\langle N^{\Lambda}_{(\underline{\mathbf{z}}^{(1)},\ldots,\underline{\mathbf{z}}^{(p)})} \Big\rangle_{\underline{\mathbf{z}}} \qquad \text{(original proof)}$$

and

$$H^{\Lambda}_{(2p)} \le H^{\Lambda}_{\mathrm{ann}(p)} \le G^{\Lambda}_{(p)} \tag{2.17}$$

bound is non-trivial only if $G^{\Lambda}_{(2p)}$ sublinear in p

$$\begin{aligned} G^{\Lambda}_{(2p)} - p\Big\{\eta - \tfrac{1}{p}\Big\}^2 &= p\Big\{2\ln 2 - \Big(\eta - \tfrac{1}{p}\Big)^2\Big\} > p\big\{2\ln 2 - 1\big\} \\ &= 0.386p > 1 \end{aligned} \tag{2.18}$$

(3) **Finite samples: Deviation from the optimal model**

$$P\left\{ R_{(\underline{\mathbf{w}}_p)} - R_{(\underline{\mathbf{w}}_0)} > \left\{ \frac{G^{\Lambda}_{(2p)} - \ln\frac{\epsilon}{8}}{p} \right\}^{\frac{1}{2}} + \left\{ -\frac{\ln\frac{\epsilon}{2}}{2p} \right\}^{\frac{1}{2}} + \frac{1}{p} \right\} < \epsilon \tag{2.19}$$

proof: supplementary material

$$p\uparrow \begin{cases} \epsilon\downarrow \text{ for equal bound on the difference} \\ \text{bound on the difference } \downarrow \text{ for equal } \epsilon \end{cases}$$

Comments

$\Rightarrow$ bounds can be made tighter and condition on learnability made less strict if a capacity measure is used, which depends on the distribution $P_{(\underline{\mathbf{z}})}$ (entropy or annealed entropy)

$\Rightarrow$ better bounds can be achieved using other capacity measures

$\Rightarrow$ regression problems can be treated in a similar manner if every continuous loss function $Q_{(\underline{\mathbf{z}},\underline{\mathbf{w}})}$ is replaced by a set of binary indicator functions

$$I_{(\underline{\mathbf{z}},\underline{\mathbf{w}},\beta)} = \underbrace{\theta_{\left(Q_{(\underline{\mathbf{z}},\underline{\mathbf{w}})}-\beta\right)}}_{\substack{\text{step}\\ \text{function}}}, \beta \in \mathbb{R} \tag{2.20}$$

2.2 Support Vector Machines

While the bounds derived from SLT are interesting from a theoretical point of view, they more generally justify structural risk minimisation as a guiding principle for model optimization.

Support Vector Machines (SVMs) provide an important example of a learning approach that is based on structural risk minimization (SRM) rather than ERM.

2.2.1 Learning by Structural Risk Minimization

Typical bound on the generalization error for ERM:

$$P\left\{ \sup_{\underline{\mathbf{w}}\in\Lambda} \left| R_{(\underline{\mathbf{w}})} - R^{(p)}_{\text{emp}(\underline{\mathbf{w}})} \right| > \eta \right\} < \underbrace{4 \exp\left\{ G^{\Lambda}_{(2p)} - p\left(\eta - \frac{1}{p}\right)^2 \right\}}_{\overset{!}{=}\epsilon} \tag{2.21}$$

In this formulation, η depends on ϵ. Therefore with probability larger than $1-\epsilon$ we obtain:

$$R_{(\underline{\mathbf{w}})} < R^{(p)}_{\text{emp}(\underline{\mathbf{w}})} + \left\{ \frac{G^{\Lambda}_{(2p)} - \ln \frac{\epsilon}{4}}{p} \right\}^{\frac{1}{2}} + \frac{1}{p} \tag{2.22}$$

cf. supplementary material

$$R_{(\underline{\mathbf{w}})} < \underbrace{R^{(p)}_{\text{emp}(\underline{\mathbf{w}})}}_{\substack{\text{empirical}\\ \text{error}}} + \underbrace{C}_{\substack{\text{complexity}\\ \text{term}}}, C = C_{(\mathrm{d_{VC}},P)} \tag{2.23}$$

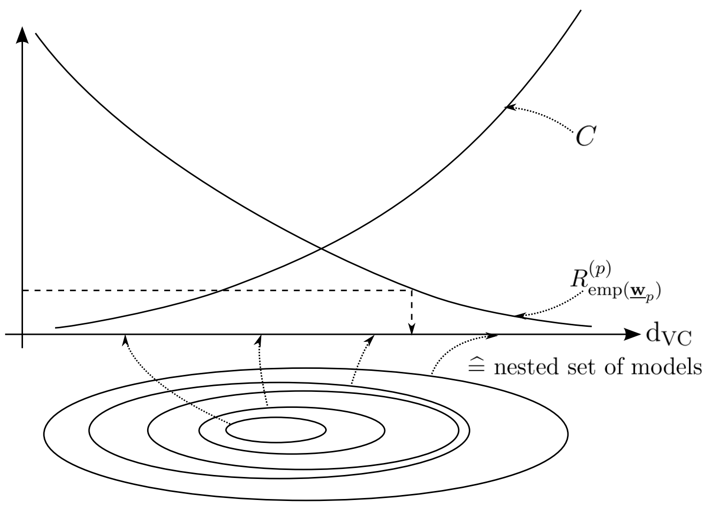

C
$R^{(p)}_{\text{emp}(\underline{\mathbf{w}}_p)}$
d_{VC}
$\widehat{=}$ nested set of models

Figure 35: Illustration of the SRM approach: Inner to outer ellipses represent models of increasing capacity (d_{VC}). Models on the very left have low capacity and high training error ($\rightarrow$ underfitting), models on the right have high capacity and low training error ($\rightarrow$ overfitting).

<u>S</u>tructural <u>R</u>isk <u>M</u>inimization: *Minimize capacity of the model class under the constraint, that the empirical error is not too large.*

$\Rightarrow$ Keeping empirical error fixed, $R_{(\underline{\mathbf{w}})}$ can be optimized by minimizing the capacity C of the model.

Note: SRM-learning is consistent (*see Vapnik, 1998, ch. 6.3)*

2.2.2 Application of SRM to Classification with Binary Connectionist Neurons

Training data: $\left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha = 1, \ldots, p, \qquad \underline{\mathbf{x}} \in \mathbb{R}^N, y_T \in \{-1, +1\}$

Connectionist neurons: $y = \text{sign}\left(\underline{\mathbf{w}}^T \underline{\mathbf{x}} + b\right)$

Classification boundary:

$$\underline{\mathbf{w}}^T\underline{\mathbf{x}} + b = 0 \qquad \text{(hyperplane)}$$

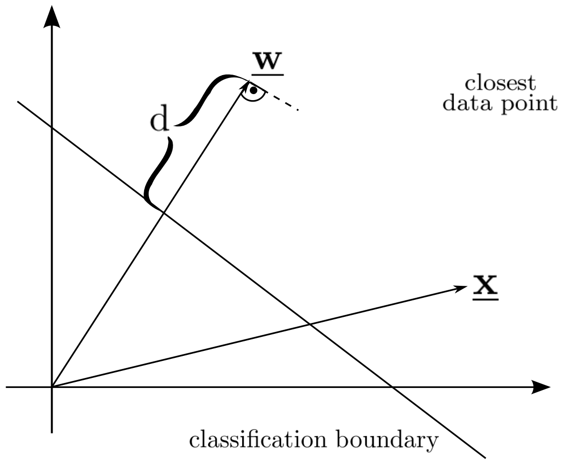
⇒ does not change if $\underline{\mathbf{w}}^T$ and b are multiplied by the same factor

Canonical hyperplanes:

Data dependent normalization of $\underline{\mathbf{w}}$ and b such that:

$$\min_{\alpha=1,\ldots,p} \Big| \underbrace{\underline{\mathbf{w}}^T\underline{\mathbf{x}}^{(\alpha)} + b}_{:=h} \Big| \overset{!}{=} 1 \tag{2.24}$$

For the closest data point, this sets the Normalized distance d to



$$\mathrm{d} = \Bigg| \underbrace{\frac{\underline{\mathbf{w}}^T\underline{\mathbf{x}}}{|\underline{\mathbf{w}}|}}_{\substack{\text{length of}\\ \text{projection of}\\ \underline{\mathbf{x}} \text{ on } \underline{\mathbf{w}}}} + \underbrace{\frac{b}{|\underline{\mathbf{w}}|}}_{\substack{\text{distance from}\\ \text{origin } (b \text{ is}\\ \text{negative because}\\ \text{origin in direction}\\ -\underline{\mathbf{w}}}} \Bigg|$$

$$\mathrm{d} = \frac{1}{|\underline{\mathbf{w}}|} \underbrace{\big|\underline{\mathbf{w}}^T\underline{\mathbf{x}} + b\big|}_{\overset{!}{=}1} = \frac{1}{|\underline{\mathbf{w}}|} \qquad \text{"margin" of a canonical hyperplane}$$

To apply the SRM principle, we need to construct a nested sequence of sets of models. These models are parametrised via their weights:

$$|\underline{\mathbf{w}}| \leq \mathrm{d}_{\mathrm{w}}^{-1} \tag{2.25}$$

⇝ for each value of $\underline{\mathbf{w}}$, this yields a set of connectionist neurons with margin larger than d_{w}.

⇝ sequence of numbers d_{w} leads to a nested sequence of sets of models

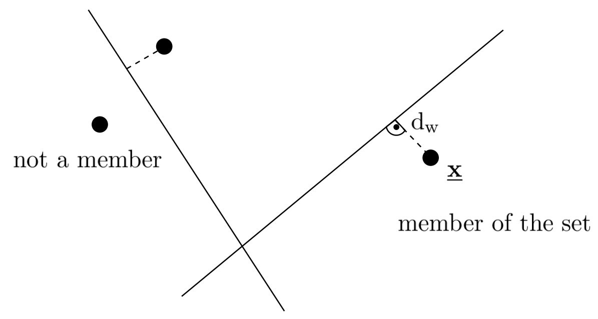


Figure 36: Application of the SRM principle: Better figure???

$\rightsquigarrow$ minimizing $|\underline{\mathbf{w}}|$ corresponds to maximizing the margin

Note: Large $\mathrm{d_w}$ means low capacity, because some labelings cannot be induced. This relation between the weight-dependent margin d_w and the capacity of the classifier is quantified in the following theorem providing a bound on the VC-Dimension.

Theorem: Let $|\underline{\mathbf{x}}| \leq R$ (bounded range of $\underline{\mathbf{x}}$ for which $P_{(\underline{\mathbf{x}})} \neq 0$), then:

$$\mathrm{d_{VC}} \leq \min\left(\underbrace{\left[\frac{R^2}{\mathrm{d_w^2}}\right]}_{\text{New!}}, N\right) + \underbrace{1}_{\substack{\mathrm{d_{VC}}\text{of the} \\ \text{connectionist} \\ \text{neuron}}} \tag{2.26}$$

proof: Vapnik (1998, ch. 8.5)

Note: $\frac{R^2}{\mathrm{d_w^2}}$ is independent of the dimension N of feature space

- $\rightsquigarrow$ VC-dimension of "fat" hyperplanes is independent of the number N of parameters
- $\rightsquigarrow$ this provides an approach to learn classification in high-dimensional spaces that avoids overfitting

2.2.3 SRM Learning for Linearly Separable Problems

We now apply this approach to learn the weights of a connectionist neuron (i.e. a linear classifier) assuming that data are linearly separable.

Training data:

$$\left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha = 1, \ldots, p, \qquad \underline{\mathbf{x}} \in \mathbb{R}^N, y_T \in \{-1, +1\}$$

Model selection through SRM: Primal Problem

$\frac{1}{2}|\underline{\mathbf{w}}|^2 \overset{!}{=} \min$ minimization of model complexity

$y_T^{(\alpha)}\left(\underline{\mathbf{w}}^T\underline{\mathbf{x}}^{(\alpha)} + b\right) \geq 1 \quad, \forall\alpha$

Note: The "$\geq$ 1" guarantees correct classification of all training data points for the normalized margin of 1 (see eq. 2.24).

Note: This is a convex (quadratic!) optimization problem $\rightsquigarrow$ only one (global!) minimum. There are standard (quadratic programming) algorithms to find the solution!

Theorem by Kuhn and Tucker

This theorem provides a strategy to efficiently solve the optimization problem of finding the optimal weight vector $\underline{\mathbf{w}}$. It links the solution for the original ("primal") problem to a "dual" problem which is often easier to solve (see below, generalization to nonlinear problems).

Preliminaries: Consider

- $\underline{\mathbf{x}} \in \mathrm{X}$ linear space X ($\rightsquigarrow$ Jensen's equality holds)
- $A \subset \mathrm{X}$ where A is a convex set
- $f_{k(\underline{\mathbf{x}})}$ convex functions $f_k : \mathrm{X} \to \mathbb{R}, k = 0, \ldots, m$
- $\exists\underline{\mathbf{x}}$ such that $f_{k(\underline{\mathbf{x}})} < 0 \quad \forall k \in 1, \ldots, m$

The last point implies that there is a solution for which all constraints (formulated via the functions $f_{k(\underline{\mathbf{x}})}$) can be fulfilled.

(primal) Optimization problem
For $\underline{\mathbf{x}} \in A$ find

$$\underbrace{f_{0(\underline{\mathbf{x}})} \overset{!}{=} \min}_{\text{minimization}} \quad \text{subject to} \quad \underbrace{f_{k(\underline{\mathbf{x}})} \leq 0, k = 1, \ldots, m}_{\text{constraints}} \tag{2.27}$$

Lagrangian

$$L_{(\mathrm{x},\{\lambda_k\})} \overset{!}{=} f_{0(x)} + \sum_{k=1}^{m} \underbrace{\lambda_k}_{\substack{\text{lagrange}\\\text{multipliers}}} \underbrace{f_{k(\underline{\mathbf{x}})}}_{\text{constraints}} \tag{2.28}$$

Equivalent optimization problem (existence of a saddle point):

$$\min_{\mathrm{x}\in A} L_{(\mathrm{x},\{\lambda_k^*\})} = L_{(\mathrm{x}^*,\{\lambda_k^*\})} = \max_{\lambda_k\geq 0} L_{(\mathrm{x}^*,\{\lambda_k\})} \tag{2.29}$$

$\Rightarrow$ If the Lagrangian has such a saddle point, then x^* is a solution to the constrained optimization problem and can be found by maximizing the right hand side with respect to the λ_k.

Comments

$\rightsquigarrow$ violation of constraints

- $\rightarrow$ f_k positive
- $\rightarrow$ maximization with respect to λ_k leads to divergence of L
- $\rightarrow$ minimization with respect to x assures, that constraints are fulfilled

$\rightsquigarrow$ maximization with respect to λ_k

$\lambda_k^f \neq 0$ only for values x* for which the equality sign holds in the constraints

Application of the theorem:

$$f_0: \quad \tfrac{1}{2}|\underline{\mathbf{w}}|^2$$

$$f_\alpha: \quad -\Big\{y_T^{(\alpha)}\Big(\underline{\mathbf{w}}^T\underline{\mathbf{x}}^{(\alpha)}+b\Big)-1\Big\} \leq 0$$

Lagrangian:

$$L = \frac{1}{2}|\underline{\mathbf{w}}|^2 - \sum_{\alpha=1}^{p}\lambda_\alpha\Big\{y_T^{(\alpha)}\Big(\underline{\mathbf{w}}^T\underline{\mathbf{x}}^{(\alpha)}+b\Big)-1\Big\} \tag{2.30}$$

$\underline{\mathbf{w}}, b:$	"primal variables"	$\sim$	dimension of feature space
$\lambda_\alpha:$	"dual variables"	$\sim$	number of training data

Finding the saddle point in 2 steps

(a) Minimization w.r.t. $\underline{\mathbf{w}}$

$$\frac{\partial L}{\partial \mathrm{w}_l} = \mathrm{w}_l - \sum_{\alpha=1}^{p}\lambda_\alpha y_T^{(\alpha)}\mathrm{x}_l^{(\alpha)} \overset{!}{=} 0$$

$$\Rightarrow \underbrace{\underline{\mathbf{w}} = \sum_{\alpha=1}^{p}\lambda_\alpha y_T^{(\alpha)}\underline{\mathbf{x}}^{(\alpha)}}_{\substack{\text{expansion of weight vector}\\ \text{in terms of data points}}} \tag{2.31}$$

(b) Minimization w.r.t. b:

$$\frac{\partial L}{\partial b} = -\sum_{\alpha=1}^{p} \lambda_\alpha y_T^{(\alpha)} \overset{!}{=} 0 \tag{2.32}$$

Problem: the λ_α are unknown

$\Rightarrow$ inserting results into L (eq. 2.30):

$$\begin{aligned} L &= \tfrac{1}{2} \sum_{\alpha,\beta=1}^{p} \lambda_\alpha \lambda_\beta y_T^{(\alpha)} y_T^{(\beta)} \left(\underline{\mathbf{x}}^{(\alpha)}\right)^T \underline{\mathbf{x}}^{(\beta)} \\ &\quad - \sum_{\alpha=1}^{p} \lambda_\alpha y_T^{(\alpha)} \sum_{\beta=1}^{p} \lambda_\beta y_T^{(\beta)} \left(\underline{\mathbf{x}}^{(\beta)}\right)^T \underline{\mathbf{x}}^{(\alpha)} \\ &\quad - \underbrace{\left(\sum_{\alpha=1}^{p} \lambda_\alpha y_T^{(\alpha)}\right)}_{=0} b + \sum_{\alpha=1}^{p} \lambda_\alpha \end{aligned} \tag{2.33}$$

$$L = -\tfrac{1}{2} \sum_{\alpha,\beta=1}^{p} \lambda_\alpha \lambda_\beta y_T^{(\alpha)} y_T^{(\beta)} \underbrace{\left(\underline{\mathbf{x}}^{(\alpha)}\right)^T \underline{\mathbf{x}}^{(\beta)}}_{\circledast} + \sum_{\alpha=1}^{p} \lambda_\alpha$$

this dual optimization problem is typically solved using SMO methods (see section 2.2.6, software implementation: `libsvm`):

$$L = \max \lambda_\alpha \tag{2.34}$$

constraints:

$$\lambda_\alpha \geq 0, \alpha = 1, \ldots, p$$

$$\sum_{\alpha=1}^{p} \lambda_\alpha y_T^{(\alpha)} = 0$$

If the Lagrange parameters λ_α are known, the weights are calculated as a linear combination of data points:

$$\underline{\mathbf{w}} = \sum_{\alpha=1}^{p} \lambda_\alpha y_T^{(\alpha)} \underline{\mathbf{x}}^{(\alpha)} \qquad \text{(calculation of } \underline{\mathbf{w}}\text{)}$$

and b can be calculated using the fact that

$$y_T^{(\alpha)} \left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} + b\right) = 1 \text{ for "support vectors"} \tag{2.35}$$

because $\lambda_\alpha \neq 0$ holds only for those data points (i.e. the support vectors)! *(cf. theorem by Kuhn and Tucker)*

$$y_T^{(\alpha)} \sum_{\beta=1}^{p} \lambda_\beta y_T^{(\beta)} \left(\underline{\mathbf{x}}^{(\beta)}\right)^T \underline{\mathbf{x}}^{(\alpha)} + y_T^{(\alpha)} b = 1 \tag{2.36}$$

$$b = y_T^{(\alpha)} - \sum_{\beta=1}^{p} \lambda_\beta y_T^{(\beta)} \left(\underline{\mathbf{x}}^{(\beta)}\right)^T \underline{\mathbf{x}}^{(\alpha)} \tag{2.37}$$

because $\left(y_T^{(\alpha)}\right)^2 = 1$

$$b = \frac{1}{\#\mathrm{SV}} \sum_{\mathrm{SV}} \left(y_T^{(\alpha)} - \sum_{\mathrm{SV}} \lambda_\beta y_T^{(\beta)} \underbrace{\left(\underline{\mathbf{x}}^{(\beta)}\right)^T \underline{\mathbf{x}}^{(\alpha)}}_{\circledast} \right) \qquad \text{(calculation of } b\text{)}$$

Classification:

$$\begin{aligned} y &= \operatorname{sign}\left(\underline{\mathbf{w}}^T \underline{\mathbf{x}} + b\right) \\ &= \operatorname{sign}\left\{ \sum_{\mathrm{SV}} \lambda_\alpha y_T^{(\alpha)} \underbrace{\left(\underline{\mathbf{x}}^{(\alpha)}\right)^T \underline{\mathbf{x}}}_{\circledast} + b \right\} \end{aligned} \tag{2.38}$$

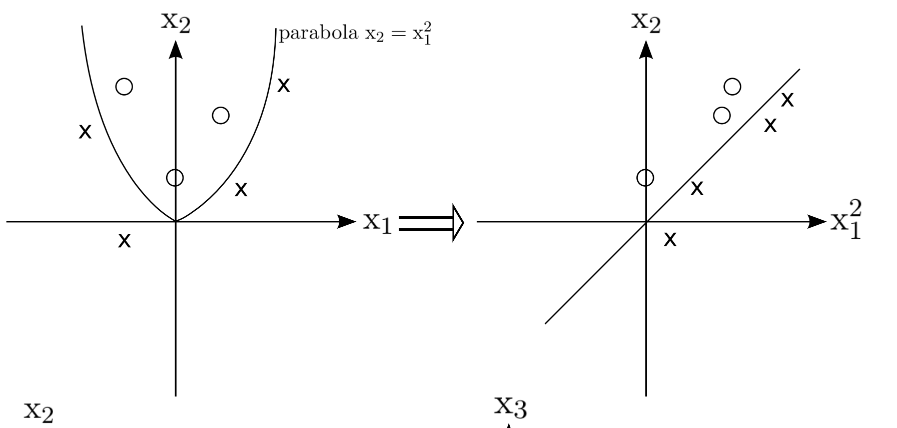
Notes: This classifier, together with aforementioned learning rules for λ_α and b, is called "*support vector machine*".

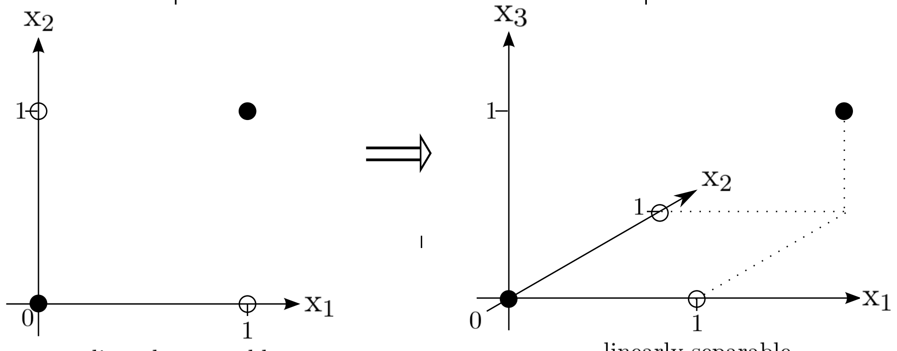
Comments

- only points on the margin are support vectors (i.e. fulfill the equality sign of the constraint) $\rightsquigarrow$ usually only few SVs
- classification boundary has equal distance between examples from class +1 and class -1
- learning and classification depends only on scalar products ($\to$ similarities) between data points ($\circledast$)

2.2.4 SRM Learning for Non-linear Classification Boundaries

Idea: Projection into a new feature space, where decision boundaries are linear:





not linearly separable linearly separable

Two steps

$$\underbrace{\underline{\mathbf{x}}}_{\substack{\text{elementary}\\ \text{features}\\ (\text{in } \mathbb{R}^N)}} \longrightarrow \underbrace{\underline{\phi}_{(\underline{\mathbf{x}})}}_{\substack{\text{new feature space } F\\ \text{(non-linear combination}\\ \text{of elementary features}}}$$ (”intelligent” preprocessing)

Kernel trick

⇒ avoid direct transformation $\underline{\phi}$

⇒ replace all scalar products by ”kernel functions”

$$\underline{\phi}^T_{(\underline{\mathbf{x}})}\underline{\phi}_{(\underline{\mathbf{x}}')} \to K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')}$$

Only scalar products are needed for SVM learning and prediction.

⇒ under which conditions is this possible?

Mercer’s theorem establishes the important link between a positive definite Kernel K and a scalar product in some corresponding metric feature space F. Given the following setting:

$\chi:$	compact subset of $\mathbb{R}^N$
$K : \chi \times \chi \to \mathbb{R}, K \in L_\alpha,$	symmetric function ("kernel")

$$\left.\begin{array}{l} T_k : L_{2(\chi)} \to L_{2(\chi)} \\ \big(T_k f\big)_{(\underline{\mathbf{x}})} := \int\limits_{\chi} K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} f_{(\underline{\mathbf{x}}')} d\underline{\mathbf{x}} \end{array}\right\} \text{corresponding integral operator}$$

$$\left.\begin{array}{ll} \lambda_j : & \text{eigenvalues} \\ \psi_j \in L_{2(\chi)} : & \text{normalized eigenfunctions} \end{array}\right\} \text{of } T_k$$

under the *essential condition* for T_k positive definite, i.e.

$$\int\limits_{\lambda\in\chi} K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} f_{(\underline{\mathbf{x}})} f_{(\underline{\mathbf{x}}')} d\underline{\mathbf{x}} d\underline{\mathbf{x}}' > 0, \forall f \in L_{2(\chi)} \tag{2.39}$$

Mercers theorm then establishes that:

$$\begin{aligned} K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} &= \sum_{j=1}^{n} \underbrace{\lambda_j \psi_{j(\underline{\mathbf{x}})} \psi_{j(\underline{\mathbf{x}}')}}_{\substack{\text{eigenvalue} \\ \text{decomposition}}} \\ n \to \infty &: \underbrace{\text{absolute and uniform convergence}}_{\text{non-trivial part}} \end{aligned} \tag{2.40}$$

Consequences of Mercer's theorem

$$\underline{\phi} : \underline{\mathbf{x}} \to \Big(\sqrt{\lambda_1}\psi_{1(\underline{\mathbf{x}})}, \sqrt{\lambda_2}\psi_{2(\underline{\mathbf{x}})}, \ldots, \sqrt{\lambda_n}\psi_{n(\underline{\mathbf{x}})},\Big)^T \tag{2.41}$$

$$K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} = \underline{\phi}^T_{(\underline{\mathbf{x}})} \underline{\phi}_{(\underline{\mathbf{x}}')} \tag{2.42}$$

Typical kernel functions

$K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} = \big(\underline{\mathbf{x}}^T\underline{\mathbf{x}}' + 1\big)^d$	polynomial kernel of degree d $\to$ image processing: pixel correlations
$K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} = \exp\Big\{-\frac{(\underline{\mathbf{x}}-\underline{\mathbf{x}}')^2}{2\sigma^2}\Big\}$	RBF-kernel with range σ $\to$ infinite dimensional feature space
$K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} = \tanh\big\{\kappa\underline{\mathbf{x}}^T\underline{\mathbf{x}}' + \theta\big\}$	neural network kernel with parameters κ and θ $\to$ not positive definite!
$K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} = \frac{1}{\lvert\underline{\mathbf{x}}-\underline{\mathbf{x}}'+\epsilon\rvert^N}$	Plummer kernel with parameter ϵ $\to$ scale invariant kernel

Hyperparameter selection is typically done via cross-validation methods.

Comments

(1) Kernel trick allows to work in high-dimensional feature space

$$K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} = \left(\underline{\mathbf{x}}^T\underline{\mathbf{x}}' + 1\right)^{10} \rightsquigarrow \text{ space of all monomials up to a degree of 10}$$

(2) SVMs vs. RBF-networks

$$y_{(\underline{\mathbf{x}})} = \operatorname{sign}\left(\sum_i \mathrm{w}_i K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')}\right)$$

$\Rightarrow$ same architecture, but different learning rules
(slide: Schoelkopf / Smola, figs. 7.7 & 7.8)

(3) Mercer's theorem can be used to "kernelize" many different linear methods, both supervised or unsupervised. Important examples:

- Fisher discriminant analysis
- principal component analysis *(see MI 2)*
- K-means clustering & self-organizing maps *(see MI 2)*
- canonical correlation analysis

2.2.5 The C-Support Vector Machine

Many real-world problems are non-separable due to overlapping classes or noise in the observation process. Given such a dataset, we might still find a feature-representation allowing to obtain perfect prediction. Such a solution would, however, not generalize well to new observations and therefore illustrates the need to control overfitting in the SRM-framework. The C-SVM introduces a hyper-parameter C to address this problem.

$\Rightarrow$ empirical cost $R_{\text{emp}}^{(p)} \neq 0$

$\Rightarrow$ trade-off between minimization of $R_{\text{emp}}^{(p)}$ and capacity of the model class

"Primal" problem

$$\frac{1}{2}|\underline{\mathbf{w}}|^2 + \frac{C}{p}\sum_{\alpha=1}^{p}\varphi_\alpha \overset{!}{=} \min \qquad \begin{array}{l}\text{left: minimization of bound on VC dimension}\\ \text{right: minimization of (approx.) margin error}\end{array} \tag{2.43}$$

Constraints:

$$\begin{array}{ll} y_T^{(\alpha)}\left(\underline{\mathbf{w}}^T\underline{\mathbf{x}}^{(\alpha)} + b\right) \geq 1 - \varphi_\alpha & \begin{array}{c}\scriptstyle\text{correct classification of all data points}\\ \scriptstyle\text{with margin } |\underline{\mathbf{w}}| \text{ for } \varphi=0\end{array} \\ \varphi_\alpha \geq 0 & \text{"margin errors" for } \varphi \neq 0 \end{array} \tag{2.44}$$

$\Rightarrow$ exact margin error: $\frac{1}{p}\sum_{\alpha=1}^{p}\theta_{(\varphi_\alpha)} \rightarrow$ NP hard optimization problem

$\Rightarrow$ C: hyperparameter[8] $\rightsquigarrow$ model selection ($\rightsquigarrow$ crossvalidation).

$\Rightarrow$ Why margin error? $\rightsquigarrow$ margin necessary for bounding d_{VC} (i.e. to be smaller than N!)

Dual problem

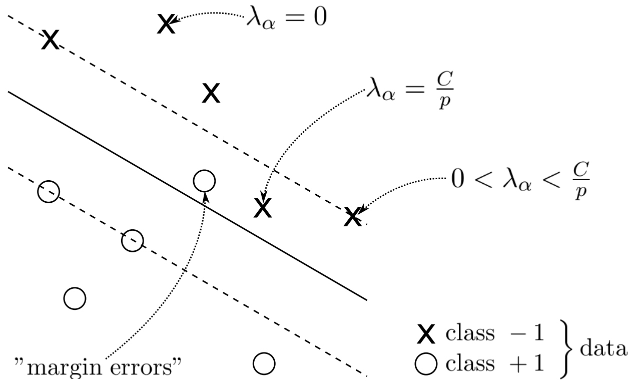
$$-\frac{1}{2}\sum_{\alpha,\beta=1}^{p}\lambda_\alpha\lambda_\beta y_T^{(\alpha)}y_T^{(\beta)}\underbrace{\left(\underline{\mathbf{x}}^{(\alpha)}\right)^T\underline{\mathbf{x}}^{(\beta)}}_{\text{kernel!}}+\sum_{\alpha=1}^{p}\lambda_\alpha \overset{!}{=} \max_{(\lambda_\alpha)} \tag{2.45}$$

Constraints:

$$0 \le \underbrace{\lambda_\alpha \le \frac{C}{p}}_{\substack{\text{difference to linearly}\\ \text{separable case}}} \tag{2.46}$$

$$\sum_{\alpha=1}^{p}\lambda_\alpha y_T^{(\alpha)} = 0$$

(derivation: see supplementary material)



$\lambda_\alpha = 0$
$\lambda_\alpha = \frac{C}{p}$
$0 < \lambda_\alpha < \frac{C}{p}$
"margin errors"
X class -1
○ class $+1$
data

Calculation of $\underline{\mathbf{w}}$ and b:

$$\underline{\mathbf{w}} = \sum_{\alpha=1}^{p}\lambda_\alpha y_T^{(\alpha)}\underline{\mathbf{x}}^{(\alpha)} \rightsquigarrow \lambda_\alpha \neq 0 \text{ only for support vectors} \tag{2.47}$$

let $SV_<$ be the SVs with $\lambda_\alpha < \frac{C}{p}$ (SVs on the margin)

$$b = \frac{1}{\#SV_<}\sum_{SV_<}\left(y_T^{(\alpha)} - \sum_{SV}\lambda_\beta y_T^{(\beta)}\underbrace{\left(\underline{\mathbf{x}}^{(\beta)}\right)^T\underline{\mathbf{x}}^{(\alpha)}}_{\text{kernel!}}\right) \tag{2.48}$$

[8]Note: "C" here refers to a parameter in eq. (2.43), not "capacity" directly, as in figure 35

(via constraints of the primal problem)

Classifier

$$y = \operatorname{sign}\left(\underline{\mathbf{w}}^T \underline{\mathbf{x}} + b\right) = \operatorname{sign}\left(\sum_{SVs} \lambda_\alpha y_T^{(\alpha)} \underbrace{\left(\underline{\mathbf{x}}^{(\alpha)}\right)^T \underline{\mathbf{x}}}_{\text{kernel!}} + b\right) \tag{2.49}$$

2.2.6 Sequential Minimal Optimization

Sequential Minimal Optimization (SMO) is an efficient procedure to solve the dual problem.

The kernel matrix (or "Gram matrix") is defined as

$$K_{\alpha\beta} = K_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{x}}^{(\beta)})} \qquad \text{("Gram matrix")}$$

$$\begin{array}{c|ccccc}
 & 1 & 2 & 3 & \dots & j \\
\hline
1 & K_{11} & K_{12} & \dots & \dots & K_{1j} \\
2 & \vdots & \vdots & K_{23} & \dots & K_{2j} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
i & K_{i1} & K_{i2} & \dots & \dots & K_{ij}
\end{array}$$

and is typically pre-computed to speed up subsequent computations.

$\Rightarrow$ SVMs operate on pairwise (similarity) data!

$\Rightarrow$ positive definite kernel

- $\rightarrow$ positive definite Gram matrix
- $\rightarrow$ well defined optimization problem

Optimization of the Lagrangian $\rightarrow$ iterative procedure

$\rightarrow$ choose two Lagrange multipliers $\lambda_\gamma, \lambda_\delta$

$\rightarrow$ optimize Lagrangian with respect to $\lambda_\gamma, \lambda_\delta$ obeying constraints (keeping all other λ_α fixed)

$\Rightarrow$ optimization with respect to two variables can be done analytically

Selection rules $\rightarrow$ good heuristics are important

Karush-Kuhn-Tucker conditions (KKT conditions)

$$\underbrace{\left[y_T^{(\alpha)}\left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} + b\right) - 1 + \varphi_\alpha\right]}_{\substack{\text{constraint of the primal problem:} \\ \text{=0 for all data points on and} \\ \text{within the margin}}} \underbrace{\lambda_\alpha}_{\substack{\text{Lagrange} \\ \text{parameter} \\ \text{of the} \\ \text{dual problem}}} = 0 \tag{2.50}$$

1. `loop over all` λ_γ `wich violate KKT-conditions`
   (and additional "threshold"-conditions due to errors in determination of b); typically done using the "primal problem" (2.50), i.e. pick λ_α for which 2.50$\neq 0$

2. `for picked` λ_γ `, select` λ_δ `in order to make ''large steps' towards the optimum`
   (general heuristics)

Reduced optimization problem - Lagrangian C-SVM:

$$\frac{1}{2}\sum_{\alpha\beta}\lambda_\alpha\lambda_\beta y_T^{(\alpha)}y_T^{(\beta)}K_{\alpha\beta}-\sum_\alpha\lambda_\alpha \quad \overset{!}{=} \quad \min_{(\lambda_\alpha)}$$

$$\frac{1}{2}\left[\lambda_\gamma^2\underbrace{\left(y_T^{(\gamma)}\right)^2}_{=1}\alpha_{\gamma\gamma}+\lambda_\delta^2\underbrace{\left(y_T^{(\delta)}\right)^2}_{=1}\alpha_{\delta\delta}+2\lambda_\gamma\lambda_\delta y_T^{(\gamma)}y_T^{(\delta)}K_{\gamma\delta}\right]$$

$$+\lambda_\gamma\underbrace{\left[\sum_{\beta\neq\delta}\lambda_\beta y_T^{(\gamma)}y_T^{(\beta)}K_{\gamma\beta}-1\right]}_{C_\gamma}$$

$$+\lambda_\delta\underbrace{\left[\sum_{\beta\neq\gamma}\lambda_\beta y_T^{(\delta)}y_T^{(\beta)}K_{\delta\beta}-1\right]}_{C_\delta}+\text{const}_{(\lambda_\delta,\lambda_\gamma)} \quad \overset{!}{=} \quad \min_{(\lambda_\delta,\lambda_\gamma)}$$

$$\circledast \quad \frac{1}{2}\left[\lambda_\gamma^2Q_{\gamma\gamma}+\lambda_\delta^2Q_{\delta\delta}+2\lambda_\gamma\lambda_\delta\right]+C_\gamma\lambda_\gamma+C_\delta\lambda_\delta \quad \overset{!}{=} \quad \min_{(\lambda_\delta,\lambda_\gamma)} \tag{2.51}$$

Constraints:

$$\circledast \quad 0\leq\lambda_{\gamma,\delta}\leq\frac{C}{p} \qquad \text{"box constraints"}$$

$$\lambda_\gamma+\underbrace{\frac{y_T^{(\delta)}}{y_T^{(\gamma)}}}_{s}\lambda_\delta=\underbrace{\frac{1}{y_T^{(\gamma)}}\sum_{\beta\neq\gamma,\delta}\lambda_\beta y_T^{(\beta)}}_{d} \qquad \text{"equality constraint"} \tag{2.52}$$

$$\circledast \quad \lambda_\gamma+s\lambda_\delta=d$$

(slide: analytical solution of constrained optimization (Schoelkopf / Smola, p. 308))
(supplementary material: Pseudocode (Schoelkopf / Smola, p. 313))

Optimization software: libsvm-package

`http://www.csie.ntu.edu.tw/∼cjlin/libsvm`

⇒ also different variants, multiclass problems, support vector regression, one-class SVMs.

Note on Validation: SVMs implement the SRM principle and therefore favor simple models with a small VC-Dimension. This helps to avoid overfitting. Do we need to validate? Similar to other classification algorithms, SVMs require to make decisions regarding the exact architecture to use. For example, we need to choose shape and parameters of the kernel and how to weight simplicity vs. penalty for boundary intrusions via the parameter C. Typically, these choices are based on available data and may therefore lead to overfitting. Thus they need to be validated, e.g. using (nested) crossvalidation with the 0-1 loss function.

2.3 The P-SVM Ansatz

2.3.1 Shortcomings of Standard SVM-Approaches

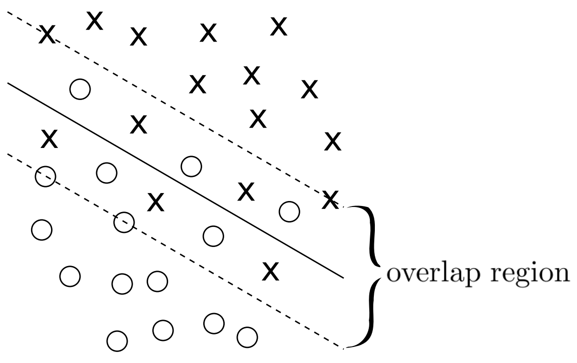
Scale sensitive solutions
(slide: NC 18(6), p. 1472 ff., Fig. 2)

All data points related to margin errors are support vectors

⇒ many support vectors for problems with overlapping classes

⇒ expansion of $\underline{\mathbf{w}}$ is not as "sparse" as it could be (even for the same classification boundary)

2D:



SVM-solution:

$$y_{(\underline{\mathbf{x}})} = \operatorname{sign}\left(\sum_{SVs} \lambda_\alpha y_T^{(\alpha)} \left(\underline{\mathbf{x}}^{(\alpha)}\right)^T \underline{\mathbf{x}} + b \right)$$

but: for 2d and linear classifiers
expansion into two data points would suffice

Only positive definite kernel functions / Gram matrices

⇒ some "interesting" kernels cannot be used
(cf. sine-kernel for periodic distributions)

$$K_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{x}}^{(\beta)})} = \sin\left\{ \mathrm{w} \left| \underline{\mathbf{x}}^{(\alpha)} - \underline{\mathbf{x}}^{(\beta)} \right| \right\}$$

here Gram matrix must have at least some negative eigenvalues (because: $T_\gamma \underline{\mathbf{K}} = 0$)

⇒ structured objects: sometimes hard to assure positive definiteness of relational measure (kernels)

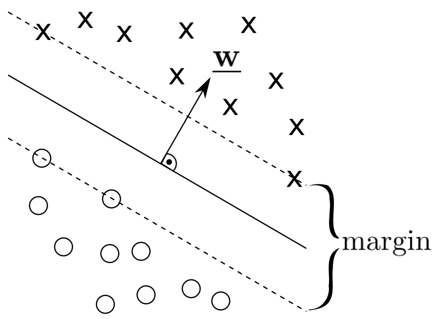
2.3.2 The Primal Optimization Problem

Training data: $\left\{ \left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)} \right) \right\}, \alpha = 1, \ldots, p, \underline{\mathbf{x}} \in \mathbb{R}^N, y_T \in \{-1, +1\}$

Connectionist neurons: $y = \operatorname{sign}\left(\underline{\mathbf{w}}^T \underline{\mathbf{x}} + b \right)$

Capacity measure for the model class: $\mathrm{d}_{\mathrm{VC}} \leq \min\left(\left[\frac{R^2}{d_{\mathrm{w}}^2} \right], N \right) + 1$
(cf. chapter 2.2.2)

① **Scale invariant objective function**



project all data into the subspace (Id) defined by $\underline{\mathbf{w}}$

$\Rightarrow$ smallest possible value for $\frac{R^2}{d_w^2}$

$$R \to \widetilde{R} = \min_{t\in\mathbb{R}} \max_{\alpha} \left| \underline{\widehat{\mathbf{w}}}^T \underline{\mathbf{x}}^{(\alpha)} + t \right|, \underline{\widehat{\mathbf{w}}} = \frac{\underline{\mathbf{w}}}{|\underline{\mathbf{w}}|} \text{ unit vector} \tag{2.53}$$

$$\begin{aligned} \frac{\widetilde{R}^2}{d_w^2} &= \underline{\mathbf{w}}^2 \left\{ \min_{t\in\mathbb{R}} \max_{\alpha} \left(\underline{\widehat{\mathbf{w}}}^T \underline{\mathbf{x}}^{(\alpha)} + t \right)^2 \right\} && \leq \underline{\mathbf{w}}^2 \left\{ \max_{\alpha} \left(\underline{\widehat{\mathbf{w}}}^T \underline{\mathbf{x}}^{(\alpha)} \right)^2 \right\} \\ &= \max_{\alpha} \left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} \right)^2 && \leq \sum_{\alpha=1}^{p} \left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} \right)^2 \end{aligned} \tag{2.54}$$

New objective function:

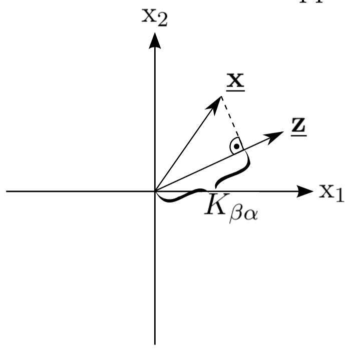
$$\sum_{\alpha=1}^{p} \left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} \right)^2 = \sum_{\alpha=1}^{p} \sum_{i,j=1}^{N} \mathrm{w}_i \mathrm{x}_i^{(\alpha)} \mathrm{w}_j \mathrm{x}_j^{(\alpha)} = \sum_{i,j=1}^{N} \mathrm{w}_i \underbrace{\left(\sum_{\alpha=1}^{p} \mathrm{x}_i^{(\alpha)} \mathrm{x}_j^{(\alpha)} \right)}_{\substack{p \cdot C_{ij} \\ \text{correlation matrix} \\ \text{of data}}} \mathrm{w}_j \tag{2.55}$$

Matrix notation:

$$\left| \underline{\mathbf{X}}^T \underline{\mathbf{w}} \right|^2 = \underline{\mathbf{w}}^T \underline{\mathbf{X}} \underline{\mathbf{X}}^T \underline{\mathbf{w}} = p \cdot \underline{\mathbf{w}}^T \underline{\mathbf{C}} \underline{\mathbf{w}}, \underline{\mathbf{X}} = \left(\underline{\mathbf{x}}^{(1)}, \underline{\mathbf{x}}^{(2)}, \ldots, \underline{\mathbf{x}}^{(p)} \right) \tag{2.56}$$

Objective function is equivalent to the "old" SVM objective $|\underline{\mathbf{w}}|^2$ if $\underline{\mathbf{C}} = \underline{\mathbf{1}}$ (i.e. for sphered data)

② **New constraints** $\rightsquigarrow$ applicability to general dyadic data



- selected complex features: $\left\{\underline{\mathbf{z}}^{(\beta)}\right\}, \beta = 1, \dots, q$
- measurement of feature values: $\left(\underline{\mathbf{z}}^{(\beta)}\right)^T \underline{\mathbf{x}}^{(\alpha)} := K_{\beta\alpha}$
  $\rightsquigarrow q \times p$ matrix[9]

Quadratic cost function:

$$R_{\text{emp}}^{(p)} = \frac{1}{2p} \sum_{\alpha=1}^{p} \left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} + b - y_T^{(\alpha)} \right)^2 \tag{2.57}$$

$\Rightarrow$ actually more natural for regression ratio than classification problems

Constraints: minimal empirical error along the selected complex feature

$$\left(\underline{\mathbf{z}}^{(\beta)}\right)^T \frac{\partial R_{\text{emp}}^{(p)}}{\partial \underline{\mathbf{w}}} = \frac{1}{p} \sum_{\alpha=1}^{p} \underbrace{\left(\underline{\mathbf{z}}^{(\beta)}\right)^T \underline{\mathbf{x}}^{(\alpha)}}_{K_{\beta\alpha}} \left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} + b - y_T^{(\alpha)} \right) \overset{!}{=} 0 \tag{2.58}$$

$\Rightarrow$ one constraint per feature

$$\frac{\partial}{\partial b} R_{\text{emp}}^{(p)} = \frac{1}{p} \sum_{\alpha=1}^{p} \left(\underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} + b - y_T^{(\alpha)} \right) \overset{!}{=} 0 \tag{2.59}$$

$$\rightsquigarrow b = \frac{1}{p} \sum_{\alpha=1}^{p} \left(y_T^{(\alpha)} - \underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} \right) \tag{2.60}$$

Normalization of $K_{\beta\alpha}$:

$$\begin{aligned} &\tfrac{1}{p} \sum_{\alpha=1}^{p} K_{\beta\alpha} = 0 \quad \text{zero mean} \qquad \text{because} \left(\tfrac{1}{p} \sum_{\alpha=1}^{p} K_{\beta\alpha} \right) b = 0 \\ &\tfrac{1}{p} \sum_{\alpha=1}^{p} K_{\beta\alpha}^2 = 1 \quad \text{unit variance} \end{aligned} \tag{2.61}$$

Primal problem of the P-SVM

$$\frac{1}{2} \left| \underline{\mathbf{X}}^T \underline{\mathbf{w}} \right| \overset{!}{=} \min \tag{2.62}$$

Constraints:

$$\underline{\mathbf{K}} \left\{ \underline{\mathbf{X}}^T \underline{\mathbf{w}} - \underline{\mathbf{y}}_T \right\} \overset{!}{=} 0 \tag{2.63}$$

Offset b is given by:

$$b = \frac{1}{p} \sum_{\alpha=1}^{p} \left(y_T^{(\alpha)} - \underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} \right) \tag{2.64}$$

[9] Please note that this definition of K corresponds to K^T in J. Hochreiter and Obermayer (2006)

but: if $\underline{\mathbf{K}} = \left(K_{\beta\alpha}\right)$ has at least rank p (no. of complex features equal or larger than number of data points), then constraints are always fulfilled with zero empirical error

$\Rightarrow$ overfitting

2.3.3 Regularization

Trade-off between violation of constraints and minimization of capacity measure.

Primal problem:

$$\frac{1}{2}\left|\underline{\mathbf{X}}^T\underline{\mathbf{w}}\right|^2 + C\underline{\mathbf{1}}^T\left(\underline{\varphi}^+ + \underline{\varphi}^-\right) \overset{!}{=} \min_{\left(\underline{\mathbf{w}},\underline{\varphi}^+,\underline{\varphi}^-\right)} \tag{2.65}$$

Constraints:

$$\begin{aligned} \underline{\mathbf{K}}\Big(\underline{\mathbf{X}}^T\underline{\mathbf{w}} - \underline{\mathbf{y}}_T\Big) + \underline{\varphi}^+ + \epsilon\underline{\mathbf{1}} &\geq 0 \\ \underline{\mathbf{K}}\Big(\underline{\mathbf{X}}^T\underline{\mathbf{w}} - \underline{\mathbf{y}}_T\Big) - \underline{\varphi}^- - \epsilon\underline{\mathbf{1}} &\leq 0 \\ \underline{\varphi}^+, \underline{\varphi}^- &\geq 0 \end{aligned} \tag{2.66}$$

C-regularization

- $\rightsquigarrow$ violation of constraints for individual features ("outliers")
- $\rightsquigarrow$ large values for φ_β^+ and φ_β^- $\Rightarrow$ corresponding features influence the classification boundary only weakly

ϵ-regularization

- $\rightsquigarrow$ deviations from the optimal residual error are tolerated if small enough ("small" determined by value of ϵ)
- $\rightsquigarrow$ these features then do not influence the classification boundary

2.3.4 Dual Formulation and P-SVM Classifier

Lagrangian:

$$\begin{aligned} L \;=\; & \tfrac{1}{2}\underline{\mathbf{w}}^T\underline{\mathbf{X}}\underline{\mathbf{X}}^T\underline{\mathbf{w}} + C\underline{\mathbf{1}}^T\left(\underline{\varphi}^+ + \underline{\varphi}^-\right) && \text{cost function} \\ & \left.\begin{aligned} &-\left(\underline{\lambda}^+\right)^T\Big\{\underline{\mathbf{K}}\left(\underline{\mathbf{X}}^T\underline{\mathbf{w}} - \underline{\mathbf{y}}_T\right) + \underline{\varphi}^+ + \epsilon\underline{\mathbf{1}}\Big\} \\ &+\left(\underline{\lambda}^-\right)^T\Big\{\underline{\mathbf{K}}\left(\underline{\mathbf{X}}^T\underline{\mathbf{w}} - \underline{\mathbf{y}}_T\right) + \underline{\varphi}^- - \epsilon\underline{\mathbf{1}}\Big\} \end{aligned}\right\} && \text{constraints for complex features} \\ & -\left(\underline{\mu}^+\right)^T\underline{\varphi}^+ - \left(\underline{\mu}^-\right)^T\underline{\varphi}^- && \text{positivity of slack variables} \end{aligned} \tag{2.67}$$

Derivatives:

$$\frac{\partial L}{\partial \underline{\mathbf{w}}} = \underline{\mathbf{w}}^T \underline{\mathbf{X}}\underline{\mathbf{X}}^T - \left(\underline{\lambda}^+\right)^T \underline{\mathbf{K}}\underline{\mathbf{X}}^T + \left(\underline{\lambda}^-\right)^T \underline{\mathbf{K}}\underline{\mathbf{X}}^T \overset{!}{=} 0 \tag{2.68}$$

$$\underline{\mathbf{w}}^T \underline{\mathbf{X}} = \left(\underline{\lambda}^+ - \underline{\lambda}^-\right)\underline{\mathbf{K}} = \left(\underline{\lambda}^+ - \underline{\lambda}^-\right)\underline{\mathbf{Z}}^T \underline{\mathbf{X}} \tag{2.69}$$

Expansion of weight vector into ”support features” rather than ”support data”.

$$\begin{aligned} \tfrac{\partial L}{\partial \underline{\varphi}^+} &= C\underline{\mathbf{1}} - \underline{\lambda}^+ - \underline{\mu}^+ \quad \overset{!}{=} \underline{\mathbf{0}} \quad \rightsquigarrow \quad \underline{\mu}^+ = C\underline{\mathbf{1}} - \underline{\lambda}^+ \\ \tfrac{\partial L}{\partial \underline{\varphi}^-} &= -C\underline{\mathbf{1}} + \underline{\lambda}^- + \underline{\mu}^- \quad \overset{!}{=} \underline{\mathbf{0}} \quad \rightsquigarrow \quad \underline{\mu}^- = C\underline{\mathbf{1}} - \underline{\lambda}^- \end{aligned} \tag{2.70}$$

Derivation of the dual[10] - inserting above equations into L:

$$\begin{aligned} L \;=\; & \tfrac{1}{2}\left(\underline{\lambda}^+ - \underline{\lambda}^-\right)^T \underline{\mathbf{Z}}^T \underline{\mathbf{X}}\underline{\mathbf{X}}^T \underline{\mathbf{Z}}\left(\underline{\lambda}^+ - \underline{\lambda}^-\right) + C\underline{\mathbf{1}}^T\left(\underline{\varphi}^+ + \underline{\varphi}^-\right) \\ & -\left(\underline{\lambda}^+\right)^T\left\{\underline{\mathbf{K}}\left(\underline{\mathbf{X}}^T\underline{\mathbf{Z}}\left(\underline{\lambda}^+ - \underline{\lambda}^-\right) - \underline{\mathbf{y}}_T\right) + \underline{\varphi}^+ + \epsilon\underline{\mathbf{1}}\right\} \\ & +\left(\underline{\lambda}^-\right)^T\left\{\underline{\mathbf{K}}\left(\underline{\mathbf{X}}^T\underline{\mathbf{Z}}\left(\underline{\lambda}^+ - \underline{\lambda}^-\right) - \underline{\mathbf{y}}_T\right) - \underline{\varphi}^- - \epsilon\underline{\mathbf{1}}\right\} \\ & \left(C\underline{\mathbf{1}} - \underline{\lambda}^+\right)^T \underline{\varphi}^+ - \left(C\underline{\mathbf{1}} - \underline{\lambda}^-\right)\underline{\varphi}^- \\ \;=\; & -\tfrac{1}{2}\left(\underline{\lambda}^+ - \underline{\lambda}^-\right)^T \underline{\mathbf{K}}\underline{\mathbf{K}}^T\left(\underline{\lambda}^+ - \underline{\lambda}^-\right) + \left(\underline{\lambda}^+ - \underline{\lambda}^-\right)\underline{\mathbf{K}}\underline{\mathbf{y}}_T \\ & -\epsilon\left(\underline{\lambda}^+ - \underline{\lambda}^-\right)^T \underline{\mathbf{1}} \overset{!}{=} \max_{(\underline{\lambda}^+,\underline{\lambda}^-)} \end{aligned} \tag{2.71}$$

$$\frac{1}{2}\left(\underline{\lambda}^+ - \underline{\lambda}^-\right)^T \underline{\mathbf{K}}\underline{\mathbf{K}}^T\left(\underline{\lambda}^+ - \underline{\lambda}^-\right) - \underline{\mathbf{y}}_T^T \underline{\mathbf{K}}^T\left(\underline{\lambda}^+ - \underline{\lambda}^-\right) + \epsilon\underline{\mathbf{1}}^T\left(\underline{\lambda}^+ - \underline{\lambda}^-\right) \overset{!}{=} \min_{(\underline{\lambda}^+,\underline{\lambda}^-)} \tag{2.72}$$

$\Rightarrow$ convex optimization problem for all $\underline{\mathbf{K}}$

$\rightsquigarrow$ $\underline{\mathbf{K}}\underline{\mathbf{K}}^T$ is always positive (semi-)definite

$\Rightarrow$ $\epsilon\underline{\mathbf{1}}^T\left(\underline{\lambda}^+ - \underline{\lambda}^-\right)$: sparseness constraint for the Lagrange multipliers

Evaluating the constraints we obtain (”box constraints”):

$$\begin{aligned} \underline{\mathbf{0}} \leq \underline{\lambda}^+ \leq C\underline{\mathbf{1}}, \quad & \text{because } \underline{\mu}^+ \geq \underline{\mathbf{0}} \quad \rightsquigarrow C\underline{\mathbf{1}} - \underline{\lambda}^+ \geq \underline{\mathbf{0}} \\ \underline{\mathbf{0}} \leq \underline{\lambda}^- \leq C\underline{\mathbf{1}}, \quad & \text{because } \underline{\mu}^- \geq \underline{\mathbf{0}} \quad \rightsquigarrow C\underline{\mathbf{1}} - \underline{\lambda}^- \geq \underline{\mathbf{0}} \end{aligned} \tag{2.73}$$

Optimization: adapted SMO method

[10]Please note that here K corresponds to K^T in J. Hochreiter and Obermayer (2006)

`http://ni.cs.tu-berlin.de/software/psvm/index.html`

P-SVM classifier

$$\begin{aligned} y &= \operatorname{sign}\left(\underline{\mathbf{w}}^T \underline{\mathbf{x}} + b\right) \\ &= \operatorname{sign}\left\{\left(\underline{\lambda}^+ - \underline{\lambda}^-\right)^T \underline{\mathbf{Z}}^T \underline{\mathbf{x}} + b\right\} \end{aligned} \tag{2.74}$$

$$y_{\mathrm{x}} = \operatorname{sign}\left\{ \sum_{\beta \in \mathrm{SV}^+} K_{\beta \mathrm{x}} \lambda_\beta^+ - \sum_{\beta \in \mathrm{SV}^-} K_{\beta \mathrm{x}} \lambda_\beta^- + b \right\} \tag{2.75}$$

where:

$$\begin{aligned} b &= \tfrac{1}{p} \sum_{\alpha=1}^{p} \left(y_T^{(\alpha)} - \underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(\alpha)} \right) \\ &= \tfrac{1}{p} \sum_{\alpha=1}^{p} \left(y_T^{(\alpha)} - \sum_{\beta=1}^{q} \left(\lambda_\beta^+ - \lambda_\beta^-\right) \left(\underline{\mathbf{Z}}^{(\beta)}\right)^T \underline{\mathbf{x}}^{(\alpha)} \right) \\ &= \underbrace{\frac{1}{p} \sum_{\alpha=1}^{p}}_{\overset{!}{=}0} \left(y_T^{(\alpha)} - \sum_{\beta=1}^{q} \left(\lambda_\beta^+ - \lambda_\beta^-\right) \underbrace{K_{\beta\alpha}}_{\overset{!}{=}0} \right) \\ &= \tfrac{1}{p} \sum_{\alpha=1}^{p} y_T^{(\alpha)} \end{aligned} \tag{2.76}$$

2.3.5 The Kernel Trick

Under fairly mild assumptions *(cf. Hochreiter & Obermayer(2006), New. Comp. 18, 1427ff, Appendix)* one obtains:

$$\begin{aligned} \underline{\psi} :\ & \underbrace{\mathrm{z}}_{\substack{\text{from a} \\ \text{Hilbert space}}} \rightarrow \underbrace{\underline{\psi}_{(\mathrm{z})}}_{\text{from } l^2} \\ \underline{\phi} :\ & \underbrace{\mathrm{x}}_{\substack{\text{from a} \\ \text{Hilbert space}}} \rightarrow \underbrace{\underline{\phi}_{(\mathrm{x})}}_{\text{from } l^2} \\ K :\ & \mathrm{z}, \mathrm{x} \rightarrow K_{(\mathrm{z},\mathrm{x})} \end{aligned} \tag{2.77}$$

then:

$$\underbrace{K_{(\mathrm{x},\mathrm{z})}}_{\text{kernel}} = \underbrace{\underline{\psi}_{(\mathrm{z})}^T \underline{\phi}_{(\mathrm{x})}}_{\text{scalar product}} \tag{2.78}$$

2.3.6 Properties of the P-SVM

General dyadic data:

	objects (attributes to be predicted)
descriptions	Gram matrix

examples:	objects		descriptions
	$\Downarrow$		$\Downarrow$
DNA microarrays	probes	vs.	genes
documents	text	vs.	indexwords
web-pages	page	vs.	links

$\Rightarrow$ arbitrary rectangular Gram matrices

$\Rightarrow$ non-definite (square) matrices

$\Rightarrow$ Gram matrix measured or constructed

slide: NC paper, Tab.3

Standard metric data: indefinite kernels:
e.g.: sine-kernel

$$K_{(\underline{\mathbf{x}},\underline{\mathbf{x}}')} = \sin\left\{\theta\left|\underline{\mathbf{x}} - \underline{\mathbf{x}}'\right|\right\}$$

slide: NC paper, Fig. 4

Feature selection:

$\Rightarrow$ "support features" are important for prediction

$\Rightarrow$ sparse set of support features (determined via regularization parameter ϵ)

$\Rightarrow$ binary classification problem

$$\begin{array}{rl} \lambda_\beta^+ \neq 0 : & \text{features supporting class "} + 1\text{"} \\ \lambda_\beta^- \neq 0 : & \text{features supporting class "} - 1\text{"} \end{array}$$

$\Rightarrow$ Lagrange parameters are directly related to the increase in empirical error when one feature is removed (*proof: see supplementary material*)

$$\Delta_\gamma := R^{(p)}_{\text{emp}[\underline{\mathbf{w}} - \lambda_\gamma \underline{\mathbf{z}}^{(\gamma)}, b]} - R^{(p)}_{\text{emp}[\underline{\mathbf{w}}, b]} \leq \frac{\epsilon |\lambda_\gamma|}{p} + \frac{\lambda_\gamma^2}{2}$$

$\Rightarrow$ P-SVM is an example of a wrapper method for feature selection
slide: NC paper, Tab. 7

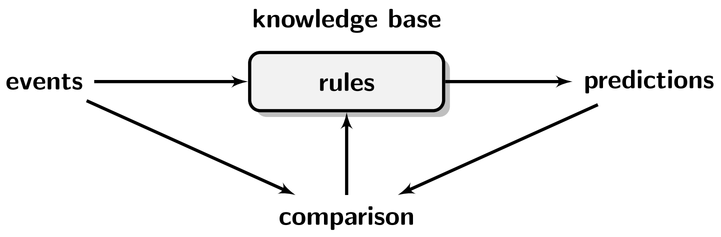
3 Probabilistic Methods I: Bayesian Inference

This chapter introduces a probabilistic approach to describe and exploit the structure of stochastic relationships between multiple variables. The general idea is to find a formally consistent description of the world allowing to describe knowledge in terms of "degrees of belief". In this way, all knowledge can be stored in a *knowledge base* by assigning a degree of belief to each *atomic event*. Knowledge represented in this way can then be exploited to *infer* unobserved quantities, i.e. to reason about the probability of making certain observations.

3.1 Uncertainty and Inference

3.1.1 Degrees of Belief

Knowledge can be stored in form of rules. Using deductive reasoning and logic ($\rightarrow$ true or false) we can apply those rules to observed data to infer further information beyond what is observed.



But: Does this really work in the real world?

Diagnosis example: toothache (using first order logic)

(1) $\forall p \; symptom(p, toothache) \Rightarrow disease(p, cavity)$
$\rightsquigarrow$ rule is wrong: not all patients with toothache have cavities (gum disease, abscess, ...)

(2) $\forall p \; symptom(p, toothache) \Rightarrow$
$disease(p, cavity) \vee disease(p, gumdisease) \vee disease(p, abscess) \vee \ldots$
$\rightsquigarrow$ almost unlimited list of possible causes

(3) diagnostic rule to causal rule
$\forall p \; disease(p, cavity) \Rightarrow symptom(p, toothache)$
$\rightsquigarrow$ rule is wrong: not all cavities cause pain

$\Rightarrow$ First order logic fails in many situations! Examples:

- complete set of antecedents and consequences too large
- no complete theory for the domain

- incomplete observations
- stochastic environments

$\Rightarrow$ How can we deal with the fact that we are almost never 100% sure about our rules?

Degrees of belief
Proposition H: Patient p has a cavity.

$P(H) : H \to [0,1]$	assignment of numbers
$P(H) = 0$	H is false
$P(H) = 1$	H is true
$0 < P(H) < 1$	quantifies our degree of belief

Formal treatment: Assume $P(H)$ to obey the laws of probability theory

$\leadsto$ "probabilities"

$\leadsto$ but: no justification via repeated observations and stochastic outcome

(for detailed treatment, see Jaynes and Bretthorst, 2003)

Application to Betting agents: (de Finetti, 1931)
"If agent 1 expresses a set of degrees of belief that violate the axioms of probability theory, then there is a combination of bets by agent 2 that guarantees that agent 1 will lose money all the time."

Example: Russell and Norvig (2003, p.474) □ betting agents

3.1.2 The Description of the World

We start with a "sufficiently complete" set of random variables to describe the state of the world.

Random variables and propositions

Random variable: a "part" of the world whose "status" is initially unknown
$\leadsto X, Y, \ldots$

Domain of a random variable: "values" the variable can take on $\leadsto x, y, \ldots$

Examples
Boolean variables:
 variable: $cavity$; domain: $\{true, false\}$
 proposition: $cavity = true$
Discrete ordinal variables:
 variable: $weather$; domain: $\{sunny, rainy, cloudy, snow\}$
 proposition: $weather = sunny$
Continuous variables:
 variable: $temperature$; domain: $\mathbb{R}_0^+$
 proposition: $temperature \in [290K, 291K]$

Atomic events
Description of the "world": complete set of random variables
 e.g. $cavity$, $toothache$, $weather$

Atomic event: complete specification of the state of the world
 e.g. $cavity = true \wedge toothache = false \wedge weather = sunny$

$\leadsto$ atomic events are mutually exclusive

$\leadsto$ set of atomic events must be exhaustive

$\leadsto$ proposition $\widehat{=}$ disjunction of atomic events

$cavity = true$ is equivalent to:
 $(cavity = true \wedge toothache = false \wedge weather = sunny) \vee$
 $(cavity = true \wedge toothache = true \wedge weather = rainy) \vee \ldots$

Prior (unconditional) probabilities
Specification of one's knowledge about the world - in the absence of any other information
 e.g. $P(cavity = true) = 0.1$
  $P(cavity = true, toothache = false, weather = snow) = 0.02$

Complete specification of domain knowledge: table of degrees of belief for all atomic events
 e.g. $P(cavity, toothache, weather)$

Conditional probabilities
Specification of one's knowledge about the world - given a set of observations (the "evidence").

$$P(cavity = true | toothache = true) = 0.8$$
$$P(\underbrace{cavity = false}_{\text{proposition}} | \underbrace{toothache = true}_{\substack{\text{observation /} \\ \text{evidence}}}) = 0.2$$

$\Rightarrow$ conditional probability

$$P(\underbrace{C}_{\text{variable}} | \underbrace{t}_{\text{value}}) = \frac{\overbrace{P(C,t)}^{\substack{\text{joint}\\ \text{probability}}}}{P(t)} \tag{3.1}$$

$$P(C,t) = P(C|t)P(t) \qquad \text{(product rule)}$$

$P(C|t)$: degree of belief in C, given <u>all</u> we know is t.

3.1.3 Probabilistic Inference Using Joint Probabilities

Knowledge base: $P(x, y, \ldots)$ degrees of belief for all atomic events

What is the probability of a proposition given a set of observations?

query variable:	X	$\leftarrow$ probability of values to be inferred
evidence variable:	E_j	$\leftarrow$ observed variables
unobserved variables:	Y_j	"hidden" or "nuisance" variables

$$P(x|\underline{\mathbf{e}}) = \underbrace{\frac{P(x,\underline{\mathbf{e}})}{P(\underline{\mathbf{e}})}}_{\substack{\text{def. of cond.}\\ \text{probability}}} = \underbrace{\alpha P(x,\underline{\mathbf{e}})}_{\text{normalization}} = \underbrace{\alpha \sum_{\underline{\mathbf{y}}} P(x,\underline{\mathbf{e}},\underline{\mathbf{y}})}_{\text{marginalization}} \tag{3.2}$$

$\frac{1}{\alpha} = P(\underline{\mathbf{e}}) = \sum_{x,\underline{\mathbf{y}}} P(x,\underline{\mathbf{e}},\underline{\mathbf{y}})$ involves a sum over many different combinations of x and $\underline{\mathbf{y}}$ which can be costly to evaluate. However, this does not have to be computed explicitly, because:

$$\sum_x P(x|\underline{\mathbf{e}}) \overset{!}{=} 1 \qquad \text{(normalization of probabilities)}$$

This only uses knowledge $P(x,\underline{\mathbf{e}},\underline{\mathbf{y}})$ stored in the knowledge-base, the *product rule* and normalization (for an example, see Russell and Norvig, 2003, pp. 475).

□marginalisation

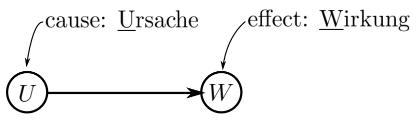
Note: If we have the complete probability table for all atomic events, we have all the relevant information to do inference.

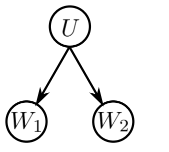
Problem: this ansatz does not scale! Assume N binary random variables:

$\rightsquigarrow$ table of joint probabilities has 2^N entries

$\rightsquigarrow$ summation over approx. 2^N entries for inference
$N = 100 \rightsquigarrow 2^N \approx 1.3 \cdot 10^{30} \rightsquigarrow$ additional assertion needed

3.1.4 Conditional Independence





Cause and effect: $P(W|U) \rightarrow$ "causal rule"
One cause and two effects: $P(W_1, W_2|U) = P(W_1|U)P(W_2|U)$

Example: $P(\text{toothache}, \text{catch}|\text{cavity}) = P(\text{toothache}|\text{cavity})P(\text{catch}|\text{cavity})$

Definition of conditional independence
Two random variables X and Y are conditionally independent given Z if:

$$P(X, Y|Z) = P(X|Z)P(Y|Z) \tag{3.3}$$

and we write $X \perp Y|Z$.

Note: Conditional independence is not independence:
$\Rightarrow$ X and Y are typically not independent.

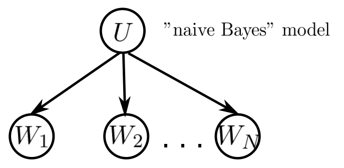
Note: Conditional independence assertions are an important step towards efficient inference algorithms: they enable a *decomposition* of the knowledge base. To illustrate this fact, consider a set of binary random variables: $w_1, w_2, \dots, w_{N-1}, U$.

Assuming conditional independence of effects given the cause:

$$\underbrace{\underbrace{P(w_1, w_2, \dots, w_{N-1}, U)}_{2^N-1 \text{ table entries}} = \underbrace{P(U) \prod_{i=1}^{N-1} P(w_i|U)}_{1+2(N-1)=2N-1 \text{ table entries}}}_{\text{probabilities sum to one}} \tag{3.4}$$

$\Rightarrow$ greatly reduces computational burden to determine joint distribution from $O(2^N)$ to $O(N)$: this would solve the scaling problem!

This approach is called "naive Bayes" can even be useful in situations where conditional independence does *not* strictly hold.



$\Rightarrow$ use conditional probabilities as part of the model base

3.1.5 Bayes' Theorem

Common inference task: Extract the causes underlying observations!

causal rule: $P(W|U)P(U)$

$$\begin{tikzcd} \boxed{U} \arrow[r] & \boxed{W} \end{tikzcd}$$

diagnostic rule: $P(U|W)P(W)$

$$\begin{tikzcd} \boxed{W} \arrow[r] & \boxed{U} \end{tikzcd}$$

Note: In this graph, arrows do not represent causation but statistical dependency.

Approach 1: causal knowledge base → diagnostic rule

More generally, the joint probability $P(W, U)$ can be written as:

$$P(W|U)P(U) = P(W,U) = P(U|W)P(W) \qquad \text{(Product rule)}$$

This can be used to infer $P(U|W)$ given $P(W|U)$ and $P(U)$ are known:

$$P(U|W) = \frac{P(W|U)P(U)}{P(W)} = \alpha\, P(W|U)P(U) \qquad \text{(Bayes' Theorem)}$$

Comment 1: Bayes' theorem holds for arbitrary random variables. "Causes" and "effects" were only used for illustration.

Approach 2: diagnostic knowledge base

Advantage: no calculations necessary to determine $P(U|W)$

Comment 2: Causal rules vs. diagnostic rules

$$\left.\begin{array}{ll} \text{M:} & \text{meningitis} \\ \text{S:} & \text{stiff neck} \end{array}\right\} P(M|S)$$

Diagnostic knowledge base: $P(M|S)$ is directly stored. No computation

causal knowledge base: $P(M|S) = \underbrace{\alpha P(S|M)P(M)}_{\text{Bayes' theorem}}$

→ constructed from hospital records

Problem: Consider a sudden epidemic of meningitis: $P(M) \uparrow$

$\rightsquigarrow$ diagnostic knowledge base $p(M|S)$ cannot be updated easily

3.2 Bayesian Networks

Bayesian network $\left\{ \begin{array}{l} \text{- representation of the joint probability distribution} \\ \text{- encoding of a collection of conditional independence statements} \end{array} \right.$

Directed Acyclic Graphs (DAG)

- set of random variables

  $\rightsquigarrow$ nodes of the graph

- direct influence between variables (e.g. causal relationships)

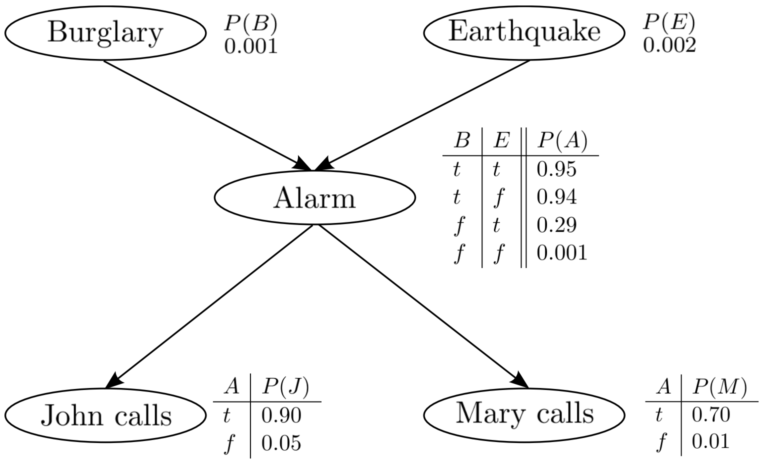
  $\rightsquigarrow$ directed links between nodes

- nodes are annotated with conditional probability distributions

$$P(x_i|parents(x_i))$$

For details, see Russell and Norvig (2003, ch. 14: Probabilistic Reasoning).

Example: Burglary detection in california: What is the probability of a burglary given that John and/or Mary call?



B	E	$P(A)$
t	t	0.95
t	f	0.94
f	t	0.29
f	f	0.001

A	$P(J)$
t	0.90
f	0.05

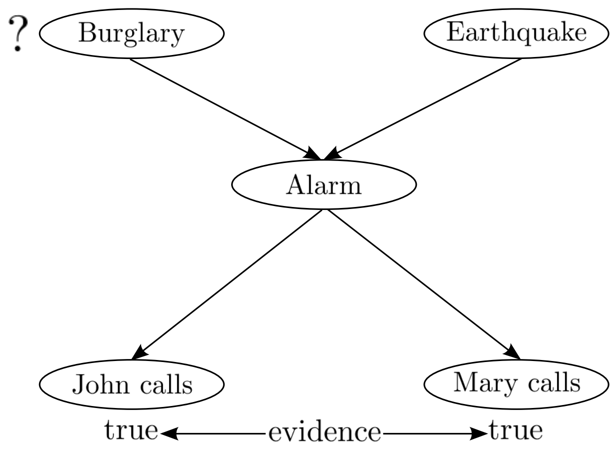
A	$P(M)$
t	0.70
f	0.01

$$P(J, M, A, B, E) = P(J|A)P(M|A)P(A|B, E)P(B)P(E) \tag{3.5}$$

Bayesian network:

$$\begin{array}{c}\text{directed acyclic graph} \\ \text{annotated nodes}\end{array} \longleftrightarrow P(x_1, \ldots, x_N) = \prod_{i=1}^{n} P(x_i|\text{parents}(x_i))$$

Inference: What is the probability of burglary - given that both Mary and John call?



To determine this probability, we need to marginalize over the two nuisance variables `Alarm` (a) and `Earthquake` (e):

$$
\begin{aligned}
P(B|J=true, M=true) &= \underbrace{\alpha P(B, J=true, M=true)}_{\substack{\text{normalization}\\ \text{c.f. section 3.1.3}}} \\
&= \underbrace{\alpha \sum_{a,e} P(B, e, a, J=true, M=true)}_{\text{marginalization}} \\
&= \alpha P(B) \sum_e P(e) \sum_a P(a|B,e) \\
&\quad \cdot P(J=true|a) P(M=true|a) \\
&= \alpha \cdot \begin{cases} 0.00059224 & \text{for } B=true \\ 0.0014919 & \text{for } B=false \end{cases}
\end{aligned} \tag{3.6}
$$

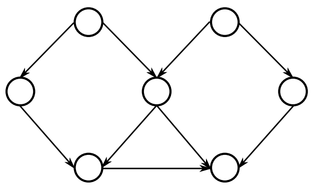
Choose α, such that sum is 1:

$$
P(B|J=true, M=true) = \begin{cases} 0.284 & \text{for } B=true & 0.001 \\ \underbrace{0.716}_{\substack{\text{evidence changed}\\ \text{our assessment}}} & \text{for } B=false & \underbrace{0.999}_{\text{Prior } P(B)} \end{cases} \tag{3.7}
$$

Question: How can we efficiently implement inference in larger and more complex Bayesian networks?

3.2.1 Directed Acyclic Graphs

Directed Acyclic Graphs (DAG's) provide useful graph structures to represent probabilistic knowlege bases (cf. feedforward neural networks).



Graph $G = (V, K)$

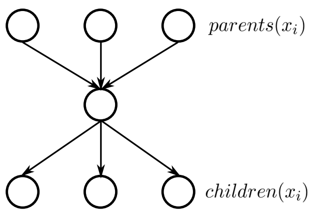
$V \rightarrow$ set of nodes

$K \rightarrow$ set of links

path: sequence $\{x_i\}, i = 1, \dots, n$ of different nodes $x_i \in V$, such that $(x_i, x_{i+1}) \in K$

cycle: path with property $x_1 = x_{n+1}$

parents and children of nodes:



DAGs and distributions: Graphical models provide a link between graph structures and probability distributions. This link allows to use results from graph-theory to implement efficient inference algorithms.

Note: Every DAG corresponds to a factorization of a joint PDF!

$$\begin{array}{ccc} P(x_1, \dots, x_n) & = & \prod_{i=1}^{n} P(x_i|parents(x_i)) \\ \text{"causal flow"} & \longleftrightarrow & \text{well ordered nodes} \end{array} \tag{3.8}$$

useful for construction of compact networks

index of a node x_i should always be larger than all indices of its parents

Topological sorting: Naming nodes according to causal flow. Application of `algorithm 6` results in a well ordered sorting of nodes corresponding to a factorization of the joint pdf.

Comment: conditional independence $\Rightarrow$ encoded by graph structure

(1) A node is conditionally independent of its non-descendants - given its parents

Algorithm 6: Topological sorting

```
DAG with nodes without indices
$i = 1$
while nodes are left within DAG do
    choose node without parents
    set index of node to $i$
    delete node and all its links from DAG
    $i \leftarrow i + 1$
end
```

(2) A node is conditionally independent of all other nodes in the network - given its parents, children and children's parents (Markov blanket)

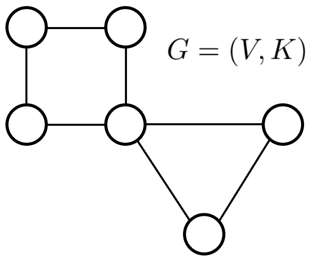
for further details, see Russell and Norvig (2003, ch.14, Fig 14.4). □ Markov Blanket

Properties of DAGs

- efficient representation of knowledge about (causal) relationships between variables
- *topology*: qualitative relationships (causality, conditional independence)
- *annotation*: quantitative information (probability tables of pdfs)
- straightforward to construct

Problem: DAGs do not provide an efficient representation for inference (one of the central aims of using graphical models). Such a respresentation, however, can be constructed on the basis of the corresponding decomposable undirected graph.

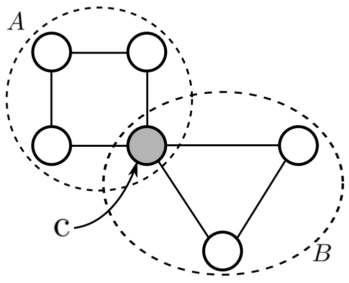
3.2.2 Decomposable Undirected Graphs



undirected graph:
graph with undirected links

Separator
Consider $A, B, C \subset V$ (not necessarily disjoint)

$\Rightarrow C$ separates A and B, if every path from an arbitrary node from $A(x_1 \in A)$ to an arbitrary node from $B(x_n \in B)$ passes through at least one node from C.



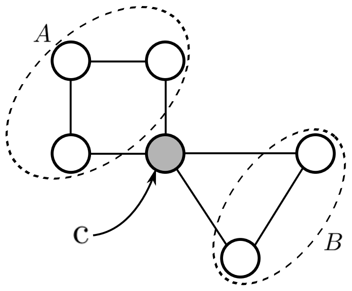
C separates A and B

Complete graph: a graph in which every pair of nodes is connected by an edge.

Proper decomposition of $G = (V, K)$
Consider $V = A \cup B \cup C$ with A, B, C non-empty and disjoint

$\Rightarrow A, B, C$ is a proper decomposition of G if:

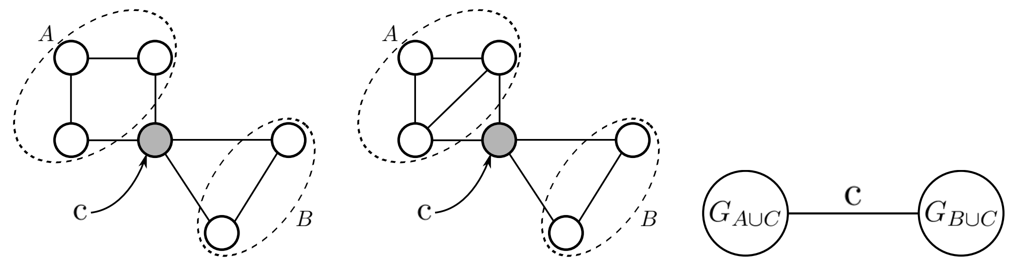
- $\rightsquigarrow$ C separates A and B
- $\rightsquigarrow$ C is complete



C separates A and B
A,B,C is proper decomposition

Decomposable graph

- $\rightsquigarrow$ complete graph <u>or</u>
- $\rightsquigarrow$ there exists a proper decomposition A, B, C such that both subgraphs $G_{A \cup C}$ and $G_{B \cup C}$ are both proper decomposable

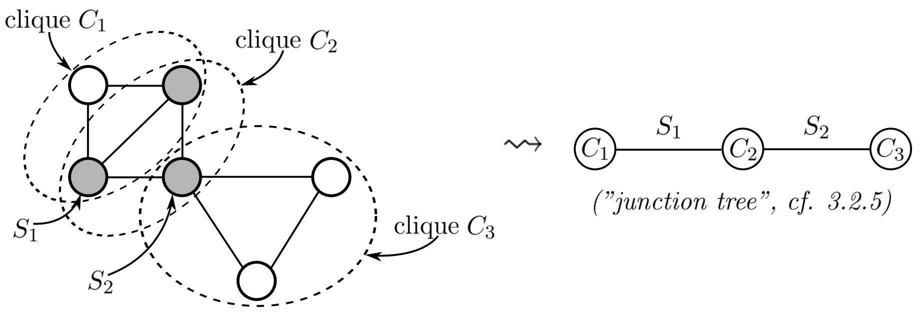


not decomposable
(no separator of $A\cup C$ is complete
nor is $A\cup C$ complete)

decomposable

decomposable or complete

⇒ decomposition into <u>maximally complete subgraphs ("cliques")</u> separated by "separators"

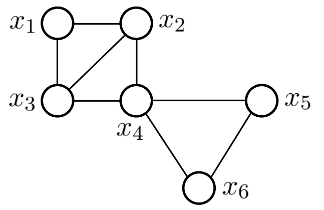


Decomposable graphs and distributions: Decomposable graphs provide a useful factorization of the joint distribution allowing to efficiently calculate marginals (→ inference, see 3.2.4/5).

(1) A is conditionally independent of B given C
⇔ A, B, C is a proper decomposition of G

(2) $$P(\underline{\mathbf{x}}) = \frac{\prod\limits_{\text{cliques } C} P_C(\underline{\mathbf{x}}_C)}{\prod\limits_{\text{separators } S} P_S(\underline{\mathbf{x}}_S)}$$ decomposable graph $\longleftrightarrow$ factorization into marginal distributions

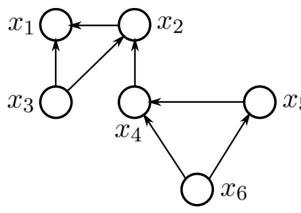
Note: The cliques (separators) are annotated by the marginal probability distributions over the corresponding clique (separator) variables.



$$P(x_1, \ldots, x_6) = \frac{\overbrace{P(x_1, x_2, x_3)P(x_2, x_3, x_4)P(x_4, x_5, x_6)}^{\text{cliques}}}{\underbrace{P(x_2, x_3)P(x_4)}_{\text{separators}}} \tag{3.9}$$

This is a valid distribution, because:

$$\begin{aligned}
\frac{\prod_{C\in\mathcal{C}} P_C(\underline{\mathbf{x}}_C)}{\prod_{S\in\mathcal{S}} P_S(\underline{\mathbf{x}}_S)} &= \frac{P(x_1,x_2,x_3)P(x_2,x_3,x_4)P(x_4,x_5,x_6)}{P(x_2,x_3)P(x_4)} \\
&= P(x_1|x_2,x_3)P(x_2,x_3|x_4)P(x_4,x_5,x_6) \\
&= P(x_1|x_2,x_3)P(x_2|x_3,x_4)P(x_3)P(x_4|x_5,x_6)P(x_5|x_6)P(x_6) \\
&= \prod_{i=1}^{6} P(x_i|x_{i+1},\dots,x_6) \\
&= P(x_1,x_2,\dots,x_6)
\end{aligned}$$



→ every decomposable undirected graph represents a valid probability distribution

→ full marginalization: complexity $\mathcal{O}(2^n)$, n: cardinality of the largest clique

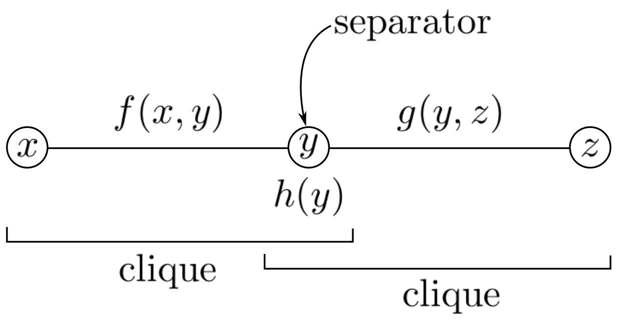
Note: The general factorization into potentials is not unique!

$$P(\underline{\mathbf{x}}) \sim \frac{\prod\limits_{\text{cliques } C} \psi_C(\underline{\mathbf{x}}_C)}{\prod\limits_{\text{separators } S} \psi_S(\underline{\mathbf{x}}_S)} \tag{3.10}$$

Shifting factors between terms in the numerator and denominator results in the same value. This will become important for inference.

3.2.3 Marginal Distributions and Inference on Decomposable Graphs

The following example illustrates how the representation of the joint distribution in terms of potential functions can be transformed into a representation based on the marginal probabilities over the corresponding cliques (cmp. Cowell et al., 2003, p. 84). Consider the distribution over 3 random variables X, Y, Z described by the following graph:



$$P(x,y,z) = \alpha \frac{f(x,y)g(y,z)}{h(y)} \tag{3.11}$$

Goal: calculate factorization into marginals
Starting point: factorization into potentials (here: f, g, h) often straightforward to construct, e.g.

$$P(x,y,z) = \frac{\overbrace{P(x|y)}^{\widehat{=} f(x,y)} \overbrace{P(y|z)P(z)}^{\widehat{=} g(y,z)}}{\underbrace{1}_{\widehat{=} h(y)}} \tag{3.12}$$

$$\begin{tikzcd} x & y \arrow[l] & z \arrow[l] \end{tikzcd}$$

Furthermore, inference problems lead to a potential-based representation for the conditional probability distributions.

Consider, e.g. inference given observed evidence: $Z = z$

$$P(x,y|z) = \frac{P(x,y,z)}{P(z)} = \alpha' P(x,y,z) = \alpha \frac{f(x,y)g(y,z)}{h(y)} \tag{3.13}$$

indicator functions:

$$E(z) = \begin{cases} 1, & \text{if } Z = z \\ 0, & \text{else} \end{cases} \tag{3.14}$$

$$\underbrace{P(x,y|z)}_{P^{(z)}(X,Y,Z)} = \alpha \frac{f(x,y) \overbrace{g(y,z)E(z)}^{\substack{\text{new clique} \\ \text{- potential -} \\ \text{after observing} \\ \text{the evidence}}}}{h(y)} \tag{3.15}$$

$$\Rightarrow P^{(z)}(x,y,z) = \alpha \frac{f(x,y)g^{(z)}(y,z)}{h(y)} \tag{3.16}$$

Calculation through message passing: The representation of the joint probability in terms of clique potentials (eq. 3.11) can be transformed into a representation in terms of the marginal probabilities over the corresponding cliques (cmp. 3.9) which is useful e.g. to infer conditional probabilities.

(1) The marginal distribution of $P(x,y)$ can be calculated from the clique-representation

$$\begin{aligned} P(x,y) &= \sum_z P(x,y,z) \\ &= \alpha \tfrac{f(x,y)}{h(y)} \underbrace{\sum_z g(y,z)}_{\overset{!}{=} h^*(y)} \\ &= \alpha \underbrace{f(x,y)\frac{h^*(y)}{h(y)}}_{\overset{!}{=} f^*(x,y)} \end{aligned} \tag{3.17}$$

This means that $f^*(x,y) \sim P(x,y)$ and can be calculated from the potential $f(x,y)$ by passing to it a "message" containing the *update ratio* $\frac{h^*(y)}{h(y)}$ calculated from the clique potential $g(y,z)$ and the separator potential $h(y)$. This way we get a representation of the joint distribution in terms of the updated (*) potentials:

$$\begin{aligned} P(x,y,z) &= \alpha f(x,y) \tfrac{1}{h(y)} \tfrac{h^*(y)}{h^*(y)} g(y,z) \\ &= \alpha f^*(x,y) \tfrac{1}{h^*(y)} g(y,z) \end{aligned} \tag{3.18}$$

(2) The marginal distribution of clique $P(y,z)$ can be calculated in the same way

$$\begin{aligned} P(y,z) &= \sum_x P(x,y,z) \\ &= \alpha \underbrace{\left(\sum_x f^*(x,y) \right)}_{\overset{!}{=} h^+(y)} \tfrac{1}{h^*(y)} g(y,z) \\ &= \alpha \underbrace{\frac{h^+(y)}{h^*(y)} g(y,z)}_{\overset{!}{=} g^+(y,z)} \end{aligned} \tag{3.19}$$

Yielding the updated potential $g^+(y,z) \sim P(y,z)$ and the joint representation in terms of both updated potentials:

$$\begin{aligned} P(x,y,z) &= \alpha f^*(x,y) \tfrac{h^+(y)}{h^+(y)} \tfrac{1}{h^*(y)} g(y,z) \\ &= \alpha f^*(x,y) \tfrac{1}{h^+(y)} g^+(y,z) \end{aligned} \tag{3.20}$$

(3) Marginal distribution of the separator $P(y)$

$$\begin{aligned} P(y) &= \sum_x P(x,y) \\ &= \alpha \sum_x f^*(x,y) \\ &= \alpha h^+(y) \\ &\Rightarrow h^+(y) \sim P(y) \end{aligned} \tag{3.21}$$

(4) Joint distribution: collecting terms shows that the updated representation is, indeed, based on the clique-marginal distributions.

$$\begin{aligned} P(x,y,z) &= \alpha \tfrac{P(x,y)}{\alpha} \tfrac{\alpha}{P(y)} \tfrac{P(y,z)}{\alpha} \\ &= \tfrac{P(x,y)P(y,z)}{P(y)} \end{aligned} \tag{3.22}$$

□ message passing & local computation

$$\text{message passing} \begin{cases} \text{for calculating prior marginals} \\ \text{for calculating posteriors marginals (after observation of evidence)} \end{cases}$$

3.2.4 Belief Propagation and the Junction Tree Algorithm

Overview

(1) *definition of state variables & construction of the Bayesian network:*

-a- directed acyclic graph (expert analysis, causal relationship)

-b- topological sorting

-c- annotation with conditional probabilities (expert knowledge, extraction of fractions from database, ... inductive learning?)

(2) *construction of the inference engine*

-a- directed acyclic graph $\leftarrow$ annotated with conditional pdfs

$\downarrow$

-b- moral graph

$\downarrow$

-c- undirected decomposable graphs $\leftarrow$ annotated with clique and separator potentials

$\downarrow$

-d- junction tree $\leftarrow$ final data structure for message passing

(3) *inference:* processing the evidence

-a- initialization

-b- modification of clique potentials by observed evidence (not when just calculating prior)

-c- message passing (”belief propagation”)

-d- final marginalization within relevant clique

Details ad (2) – Construction of the inference engine

-a- directed acyclic graph

-b- construction of the related moral graph

□ DAG $\downarrow$ moral graph

$\rightarrow$ connect all parents of a node with undirected links (for all nodes)

$\rightarrow$ replace all directed links with undirected links

-c- construction of a related undirected decomposable graph

□ undirected decompos. graph

$\rightarrow$ add undirected links such that all cycles of length four and larger contain a chord

$\rightarrow$ chordal graph is always decomposable[11]

□ triangulation algorithm

$\rightarrow$ chordal graph is not unique

$\rightarrow$ construction of a chordal graph with cliques of minimal size is NP hard

-d- construction of a related junction tree

$\rightarrow$ *nodes of the junction tree*:
maximal cliques of the decomposable graph

$\rightarrow$ *links of the junction tree*:
connect neighboring maximal cliques annotated by the corresponding separators

□ construction of the junction tree

[11]for a proof, see Cowell et. al. theorem 4.4

→ after the first loop, the running intersection property holds, i.e. for all $1 < j \leq k$ there exists one $i < j$, such that $C_j \cap \big(C_1 \cup \ldots \cup C_{i-1}\big) \subseteq C_i$

→ but: nodes may not only be cliques, but could be other complete subgraphs

→ second loop estimates those nodes

Details ad (3) – Inference: processing of the evidence

-a- initialization of the clique and separator potentials

☐ initialization of cliques and separators

$$P(\underline{\mathbf{x}}) = \prod_k P(x_k|parents(x_k)) \leftarrow \text{ from annotated DAG} \tag{3.23}$$

Algorithm 7: Initialization of clique potentials

```
Set all clique and separator potentials to $\psi_{C,S}(\underline{\mathbf{x}}_{C,S}) = 1$
for all nodes $x_k$ of the DAG do
    Find a node of the junction tree which contains $x_k$ and its parents
    Multiply corresponding clique potential with $P(x_k|parents(x_k))$
end
```

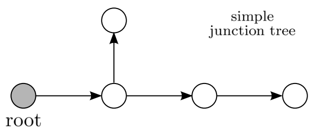
-b- modification of clique potentials by observed evidence

→ for given observations $\underline{\mathbf{X}}_e = \underline{\mathbf{x}}_e$, set clique potentials to:

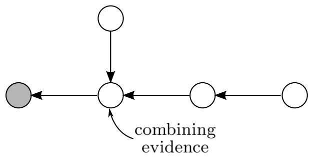
$$P(\underline{\mathbf{x}}/\underline{\mathbf{x}}_e|\underline{\mathbf{x}}_e) = \alpha P(\underline{\mathbf{x}}) \underbrace{\prod_{j\in e}}_{\substack{\text{multiplied to}\\ \text{the proposed}\\ \text{clique potential}}} \underbrace{E(x_j)}_{\text{indicator functions}} \qquad E(x_j) = \begin{cases} 1, & \text{if } X_j = x_j \\ 0, & \text{else} \end{cases} \tag{3.24}$$

-c- calculation of marginal probabilities: belief propagation

→ "collect evidence" (1st pass): choose a root node from the junction tree, broadcast "request" & collect

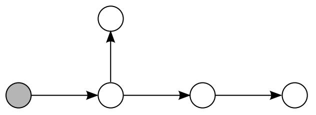


broadcast of request



transmission of messages (and modification of potentials)

→ "distribute evidence" (2nd pass)



transmission of messages (and modification of potentials)

⇒

$$P(\underline{\mathbf{x}}) = \frac{\prod\limits_{\text{cliques } C} P_C(\underline{\mathbf{x}}_C)}{\prod\limits_{\text{separators } S} P_S(\underline{\mathbf{x}}_S)} \Bigg\} \text{marginal distributions} \tag{3.25}$$

-d- marginalization within relevant clique

3.3 Bayesian Inference and Neural Networks

3.3.1 Generative Models

observations: $\underline{\mathbf{z}}^{(\alpha)} = (\underbrace{\underline{\mathbf{x}}^{(\alpha)}}_{\substack{\text{independent}\\ \text{variable}}} , \underbrace{\underline{\mathbf{y}}_T^{(\alpha)}}_{\substack{\text{associated}\\ \text{variable}}}), \alpha = 1, \ldots, p$

true distribution:

$$p_{(\underline{\mathbf{z}})} = \underbrace{p_{(\underline{\mathbf{y}}|\underline{\mathbf{x}})}}_{\substack{\text{conditional}\\ \text{probability}}} \; p_{(\underline{\mathbf{x}})} \tag{3.26}$$

previous approach:

$\rightsquigarrow$ construct a parametrized class $y_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}$ of (deterministic) predictors

$\rightsquigarrow$ inference is based on a single selected (optimal) predictor $y_{(\underline{\mathbf{x}};\underline{\mathbf{w}}^*)}$

generative model approach:

$\rightsquigarrow$ construct a parametrized class $p_{(\underline{\mathbf{y}}|\underline{\mathbf{x}};\underline{\mathbf{w}})}$ of (conditional) densities

$\rightsquigarrow$ inference is based on good "generative models"

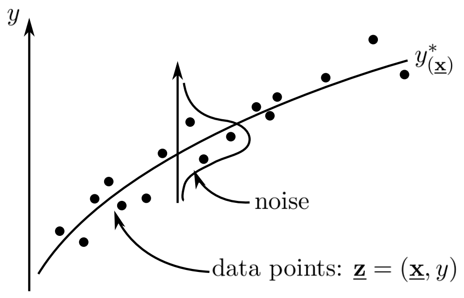
$$\underbrace{\boxed{\text{"generative" model of a data source}}}_{\substack{\text{description of the data}\\ \text{generation process}}} \longrightarrow \boxed{\text{observations}}$$

Comment:

models $p_{(\underline{\mathbf{z}};\underline{\mathbf{w}})}$ for unconditional densities $\rightsquigarrow$ unsupervised learning (e.g. ICA, mixture models)

models $p_{(\underline{\mathbf{y}}|\underline{\mathbf{x}};\underline{\mathbf{w}})}$ for conditional densities $\rightsquigarrow$ supervised learning

Example I: Generative model for simple regression


y
$y^*_{(\underline{\mathbf{x}})}$
noise
data points: $\underline{\mathbf{z}} = (\underline{\mathbf{x}}, y)$

Description of the data generation process:

$$y_{(\underline{\mathbf{x}})} = \underbrace{\widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}}_{\substack{\text{model of a deterministic}\\ \text{relationship}}} + \underbrace{\widehat{\eta}}_{\substack{\text{model of the noise}\\ \text{here: additive noise}}} \tag{3.27}$$

The deterministic relationship is typically modeled with a parametrized function e.g. a polynomial or a neural network. Noise is often assumed to be additive but could also be multiplicative. Here, we model noise with a parametrized distribution $\widehat{p}_{(\widehat{\eta};\underline{\sigma})}$.

Common noise models
Additive Gaussian noise:

$$\widehat{p}_{(y|\underline{\mathbf{x}};\underline{\mathbf{w}})} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{\left(y - \widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}\right)^2}{2\sigma^2} \right\} \tag{3.28}$$

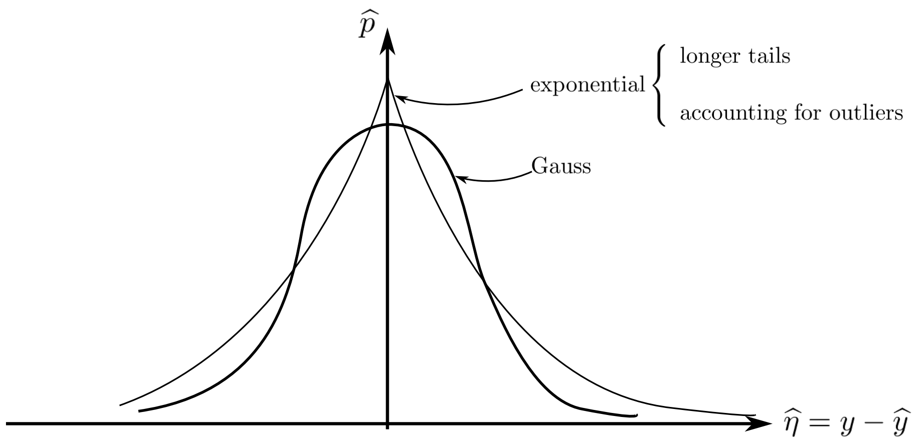
Additive Minkowski noise:

$$\widehat{p}_{(y|\underline{\mathbf{x}};\underline{\mathbf{w}})} = \frac{d\beta^{\frac{1}{d}}}{2\,\underbrace{\Gamma_{(\frac{1}{d})}}_{\substack{\text{Gamma}\\ \text{function}}}} \exp\left\{ -\beta \left| y - \widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}})} \right|^d \right\} \tag{3.29}$$

$d = 1$: exponential function

$$\widehat{p}_{(y|\underline{\mathbf{x}};\underline{\mathbf{w}})} = \frac{\beta}{2} \exp\left\{ -\beta \left| y - \widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}})} \right|^d \right\} \tag{3.30}$$

$d = 2$: Gaussian distribution



Example II: Classification for c classes $C_k, k = 1, \dots, c$

Description of the data generation process:

$$p_{(C_k|\underline{\mathbf{x}})} = y_{k(\underline{\mathbf{x}})} \tag{3.31}$$

$\rightsquigarrow$ label noise (e.g. overlapping classes)

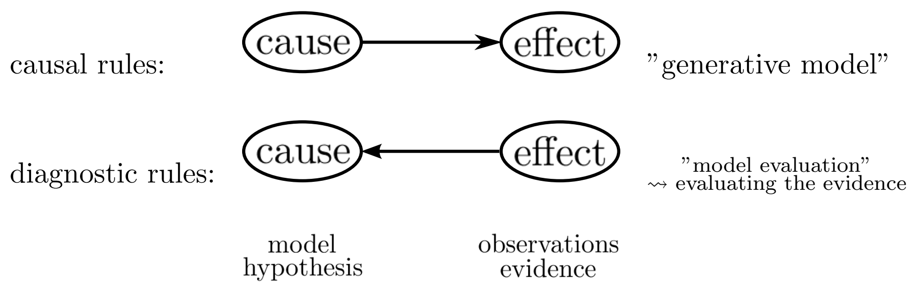
Model:

$$\widehat{p}_{(C_k|\underline{\mathbf{x}};\underline{\mathbf{w}})} = y_{k(\underline{\mathbf{x}};\underline{\mathbf{w}})} \tag{3.32}$$

$\rightsquigarrow$ general parametrized model (e.g. neural network: section 1.4.7)

3.3.2 Bayesian Model Selection

Reasoning under uncertainty (cf. section 3.1.4)



Degree of belief in a given model

$\rightsquigarrow$ set $\{M_i\}$ of disjunct models (hypotheses) M_i

$\rightsquigarrow$ a random observed event E (evidence)

Bayes rule:

$$\underbrace{P_{(M_i|E)}}_{\text{posterior}} = \frac{\overbrace{P_{(E|M_i)}}^{\text{likelihood}} \overbrace{P_{(M_i)}}^{\text{prior}}}{\underbrace{P_{(E)}}_{\substack{\text{normalization constant} \\ (\text{"evidence"})}}} \tag{3.33}$$

Likelihood $P_{(E|M_i)}$: probability of observing the evidence E, given that model M_i is true $\Leftarrow$ $\boxed{\text{"generative model"}}$

Prior $P_{(M_i)}$: our degree of belief in M_i, before E has been observed

Initialization of prior beliefs $\rightarrow$ maximum entropy methods

$$-\sum_i P_{(M_i)} \ln P_{(M_i)} \overset{!}{=} \max \quad \substack{\text{find least informative} \\ \text{prior beliefs}} \tag{3.34}$$

Constraints:

$$\begin{array}{cc} \sum_i P_{(M_i)} = 1 & \text{normalization of probabilities} \\ \text{information about (e.g.) moments of meanwhile quantities} & \text{formal description of prior knowledge} \end{array} \tag{3.35}$$

$\Rightarrow$ solution using Lagrange multipliers

$\Rightarrow$ if no prior knowledge exists:

$$P_{(M_i)} = \text{const.} \tag{3.36}$$

$$P_{(M_i|E)} \sim P_{(E|M_i)} \tag{3.37}$$

3.3.3 Bayesian Prediction

Predictions regarding future data e will depend on both the observed evidence E and the model M used. The *predictive distribution* combines predictions from multiple models and weights them, depending on how probable these models are given the data observed so far ($\rightarrow$ posterior distribution of models/parameters given observed evidence).

The predictive distribution:

$$\text{observations } E \xrightarrow{\text{fundamental problem of prediction}} \begin{array}{c}\text{degree of belief } P_{(e|E)} \\ \text{into a new event } e\end{array} \tag{3.38}$$

$$\begin{aligned} P_{(e|E)} &= \sum_i P_{(e,M_i|E)} && \text{marginalization} \\ &= \sum_i P_{(e|M_i,E)} P_{(M_i|E)} && \text{def. of conditional probability} \\ &\overset{!}{=} \sum_i P_{(e|M_i)} P_{(M_i|E)} && \text{conditional independence assumption} \end{aligned} \tag{3.39}$$

$\rightarrow$ only ok if model fully describes data generation

$\rightarrow$ only approximately fulfilled in reality

$\Rightarrow$ committee-ansatz (Bayesian committee)

Prediction and evaluation of distribution: After selection of a "predicted attribute" (see below) one can make a decision based on $P_{(e|E)}$. In many cases it might not be optimal to simply choose the maximally probable value because there are additional constraints to consider, e.g. estimated loss if a prediction error occurs.

Loss function:

$$C_{(\underbrace{e}_{\substack{\text{true}\\ \text{value}}}, \underbrace{\widehat{e}}_{\substack{\text{predicted}\\ \text{value}\\ \text{(based on}\\ P_{(e|E)})}})} \tag{3.40}$$

Instead of choosing the maximally probable value, it is therefore a common strategy to minimize the expected loss

$$\widehat{e} = \underset{\tilde{e}}{\operatorname{argmin}} \int de C_{(e,\tilde{e})} P_{(e|E)} \tag{3.41}$$

3.3.4 Application: MLPs with weight decay

For some simple models such as linear regression, the posterior parameter distribution and the predictive distribution can be analysed in closed form ($\rightsquigarrow$ Bayesian linear regression). For more flexible models, exact analytical treatment is not available such that evaluation of the predictive distribution requires approximation techniques (e.g. sampling, variational inference).

Here, we discuss another alternative, the *Laplace approximation*, which (a) approximates the true posterior parameter distribution with a Gaussian centered on a MAP estimate and (b) assumes the network function to be approximately linear around this point.

To determine the posterior parameter distribution from a given set of training data we will (a) formulate the likelihood function for a given class of generative models, (b) determine the maximum entropy prior distribution, and (c) use Bayes rule to combine them.

training data: $\left\{ \left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right) \right\}, \alpha = 1, \ldots, p$

abbreviations: $X = \left\{ \underline{\mathbf{x}}^{(\alpha)} \right\}, Y = \left\{ \underline{\mathbf{y}}_T^{(\alpha)} \right\}$

(a) Construction of the model class $\rightarrow$ likelihood of the data

$$P_{\left(y_T^{(\alpha)}|\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}\right)} \sim \exp\Big\{ -\beta \underbrace{e^T_{\left(y_T^{(\alpha)},\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}\right)}}_{\substack{\text{almost always possible,}\\ \text{because } P \text{ positive}}} \Big\} \tag{3.42}$$

$$\begin{aligned} P_{(Y|\underline{\mathbf{x}};\underline{\mathbf{w}})} &\sim \prod_\alpha \exp\Big\{ -\beta e^T_{\left(y_T^{(\alpha)},\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}\right)} \Big\} \\ &\sim \exp\Big\{ -\beta \sum_\alpha e^T_{\left(y_T^{(\alpha)},\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}\right)} \Big\} \qquad (3.43) \\ &\sim \exp\left\{ -\beta E^T \right\} \end{aligned}$$

where $E^T := E_{(Y,\mathrm{x};\underline{\mathbf{w}})}$ denotes the training error. This shows a direct relation between minimisation of the mean squared error and Maximum likelihood estimation under the assumption of additive Gaussian noise:

$$P_{(Y|\underline{\mathbf{x}};\underline{\mathbf{w}})} = \frac{1}{(2\pi\sigma^2)^{\frac{p}{2}}} \exp\left\{ \underbrace{-\frac{1}{2\sigma^2} \overbrace{\underbrace{\sum_{\alpha=1}^{p}}_{\text{iid assumption}} \left(y_T^{(\alpha)} - \underbrace{\widehat{y}_{\left(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}\right)}}_{\to\text{ MLP}} \right)^2}^{\text{"quadratic error"}}}_{\text{additive Gaussian noise}} \right\} \tag{3.44}$$

(b) Construction of the prior: Maximum entropy method

$$-\sum_{\underline{\mathbf{w}}} P_{(\underline{\mathbf{w}})} \ln P_{(\underline{\mathbf{w}})} \overset{!}{=} \max \tag{3.45}$$

$$\sum_{\underline{\mathbf{w}}} P_{(\underline{\mathbf{w}})} = 1 \tag{3.46}$$

$$\sum_{\underline{\mathbf{w}}} E^R_{(\underline{\mathbf{w}})} P_{(\underline{\mathbf{w}})} = \underbrace{E_0}_{\substack{\text{prior}\\\text{knowledge}}} \tag{3.47}$$

Solution using Lagrange multipliers:

$$-\sum_{\underline{\mathbf{w}}} P_{(\underline{\mathbf{w}})} \ln P_{(\underline{\mathbf{w}})} + \lambda\left(\sum_{\underline{\mathbf{w}}} P_{(\underline{\mathbf{w}})} - 1 \right) - \alpha\left(\sum_{\underline{\mathbf{w}}} E^R_{(\underline{\mathbf{w}})} P_{(\underline{\mathbf{w}})} - E_0 \right) \overset{!}{=} \max \tag{3.48}$$

$$-\ln P_{(\underline{\mathbf{w}})} - 1 + \lambda - \alpha E^R_{(\underline{\mathbf{w}})} = 0$$

$$\ln P_{(\underline{\mathbf{w}})} = \lambda - 1 - \alpha E^R_{(\underline{\mathbf{w}})} \tag{3.49}$$

$$P_{(\underline{\mathbf{w}})} \sim \exp\left(-\alpha E^R_{(\underline{\mathbf{w}})} \right)$$

λ is found through normalization of prior probabilities $\leftarrow$ equivalent to choosing a normalization factor

α: can be calculated - in principle - from the corresponding constraint $\to$ often used as a hyperparameter

Comment: This gives the “least informative” prior distribution $P(\underline{\mathbf{w}})$ constraining the final solution as little as possible. However, prior knowledge already explicitly put in

$\rightsquigarrow$ choice of parametrization (i.e. model class)

$\rightsquigarrow$ choice of noise model

(c) Application of Bayes rule: making use of the distributions from eqs. (3.43) and (3.49), we get the posterior parameter distribution

$$\begin{aligned} P_{(\underline{\mathbf{w}}|Y,X)} &\sim P_{(Y|X;\underline{\mathbf{w}})} P_{(\underline{\mathbf{w}})} \\ &\sim \exp\big\{-\beta E^T - \alpha E^R\big\} \\ &= \exp(-\beta R) \end{aligned} \tag{3.50}$$

where:

$$R = \underbrace{p}_{\sim\#\text{data}} E^T + \underbrace{\alpha^{'}}_{\sim\#\text{hyperparameters}} E^R \tag{3.51}$$

$$\alpha^{'} = \frac{\alpha}{\beta} \quad \begin{array}{l}\text{the more data points, the}\\ \text{less important is the prior}\end{array} \tag{3.52}$$

- formal equivalence to a regularized training error
- low $R \rightarrow$ high posterior
- noise model $\rightarrow$ form of training error
- prior knowledge $\rightarrow$ regularization term

Note: The form of the posterior distribution gives a probabilistic justification of the MLP with weight decay

$$R = \frac{1}{2}\sum_{\alpha=1}^{p}\Big(y_T^{(\alpha)} - \widehat{y}_{(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}})}\Big)^2 + \underbrace{\frac{\alpha}{2}\sum_{k=1}^{d}\mathrm{w}_k^2}_{\textit{cf. section 1.4.6}} \tag{3.53}$$

Comment: Because for the MLP, $\hat{y}$ depends nonlinearly on $\underline{\mathbf{w}}$, this distribution is not simply a Gaussian and might have multiple local optima.

3.3.5 The "maximum a posteriori" method

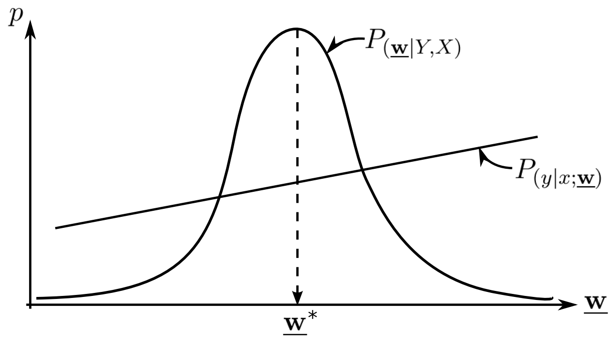
Prediction through Bayesian committee:

$$P_{(y|\underline{\mathbf{x}};Y,X)} = \int P_{(y|\underline{\mathbf{x}};\underline{\mathbf{w}})} P_{(\underline{\mathbf{w}}|Y,X)} d^d\underline{\mathbf{w}} \tag{3.54}$$

- often no closed expression for the integral
- due to large number of model parameters, numerical methods to approximate high dimensional integrals (e.g. MCMC, variational Bayes) are often either time-consuming or inaccurate
- MAP method provides an efficient *approximation*

(see also Bishop (2006, ch. 5.7))

The MAP-assumption: Posterior has a localized maximum



$$\begin{aligned}\underline{\mathbf{w}}^* &= \underset{\underline{\mathbf{w}}}{\operatorname{argmax}} P_{(\underline{\mathbf{w}}|Y,X)} \\ &= \underset{\underline{\mathbf{w}}}{\operatorname{argmin}} \underbrace{\big(pE^T + \alpha' E^R\big)}_{R} \quad \leftarrow \text{"MAP solution"}\end{aligned} \tag{3.55}$$

We will use this assumption to approximate the *predictive distribution*

$$P_{(y|\underline{\mathbf{x}};Y,X)} \sim \int \underbrace{\exp\big(-\beta e^T_{(y,x;\underline{\mathbf{w}})}\big)}_{\substack{\text{individual}\\ \text{likelihood}}} \underbrace{\exp\big(-\beta R_{(\underline{\mathbf{w}},Y,X)}\big)}_{\text{posterior}} d^d\underline{\mathbf{w}} \tag{3.56}$$

This can be a good approximation e.g. when the posterior is highly concentrated around $\underline{\mathbf{w}}^*$. The approach used here involves two approximations around the mode $\underline{\mathbf{w}}^*$:

(1) Gaussian approximation of the posterior around $\underline{\mathbf{w}}^*$: (Taylor expansion to 2nd order at $\underline{\mathbf{w}}^*$).[12]

$$R_{(\underline{\mathbf{w}},Y,X)} = R_{(\underline{\mathbf{w}}^*,Y,X)} + \frac{1}{2}\sum_{i,j}(\mathrm{w}_i - \mathrm{w}_i^*) \underbrace{\frac{\partial^2 R}{\partial \mathrm{w}_i \partial \mathrm{w}_j}}_{H_{ij}:\ \text{Hesse matrix}} \Bigg|_{\underline{\mathbf{w}}^*} (\mathrm{w}_j - \mathrm{w}_j^*) \tag{3.57}$$

(2) Linear approximation of the individual likelihood: Taylor expansion of the models input-output function around $\underline{\mathbf{w}}^*$ to 1st order

$$e^T_{(y,\underline{\mathbf{x}};\underline{\mathbf{w}})} = e^T_{(y,x;\underline{\mathbf{w}}^*)} + \sum_i \frac{\partial e^T}{\partial \mathrm{w}_i}\Bigg|_{\underline{\mathbf{w}}^*} (\mathrm{w}_i - \mathrm{w}_i^*) \tag{3.58}$$

[12]As the expansion is around the maximum $\underline{\mathbf{w}}^*$, first order terms vanish here.

Result of the integration (*calculation see supplementary material*)

$$P_{(y|\underline{\mathbf{x}};Y,X)} \sim \exp\left\{ \underbrace{-\beta e^T}_{\substack{\text{individual}\\\text{likelihood}\\\text{for the MAP}\\\text{"model"}\underline{\mathbf{w}}^*}} + \frac{\beta}{2}\Bigg(\underbrace{\frac{\partial e^T}{\partial \underline{\mathbf{w}}}}_{\substack{\text{correction for}\\\text{uncertainty}\\(2^{\text{nd}}\text{ order})\\\text{in estimating}\\\text{the model}\underline{\mathbf{w}}}} \Bigg) \underline{\mathbf{H}}^{-1} \frac{\partial e^T}{\partial \underline{\mathbf{w}}} \right\}\Bigg|_{\underline{\mathbf{w}}^*} \tag{3.59}$$

For the MLP with additive Gaussian noise and weight decay, the corresponding terms are:

$$\beta = \frac{1}{\sigma^2} \qquad e^T_{(\underline{\mathbf{x}},y;\underline{\mathbf{w}})} = \frac{1}{2}\big(y - \underbrace{\widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}}_{\text{MLP}}\big)^2 \qquad \frac{\partial e^T}{\partial \underline{\mathbf{w}}} = -\big(y - \widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}})}\big) \underbrace{\frac{\partial \widehat{y}}{\partial \underline{\mathbf{w}}}}_{\overset{!}{=}\underline{\mathbf{g}}}$$

inserting into (3.59) yields

$$P_{(y|\underline{\mathbf{x}};Y,X)} \sim \exp\left\{ -\frac{1}{2\sigma^2}\Big(\underbrace{1}_{\substack{\text{noise}\\\text{model}}} - \underbrace{\underline{\mathbf{g}}^T\underline{\mathbf{H}}^{-1}\underline{\mathbf{g}}}_{\substack{\text{correction for}\\\text{uncertainty in the}\\\text{determination of}\\\text{model parameters}}} \Big)\Big|_{\underline{\mathbf{w}}^*} \big(y - \widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}}^*)}\big)^2 \right\} \tag{3.60}$$

This is a Gaussian centered on the model prediction ($M_{\underline{\mathbf{w}}^*}$) whose variance depends on the width of the posterior ($\rightarrow$ noise model):

$$\sigma_y^2 \overset{!}{=} \frac{\sigma^2}{1 - \underline{\mathbf{g}}^T\underline{\mathbf{H}}^{-1}\underline{\mathbf{g}}}\Bigg|_{\underline{\mathbf{w}}^*} \tag{3.61}$$

width of posterior $\uparrow \rightsquigarrow$ "$\underline{\mathbf{H}}^{-1} \uparrow$" $\rightsquigarrow (1 - \underline{\mathbf{g}}^T\underline{\mathbf{H}}^{-1}\underline{\mathbf{g}}) \downarrow \rightsquigarrow \sigma_y^2 \uparrow$. □MacKay(1992) Fig.4b

For a more detailed discussion of the theoretical approach, see MacKay (1992).

Comments

(1) $\underline{\mathbf{w}}^*$ is referred to as the "MAP-solution"

$$\underline{\mathbf{w}}^* = \underset{\underline{\mathbf{w}}}{\operatorname{argmin}} \big(pE^T + \alpha' E^R\big) \tag{3.62}$$

Formal equivalence between MAP solution and regularized ERM:

$$E^T \widehat{=} -\log \text{likelihood} \qquad E^R \widehat{=} -\log \text{prior} \tag{3.63}$$

(2) *Efficient calculation of the relevant terms for MLPs*
$\underline{\mathbf{g}}$ can be calculated via backpropagation, for calculation of $\underline{\mathbf{H}}^{-1}$, see e.g. Bishop (2006, ch. 5.4).

(3) $P_{(y|\underline{\mathbf{x}};Y,X)} \sim \exp(-\beta e^T)\big|_{\underline{\mathbf{w}}^*}$ is sometimes referred to as the *MAP solution* for the output distribution

(4) The solution illustrates *2 types of uncertainty:*

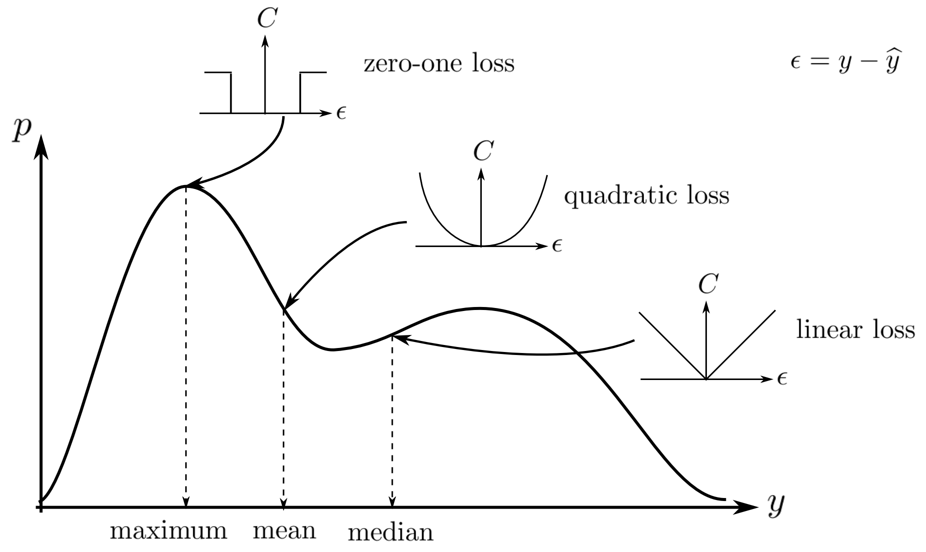
- uncertainty inherent in the generating process ($\rightarrow \sigma^2$)
- precision of estimated model ($\rightarrow 1 - \underline{\mathbf{g}}^T \underline{\underline{\mathbf{H}}}^{-1} \underline{\mathbf{g}}$)

3.3.6 Prediction of attributes (point prediction)

Loss-function: $C_{(y,\widehat{y})}$, real valued attributes

Minimization of expected loss:

$$\widehat{y}_{(\underline{\mathbf{x}})} = \underset{\tilde{y}}{\operatorname{argmin}} \int dy C_{(y,\tilde{y})} P_{(y|x;Y,X)} \tag{3.64}$$



For the Gaussian density: maximum $\widehat{=}$ mean $\widehat{=}$ median
$\Rightarrow$ predictions coincide

Example: MLP with weight-decay, Gaussian approximation (MAP)

$$\begin{aligned}\underline{\mathbf{w}}^* &= \underset{\underline{\mathbf{w}}}{\operatorname{argmin}} \left(pE^T + \alpha' E^R\right) \\ &= \underset{\underline{\mathbf{w}}}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{\alpha=1}^{p} \left(y_T^{(\alpha)} - \widehat{y}_{\left(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}\right)} \right)^2 + \frac{\alpha'}{2} \sum_{k=1}^{d} \mathrm{w}_k^2 \right\}\end{aligned} \tag{3.65}$$

predictor: "optimal" network: $\widehat{y}_{(\underline{\mathbf{x}};\underline{\mathbf{w}}^*)}$

References

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0387310738.

Cover, T. M. (1965). "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition". In: *Electronic Computers, IEEE Transactions on* EC-14.3, pp. 326–334.

Cowell, R. et al. (2003). *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks*. Information Science and Statistics. Springer New York. ISBN: 9780387987675.

Funahashi, K. (1989). "On the Approximate Realization of Continuous Mappings by Neural Networks". In: *Neural Netw.* 2.3, pp. 183–192. ISSN: 0893-6080.

Hochreiter, J. and K. Obermayer (2006). "Support Vector Machines for Dyadic Data". In: *Neural Comput.* 18, pp. 1472–1510.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hornik, K., M. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5, pp. 359–366.

Jaynes, E. T. and G. L. Bretthorst, eds. (2003). *Probability theory : the logic of science*. Cambridge, UK, New York: Cambridge University Press. ISBN: 0-521-59271-2.

MacKay, D. J. C. (1992). "Bayesian Interpolation". In: *Neural Computation* 4.3, pp. 415–447. ISSN: 0899-7667. DOI: `10.1162/neco.1992.4.3.415`.

— (2003). *Information Theory, Inference & Learning Algorithms*. New York, NY, USA: Cambridge University Press. ISBN: 978-0521642989.

Russell, S. and P. Norvig (2003). *Artificial Intelligence: a modern approach*. Prentice-Hall International.

Vapnik, V. (1998). *Statistical learning theory*. Adaptive and learning systems for signal processing, communications, and control. Wiley. ISBN: 9780471030034.