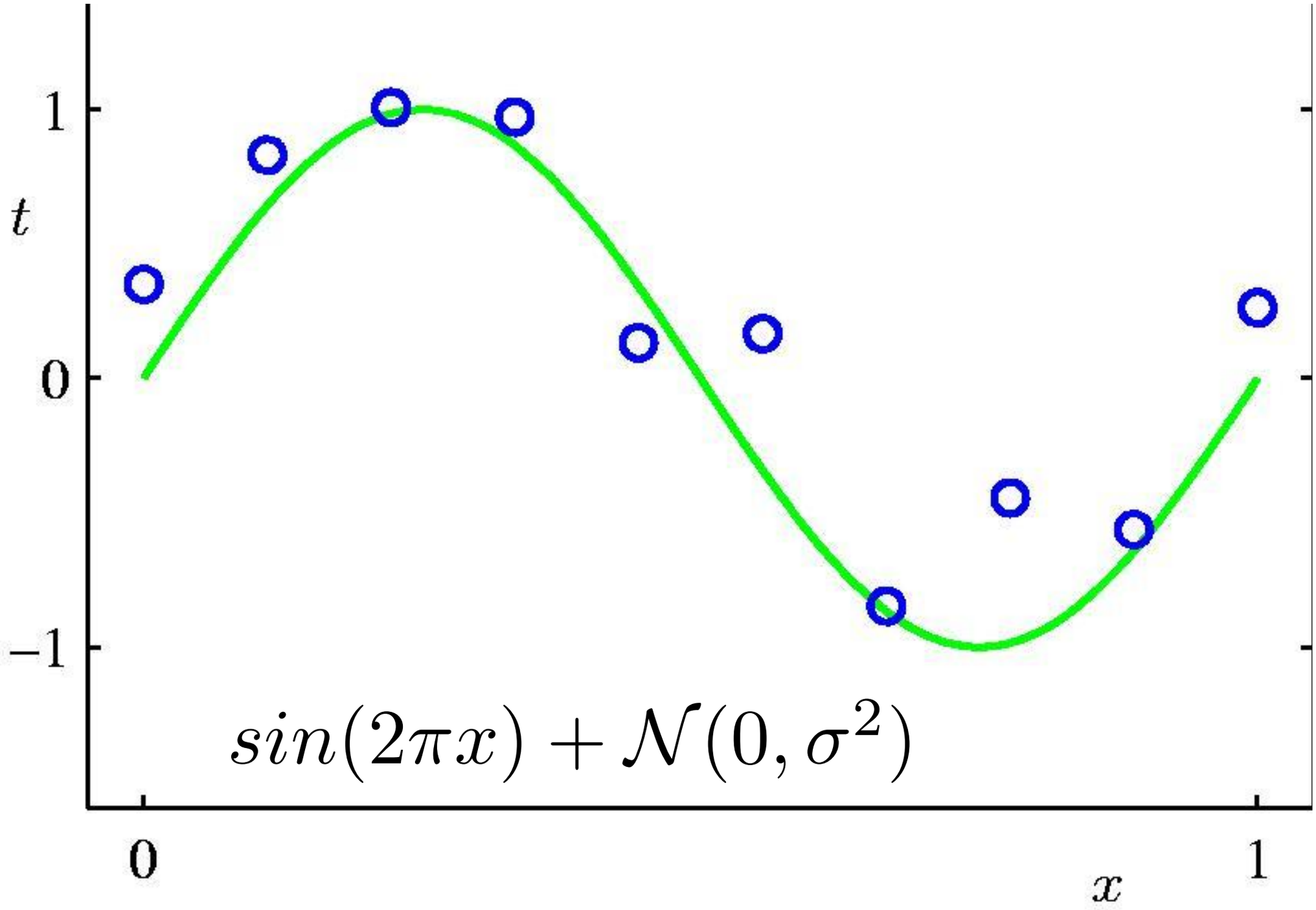
Wiederholung: Wahrscheinlichkeitsbasierte Modellierung

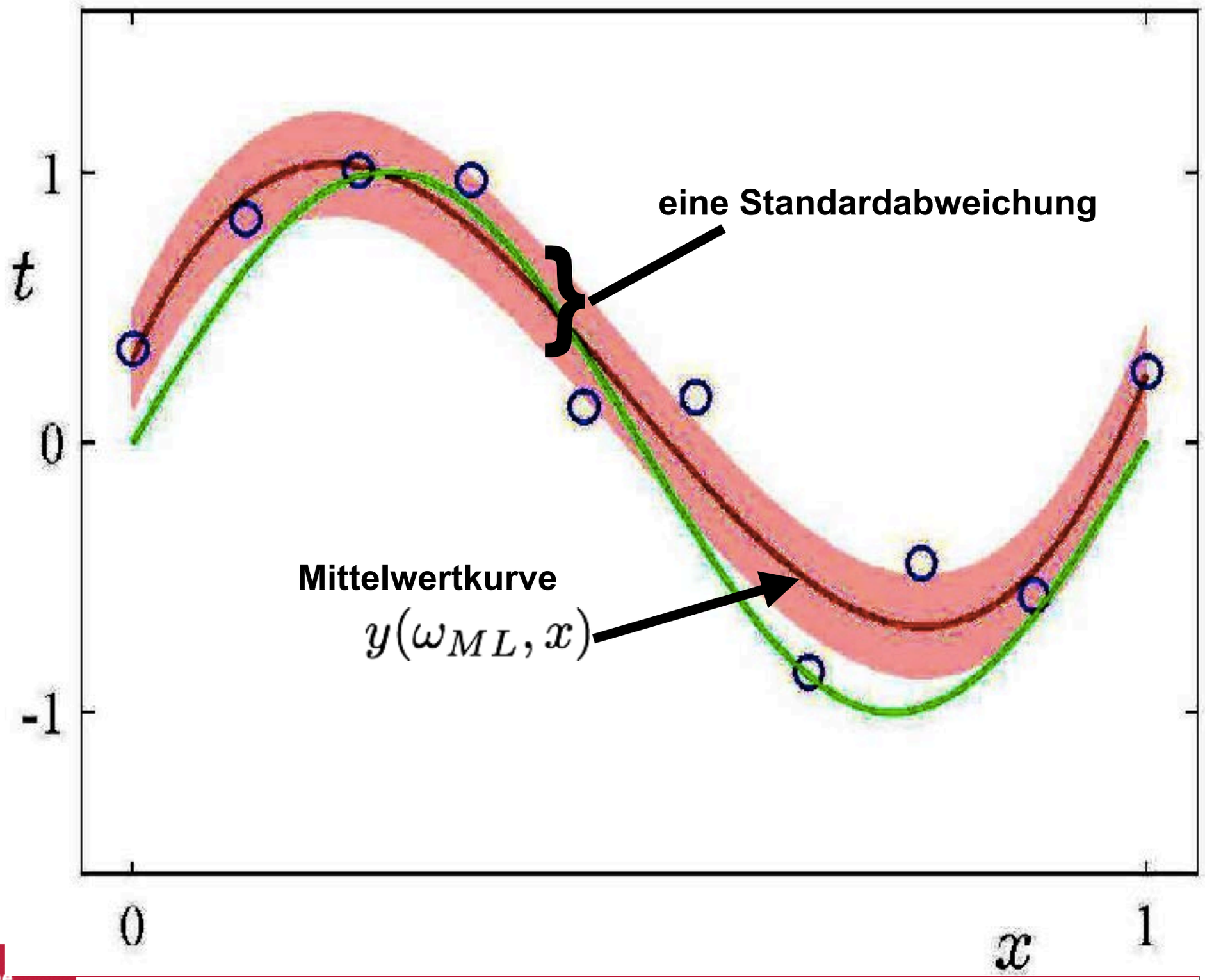


Slide from Bishop, machine learning

Vorgehen

- Rauschen/Unsicherheit in Daten ~ Normalverteilung
- bilde Likelihood und Data-Likelihood L(w)
- Minimierung von -log L(w):
  - argmin_w -log L(w) => maximum Likelihood Parameter
- Generalisierung durch *predictive distribution:*
  - wahrscheinlichster Wert (Mittelwert) $y(\omega_{ML}, x)$
  - auch möglich: “sampling” aus der *predictive distribution*
  - Konfidenz gegeben durch die Präzision (Inverse der Varianz der Verteilung)

predictive distribution für Maximum Likelihood



weitergehender Bayes'scher Ansatz:

- modelliere initiale Unsicherheit über die Parameter als $P(\omega)$
- $P(\omega)$ ist die a-priori W.-keit bevor Daten beobachtet werden
- modelliere den Likelihood wie vorher
- dann berechne die a-posteriori Wahrscheinlichkeit mit der Bayes-Formel:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)}$$

- führt auf maximum a-posteriori Ansatz (MAP) für optimalen Parameter(vektor):

$$w_{MAP} = \text{argmax}_w \, P(w \mid D)$$

der Inferenzschritt & die *Bayesian predictive distribution*

$$p(\mathbf{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha)$$

a-posteriori W.-keit ***is proportional to*** **likelihood x prior**

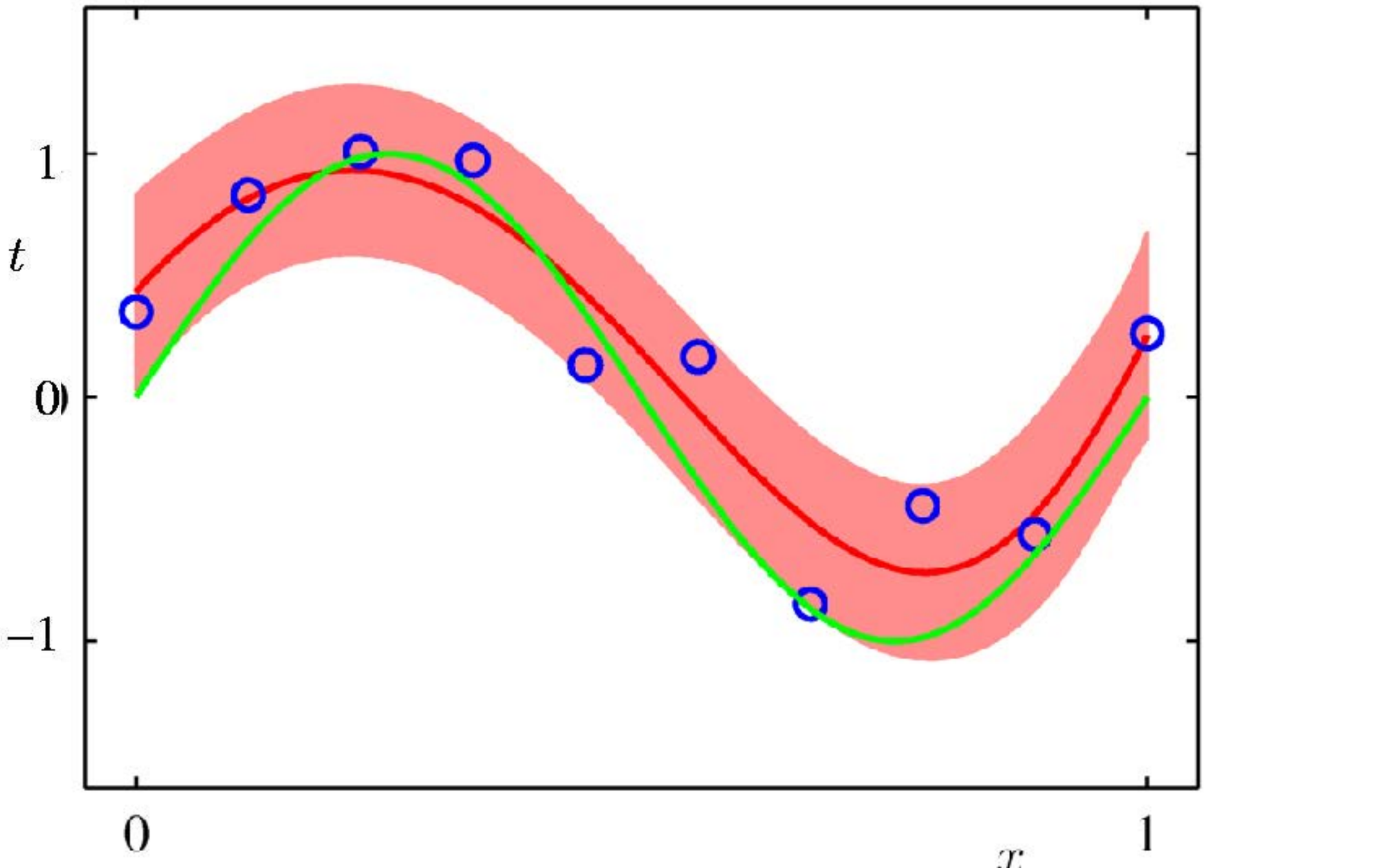
(updated knowledge)

- **schließlich: integriere über ALLE parameter w (Bayesian predictive distribution)**

$$p(t|x, \mathbf{x}, \mathbf{t}) = \int p(t|x, \mathbf{w})p(\mathbf{w}|\mathbf{x}, \mathbf{t})\,\mathrm{d}\mathbf{w} = \mathcal{N}\left(t|m(x), s^2(x)\right)$$

Bayesian Predictive Distribution

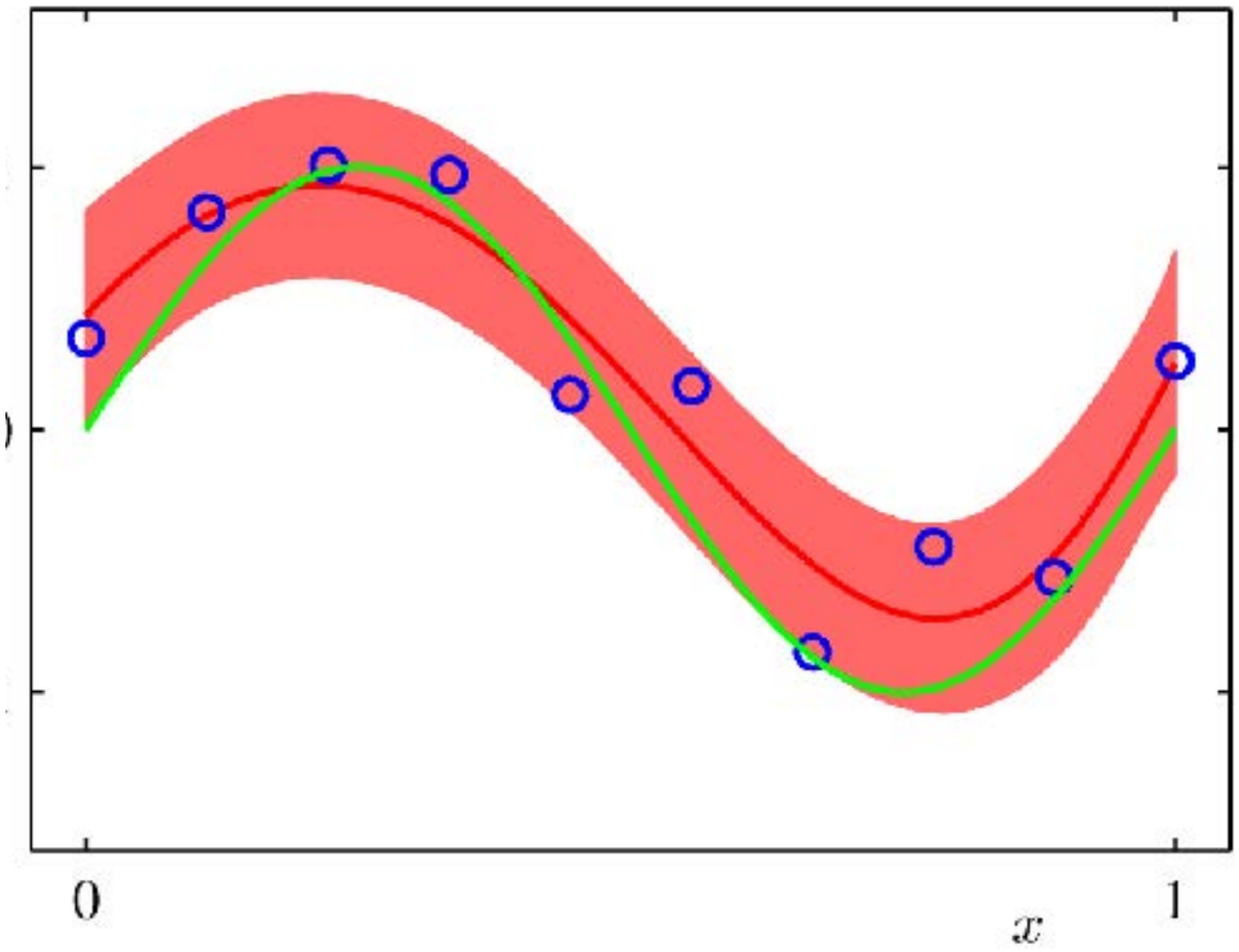
$$p(t|x, \mathbf{x}, \mathbf{t}) = \mathcal{N}\left(t|m(x), s^2(x)\right)$$

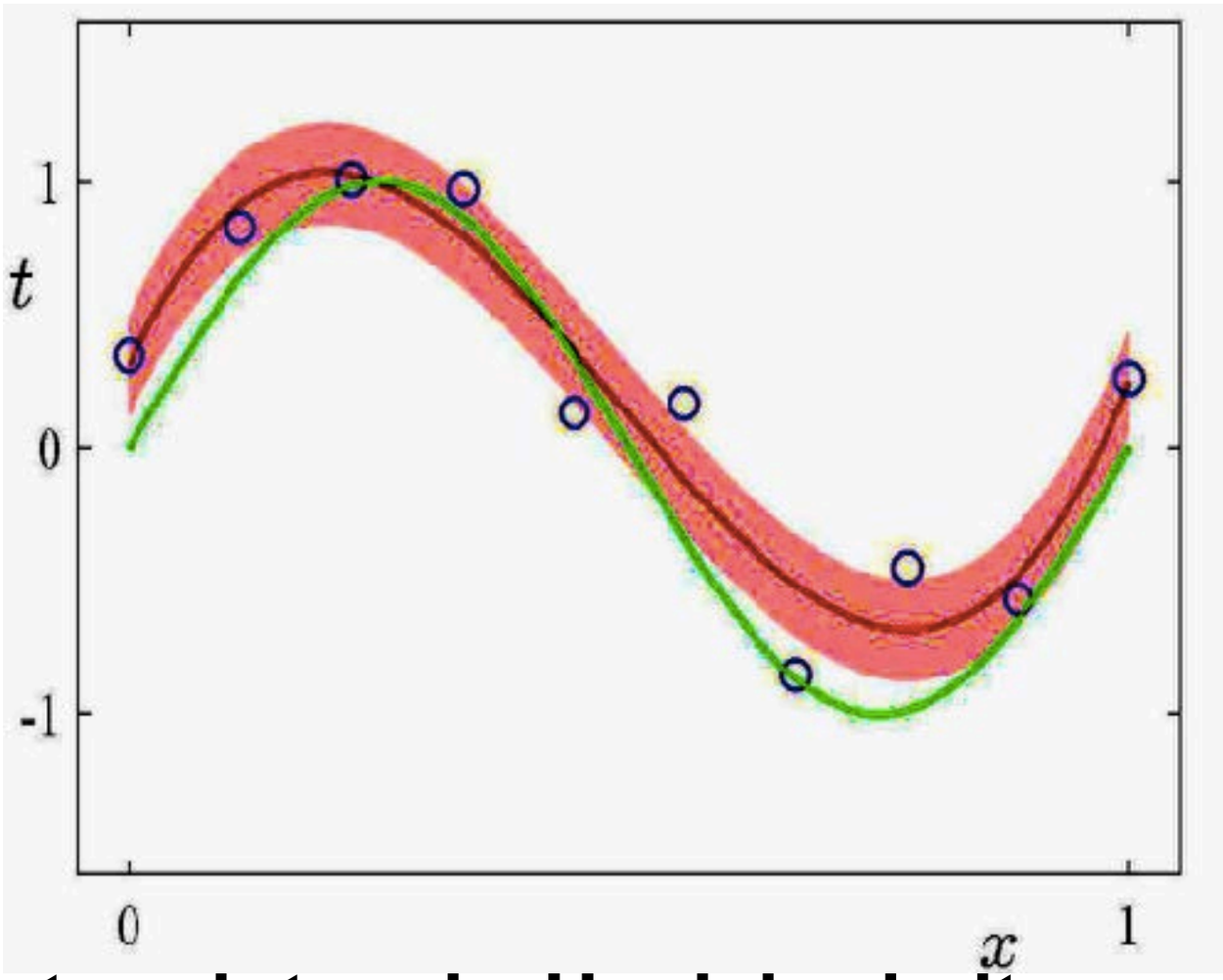


Bayesian vs. Maximum Likelihood Predictive Distribution

$$p(t|x, \mathbf{x}, \mathbf{t}) = \mathcal{N}\left(t|m(x), s^2(x)\right)$$

$$p(t|x, \mathbf{w}_{\mathrm{ML}}, \beta_{\mathrm{ML}}) = \mathcal{N}\left(t|y(x, \mathbf{w}_{\mathrm{ML}}), \beta_{\mathrm{ML}}^{-1}\right)$$





Illustrations from Bishop, machine learning

— der volle Bayes'sche Ansatz zeigt mehr Unsicherheit, da die Unsicherheit in den Parametern auch modelliert ist !

— im Bayes-Ansatz hängt Unsicherheit vom Input (x) ab !

Die a-Priori Verteilung (Prior)

Prior modelliert Vorwissen/Vorannahmen

- typische a-priori Verteilungen $p(\mathbf{w}|\alpha)$:
  - "flach" (= uniform) — aller Parameter gleichwahrscheinlich d.h. kein Vorwissen, kann durch Wahl α >>1 modelliert werden
  - Gaussverteilung mit Varianz α :

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}\right\}$$

normalization factor $1/\mathcal{N}$

  - drückt die Annahme "kleine Parameter" aus
    - α ist Hyperparameter
- vorher berechnete posterior P(w | D), wenn "neue" Daten beobachtet werden (sog. interatives/incrementelles Bayes'sches Lernen)

Inkrementelles Bayes'sches Lernen

Wenn Daten sequentiell beobachtet werden, wiederhole den Inferenzschritt durch Anwendung der Bayesregel für jeden neuen Datenpunkt oder Datensatz $D_1, D_2, D_3, \ldots$ **:**

$$P(w|D_1) = \frac{P(D_1|w)P(w)}{P(D_1)}$$

$$P(w|D_2) = \frac{P(D_2|w)P(w|D_1)}{P(D_2)}$$

$$P(w|D_{k+1}) = \frac{P(D_{k+1}|w)P(w|D_k)}{P(D_{k+1})}$$

$\ldots$

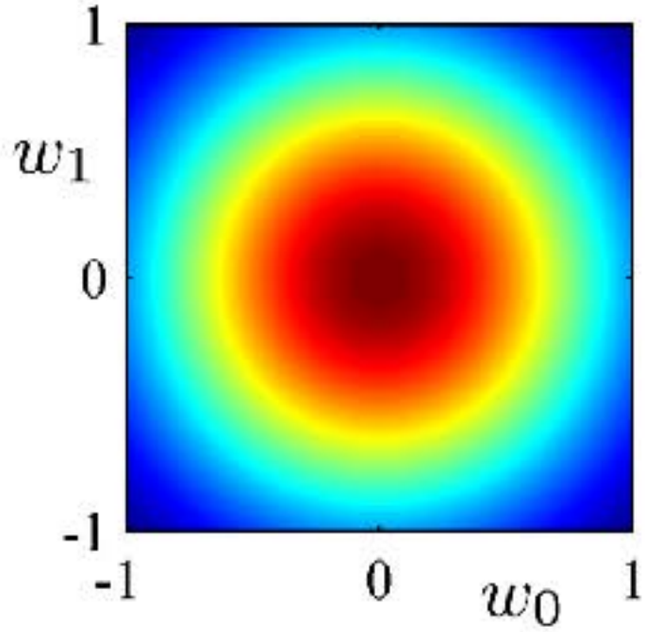
Hier wird der letzte Posterior zum neuen Prior !

Illustration: inkrementelles Bayes'sches Lernen

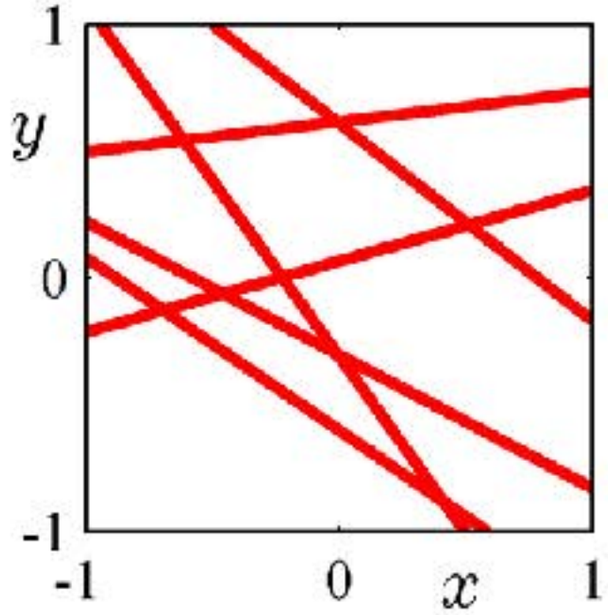
Datenmodell:
$y = w_0 + w_1 x$

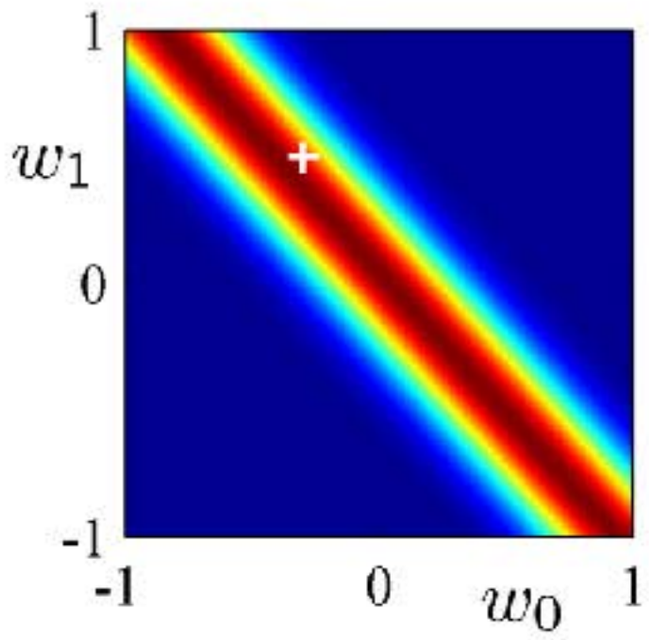
likelihood

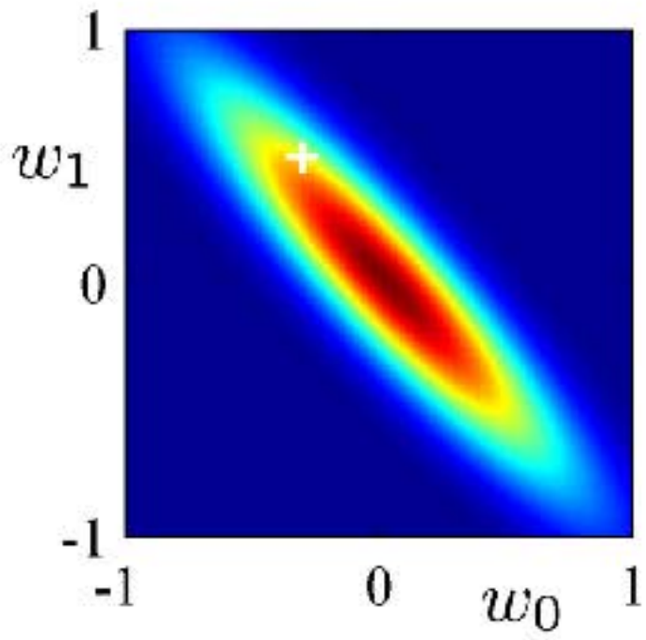
prior/posterior

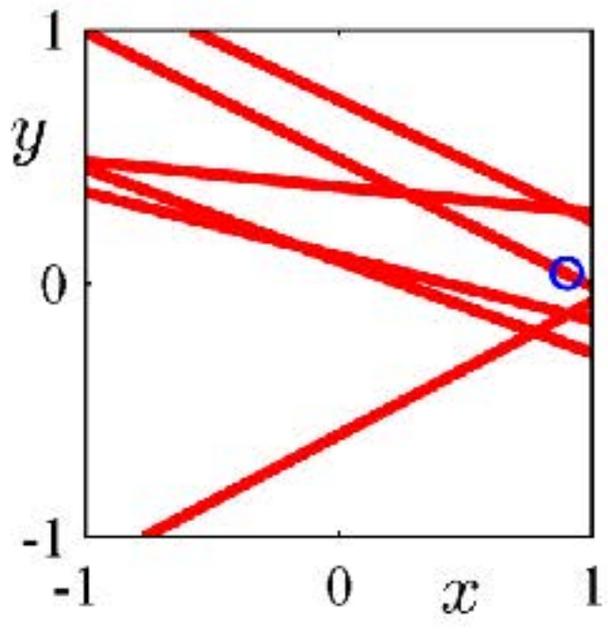


data space



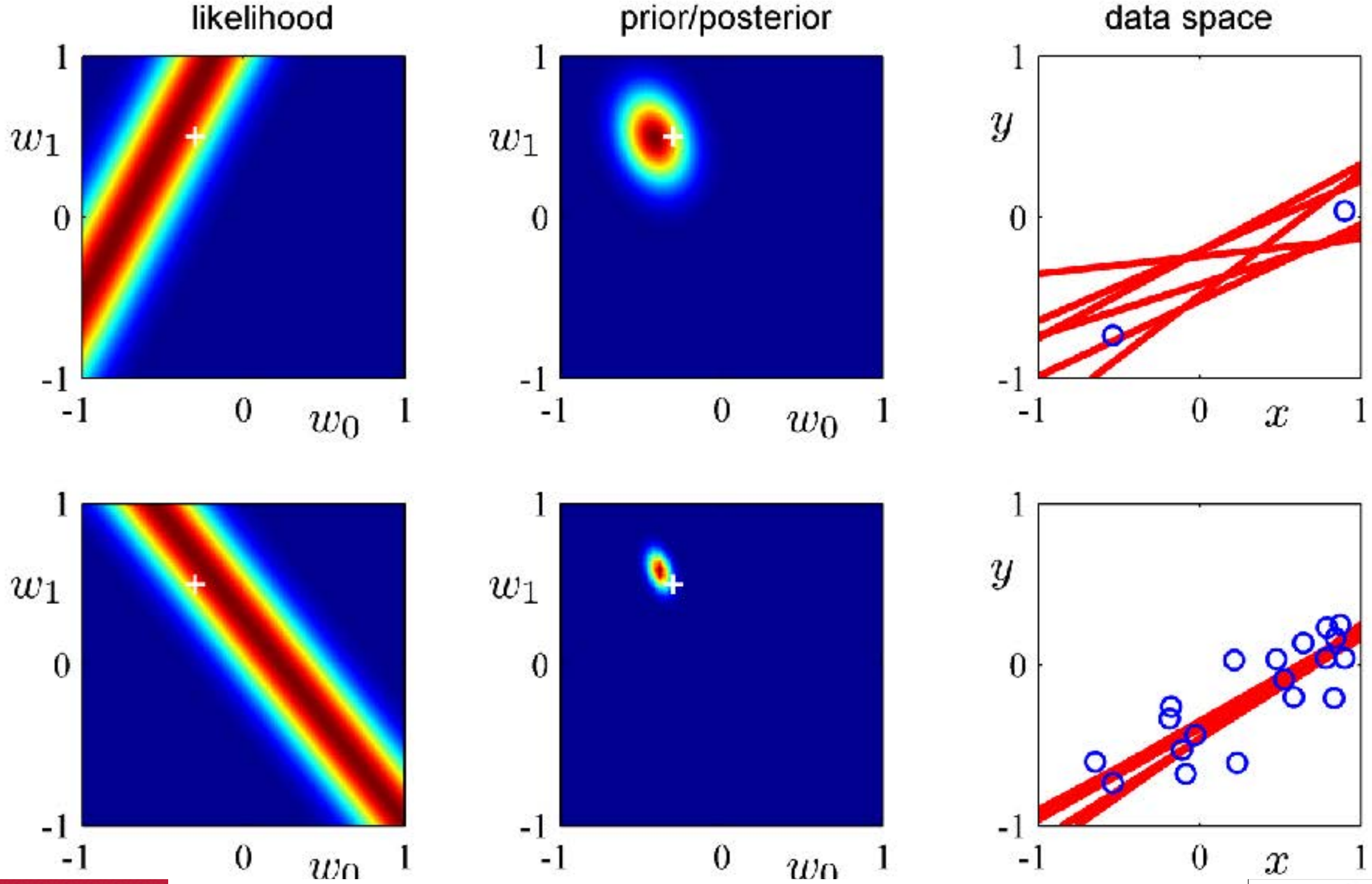






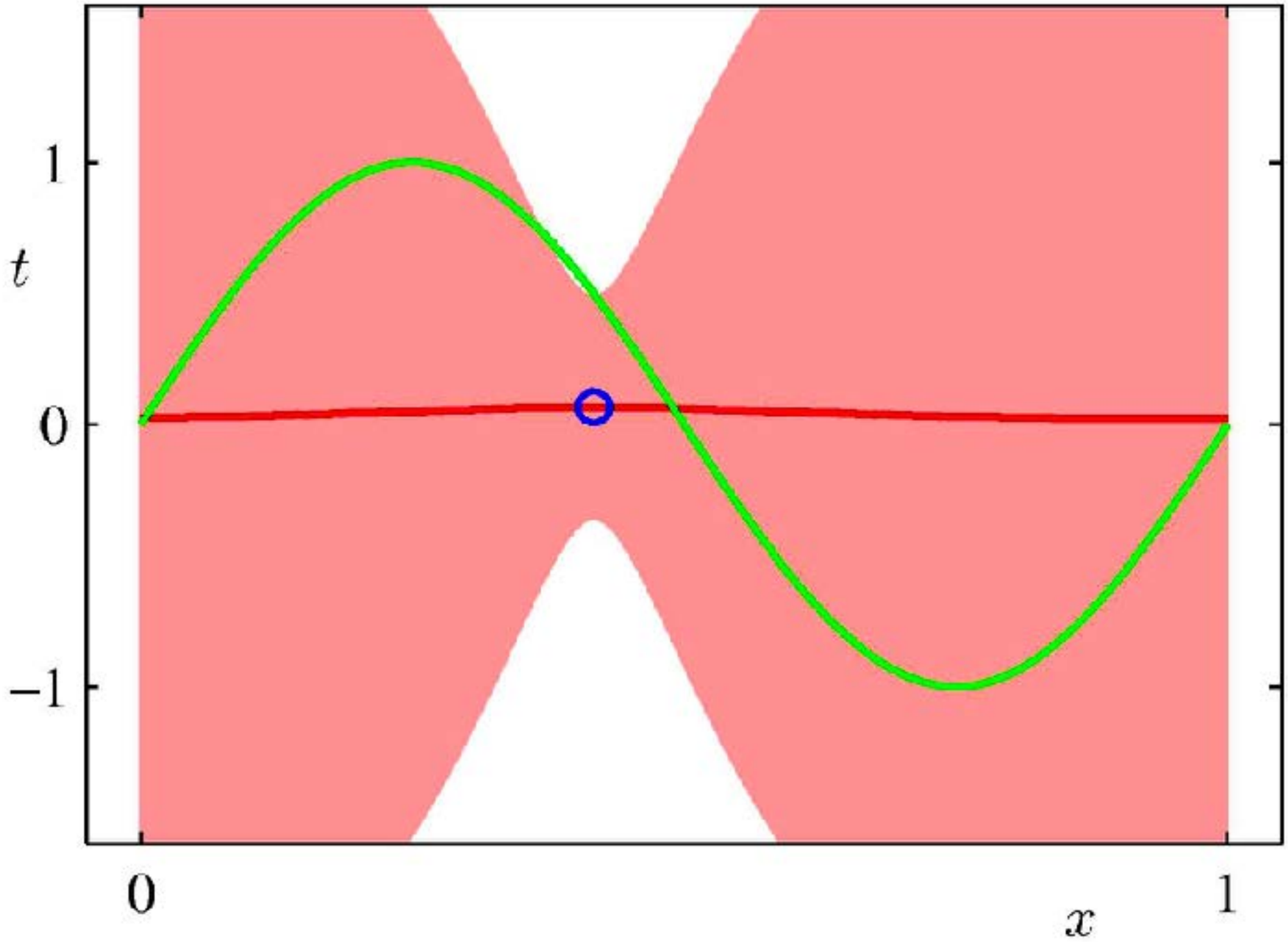
Slide from Bishop, machine learning

Illustration: inkrementelles Bayes'sches Lernen



Slide from Bishop, machine learning

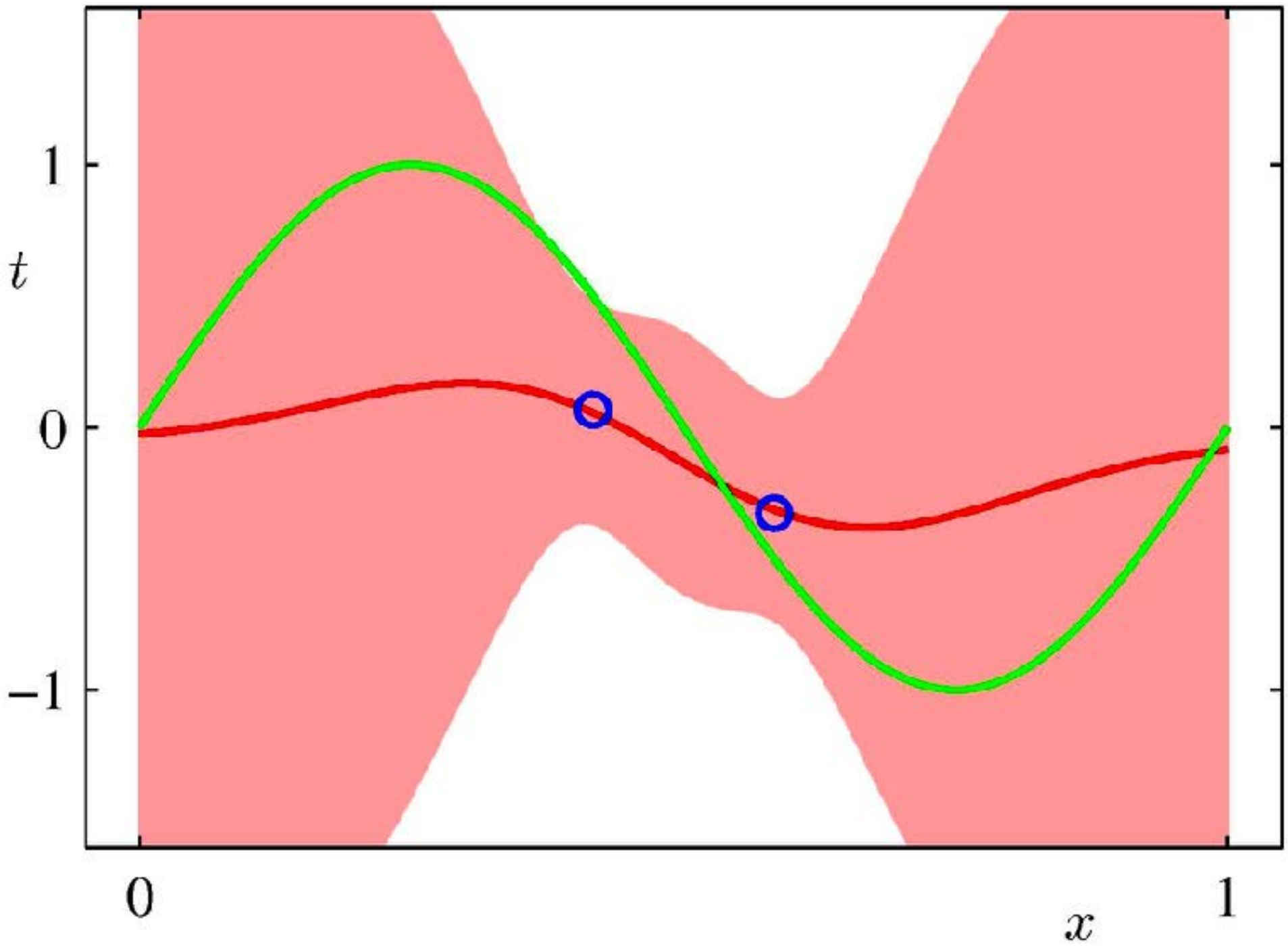
Sequential (incremental) Bayesian learning für sin()



Slide from Bishop, machine learning

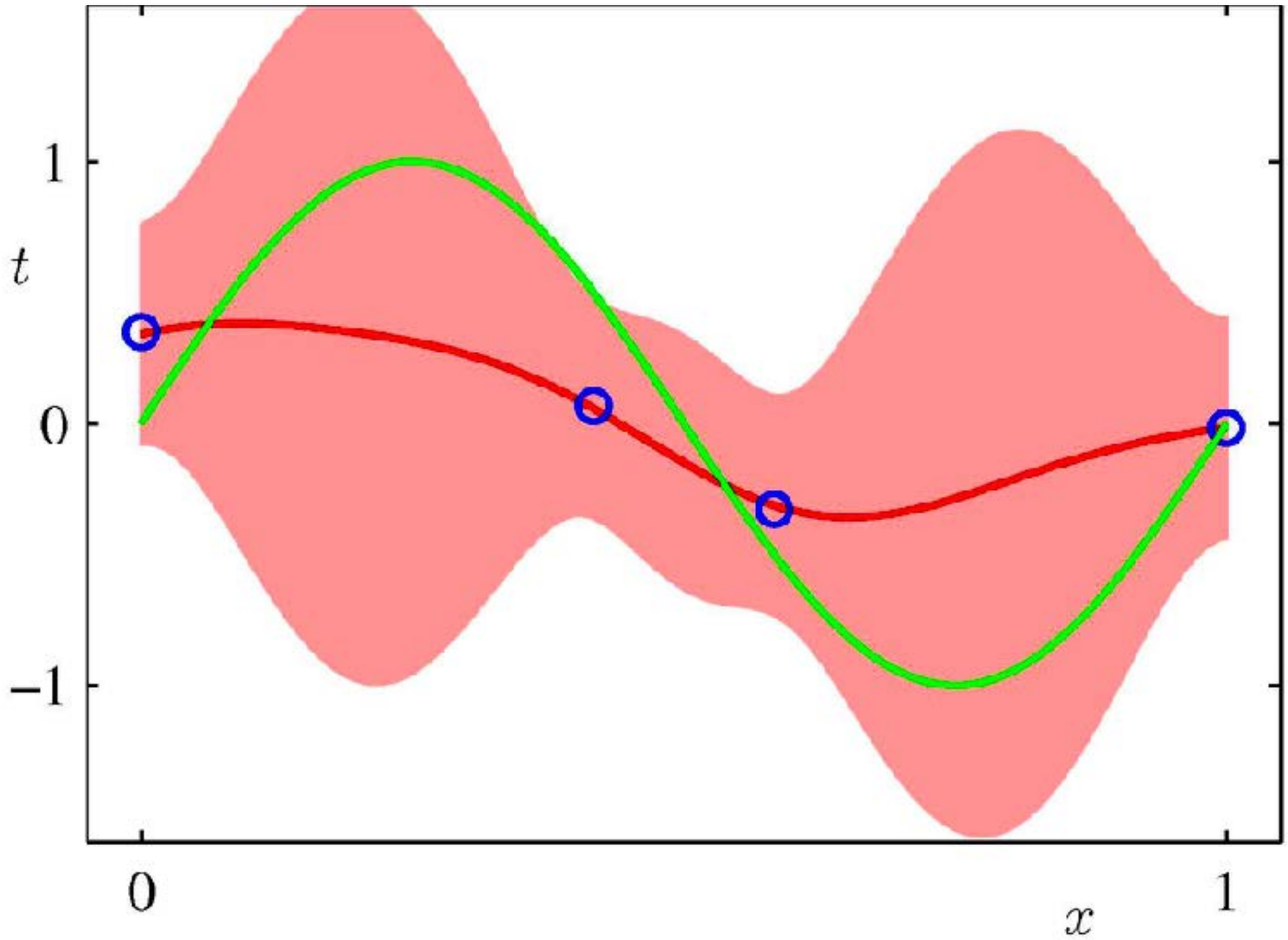
— im Bayes-Ansatz hängt Unsicherheit vom Input (x) ab !

Sequential (incremental) Bayesian learning



Slide from Bishop, machine learning

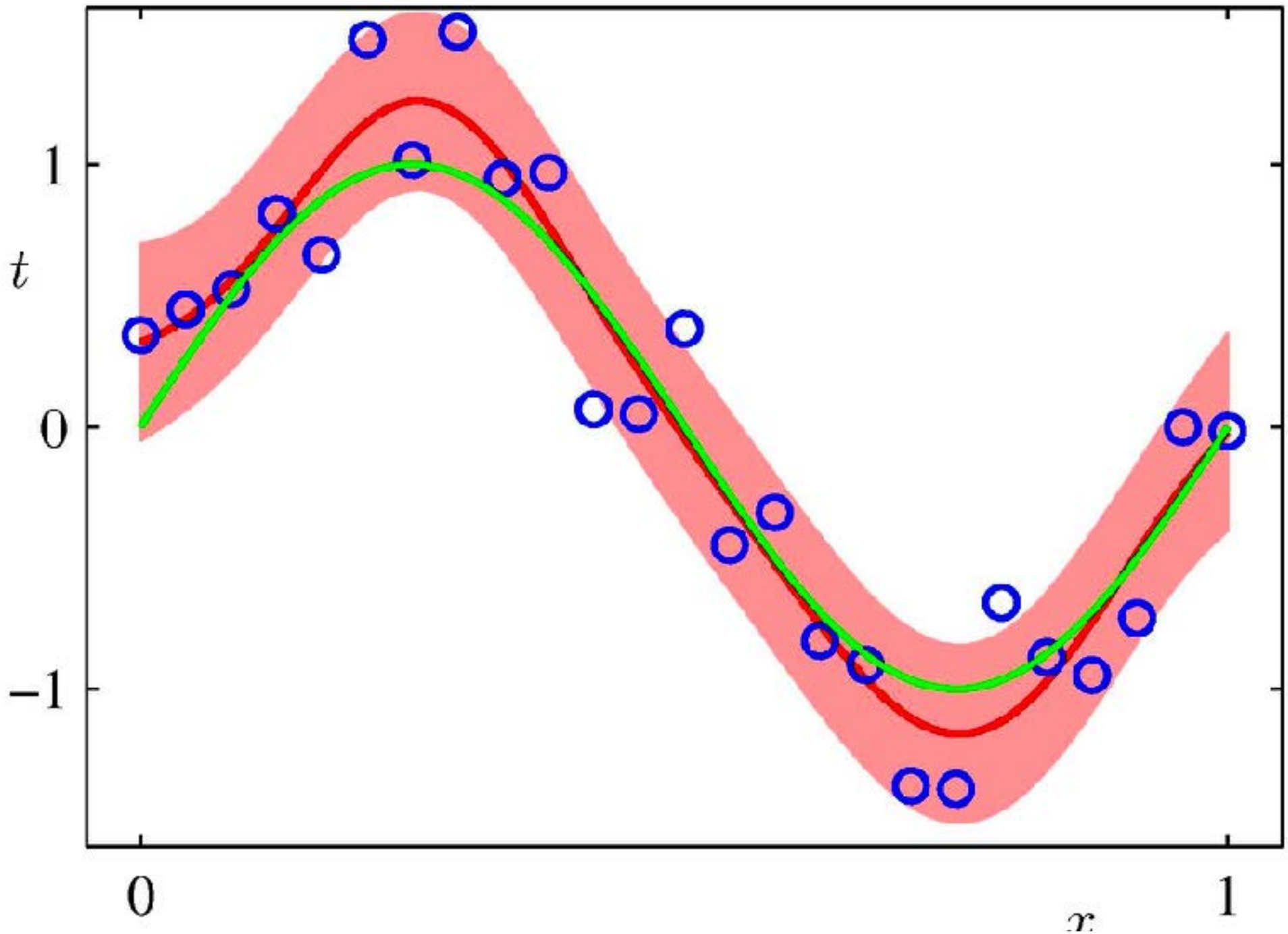
Sequential (incremental) Bayesian learning



Slide from Bishop, machine learning

Sequential (incremental) Bayesian learning

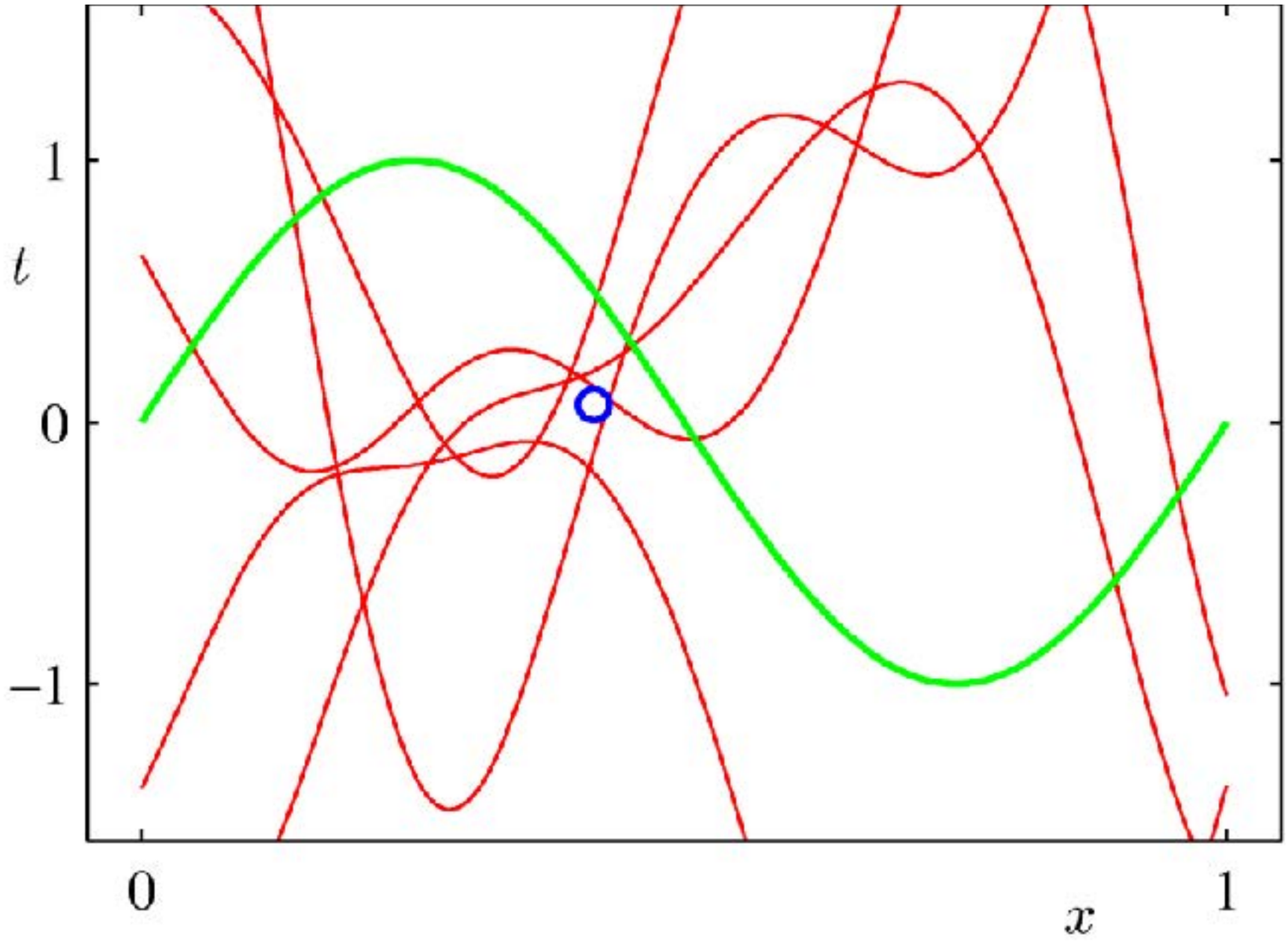
Slide from Bishop, machine learning



Sampling von der a posteriori Verteilung

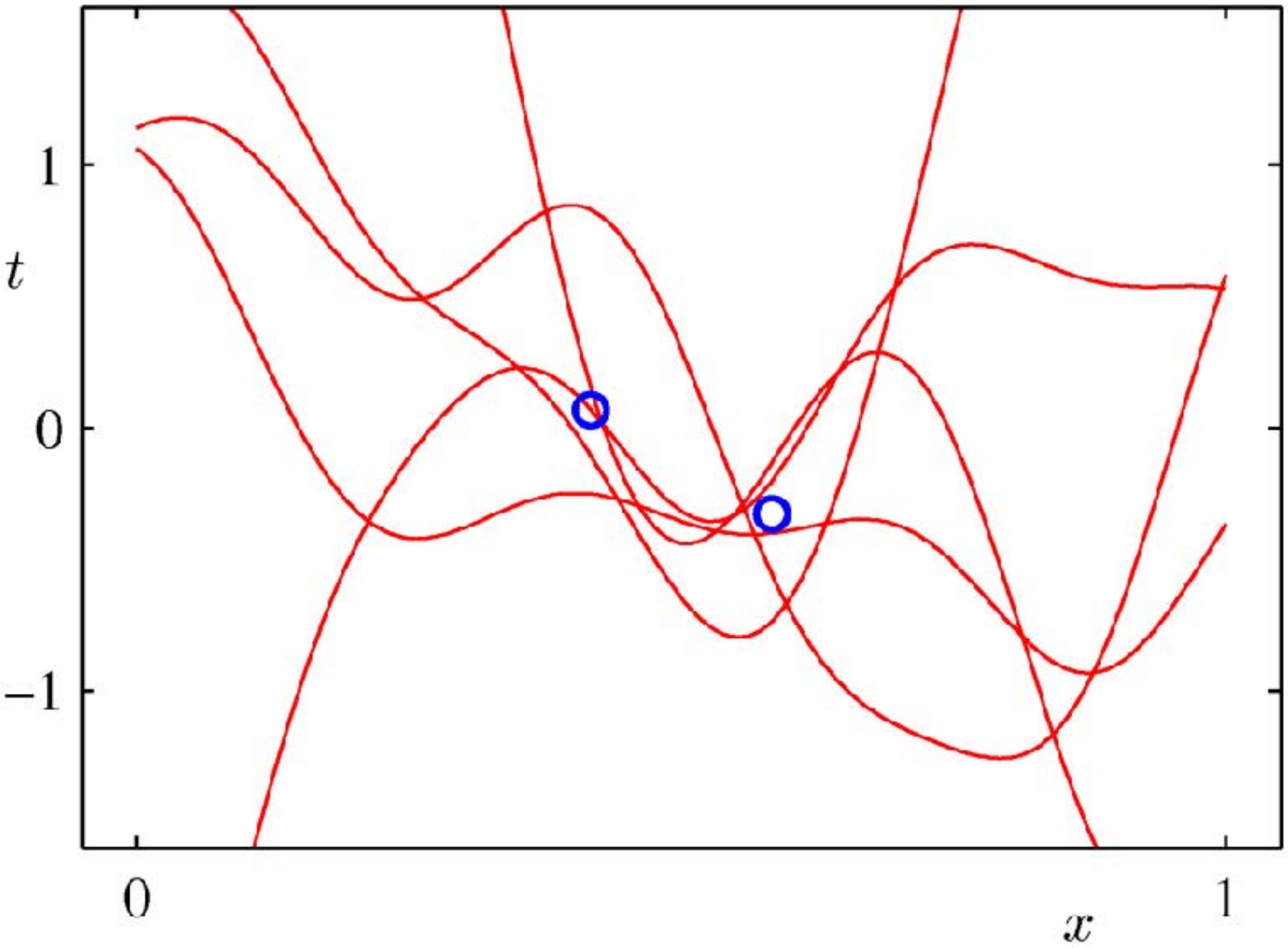
Vorgehen: **“ziehe” Parameter aus posterior P(wlD) (mehrfach)**

- **zeichne die entsprechende(n) Funktion(en)**



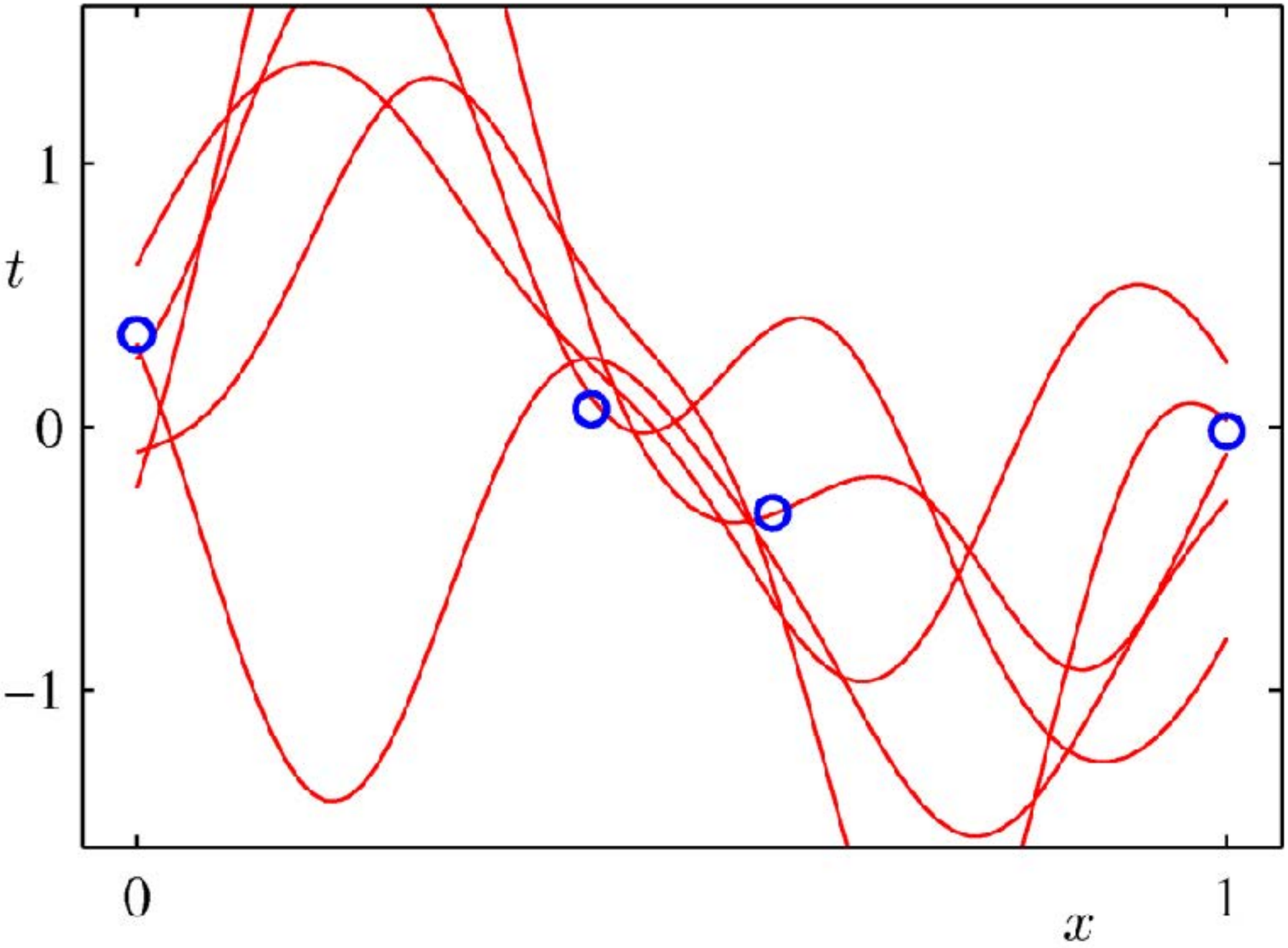
Slide from Bishop, machine learning

Sampling von der a posteriori Verteilung



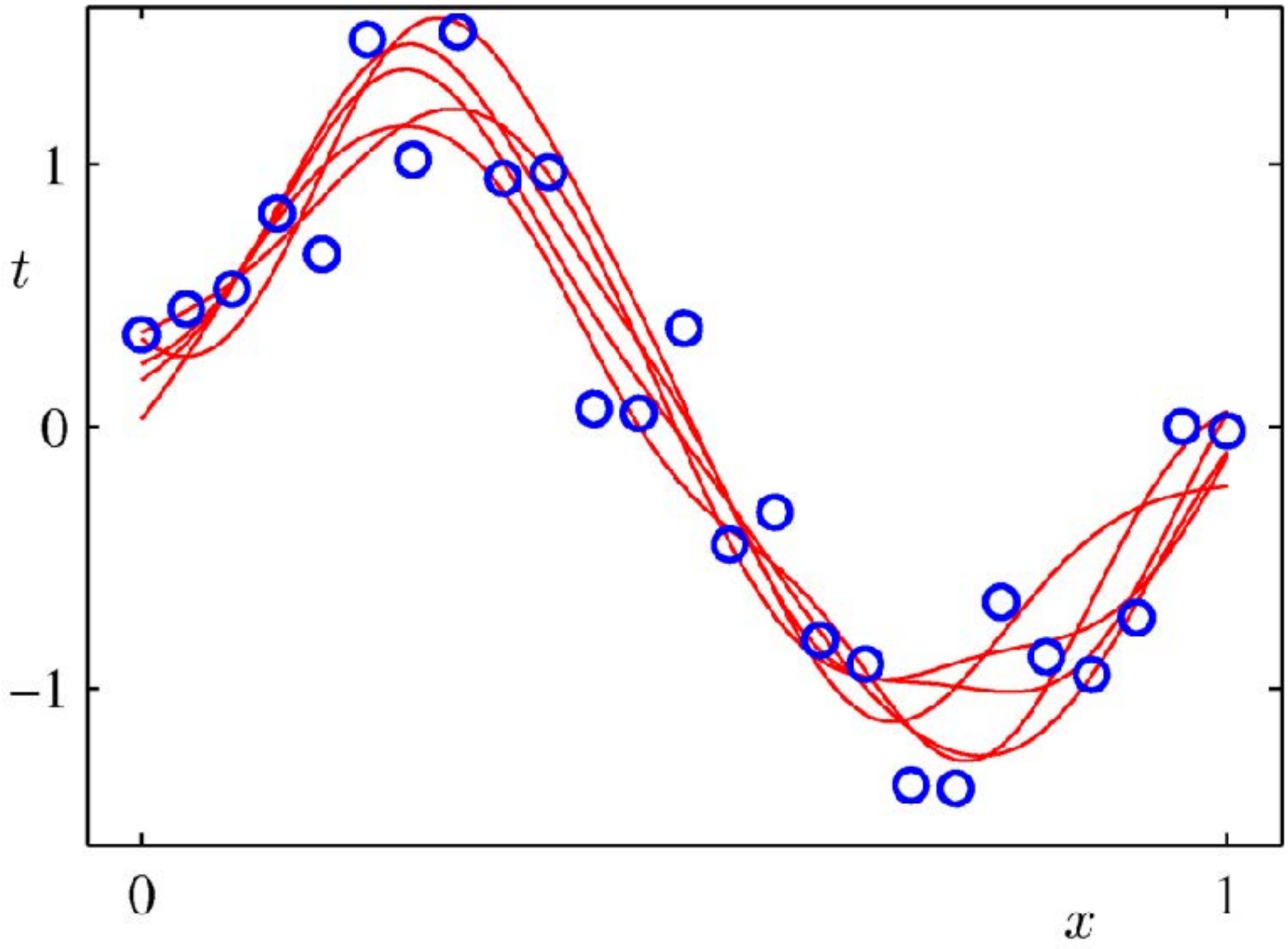
Slide from Bishop, machine learning

Sampling von der a posteriori Verteilung



Slide from Bishop, machine learning

Sampling von der a posteriori Verteilung



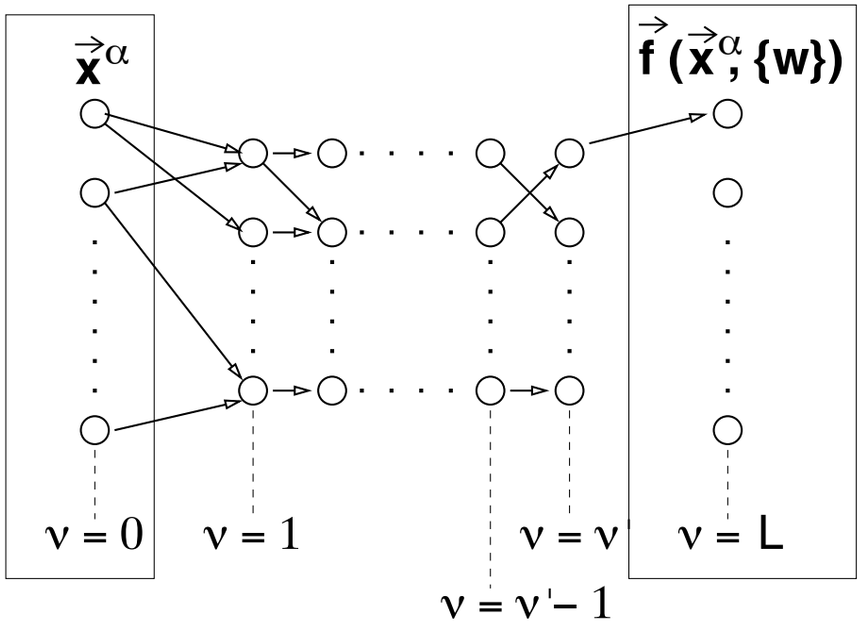
Slide from Bishop, machine learning

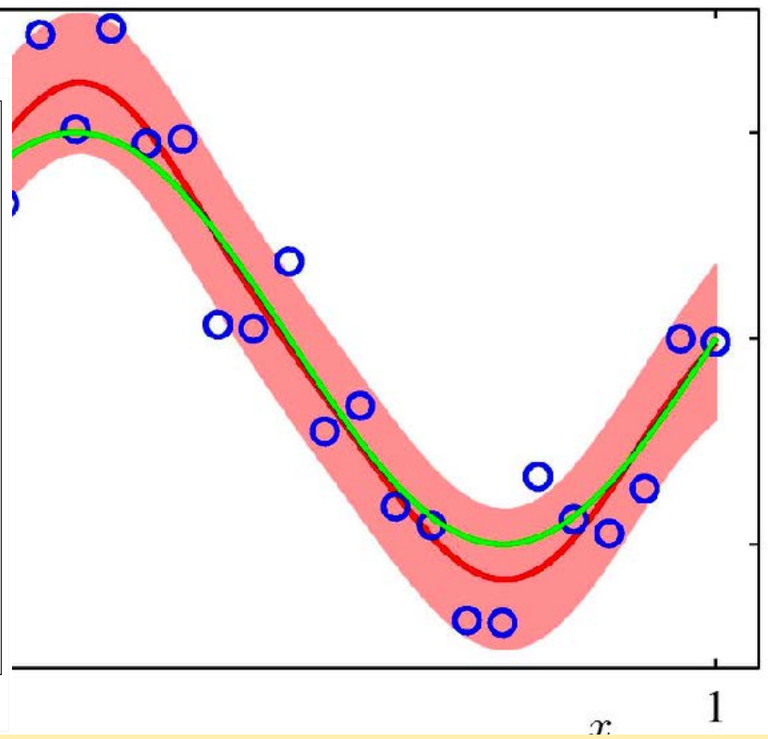
Take home:

Unsicherheit

- Rauschen/Unsicherheit in Daten & Parametern ~ Normalverteilung
- Generalisierung durch *predictive distribution:*
  - wahrscheinlichster Wert (Mittelwert)
  - Konfidenz gegeben durch die Präzision (Inverse der Varianz)
  - drei Varianten der *predictive distribution:*
    - nur mit optimalem ML-Parameter - Datenunsicherheit
    - mit optimalem MAP - Daten- & Parameterunsicherheit
    - “full” predictive distribution: Daten- und mittlere Parameterunsicherheit

Technische Universität Braunschweig





predictive distribution:

$$p(t|x,\mathbf{x},\mathbf{t}) = \int p(t|x,\mathbf{w})p(\mathbf{w}$$

$$m(x) = \beta\boldsymbol{\phi}(x)^{\mathrm{T}}\mathbf{S}\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)t_n$$

$$\mathbf{S}^{-1} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^{\mathrm{T}}$$

Parameteroptimierung und Modelle

Parameteroptimierung & Modelle: linear vs. nichtlinear

Lernziele:

- Kennenlernen verschiedener Modellansätze
- lineare Modelle vs. nichtlineare Modelle
- jeweils das Vorgehen zur Parameteroptmierung
- Grundlagen Gradientenabstieg

(Mathem.) Voraussetzungen:

- (multi-dim.) Ableitungen, Kettenregel, Matrizenrechnung

Vorgehen:

- Ansatz Modell linear/nicht-linear
- Berechnung/Suche von Minima der Fehlerfunktion
- Berechnung der Gradienten

linear Modelle = Modelle linear in den Parametern

- ein lineares Modell kann nicht-linear in den Eingaben sein !
- lineare Modelle habe die generelle Form

$$y(\mathbf{x_n}, \mathbf{w}) = \sum_{m=0}^{M} w_m \Phi_m(\mathbf{x_n}) = \mathbf{w}^T \Phi(\mathbf{x_n})$$

wobei $\Phi(\mathbf{x_n}) = (\Phi_1(\mathbf{x_n}), \ldots, \Phi_m(\mathbf{x}))^T$

- im Beispiel der Polynomapproximation

$$\Phi_m(x) = x^m$$

- $\Phi_m(x)$ kann eine beliebige (nichtlineare) Funktion der Eingabe sein

Beispiele:

polynom $\Phi_j(x) = x^j$

j-th component $\Phi_j(\mathbf{x}) = x_j$

RBF-net $\Phi_j(x) = exp\left[-\frac{||\mu_j - x||^2}{2\sigma^2}\right]$

neural net $\Phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right), \sigma(s) = \frac{1}{1 + e^a}$

$\vdots$

if parameters μ_j, s_j chosen randomly:
Extrem Learning Machine (ELM)

Die quadratische Fehlerfunktion ist für ein lineares Modell:

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

Minimierung:

- **bilde den Gradienten**
- **setze = 0 !**
- **löse auf**

für ein lineares Modell erhalten wir das Resultat:

$$\vec{w}_{ML} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\vec{t} = \mathbf{\Phi}^\sharp\vec{t}$$

wobei $\mathbf{\Phi}^\sharp = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T$

die Moore-Penrose Pseudoinverse ist

und

$$\mathbf{\Phi}(X) = \begin{pmatrix} \Phi_1(\mathbf{x_1}) & \cdots & \Phi_M(\mathbf{x_1}) \\ \vdots & \ddots & \vdots \\ \Phi_1(\mathbf{x_N}) & \cdots & \Phi_M(\mathbf{x_N}) \end{pmatrix} \in R^{N\times M}$$

ist die sogenannten Designmatrix

Minimiere Fehler / Maximiere a posteriori für lineares Modell + Regularisierung (äquivalent zu Maximum posterior)

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\Phi(\mathbf{x_n}))^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

berechne Gradienten, setze gleich null, löse

...

Resultat:

$$\vec{w}_{MAP} = \left(\lambda I + \mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\vec{t}$$

(erwünschter) Seiteneffekt:

die Matrixinversion wird numerisch stabiler

alle Schritte zusammen für lineare Modelle

- a-priori Verteilung für kleine Parameter

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) \tag{3.52}$$

and the corresponding posterior distribution over $\mathbf{w}$ is then given by (3.49) with

$$\mathbf{m}_N = \beta \mathbf{S}_N \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t} \tag{3.53}$$

$$\mathbf{S}_N^{-1} = \alpha \mathbf{I} + \beta \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}. \tag{3.54}$$

The log of the posterior distribution is given by the sum of the log likelihood and the log of the prior and, as a function of $\mathbf{w}$, takes the form

$$\ln p(\mathbf{w}|\mathbf{t}) = -\frac{\beta}{2} \sum_{n=1}^{N} \{t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)\}^2 - \frac{\alpha}{2} \mathbf{w}^{\mathrm{T}} \mathbf{w} + \text{const.} \tag{3.55}$$

wir erhalten folgendes analytisches Resultat für die *predictive distribution:*

$$p(t|x,\mathbf{x},\mathbf{t}) = \int p(t|x,\mathbf{w})p(\mathbf{w}|\mathbf{x},\mathbf{t})\,\mathrm{d}\mathbf{w} = \mathcal{N}\left(t|m(x),s^2(x)\right)$$

$$m(x) = \beta\boldsymbol{\phi}(x)^{\mathrm{T}}\mathbf{S}\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)t_n \qquad s^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^{\mathrm{T}}\mathbf{S}\boldsymbol{\phi}(x)$$

$$\mathbf{S}^{-1} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^{\mathrm{T}} \qquad \boldsymbol{\phi}(x_n) = \left(x_n^0,\ldots,x_n^M\right)^{\mathrm{T}}$$

(für Polynommodell)

Problem: Minimiere quadratischen Fehler

- keine analytische Lösung
- suche Minimum iterativ
- meist durch Gradientenabstieg
- notwendig für

— direkte Minimierung des Fehlers als Funktion der Parameter

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} E(\mathbf{w})$$

— Maximum Likelihood durch Fehlerminimierung:

$$\mathbf{w}_{ML} = \text{argmax}_{\mathbf{w}} \log P(D|\mathbf{w})$$
$$\sim \text{argmin}_{\mathbf{w}} E(\mathbf{w})$$

Problem: Minimiere quadratischen Fehler

- keine analytische Lösung
- suche Minimum iterativ
- meist durch Gradientenabstieg
- notwendig für

— direkte Minimierung des Fehlers als Funktion der Parameters + Regularisierung

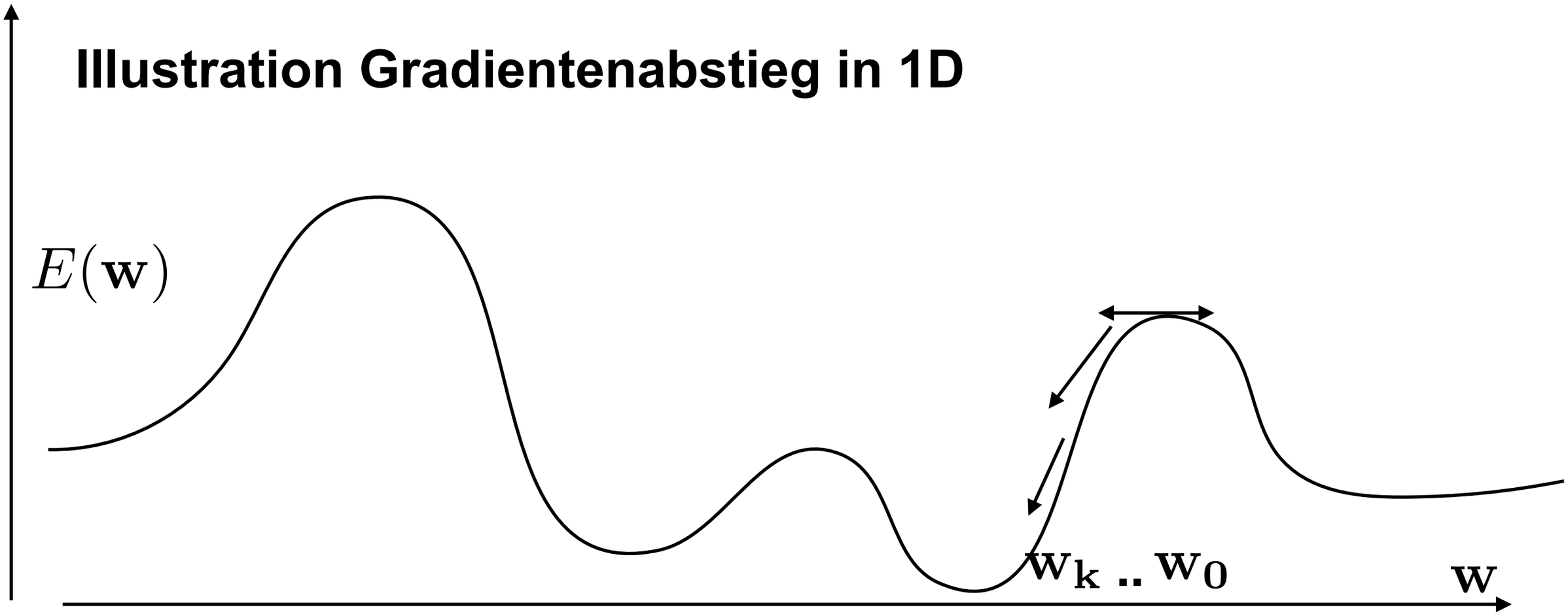
$$\mathbf{w} = \text{argmin}_{\mathbf{w}} \left[E(\mathbf{w}) + \lambda ||\mathbf{w}||^2 \right]$$

— Maximum a-posteriori durch Minimierung des Fehlers + Regularisierung

$$\mathbf{w}_{MAP} = \text{argmax}_{\mathbf{w}} \log P(\mathbf{w}|D)$$

$$\sim \text{argmin}_{\mathbf{w}} \left[E(\mathbf{w}) + \lambda ||\mathbf{w}||^2 \right]$$

Illustration Gradientenabstieg in 1D



Gradientenabstieg: iterative Optimierung, die lokales Minimum findet:

- definiere Startpunkt w_0 (e.g. a good guess or random)

- iteriere $\mathbf{w_{k+1}} = \mathbf{w_k} - \mu \nabla E(w)$

- bis $\nabla E(w) \approx 0$

Gradientenabstieg führt zu einer allgemeinen Lernregel mit "Lernrate" μ:

$$\mathbf{w_{k+1}} = \mathbf{w_k} - \mu \nabla E(w)$$

wobei

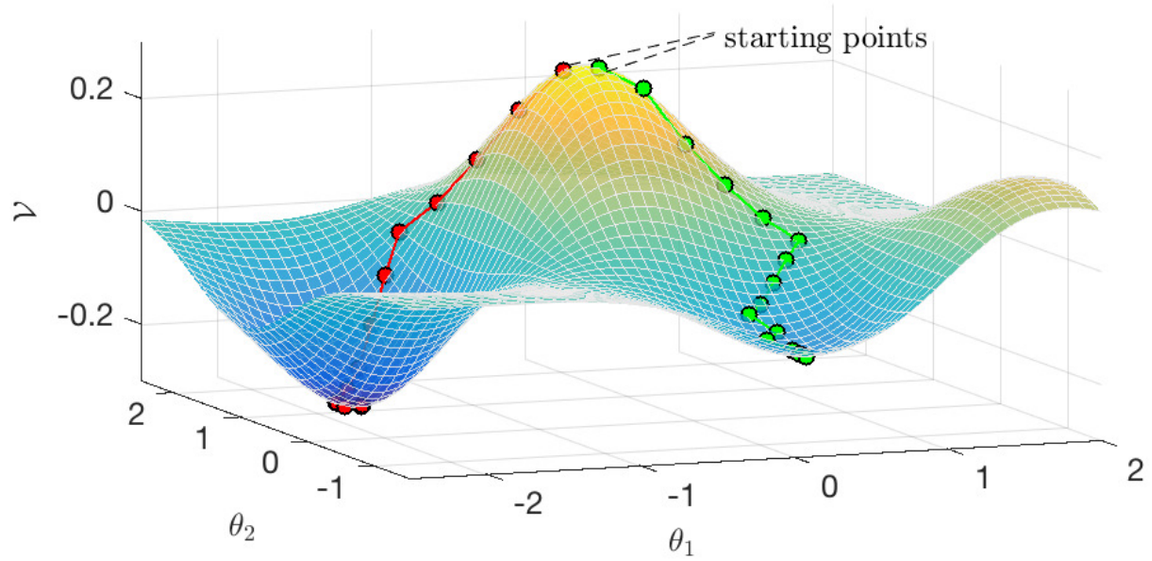
$$E(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(y(x_n, \mathbf{w}) - t_n\right)^2$$

und das Modell $y(x, \mathbf{w})$ kann eine beliebige (nichtlineare) Funktion der Parameter und des Inputs sein.

Gradient Descent für nichtlineare Modellstrukturen

Probleme mit dem Gradient Descent Schritt

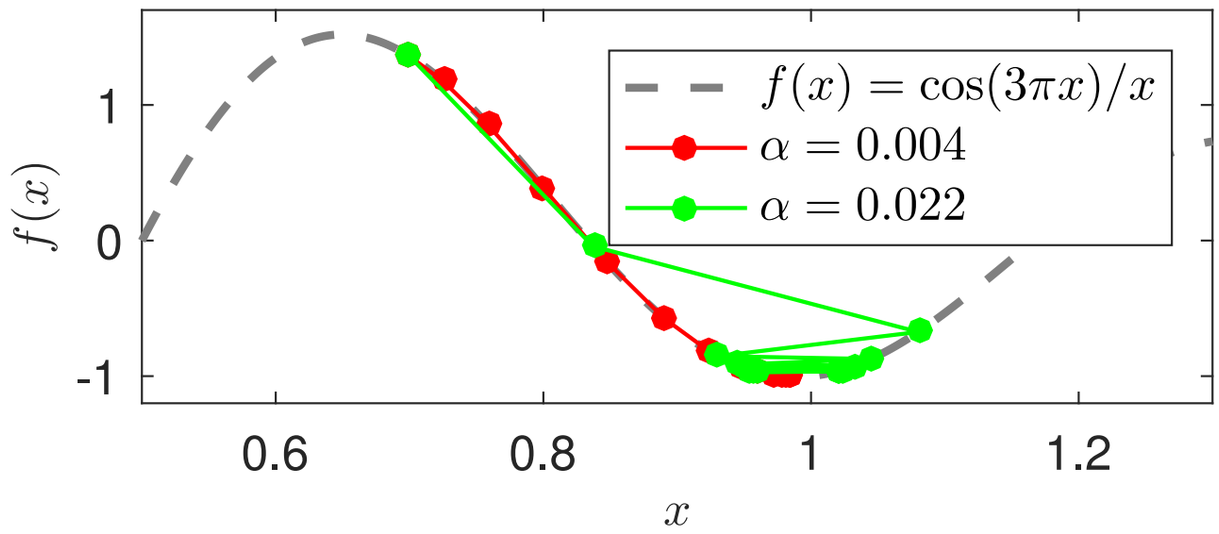
- **alle Beispiele sind Praxisrelevant**
- **Minimum hängt vom Startwert ab**



Schrittweite

- **zu große Schritte kann Minimum verfehlen/oszillieren**
- **zu kleine Schrittweite konvergiert langsam**
- **adaptive Schrittweitenbestimmung**

- **Flache Regionen (Plateaus)**
  - **Kleine Gradienten → Minimum ?**



Gradientenabstieg für nichtlineare Modellstrukturen

Das Problem: Berechnen der Gradienten mittels Kettenregel

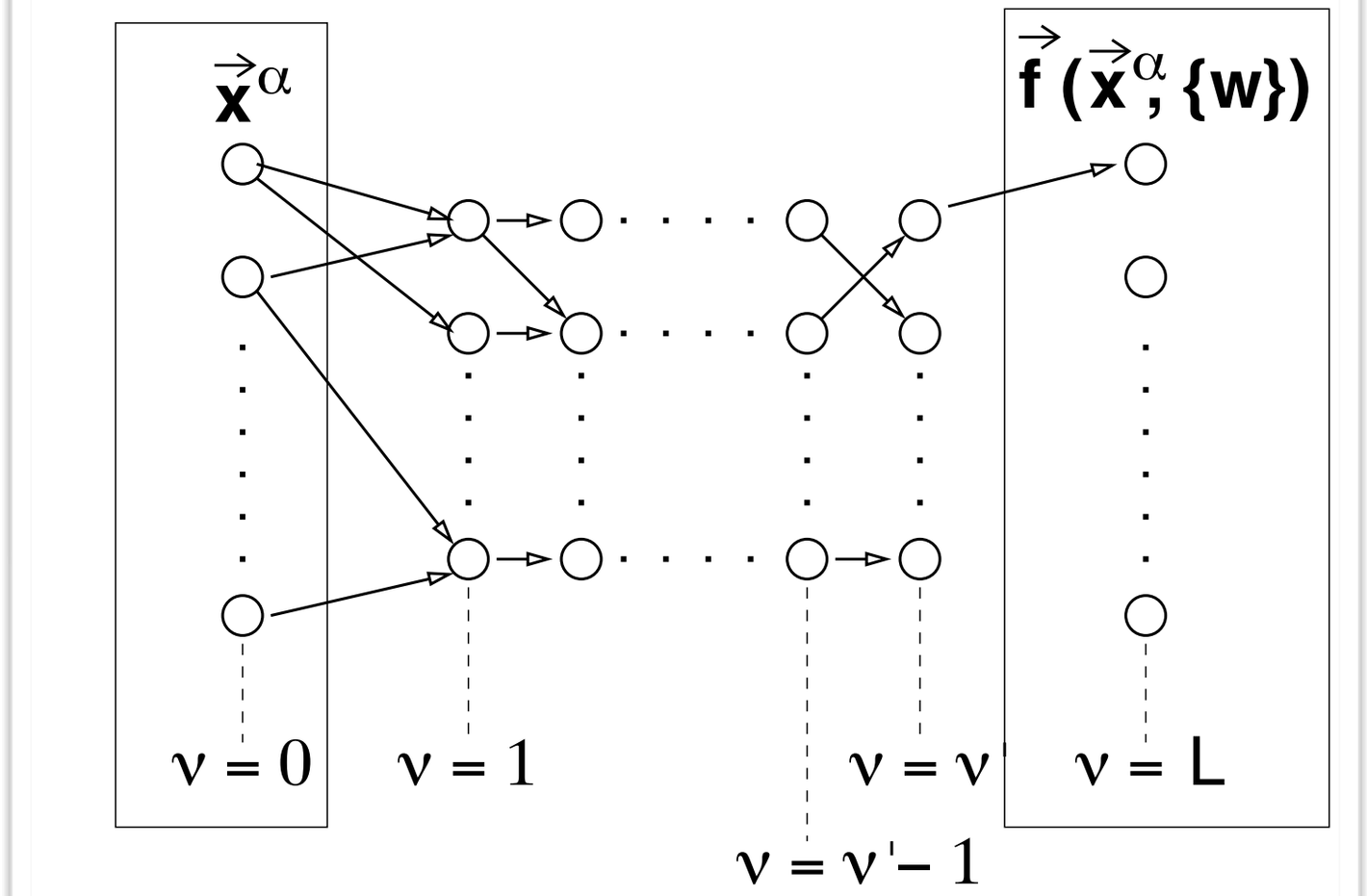
$$\nabla E(w) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$
$$= \frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{2} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2 \right)$$

$$\frac{\partial}{\partial w_i} E(\mathbf{w}) = \sum (y(x_n, \mathbf{w}) - t_n) \frac{\partial}{\partial w_i} y(x_n, \mathbf{w})$$

...

… wenn Modell komplex, muss Kettenregel rekursiv angewandt werden!

Beispiel komplexes Modell: Multilayer Perceptron (MLP)



L = Anzahl der Schichten, L-1 innere Schichten (“hidden layer”)

Für jeden Knoten: $s_i^{\nu+1} = \sigma(\sum w_{ij} s_j^{\nu})$

$$\sigma(a) = tanh(a) \text{ or } \sigma(a) = \frac{1}{1+e^{-a}}$$

Kapazität

- Jede kontinuierliche Funktion kann theoretisch mit jeglicher Genauigkeit angenähert werden (Satz von Funahashi)
- Die Anzahl notwendiger Neuronen (Knoten) ist aber unbekannt

konkrete Modellparameter

- Anzahl der Schichten
  - eine innere Schicht (“hidden layer”) ist ausreichend
  - häufig funktionieren 2 Schichten aber besser
  - modern: “deep learning” (viele, spezielle Schichten)
- Anzahl der Neuronen (bestimmt Modellkomplexität)
- Overfitting ist häufig ein Problem (Regularisierung notwendig!)
- Vorverarbeitung der Trainingsdaten (Features) ist wichtig

Berechnung der Gradienten mittels Backpropagation

- Rekursive Formel zur Berechnung der Gradienten bzgl. der inneren Parameter w_{ij}
- Analytischer Ausdruck vorhanden (direkte Lösung möglich)
- Geringer Rechenaufwand
- Vollständige Form gegeben in Bishop, Kapitel 5.3

- Backpropagation = Methode zur Berechnung von Gradienten, => funktioniert auch für andere Modelle als MLP

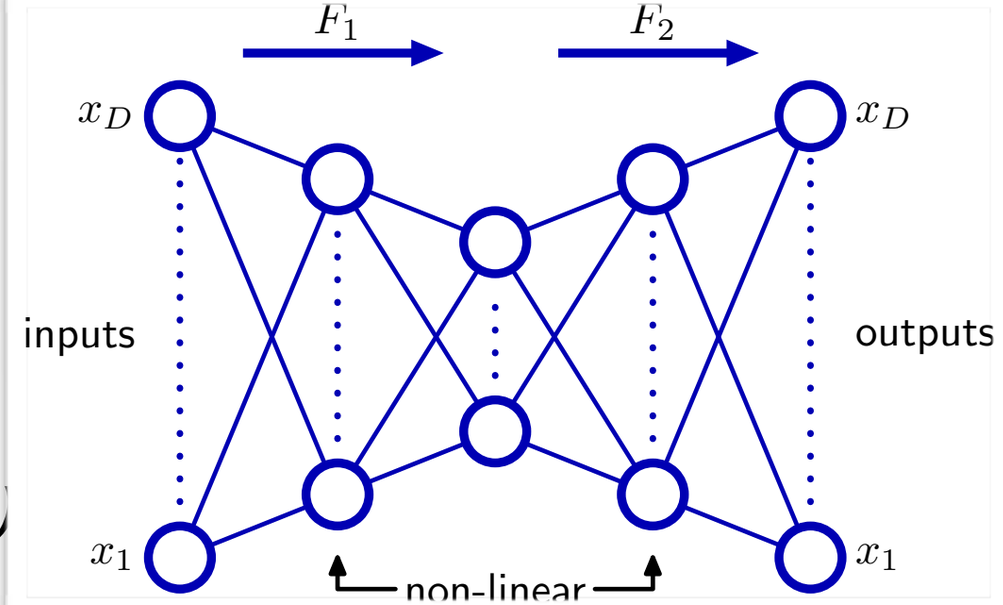
Verschiedene Entwicklungen

Deep learning

- **Viele Layer**
- **Vortrainieren (unterschiedlichste Algorithmen)**
- **Backpropagation anwenden**

Ersten Layer berechnen *Features (Vorverarbeitung)*

Bsp: “Deep Autoencoder"

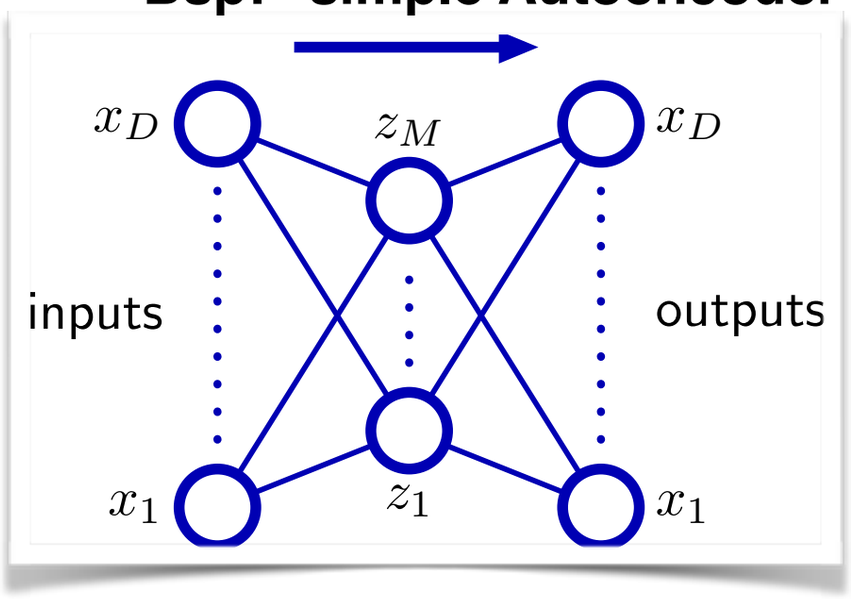


Extreme Learning Machines

Ein einziger, zufälliger, verborgener Layer

Lineare Regression

Bsp: “simple Autoencoder”



Deep-Learning Beispiel: Bildverarbeitung

Convolutional Networks

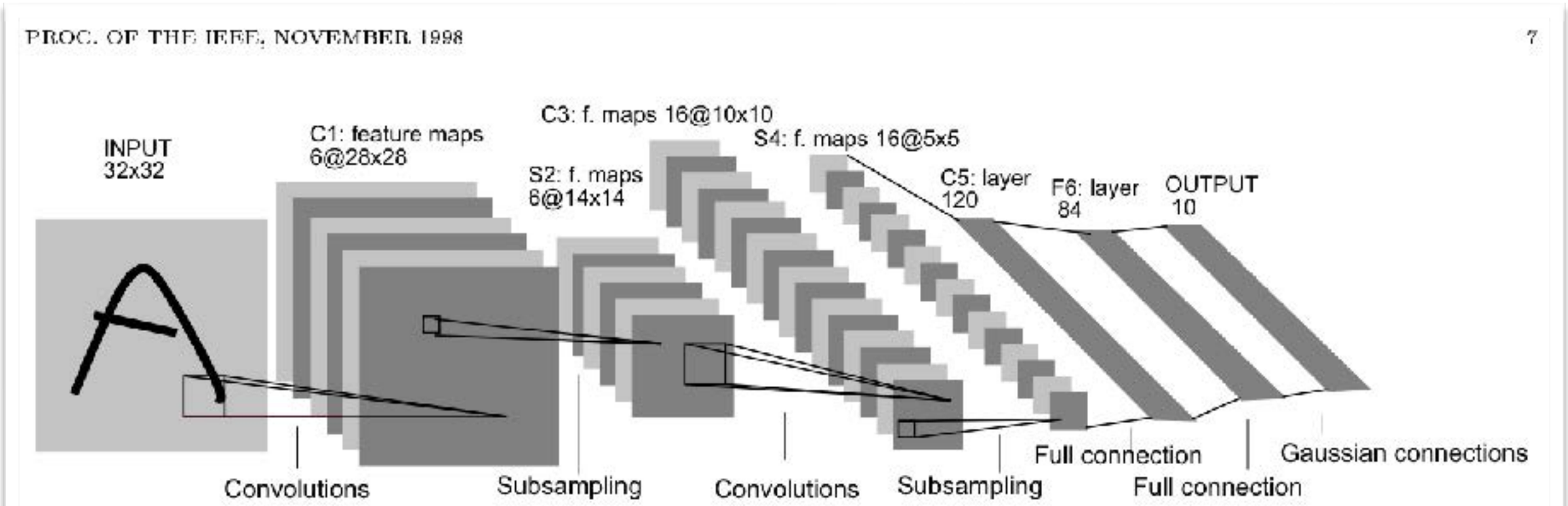
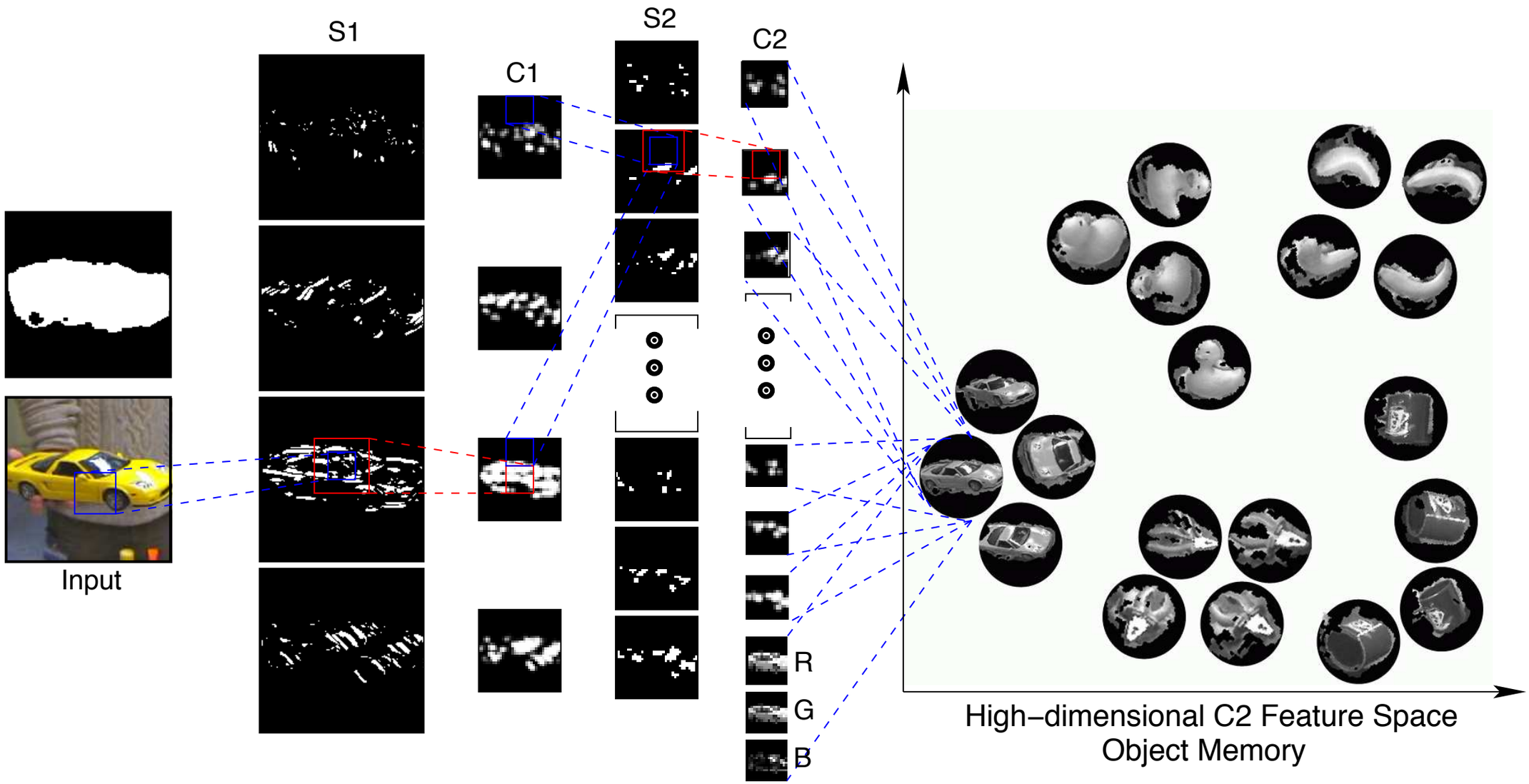


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

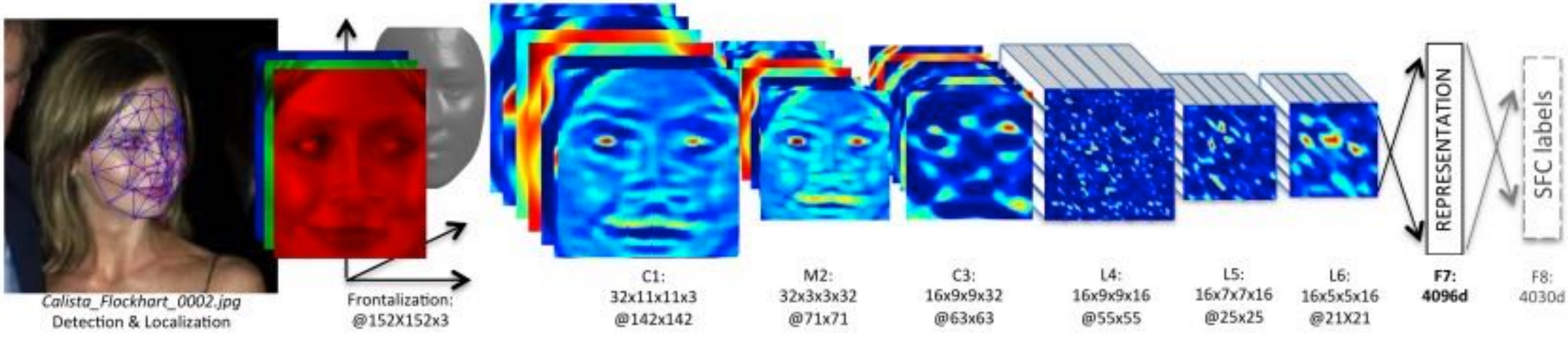
LeCun, 1998 (& early version 1989)

Deep-Learning Beispiel: Bildverarbeitung



Wersing et al., Int. J. Neural Systems, 2007; Wersing & Körner, Neural Computation, 2003

Beispiel: Deep Face Gesichtserkennung



- **“klassische Bildverarbeitung” notwendig**
- **besser als Menschen auf trainierten Gesichtern**
- **erkennt aber nichts, außer den trainierten Gesichtern**
- **(noch) *hohe Kosten für Konfiguration & Training***
- ***Daten allein (Bilder) beantworten keine Fragen !***

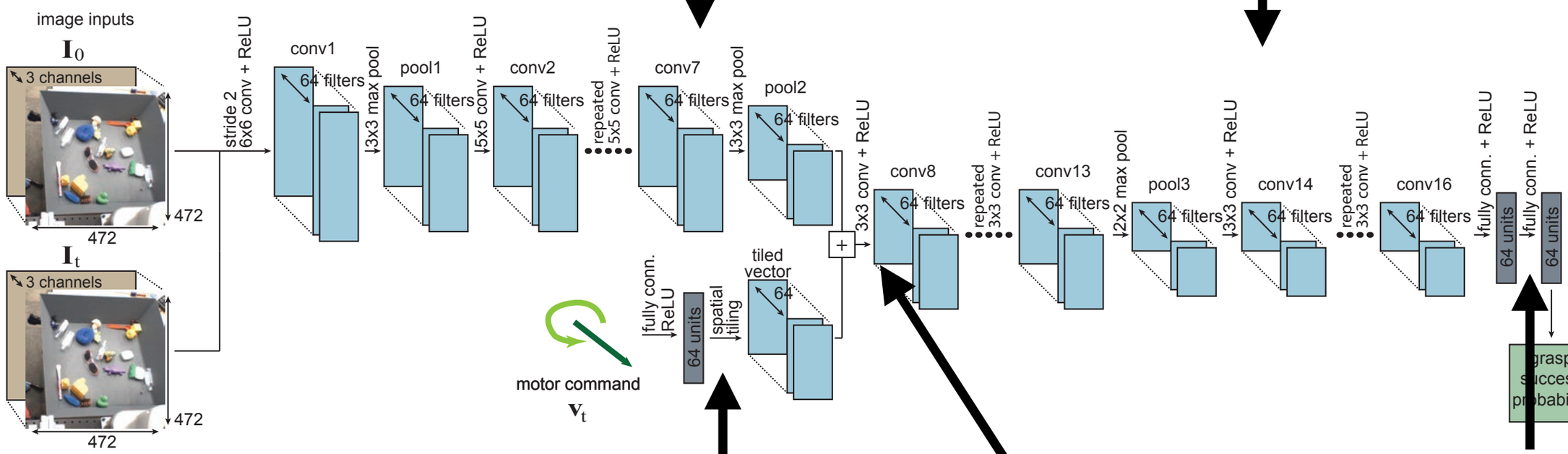
“ DeepFace: Closing the Gap to Human-Level Performance in Face Verification”, Facebook AI Research, 2014

Deep End-to-End Learning in Robotics

Architecture

- **5 convolutional layers to process the image(s) at time t**

more convolutional layers



encode motor command spatially (to help network use it)

fuse motor and image data

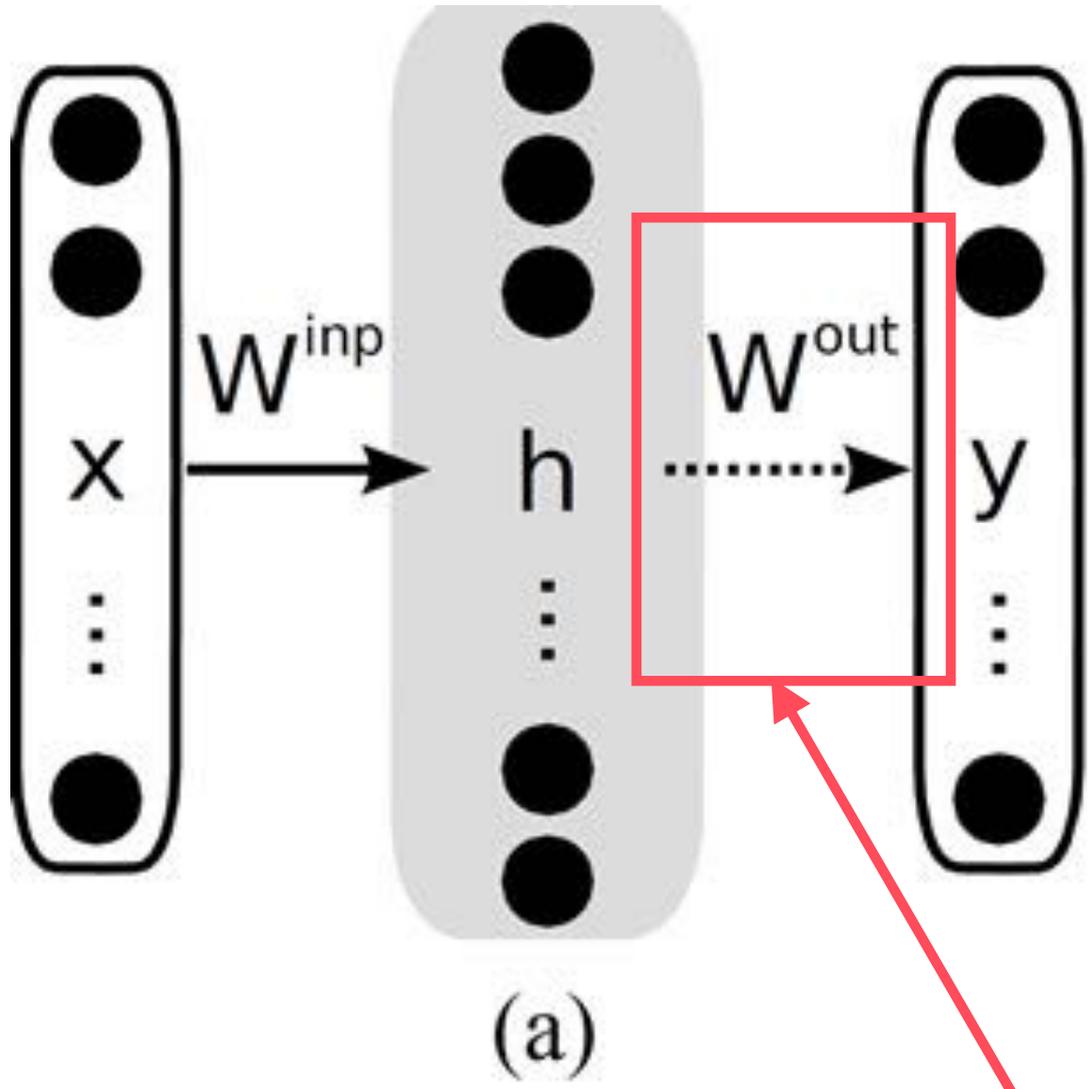
standard MLP trained with cross entropy output: p(grasp sucess)

Deep End-to-End Learning in Robotics

Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection

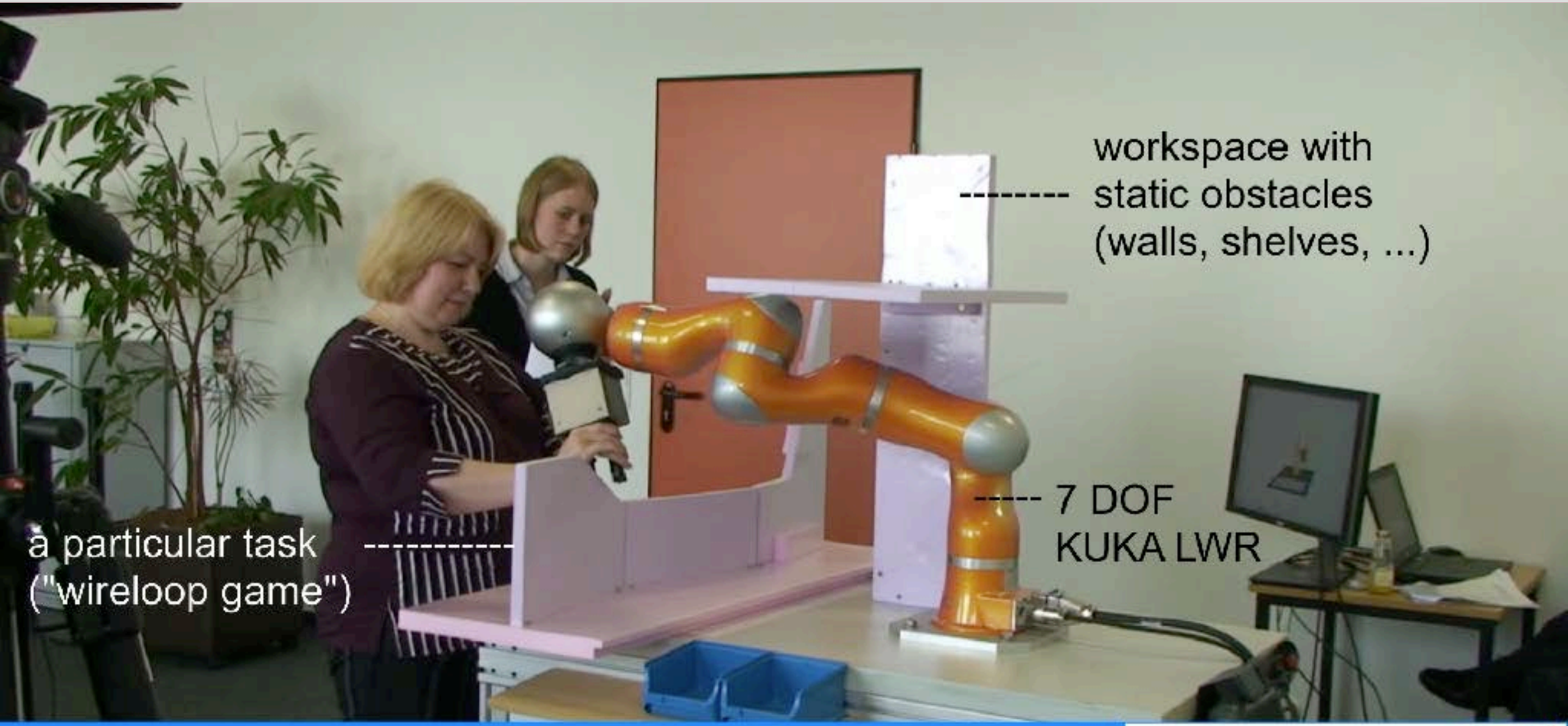
Sergey Levine Peter Pastor
Alex Krizhevsky Deirdre Quillen
Google

Beispiel: Extreme-Learning Machine (flach)



fest, zufällig, hoch dimensional, linear regression

**Neumann, K., and J. J. Steil,
"Optimizing Extreme Learning Machines via Ridge Regression and Batch Intrinsic Plasticity",
Neurocomputing, vol. 102, pp. 23-30, 2013**



Problem Statement

Teaching of redundant robots in confined spaces is difficult.

FlexIRob@HARTING

A user study on physical human-robot interaction