Machine Intelligence 1

1.3 Multi-layer Perceptrons

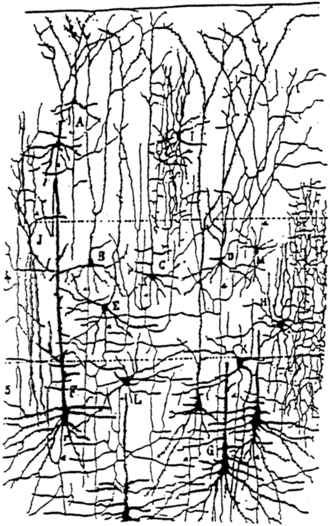
Prof. Dr. Klaus Obermayer

Fachgebiet Neuronale Informationsverarbeitung (NI)

WS 2016/2017

1.3.1 Classes of Neural Networks

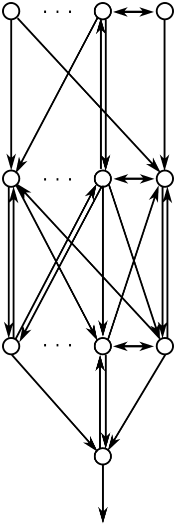
Graphs representing neural networks



Design Principles from Biology

- simple, but highly optimized hardware
  - echolocation in bats
  - sound localization in barn owls
  - ultra fast face recognition
- plasticity
  - synaptic strength
  - lifelong renewal of cells
  - drifting environments
- adaptation in sensory systems
- graceful degradation

Graphs representing neural networks



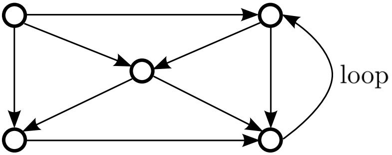
neural network
↓
directed graph

(connectionist) neuron
↓
node of the graph

neural connection
↓
weighted edge

Recurrent Neural Networks (RNNs)

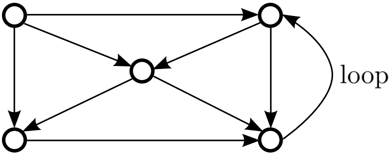
Recurrent networks $\widehat{=}$ directed graphs containing cycles



- dynamical systems
- spatio-temporal pattern analysis
- sequence processing
- associative memory and pattern completion

Recurrent Neural Networks (RNNs)

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



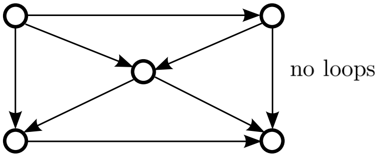
- dynamical systems
- spatio-temporal pattern analysis
- sequence processing
- associative memory and pattern completion

Example-architectures

Hopfield networks (Hopfield, 1982)
Boltzmann machines (Ackley, Hinton & Sejnowski, 1985)
Infinite **I**mpulse **R**esponse (IIR) networks (Crochiere & Oppenheim, 1975)
Long **S**hort-**T**erm **M**emory networks (LSTM, Hochreiter & Schmidhuber, 1997)
Deep Recurrent Neural Networks (Pascanu, Gulcehre, Cyho & Bengio, 2014)

Feedforward Networks (FFN)

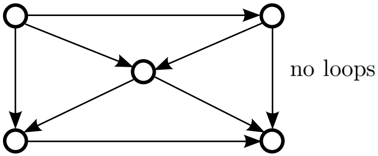
Feedforward Networks $\widehat{=}$ directed acyclic graphs (DAGs)



- association between variables
- prediction of attributes

Feedforward Networks (FFN)

Feedforward Networks $\widehat{=}$ directed acyclic graphs (DAGs)



- association between variables
- prediction of attributes

Example-architectures

Multi**l**ayer-**P**erceptron (MLP, Rumelhart, Hinton & Williams, 1986
Radial **B**asis **F**unction network (RBF, Broomhead & Lowe, 1988)
Support **V**ector **M**achines (SVM, Cortes & Vapnik, 1995)
Deep **B**elief **N**etworks (DBN, Hinton, Osidero & Teh, 2006; Mnih et al., 2016)

Prediction of attributes

real-valued attributes: regression problems

$f : \mathbb{R}^N \to \mathbb{R}$	(or subsets)	one target value
$f : \mathbb{R}^N \to \mathbb{R}^M$	(or subsets)	multivariate attributes

Prediction of attributes

real-valued attributes: regression problems

$f : \mathbb{R}^N \to \mathbb{R}$	(or subsets)	one target value
$f : \mathbb{R}^N \to \mathbb{R}^M$	(or subsets)	multivariate attributes

ordinal attributes: classification problems

$f : \mathbb{R}^N \to \mathcal{S}$	where $\mathcal{S}$ is a set of attributes $\{a_1, a_2, \ldots, a_M\}$
$f : \mathbb{R}^N \to \{-1, +1\}$	special case: two class problems

Prediction of attributes

real-valued attributes: regression problems

$f : \mathbb{R}^N \to \mathbb{R}$	(or subsets)	one target value
$f : \mathbb{R}^N \to \mathbb{R}^M$	(or subsets)	multivariate attributes

ordinal attributes: classification problems

$f : \mathbb{R}^N \to \mathcal{S}$	where $\mathcal{S}$ is a set of attributes $\{a_1, a_2, \dots, a_M\}$
$f : \mathbb{R}^N \to \{-1, +1\}$	special case: two class problems

predicting probabilities

$f_k : \mathbb{R}^N \to [0,1] \subset \mathbb{R}$	probability of attribute a_k from $\mathcal{S}$
s.t. $\sum_{k=1}^{M} f_k(\cdot) = 1$	constrains all $f_k(\cdot)$ to be probabilities

Prediction of attributes

real-valued attributes: regression problems

$f : \mathbb{R}^N \to \mathbb{R}$	(or subsets)	one target value
$f : \mathbb{R}^N \to \mathbb{R}^M$	(or subsets)	multivariate attributes

ordinal attributes: classification problems

$f : \mathbb{R}^N \to \mathcal{S}$	where $\mathcal{S}$ is a set of attributes $\{a_1, a_2, \ldots, a_M\}$
$f : \mathbb{R}^N \to \{-1, +1\}$	special case: two class problems

predicting probabilities

$f_k : \mathbb{R}^N \to [0, 1] \subset \mathbb{R}$	probability of attribute a_k from $\mathcal{S}$
s.t. $\sum_{k=1}^{M} f_k(\cdot) = 1$	constrains all $f_k(\cdot)$ to be probabilities

structured input / structured output

$f : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are sets of structures, e.g. graphs or sentences

see Baklr, Hofmann, Schölkopf, Smola, Taskar and Vishwanathan (Eds.): Predicting structured data, 2007

1.3.2 The Multi-layer-Perceptron for Regression

Overview inductive learning

data representation

↓

model class

↓

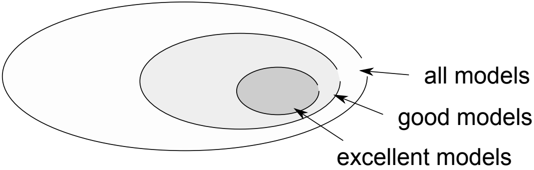
performance measure

↓

optimization

↓

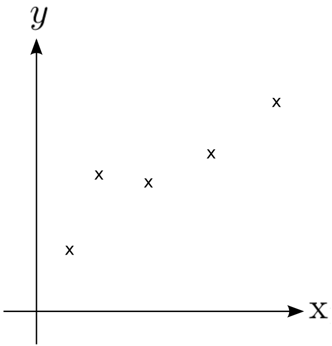
validation



data representation

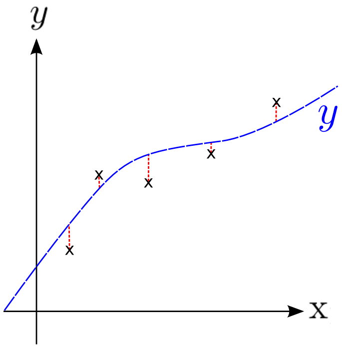
Our example: Regression with one real-valued attirbute

- input feature vectors: $\underline{\mathbf{x}} \in \mathbb{R}^N$
- output attributes: $y_T \in \mathbb{R}$
- training set: $\{\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\}_{\alpha=1}^{p}$



Our example: Regression with one real-valued attirbute

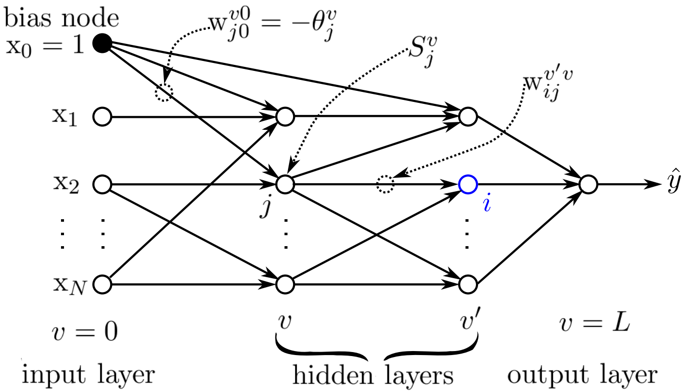
- input feature vectors: $\underline{\mathbf{x}} \in \mathbb{R}^N$
- output attributes: $y_T \in \mathbb{R}$
- training set: $\left\{\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right\}_{\alpha=1}^{p}$
- label function: $y_T = y(\underline{\mathbf{x}}) + \text{noise}$



model class

Multi-layer perceptrons (MLPs)

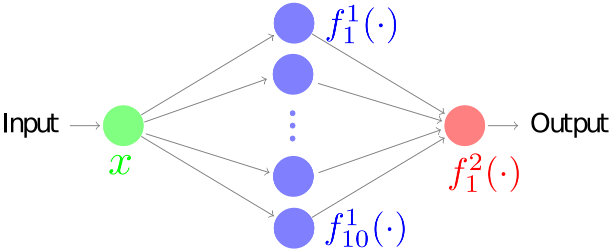
- layered FFN $\hat{y}(\cdot; \underline{\mathbf{w}})$ models label function $y(\cdot)$



- $S_i^{v'} = f_i^{v'}(h_i^{v'}) \overset{\text{e.g.}}{=} \tanh(h_i^{v'}), \quad h_i^{v'} = \sum_{j=0}^{N} w_{ij}^{v'v} S_j^v$

Model class: an example

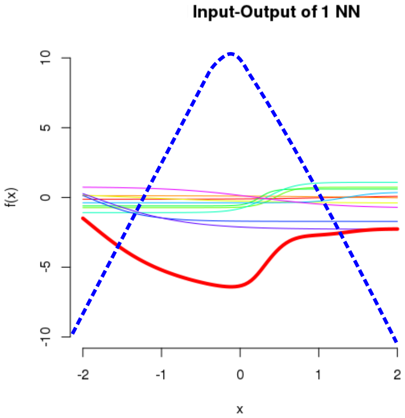
1 input, 1 output, 1 hidden layer with 10 units

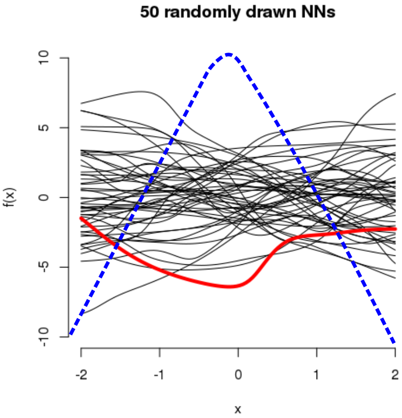


$$\hat{y}(x) \quad = \quad f_1^2\Big(\sum_{i=1}^{10} w_{1\,i}^{21}\, f_i^1(w_{i1}^{10}\, x - w_{i0}^{10})\Big)$$

$$f_i^1(a) = \tanh(a) \;\&\; f_1^2(a) = a$$

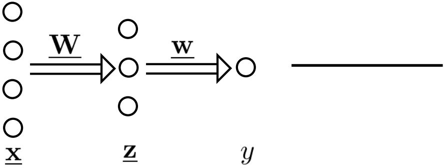
Model class: one example





(parameters drawn from Gaussian distribution)

Linear transfer functions



$$
\begin{aligned}
y &= \underline{\mathbf{w}}^\top \underline{\mathbf{z}} \quad = \quad \underline{\mathbf{w}}^\top \underline{\mathbf{W}}\, \underline{\mathbf{x}} \\
&= (\underline{\mathbf{W}}^\top \underline{\mathbf{w}})^\top \underline{\mathbf{x}} \quad = \quad \underline{\widehat{\mathbf{w}}}^\top \underline{\mathbf{x}} \\
&\widehat{=} \quad \text{connectionist neuron}
\end{aligned}
$$

MLPs are universal approximators

Funahashi (1989)

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

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

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

MLPs are universal approximators

Funahashi (1989)

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

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

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 $\varepsilon > 0$:

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

Funahashi (1989) On the approximate realization of continuous mappings by neural networks. Neur Netw, 2:183–192
Hornik et al. (1989) Multilayer Feedforward Networks are Universal Approximators. Neur Netw, 2:359–366.

1.3.3 Performance Measures and Model Selection

Cost functions

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

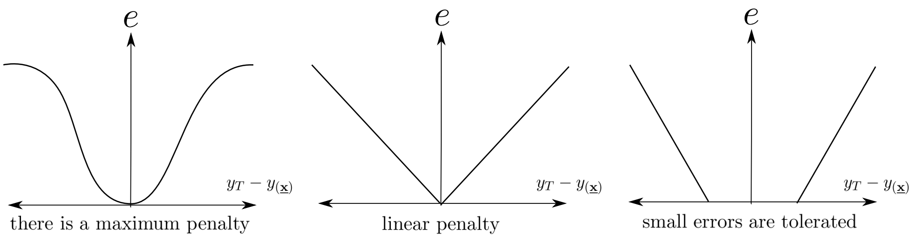
y_T: true value of attribute
$y(\underline{\mathbf{x}})$: predicted value of attribute (e.g. by MLP)

Cost functions

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

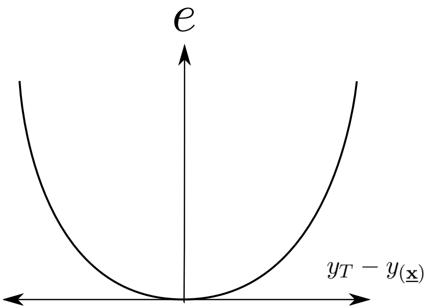
y_T: true value of attribute
$y(\underline{\mathbf{x}})$: predicted value of attribute (e.g. by MLP)

individual cost $e(y_T, \underline{\mathbf{x}})$



several choices $\Rightarrow$ predictor will depend on error measure!

Lecture example: quadratic error



$$e(y_T, \underline{\mathbf{x}}) \quad = \quad \frac{1}{2}\Big(y(\underline{\mathbf{x}}) - y_T\Big)^2$$

- Gaussian noise on attributes
- sensitive against "outliers"

Performance measure

Generalization error

$$E^G \quad := \quad < e >_{y_T, \underline{\mathbf{x}}} \quad = \quad \iint d\underline{\mathbf{x}}\, dy_T \; P_{(y_T, \underline{\mathbf{x}})} \, e_{(y_T, \underline{\mathbf{x}})}$$

$P_{(y_T, \underline{\mathbf{x}})}$: joint Probability Density Function (PDF) of observations

Performance measure

Generalization error

$$E^G \quad := \quad < e >_{y_T, \underline{\mathbf{x}}} \quad = \quad \iint d\underline{\mathbf{x}}\, dy_T\, P_{(y_T, \underline{\mathbf{x}})}\, e_{(y_T, \underline{\mathbf{x}})}$$

$P_{(y_T, \underline{\mathbf{x}})}$: joint Probability Density Function (PDF) of observations

"good" predictor: low value of E^G
"bad" predictor: high value of E^G

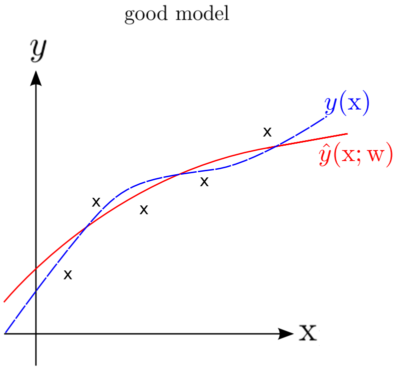
$$E^G \stackrel{!}{=} \min$$

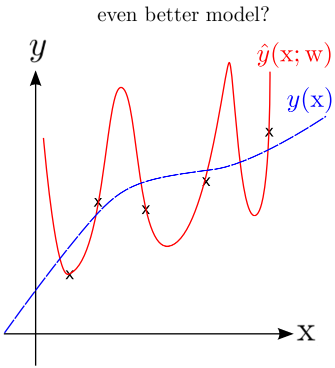
but: $P(y_T|\underline{\mathbf{x}})$ is not known

Principle of Empirical Risk Minimization (ERM)

$$
\begin{array}{rcl}
\text{math. expectation } (E^G) & \rightarrow & \text{empirical average } (E^T) \\
E^G := \int d\underline{\mathbf{x}} dy_T \, P(y_T, \underline{\mathbf{x}}) e(y_T, \underline{\mathbf{x}}) & \rightarrow & E^T := \frac{1}{p} \sum_{\alpha=1}^{p} e^{(\alpha)} \\
\text{generalization error} & \rightarrow & \text{training error} \\
E^G \stackrel{!}{=} \min & \rightarrow & E^T \stackrel{!}{=} \min
\end{array}
$$

Consequences?

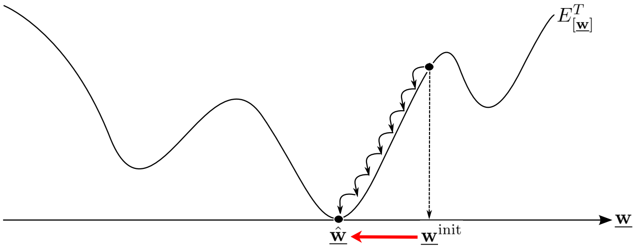




1.3.4 Optimization of Model Parameters: Gradient Descent

Gradient Descent

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



$$w_{ij}^{v'v}(t+1) = w_{ij}^{v'v}(t) - \underbrace{\hat{\eta}}_{\text{learning step}} \underbrace{\frac{\partial E^T_{[\underline{\mathbf{w}}]}}{\partial \mathrm{w}_{ij}^{v'v}}}_{\text{gradient vector}}$$

Calculation of the gradient

$$\frac{\partial E^T_{[\underline{\mathbf{w}}]}}{\partial \mathrm{w}_{ij}^{v'v}} = \frac{1}{p}\sum_{\alpha=1}^{p} \underbrace{\frac{\partial e^{(\alpha)}_{[\underline{\mathbf{w}}]}}{\partial \mathrm{w}_{ij}^{v'v}}}_{\text{individual cost}} = \frac{1}{p}\sum_{\alpha=1}^{p} \underbrace{\frac{\partial e^{(\alpha)}_{[\underline{\mathbf{w}}]}}{\partial y(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}}_{\text{factor depending on cost function}} \cdot \underbrace{\frac{\partial y(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}{\partial \mathrm{w}_{ij}^{v'v}}}_{\text{factor depending on model class (e.g. MLP)}}$$

Calculation of the error term

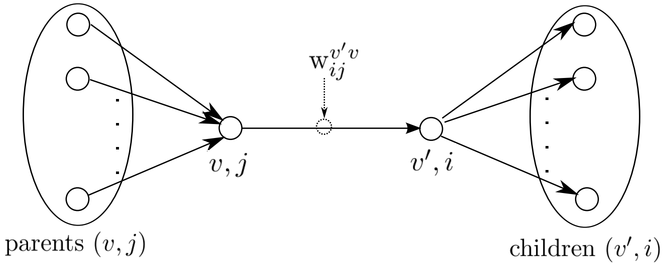
Quadratic Error

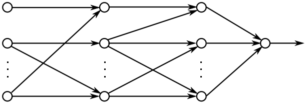
$$e(y_T, \underline{\mathbf{x}}) = \frac{1}{2}\big(y_T - y(\underline{\mathbf{x}})\big)^2 \qquad \Rightarrow \qquad \frac{\partial e^{(\alpha)}_{[\underline{\mathbf{w}}]}}{\partial y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})}} = y_{(\underline{\mathbf{x}}^{(\alpha)}, \underline{\mathbf{w}})} - y_T$$

1.3.5 The Backpropagation Method

Gradients in neural networks

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





see blackboard for derivation

1.3.6 Summary of the Gradient Descent Method

Summary of the backpropagation method

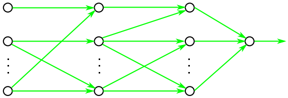
initialization of weights and thresholds

while *stopping criterion not met* **do**

$\quad \text{gradient}_{ij}^{v'v} := 0\,, \quad \forall w_{ij}^{v'v}$

$\quad$ **for** $\alpha \in \{1, \ldots, p\}$ **do**

$\quad\quad h_i^0 := x_i^\alpha\,, \quad \forall i$ // forward propagation

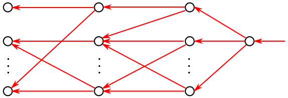


$\quad\quad$ **for** $v' \in \{1, \ldots, L\}$ **do**

$$h_i^{v'} := \sum_{(v',i) \in C(v,j)} w_{ij}^{v'v} f_j^v(h_j^v)\,, \quad \forall i$$

$\quad\quad$ **end**

$\quad\quad \delta_i^L := f'(h_i^L)\,, \quad \forall i$ // backward propagation



$\quad\quad$ **for** $v' \in \{L-1, \ldots, 0\}$ **do**

$$\delta_i^{v'} := {f'}_i^{v'}(h_i^{v'}) \sum_{(\beta,k) \in C(v',i)} \delta_k^\beta\, w_{ki}^{\beta v'}\,, \quad \forall i$$

$\quad\quad$ **end**

$\quad\quad \text{gradient}_{ij}^{v'v} := \text{gradient}_{ij}^{v'v} + \frac{\partial e^{(\alpha)}}{\partial y}\, \delta_i^{v'}\, f_j^v(h_j^v)\,, \quad \forall w_{ij}^{v'v}$ // average

$\quad$ **end**

$\quad w_{ij}^{v'v} := w_{ij}^{v'v} - \frac{\hat{\eta}}{p}\, \text{gradient}_{ij}^{v'v}\,, \quad \forall w_{ij}^{v'v}$ // gradient descend step

end

Summary of the backpropagation method

initialization of weights and thresholds
while *stopping criterion not met* **do**

 gradient$_{ij}^{v'v} := 0\,, \quad \forall w_{ij}^{v'v}$
 for $\alpha \in \{1, \ldots, p\}$ **do**
  $h_i^0 := x_i^\alpha\,, \quad \forall i$ // forward propagation
  for $v' \in \{1, \ldots, L\}$ **do**
   $$h_i^{v'} := \sum_{(v',i) \in C(v,j)} w_{ij}^{v'v} f_j^v(h_j^v)\,, \quad \forall i$$
  end
  $\delta_i^L := f'(h_i^L)\,, \quad \forall i$ // backward propagation
  for $v' \in \{L-1, \ldots, 0\}$ **do**
   $$\delta_i^{v'} := {f'}_i^{v'}(h_i^{v'}) \sum_{(\beta,k) \in C(v',i)} \delta_k^\beta\, w_{ki}^{\beta v'}\,, \quad \forall i$$
  end
  gradient$_{ij}^{v'v}$:= gradient$_{ij}^{v'v} + \frac{\partial e^{(\alpha)}}{\partial y}\, \delta_i^{v'}\, f_j^v(h_j^v)\,, \quad \forall w_{ij}^{v'v}$ // average
 end
 $w_{ij}^{v'v} := w_{ij}^{v'v} - \frac{\hat{\eta}}{p}$ gradient$_{ij}^{v'v}\,, \quad \forall w_{ij}^{v'v}$ // gradient descend step
end

computational and memory complexity

$\mathcal{O}(n)$,

n: number of weights

Start & stop for the gradient descent method

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

$\rightarrow$ too **large**?

$\rightarrow$ too **small**?

Start & stop for the gradient descent method

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

$\rightarrow$ too **large**: transfer function saturates and gradients become too small

$\rightarrow$ too **small**: neurons operate in the linear regime of f

Start & stop for the gradient descent method

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

$\rightarrow$ too **large**: transfer function saturates and gradients become too small

$\rightarrow$ too **small**: neurons operate in the linear regime of f

Stopping criteria:

$\rightsquigarrow$ fixed number of iterations

$\rightsquigarrow$ fixed CPU-time

$\rightsquigarrow$ E^T falls below a predefined value

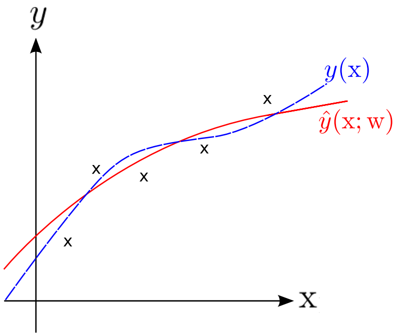
$\rightsquigarrow$ $\frac{\Delta E^T}{E^T}$ falls below a predefined value

$\rightsquigarrow$ validation criterion fulfilled

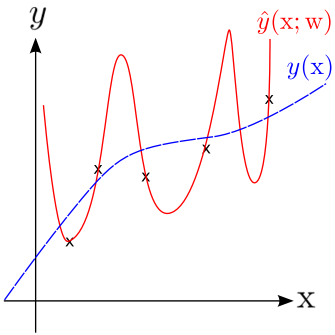
1.3.7 Validation of Model Selection

Overfitting

good model



even better model?



Assessment of prediction quality

Test Set Method

observations $\left\{ \begin{array}{l} \text{training data } \left\{ \left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)} \right) \right\}, \alpha \in \{1, \ldots, p\} \\ \rightarrow E^T \text{ selects model parameters} \\ \\ \end{array} \right.$

$$\hat{E}^T \quad = \quad \frac{1}{p} \sum_{\alpha=1}^{q} e^{(\alpha)}$$

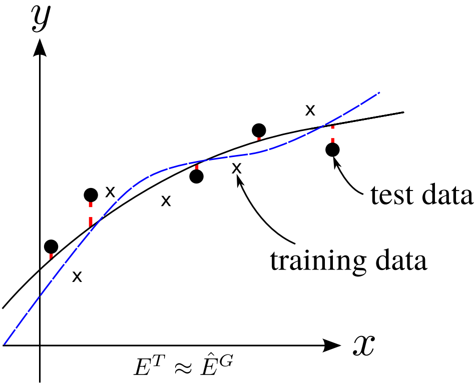
Assessment of prediction quality

Test Set Method

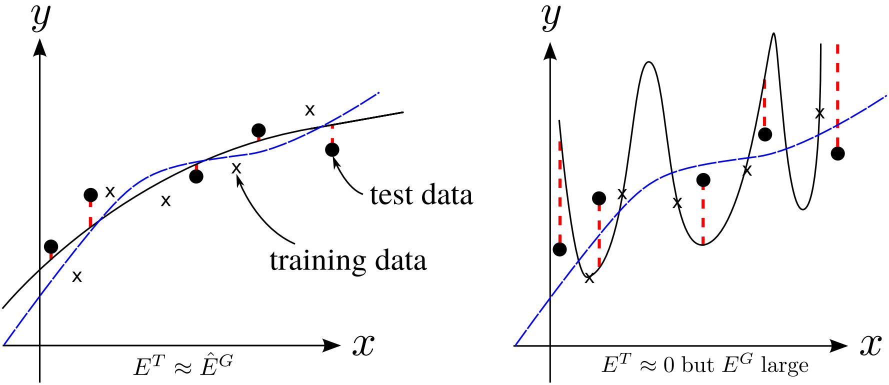
$$\text{observations} \begin{cases} \text{training data } \left\{\left(\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}\right)\right\}, \alpha \in \{1, \ldots, p\} \\ \rightarrow E^T \text{ selects model parameters} \\ \text{test data } \left\{\left(\underline{\mathbf{x}}^{(\beta)}, y_T^{(\beta)}\right)\right\}, \beta \in \{1, \ldots, q\} \\ \rightarrow \hat{E}^G \text{ estimates generalization error} \end{cases}$$

$$\hat{E}^G = \frac{1}{q} \sum_{\beta=1}^{q} e^{(\beta)}$$

Test set method

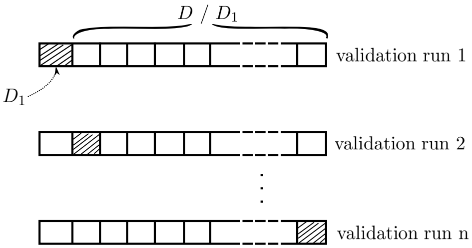


Test set method



Resampling methods: n-fold cross-validation

Observations $D \rightarrow n$ disjunct sets D_j, $\bigcup_{j=1}^{n} D_j = D$



Training of n networks on the training datasets D / D_j

Estimation of E^G:

$$\widehat{E}^G = \frac{1}{p} \sum_j \sum_{\alpha \in D_j} e^{(\alpha)}$$

Typical choice: $n \in \{5, \ldots, 10\}$ $\qquad$ $n = p$: leave-one-out cross-validation

Variance of the cross-validation estimator

Remarks

- n-fold cross-validation is only used for estimating E^G
- each of the n folds results in a different solution (set of parameters)
- <u>All data</u> are used for selecting the model parameters

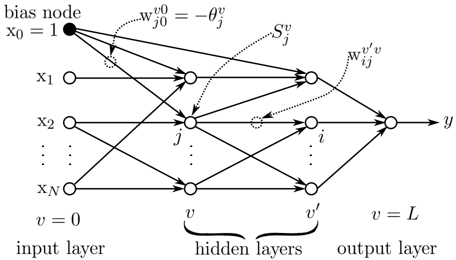
$$\operatorname{var}\left(\widehat{E}^G\right) = \frac{n-1}{n} \sum_{j=1}^{n} \Bigg(\underbrace{\widehat{E}_j^G}_{\substack{\text{test error} \\ \text{on } D_j}} - \widehat{E}^G \Bigg)^2$$

End of Section 1.3

the following slides contain

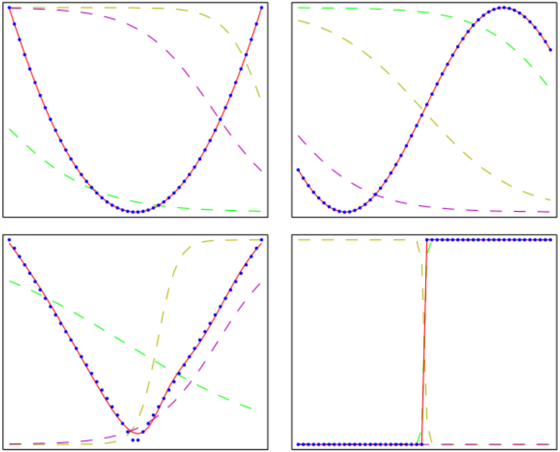
OPTIONAL MATERIAL

Multi-layer perceptrons: nomenclature



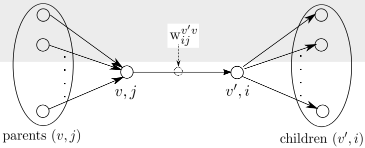
S_j^v	activity of neuron (v, j)	(layer,unit)
$S_0^v = x_0 = 1$	activity of bias neuron	
$S_i^0 = x_i$	input to MLP, i^{th} component	
$S_i^L = y$	output of MLP	
$w_{ij}^{v'v}$	connection weight between neurons (v, j) and (v', i)	
$w_{j0}^{v0} = \theta_j^v$	connection weight between bias node and neuron (v, j)	
$h_i^{v'}$	$= \sum_j w_{ij}^{vv-1} S_j^{v-1} \quad =$ total input of neuron (v', i)	
$f_i^{v'}$	transfer function of neuron (v', i)	

Illustration: ERM with MLPs (Bishop 2009)



fitted MLPs with 1 hidden layer of 3 neurons

The Credit Assignment Problem



How do the weights $w_{ij}^{v'v}$ of hidden units $S_i^{v'}$ contribute to the error?

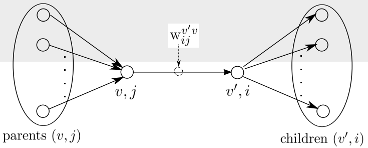
Solution: smart application of the chain rule

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

Forward propagation: calculation of activities (parents $\rightarrow$ children)

$$S_j^0 = \mathrm{x}_j^{(\alpha)} \quad \rightarrow \quad S_i^{v'} = f\Big(\sum_{(v',i)\in C(v,j)} \mathrm{w}_{ij}^{v'v} S_j^v \Big)$$

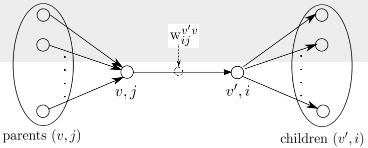
Backpropagation



Backpropagation: calculation of "local errors" (children → parents)

$$\delta_i^L = \underbrace{f'(h_i^{v'})}_{=1 \text{ for identity}} \quad \text{and} \quad \delta_i^{v'} = \sum_{(v'',k) \in C(v',i)} \frac{\partial y}{\partial h_k^{v''}} \cdot \frac{\partial h_k^{v''}}{\partial h_i^{v'}}, \quad \forall v' \neq L$$

Backpropagation



Backpropagation: calculation of "local errors" (children $\rightarrow$ parents)

$$\delta_i^L = \underbrace{f'(h_i^{v'})}_{=1 \text{ for identity}} \quad \text{and} \quad \delta_i^{v'} = \sum_{(v'',k)\in C(v',i)} \frac{\partial y}{\partial h_k^{v''}} \cdot \frac{\partial h_k^{v''}}{\partial h_i^{v'}}, \quad \forall v' \neq L$$

rewrite using $\frac{\partial y}{\partial h_k^{v''}} = \delta_k^{v''}$

$$\delta_i^{v'} = \sum_{(v'',k)\in C(v',i)} \delta_k^{v''} \cdot \mathrm{w}_{ki}^{v''v'} f'(h_i^{v'}) = f'(h_i^{v'}) \sum_{(v'',k)\in C(v',i)} \delta_k^{v''} \mathrm{w}_{ki}^{v''v'}$$

Computational complexity: $O(n)$, n: number of weights & thresholds