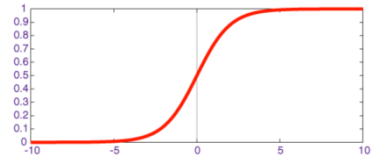
Computational Intelligence

4. Regularisierung

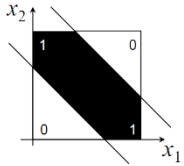
Prof. Dr. Sven Behnke

Letzte Vorlesung

- Überwachtes Training
  - Menge von Trainingsbeispielen $\{(x_1,t_1), \ldots, (x_p,t_p)\}$ (Regression, Klassifikation)
  - Fehlerfunktion, z.B. quadratisch $E(w) = 1/2 \sum_{i=1}^{p} (y_i - t_i)^2$
  - Differenzierbare Aktivierungsfunktionen, z.B. $y = \Phi(a) = \frac{1}{1+e^{-a}}$

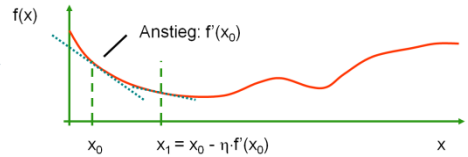


  - XOR-Problem

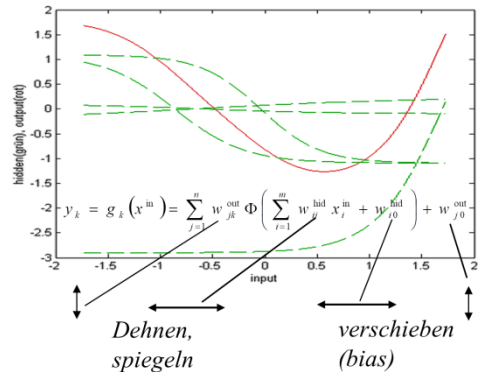


  => Multi-Layer Perzeptron (MLP)

  - Gewichtsanpassung durch Gradientenabstieg: $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j$

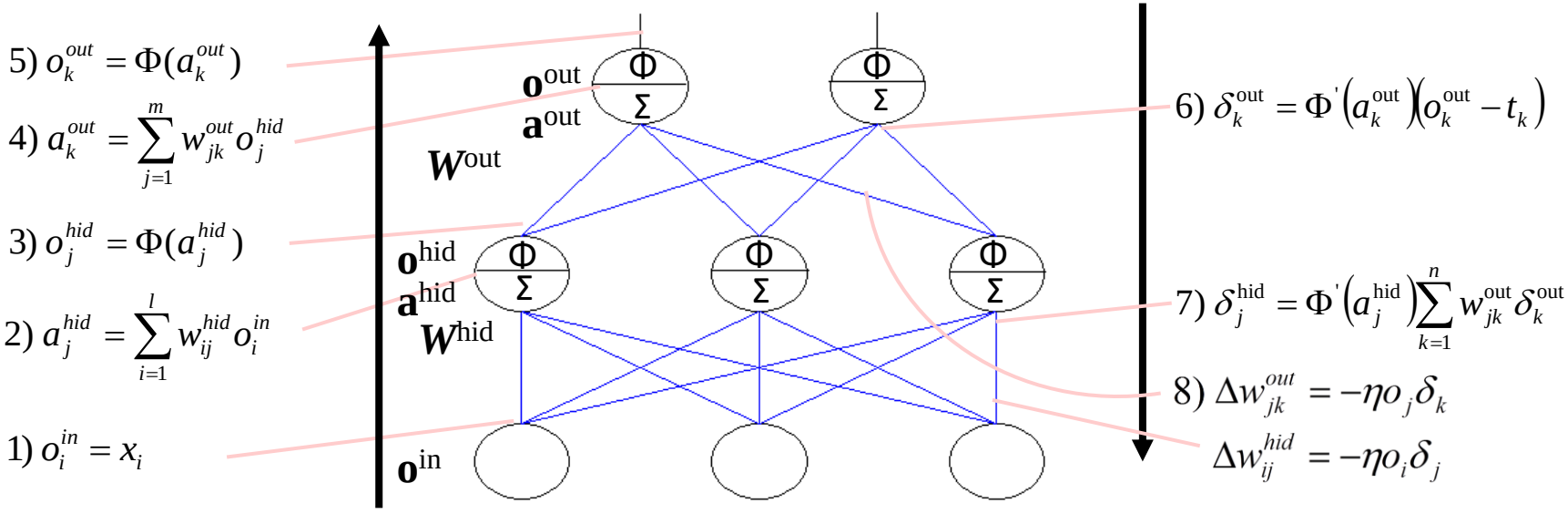


  - Backpropagation zur Berechnung des Gradienten
  - Backpropagation-Varianten
    - Resilient Propagation (RProp), Momentum, …
  - MLP als universeller Funktionsapproximator



Letzte Vorlesung: Backpropagation

- Vorwärtspropagation von Aktivität
- Rückwärtspropagation von Fehler
- Gewichtsanpassung durch Gradientenabstieg



- Zufällige Gewichtsinitialisierung
- Präsentation der Eingabemuster zufällig oder als Block

Maximum-Likelihood-Schätzung

- Gegeben iid-Beispiele $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ aus Verteilung $p_{\text{data}}(\mathbf{x})$
- Parametrische Familie von Wahrscheinlichkeitsverteilungen $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$
- Maximum-Likelihood-Schätzer:

$$\begin{aligned}
\boldsymbol{\theta}_{\text{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})
\end{aligned}$$

[Goodfellow et al. 2016]

KL-Divergenz

- Misst Unterschied zwischen Modell und Daten

$$D_{\mathrm{KL}}\left(\hat{p}_{\mathrm{data}} \| p_{\mathrm{model}}\right) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}}\left[\log \hat{p}_{\mathrm{data}}(\boldsymbol{x}) - \log p_{\mathrm{model}}(\boldsymbol{x})\right]$$

Hängt nicht vom Modell ab.

- Minimiere: $-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}}\left[\log p_{\mathrm{model}}(\boldsymbol{x})\right]$

- Entspricht ML-Schätzer:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x};\boldsymbol{\theta})$$

Maximum-Likelihood-Training

- Parametrische Ausgabeverteilung $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$
- Kostenfunktion $J(\boldsymbol{\theta}) = -\log P(y \mid \boldsymbol{x})$

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$$

- Normalverteilte Ausgaben

$$p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{I})$$

=> Quadratische Kostenfunktion

$$J(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x}; \boldsymbol{\theta})||^2 + \text{const}$$

- Angemessen für Funktionsapproximation (kontinuierliche Ausgaben)
- Ungeeignet für sigmoide Ausgabe-Transferfunktion, da Saturierung => Gradient≈0

[Goodfellow et al. 2016]

Binäre Klassifikation: Sigmoide Aktivierung

- Gewünschte Ausgabe 0 oder 1: $P(y = 1 \mid \boldsymbol{x})$

[Goodfellow et al. 2016]

- Benötigt sigmoide Transferfunktion

$$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right) \qquad z = \boldsymbol{w}^\top \boldsymbol{h} + b$$

- Bernoulli-Verteilung:

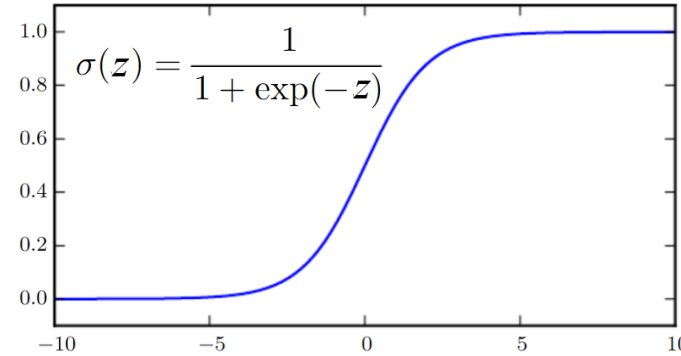
$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)}$$
$$= \sigma\left((2y-1)z\right)$$

- Kostenfunktion:

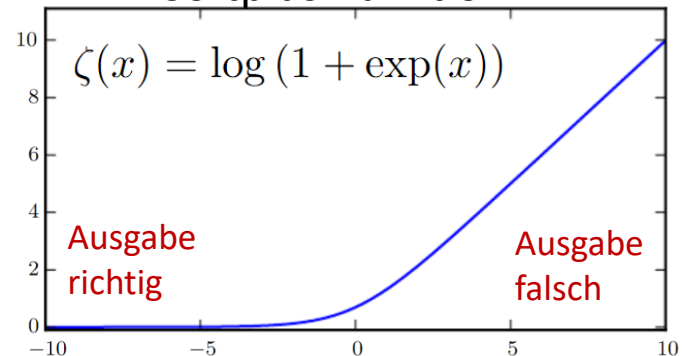
$$J(\boldsymbol{\theta}) = -\log P(y \mid \boldsymbol{x})$$
$$= -\log \sigma\left((2y-1)z\right)$$
$$= \zeta\left((1-2y)z\right)$$

  - Mindert den Gradient für falsche Ausgaben kaum

Sigmoide Transferfunktion

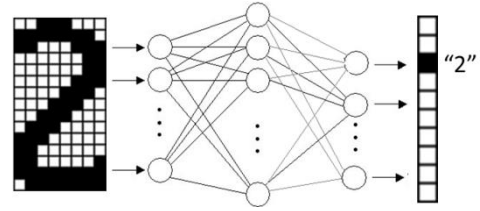


Softplus-Funktion



N-äre Klassifikation: Softmax-Aktivierung

- N Klassenwahrscheinlichkeiten: $\hat{y}_i = P(y = i \mid \boldsymbol{x})$
  - Ausgabe 1 an der richtigen Stelle
  - Alle anderen Ausgaben sollen 0 sein



- Normalisierung der Klassenwahrscheinlichkeiten durch Softmax
  - Unnormalisierte Log-Klassenwahrscheinlichkeiten: $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$
  - Normalisierung auf Summe 1:

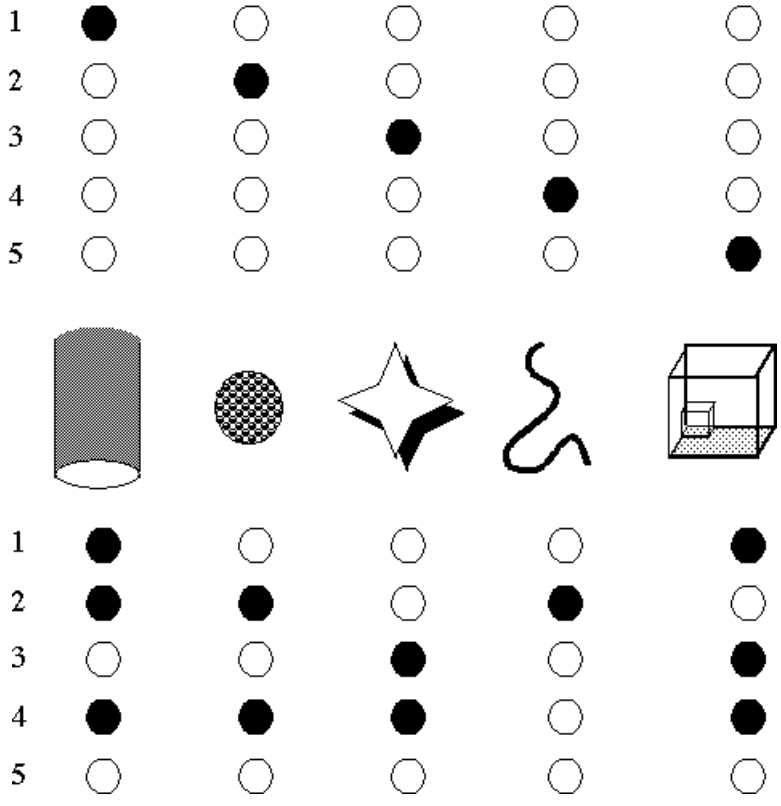
$$\hat{y} = \text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

  - Wichtig sind nicht absolute Werte der Nettoaktivitäten, sondern deren Unterschiede (Laterale Hemmung, Winner Takes All)
- Logarithmus liefert direkten Beitrag der Nettoaktivität z_i zur Kostenfunktion:

$$\log P(\text{y} = i; \boldsymbol{z}) = \log \text{softmax}(\boldsymbol{z})_i = \boxed{z_i} - \log \sum_j \exp(z_j)$$

[Goodfellow et al. 2016]

Neuronale Codierung

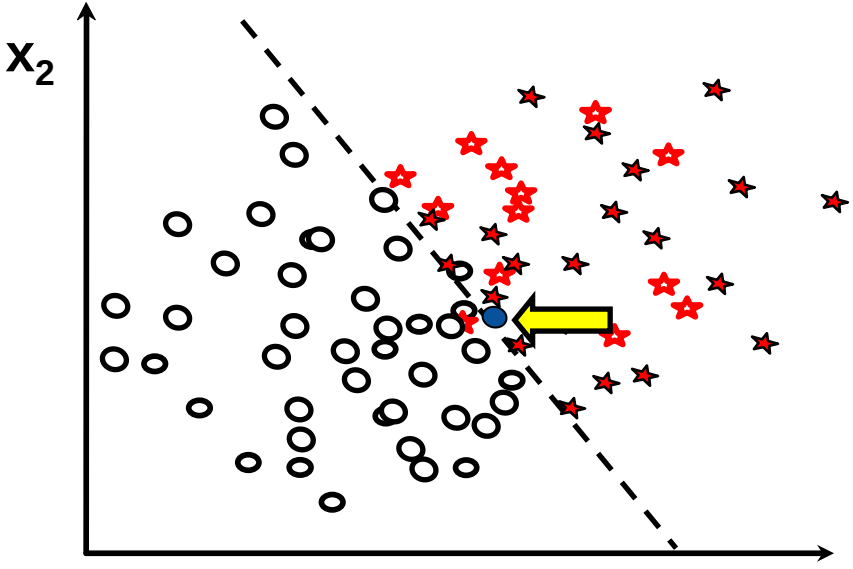


- **Lokal**
  - Ein Neuron codiert ein Objekt
  - Großmutter-Zellen
  - Skaliert nicht
  - Nicht robust

- **Verteilt**
  - Neuronen codieren Merkmale
  - Ein Neuron für mehrere Objekte aktiv
  - Ein Objekt aktiviert mehrere Neuronen
  - Robust gegen Beschädigungen

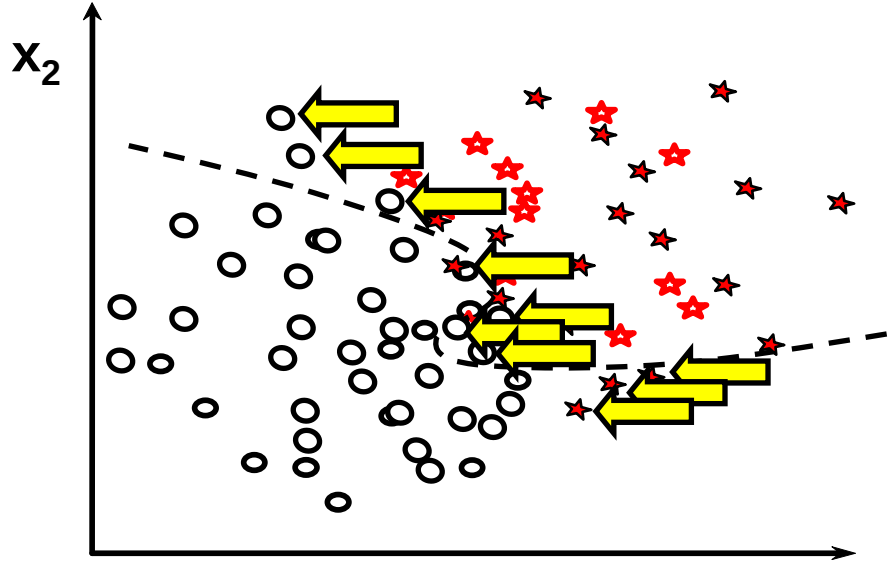
Wie genau soll die Trainingsmenge gelernt werden?

Trainingsfehler



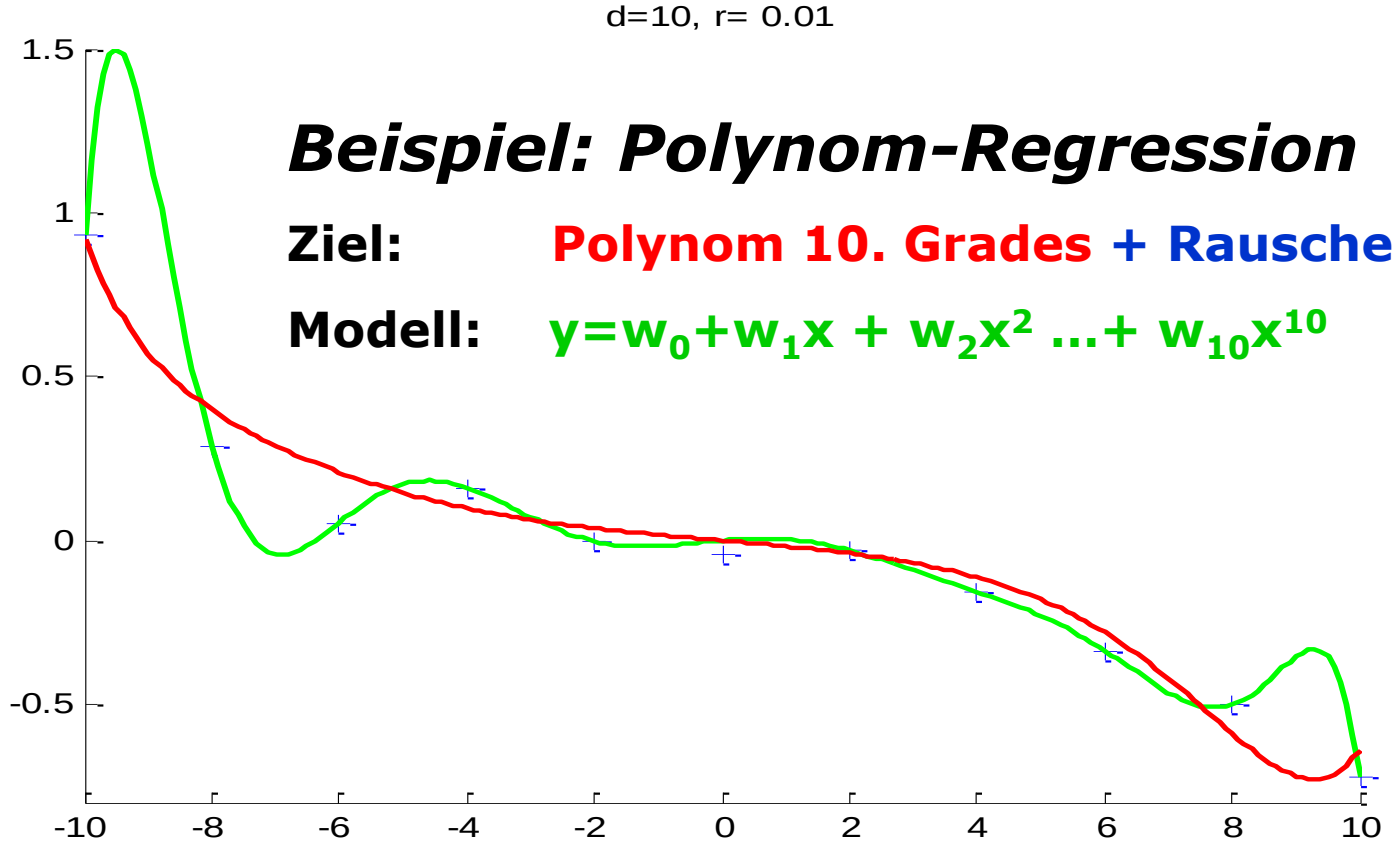
Kleiner Testfehler

Kein Trainingsfehler



Großer Testfehler

Überanpassung (Overfitting)



Beispiel: Polynom-Regression

Ziel: **Polynom 10. Grades + Rauschen**

Modell: $y=w_0+w_1x + w_2x^2 \ldots + w_{10}x^{10}$

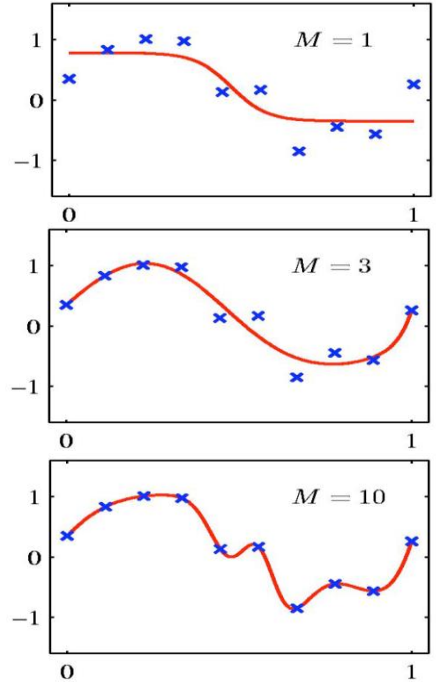
Regularisierung

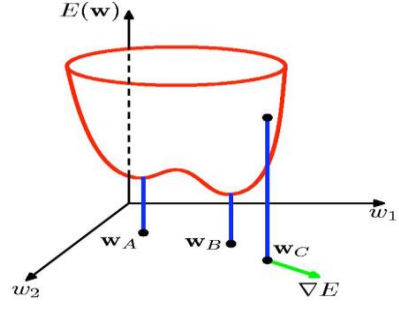
- Auswendiglernen der Trainingsmenge lässt keine gute Generalisierung auf Testmenge erwarten

- Idee: Beschränke die Kapazität der Lernmaschine durch
  - Geringe Zahl der Hidden Units
  - Early Stopping
  - Weight Decay
  - Kombination mehrerer Lerner

Effekt der Anzahl der Hidden Units

- Regression einer Sinusfunktion, gegeben durch zehn verrauschte Datenpunkte
- Quadratisches Fehlermaß
- Anzahl der Units in der verdeckten Schicht M=1,3,10
- Generalisierung hängt nicht nur von M ab, da Fehlerfunktion lokale Minima hat

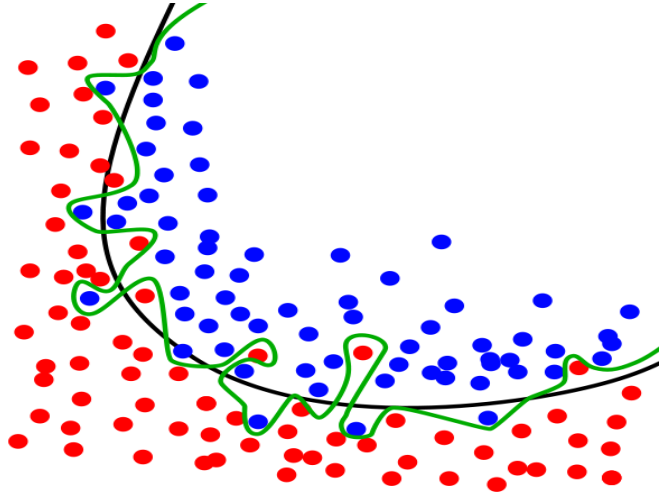


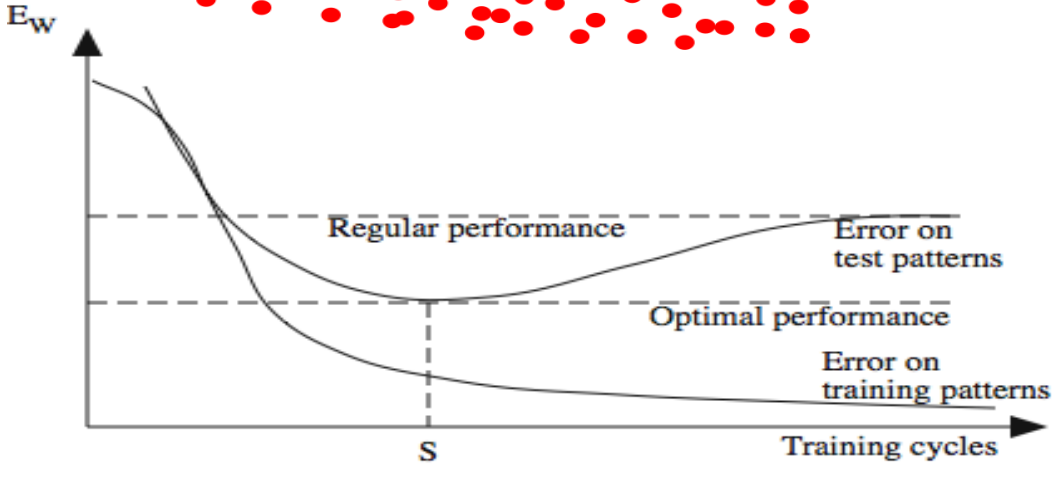


Ockham's Razor

- Prinzip vorgeschlagen von William von Ockham im 14. Jahrhundert: "**Pluralitas non est ponenda sine neccesitate**"
- Wenn zwei Theorien vergleichbar gute Vorhersagen machen, **ziehe die einfachere vor**
- Spare unnötige Modellparameter ein

Early Stopping

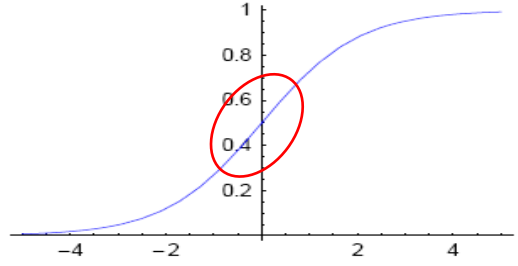


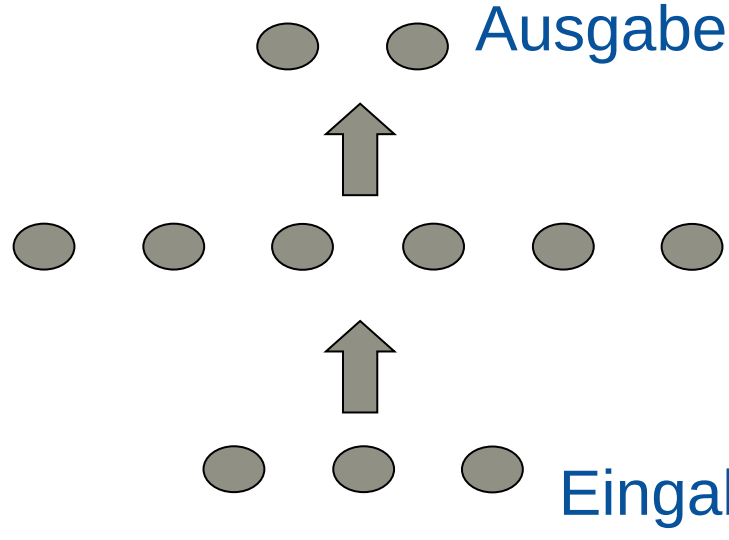


- Netzwerk soll auf neuen Daten die richtige Antwort geben (Generalisierung)
- Zu viele Gewichte führen zum Auswendigkernen der Trainings-menge (Overfitting)
- Nicht einfach, die Netzwerk-architektur zu wählen
- Lösung: Benutze Validierungsmenge
- Teile Daten in:
  - Trainingsmenge (für Gewichts-anpassung)
  - Validierungsmenge (zur Fehler-beobachtung)
- **Beende Training wenn Fehler auf der Validierungsmenge zu steigen beginnt** (early stopping)

Warum Early Stopping funktioniert

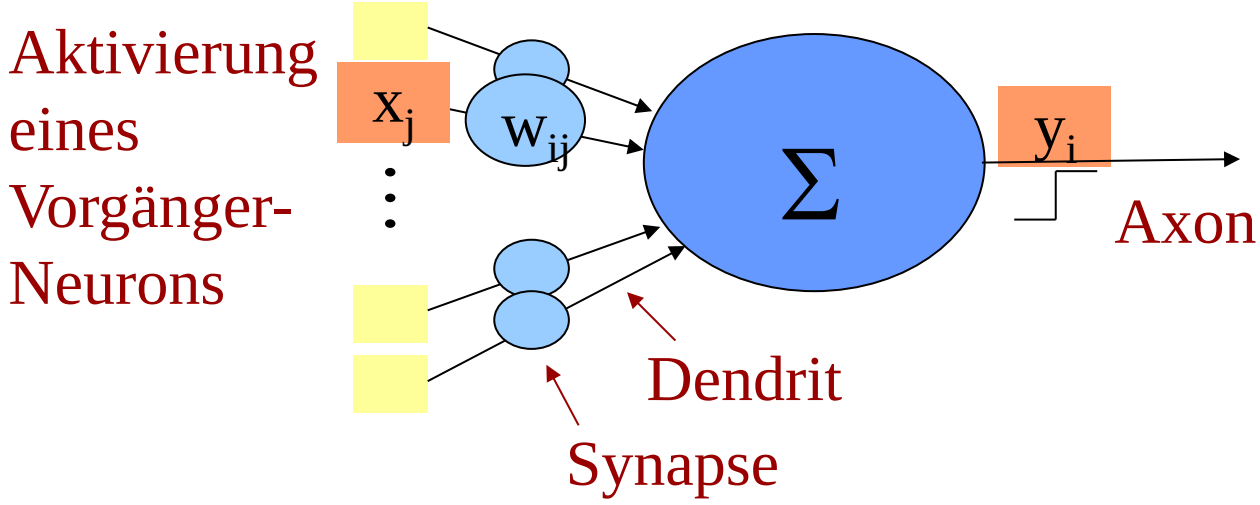
- Wenn die Gewichte klein sind, sind alle Hidden-Units linear
  - Folglich ist ein Netz mit einer großen verdeckten Schicht linear
  - Es kann nicht mehr als ein Netz mit direkten Verbindung von den Eingaben zu den Ausgaben
- Mit dem Gewichtswachstum werden die verdeckten Neuronen nichtlinearer und die Kapazität des Netzes steigt





Erinnerung: Hebb'sche Regel

$$w_{ij} \leftarrow w_{ij} + y_i \, x_j$$



Problem: Unbegrenztes Gewichtswachstum

WEIGHT DECAY

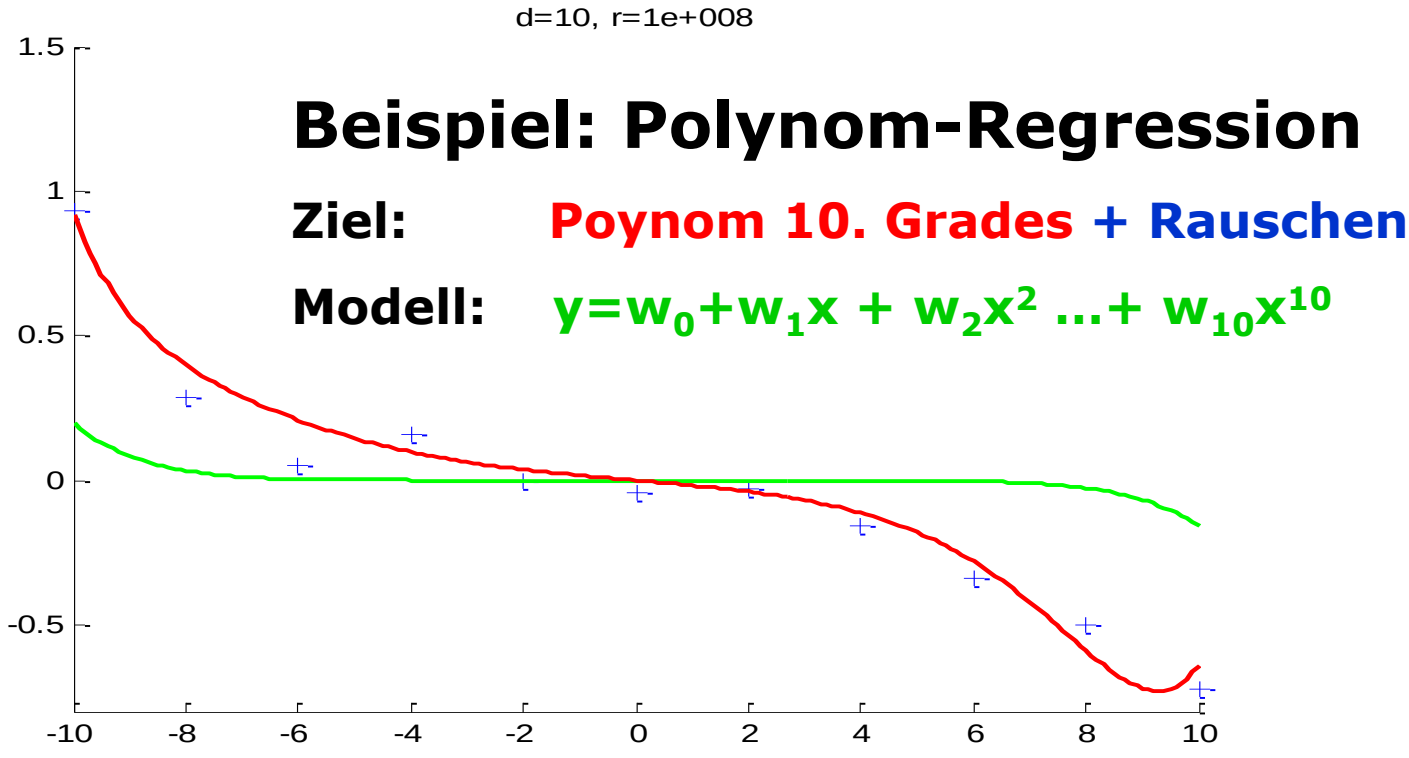
$w_{ij} \leftarrow w_{ij} + y_i x_j$ Hebb'sche Regel

$w_{ij} \leftarrow (1-\gamma) w_{ij} + y_i x_j$ Weight Decay

$\gamma \in [0, 1]$, Decay-Parameter

- Effekt: Gewichte werden kleiner,
  wenn sie nicht verstärkt werden

VERMEIDUNG VON OVERFITTING



Beispiel: Polynom-Regression

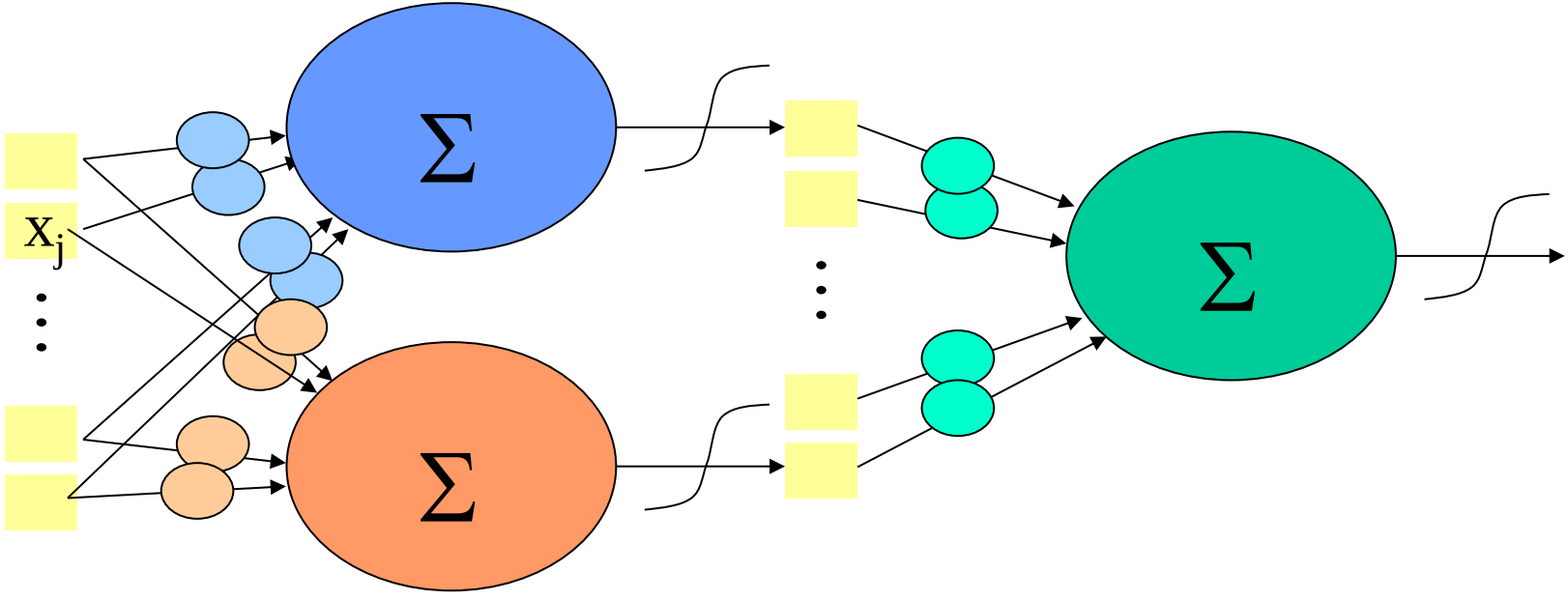
Ziel: **Poynom 10. Grades + Rauschen**

Modell: $y=w_0+w_1x + w_2x^2 \ldots + w_{10}x^{10}$

Weight Decay für MLPs

Ersetze: $w_{ij} \leftarrow w_{ij} + \text{back\_prop}(i,j)$

mit: $w_{ij} \leftarrow (1-\gamma)\, w_{ij} + \text{back\_prop}(i,j)$

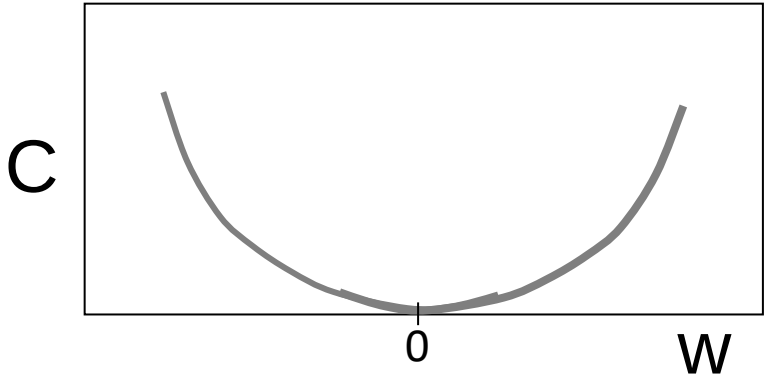


Weight Decay: Kosten für große Gewichte

- Dies entspricht Kostenfunktion erweitert um Term, der Gewichts-Quadrate bestraft

$$C = E + \frac{\lambda}{2} \sum_i w_i^2$$

$$\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$$



$$wenn \ \frac{\partial C}{\partial w_i} = 0, \quad w_i = -\frac{1}{\lambda} \frac{\partial E}{\partial w_i}$$

- Gewichte bleiben klein, es sei denn, sie werden sehr gebraucht

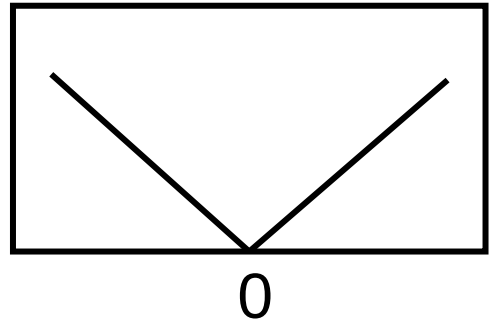
Effekt des Weight Decay

- Gewichte, die nicht gebraucht werden, werden nicht benutzt
  - Verbessert häufig Generalisierung
  - Vermeidet das Auswendiglernen von Rauschen
  - Ausgabe hängt glatter von Eingabe ab
- Wenn das Netz zwei ähnliche Eingaben hat, werden zwei kleine Gewichte benutzt, anstatt eines großen Gewichts

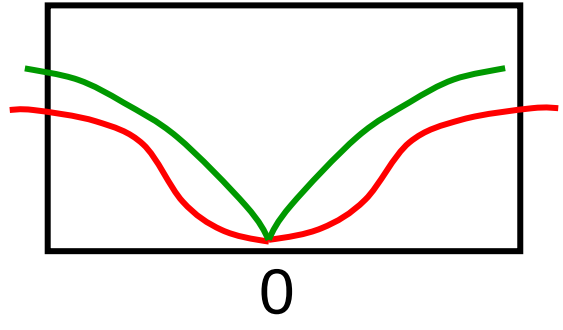
Andere Strafterme für Gewichte

- Bestrafe Absolutwerte
  - Dies reduziert viele Gewichte auf Null

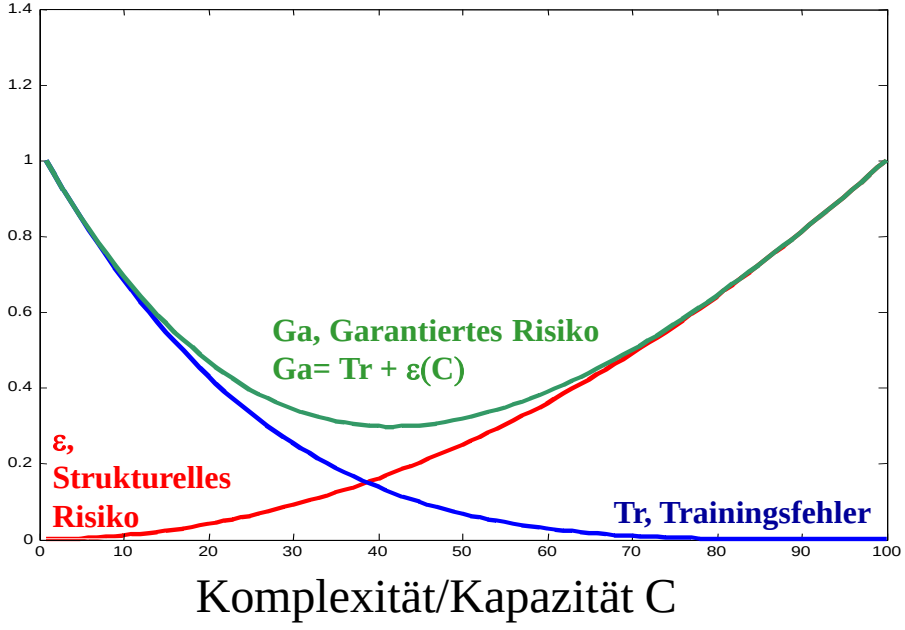
    => Einfache Interpretation
    des Netzes



- Bestrafe große Gewichte nicht so sehr



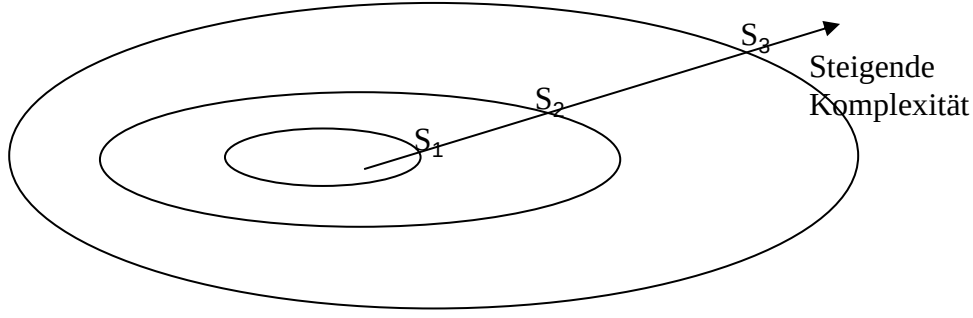
Minimierung des Strukturellen Risikos (SRM)



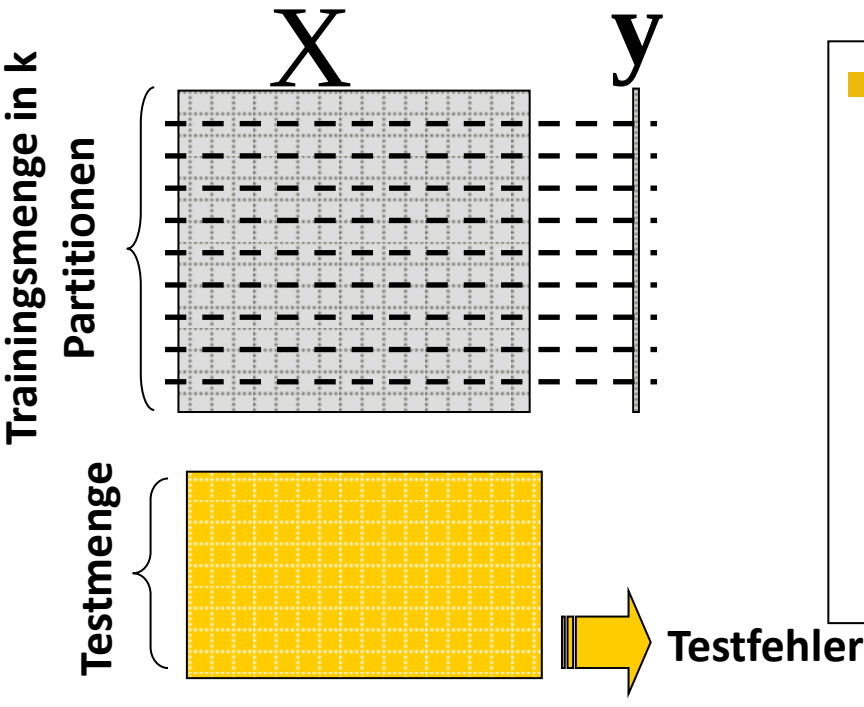
Vapnik, 1974

Modelle mit steigender Komplexität/Kapazität:

$S_1 \subset S_2 \subset \ldots S_N$



Selektion von Hyper-Parametern



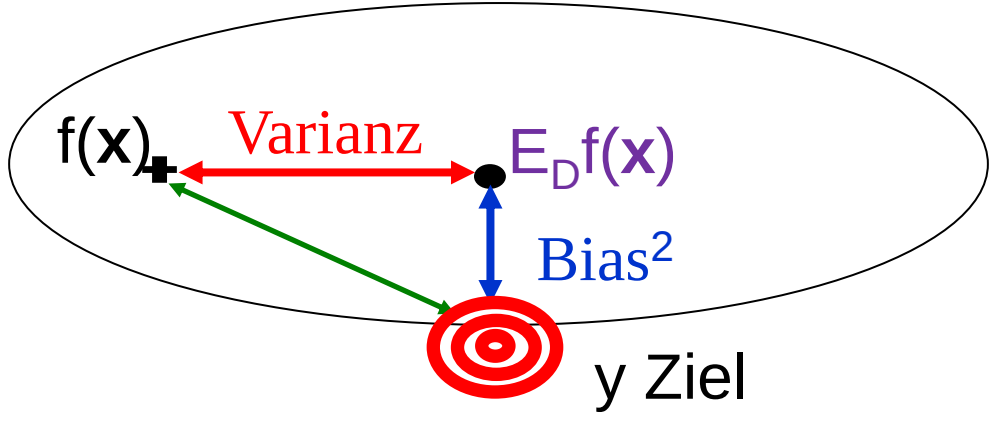
- **Lernen = Anpassung von:**
  - **Parametern** ($\mathbf{w}$ Gewichtsvektor)
  - **Hyper-Parametern** (γ, σ, κ).

- **K-fache Kreuzvalidierung:**
  - Für verschiedene Hyperparameter γ, σ, κ:
    - Lerne $\mathbf{w}$ aus (K-1) Trainingsbeispielen
    - Validiere auf verbleibenden 1/K Beispielen
    - Mittle Validierungsfehler über alle möglichen Validierungsteilmengen
  - Wähle γ, σ, κ so, dass obiger Kreuzvalidierungs-fehler minimal ist.
  - Nutze optimale Hyperparameter γ, σ, κ für Training auf gesamter Trainingsmenge

Bias-Varianz-Dilemma

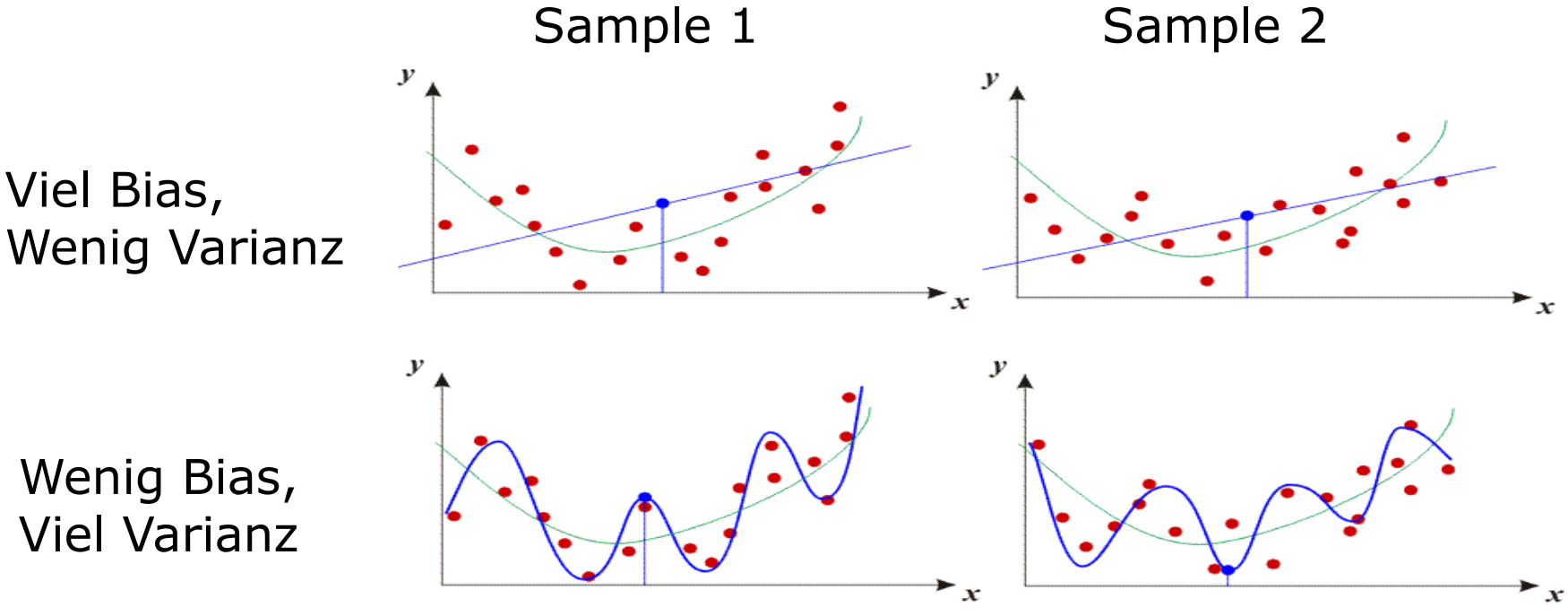
- f wird auf Trainingsmenge D der Größe m trainiert
- Quadratischer Fehler:

$$\underbrace{E_D[f(\mathbf{x})-y]^2}_{\text{Erwarteter Fehler für Trainingsmenge D der Größe m}} = \underbrace{[E_D f(\mathbf{x})-y]^2}_{\text{Bias}^2} + \underbrace{E_D[f(\mathbf{x})-E_D f(\mathbf{x})]^2}_{\text{Varianz}}$$



- Varianz kann häufig nur verringert werden, indem Bias erhöht wird

Bias-Varianz-Dilemma Beispiel



- Einführung von Bias kann Varianz so stark vermindern, dass erwarteter MSE sinkt

Kombination von Netzwerken reduziert Varianz

- Vergleiche erwarteten quadratischer Fehler
  - **Methode 1:** Wähle **zufällig** einen Prediktor aus
  - **Methode 2:** Nutze den **Durchschnitt** aller Prediktoren

Mittlerer Ausgabewert:

$$\bar{y} = \langle y_i \rangle_i = \frac{1}{N}\sum_{i=1}^{N} y_i$$

Mittlerer quadratischer Ausgabefehler:

$$\langle (t - y_i)^2 \rangle_i = \langle \big((t - \bar{y}) - (y_i - \bar{y})\big)^2 \rangle_i$$
$$= \langle (t - \bar{y})^2 + (y_i - \bar{y})^2 - 2(t - \bar{y})(y_i - \bar{y}) \rangle_i$$
$$= \langle (t - \bar{y})^2 \rangle_i + \langle (y_i - \bar{y})^2 \rangle_i \ldots$$
$$- 2(t - \bar{y}) \langle (y_i - \bar{y}) \rangle_i$$

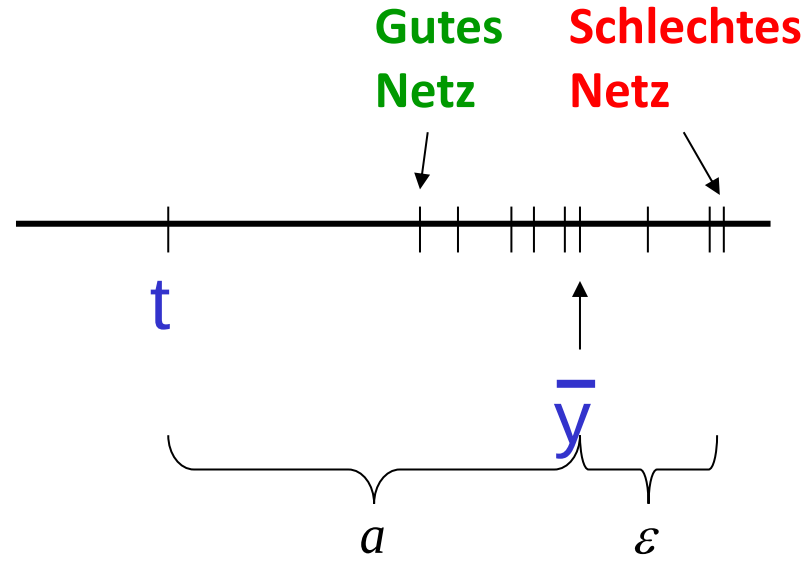
Ziel-Ausgabe

Dieser Term verschwindet

Verschwindet bei Nutzung des durchschnittlichen Prediktors

Mitteln reduziert erwarteten Fehler

- **Überdurchschnittlich** weit von t entfernte Prediktoren machen überdurchschnittliche quadratische Fehler
- **Unterdurchschnittlich** weit von t entfernte Prediktoren machen unterdurchschnittliche quadratische Fehler
- Wegen des **Quadrats** dominiert der erste Effekt



$$(a+\varepsilon)^2+(a-\varepsilon)^2=2a^2+2\varepsilon^2$$

Kann durch Verwendung des mittleren Ausgabewerts vermieden werden

Kombinierte Prediktoren vs. Einzelprediktoren

- Für jedes Testbeispiel findet man einige Einzelprediktoren, die besser sind als der kombinierte Prediktor
  - Aber: Die besseren Einzelprediktoren sind für verschiedene Testbeispiele jeweils andere

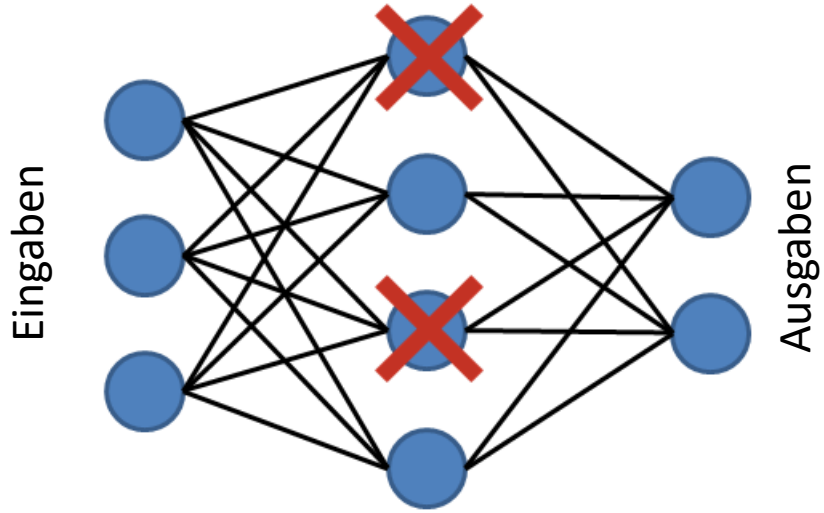
- Wenn die Einzelprediktoren unterschiedlich sind, ist der kombinierte Prediktor typischerweise besser als jeder Einzelprediktor, wenn man über die Testmenge mittelt
  - Wie kann man die Einzelprediktoren unterschiedlich machen, ohne sie wesentlich zu verschlechtern?

Ansätze zur Erzeugung unterschiedlicher Prediktoren

1. Man verlässt sich darauf, dass der Lernalgorithmus jedes Mal in einem anderen lokalen Minimum stecken bleibt
2. Man nutzt verschiedene Lernmaschinen:
   - Unterschiedliche Architektur
   - Verschiedene Lernverfahren
3. Man nutzt unterschiedliche Trainingsmengen:
   - **Bagging**: Zufallsauswahl (mit Zurücklegen) aus Trainingsmenge: a,b,c,d,e -> a c c d d
   - **Boosting**: Trainiere einen Prediktor nach dem anderen. Gewichte die schlecht gelernten Trainingsbeispiele höher.
     - Dies braucht relativ wenig Rechenzeit, da die am Anfang trainierten Prediktoren nicht mehr verändert werden

Regularisierung durch Drop-Out

- Zufälliges Deaktivieren der Hälfte der Hidden Units während des Trainings
- Ausgabegewichte mit Faktor 0.5 skalieren für Recall
- Gleichzeitiges Training exponentiell vieler Modelle



[Srivastava et al., JMLR 2014]