Neuronale Netze
Training & Regularisierung

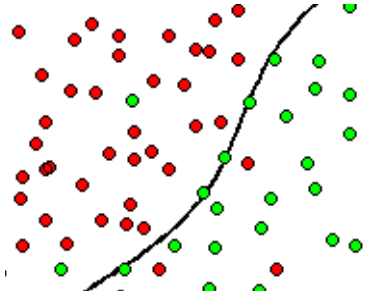
Prof. Dr.-Ing. Sebastian Stober

Artificial Intelligence Lab
Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
stober@ovgu.de

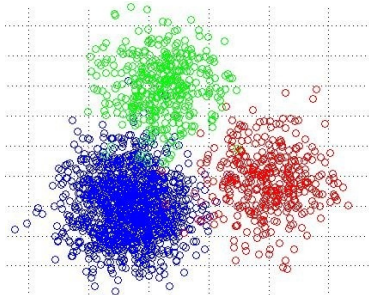
FACULTY OF
COMPUTER SCIENCE

Training
durch Gradientenabstieg

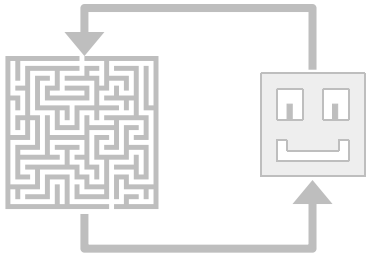
ML Problemklassen



Überwachtes Lernen
(supervised learning)



Unüberwachtes Lernen
(unsupervised learning)



Bestärkendes Lernen
(reinforcement learning)

Gradientenabstieg

Allgemeinerer Ansatz:**Gradientenabstieg**.

Notwendige Bedingung: **differenzierbare Aktivierungs- und Ausgabefunktionen**.

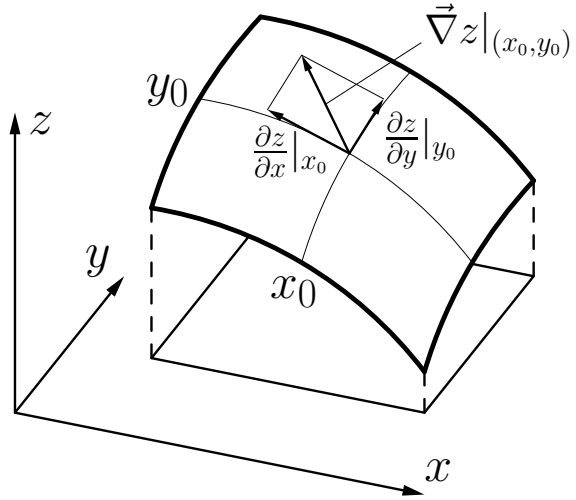


Illustration des Gradienten einer reellwertigen Funktion $z = f(x, y)$ am Punkt (x_0, y_0). Dabei ist $\vec{\nabla} z|_{(x_0,y_0)} = \left(\frac{\partial z}{\partial x}|_{x_0}, \frac{\partial z}{\partial y}|_{y_0}\right)$.

Error Backpropagation

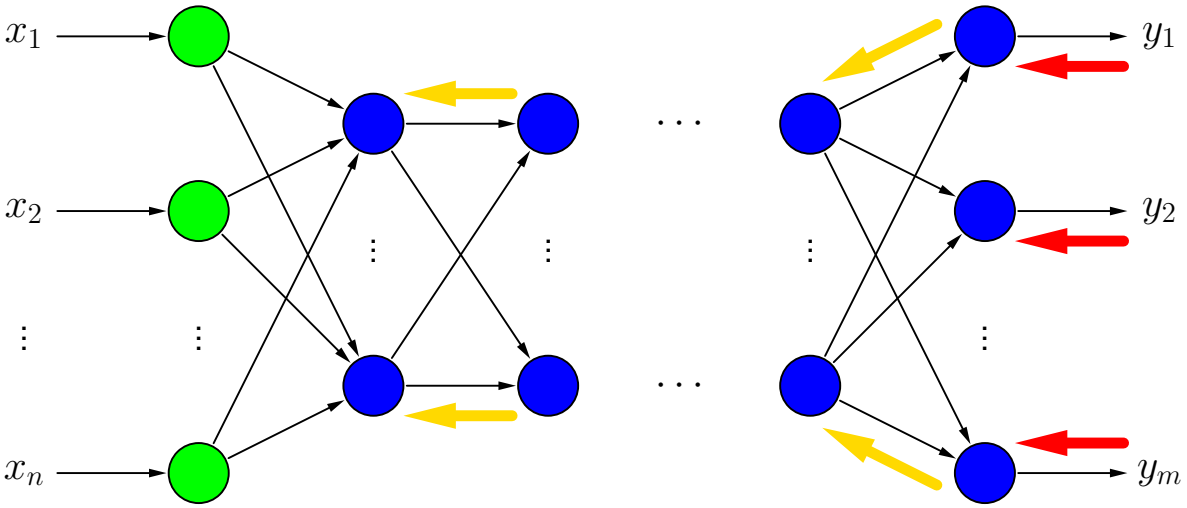
Aktivierungsfunktion: logistisch
Ausgabefunktion: Identität
impliziter Biaswert

$$\forall u \in U_{\text{in}}: \quad \text{out}_u^{(l)} = i_u^{(l)}$$

Setzen der Eingabe

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}: \quad \text{out}_u^{(l)} = \left(1 + \exp\left(-\sum_{p \in \text{pred}(u)} w_{up}\, \text{out}_p^{(l)}\right)\right)^{-1}$$

Vorwärtsweitergabe der Eingabe



Fehlerrückübertragung

Fehlerbestimmung

Rückwärts-propagation:

$$\forall u \in U_{\text{hidden}}: \quad \delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su}\right) \lambda_u^{(l)}$$

$$\forall u \in U_{\text{out}}: \quad \delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)}\right) \lambda_u^{(l)}$$

Aktivierungs-ableitung:

$$\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$$

Gewichts-änderung:

$$\Delta w_{up}^{(l)} = \eta\, \delta_u^{(l)}\, \text{out}_p^{(l)}$$

Algorithmus-Skizze (online)

gegeben: MLP mit $G = (U, C)$, Lernrate η, Trainingsbeispiele L_{fixed}

Initialisierung aller Gewichte (Zufallswerte)

wiederhole:

für jedes Trainingsbeispiel $l = (\vec{\imath}^{(l)}, \vec{o}^{(l)}) \in L_{\text{fixed}}$

Eingabe, Vorwärtsberechnung der Aktivierungen und Ausgabe:

$$\forall u \in U_{\text{in}}: \quad \text{out}_u^{(l)} = i_u^{(l)}$$

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}: \quad \text{out}_u^{(l)} = \left(1 + \exp\left(-\sum_{p \in \text{pred}(u)} w_{up}\, \text{out}_p^{(l)}\right)\right)^{-1}$$

Fehlerberechnung und Rückübertragung (Backpropagation):

$$\forall u \in U_{\text{out}}: \quad \delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)}\right) \lambda_u^{(l)}$$

$$\forall u \in U_{\text{hidden}}: \quad \delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su}\right) \lambda_u^{(l)}$$

mit Ableitung der Aktivierungsfunktion $\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$

Berechnung der Gewichtsänderung $\Delta w_{up}^{(l)} = \eta\, \delta_u^{(l)}\, \text{out}_p^{(l)}$ und Update

bis Stopkriterium erreicht

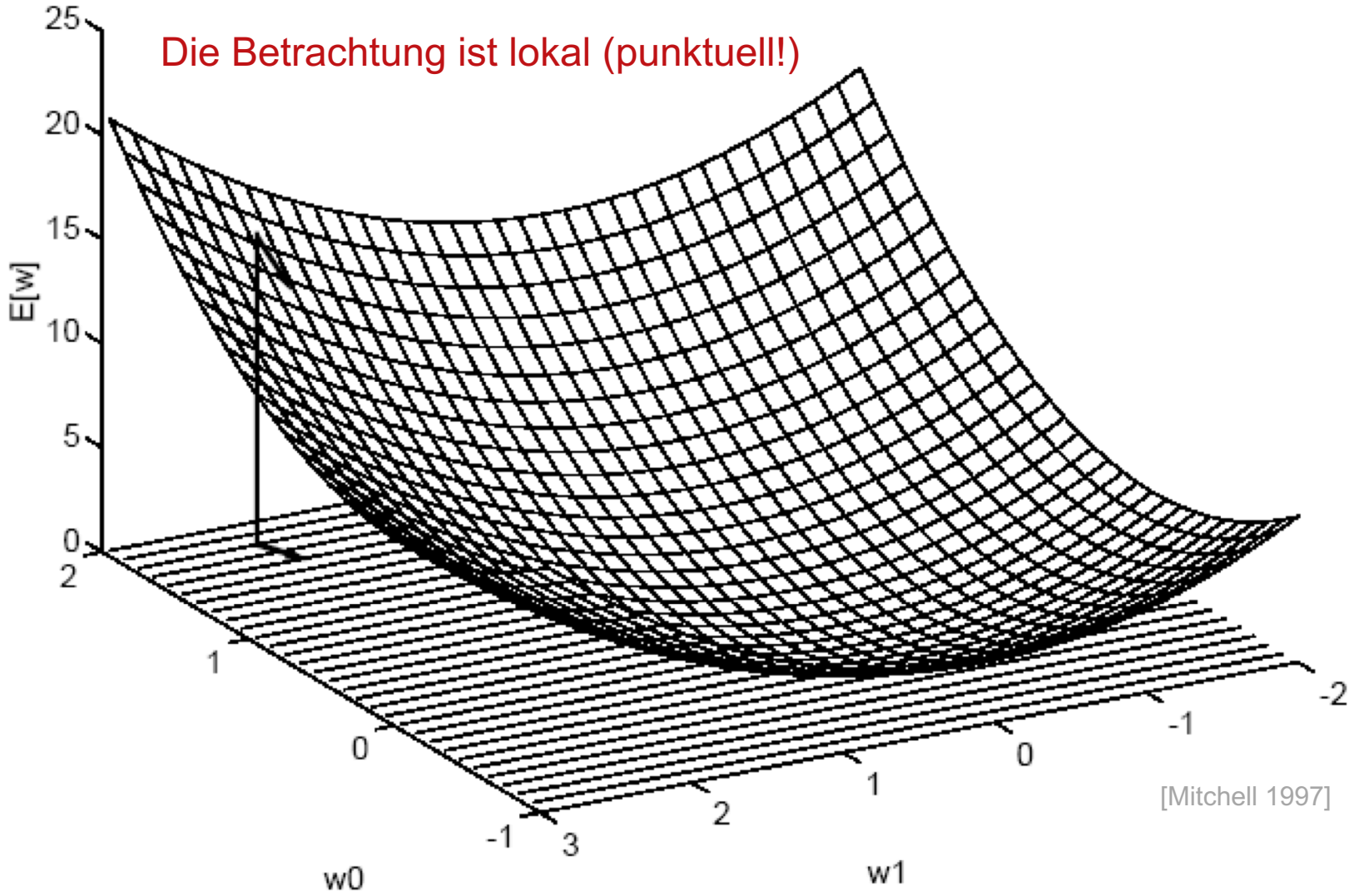
Gradientenabstieg

- In jedem Schritt
  (jeweils für das aktuelle Netzwerk mit sämtlichen Gewichten),
- für jedes Trainingsbeispiel (online) oder einen Batch,
- wird **punktuell (!)** für jedes Gewicht ein Gradient bestimmt.

d.h. insbesondere:

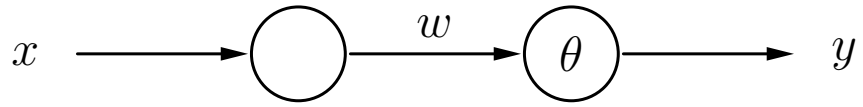
- Für jedes Trainingsbeispiel sieht das Fehlergebirge anders aus.
- Die Betrachtung ist lokal (immer nur für den Punkt im Fehlergebirge, der durch die aktuellen Gewichte gegeben ist).
- Es ist im Allgemeinen nicht praktikabel, das Fehlergebirge global zu betrachten.

Gradientenabstieg

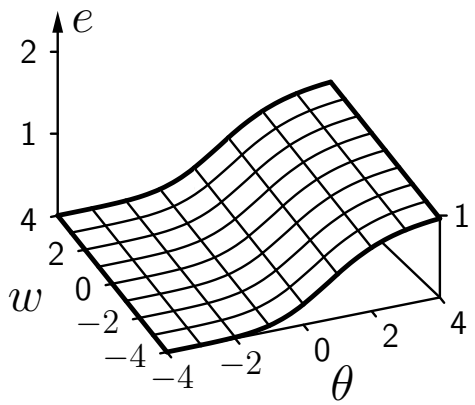


Gradientenabstieg

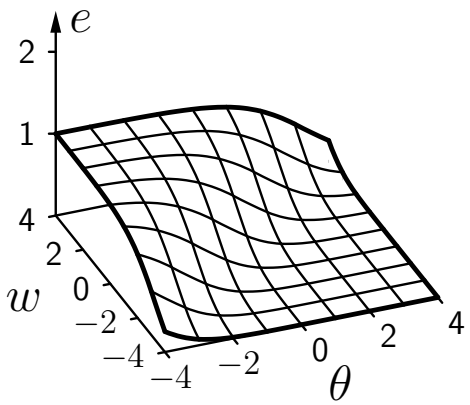
Gradientenabstieg für die Negation $\neg x$



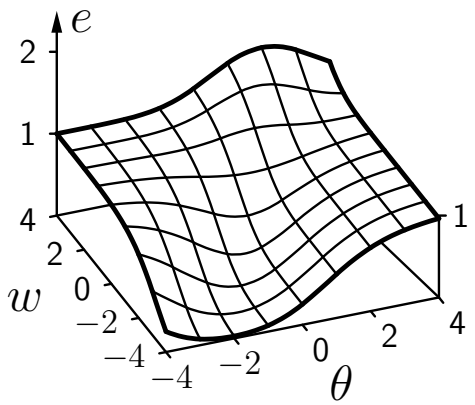
x	y
0	1
1	0



Fehler für $x = 0$



Fehler für $x = 1$

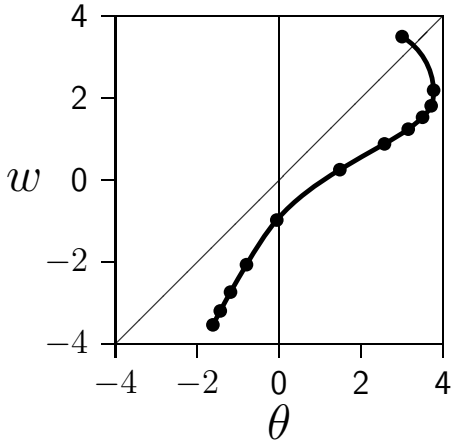


Summe der Fehler

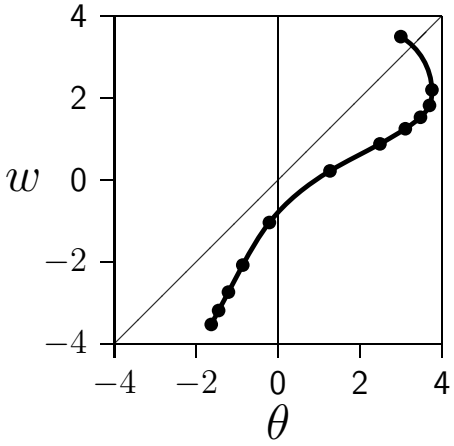
Für jedes Trainingsbeispiel sieht das Fehlergebirge anders aus.
Nur beim Batch-Training addieren sich die einzelnen Fehlergebirge zu einem.

Gradientenabstieg

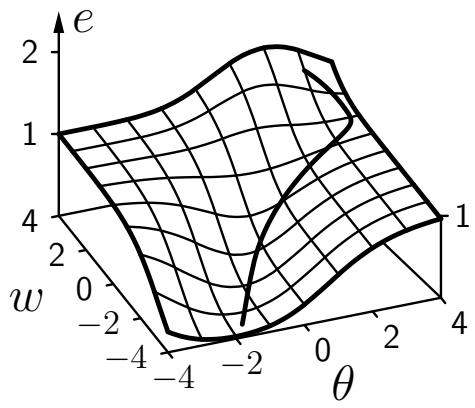
Visualisierung des Gradientenabstiegs für die Negation $\neg x$



Online-Training



Batch-Training



Batch-Training

Beim Online-Training kann es zu starkem Rauschen in den Gradienten kommen.

Beim Batch-Training besteht höhere Gefahr, in lokalen Minima stecken zu bleiben.

Allgemeine Praxis: Mini-Batch-Training

Minibatch Learning

- subset of the terms of the full cost function
- larger batches more accurate
  (less than linear return)
- compute power underutilized if too small
- memory often limiting factor
- power-of-2 sizes preferred on GPUs
- selected randomly, independent samples

SGD

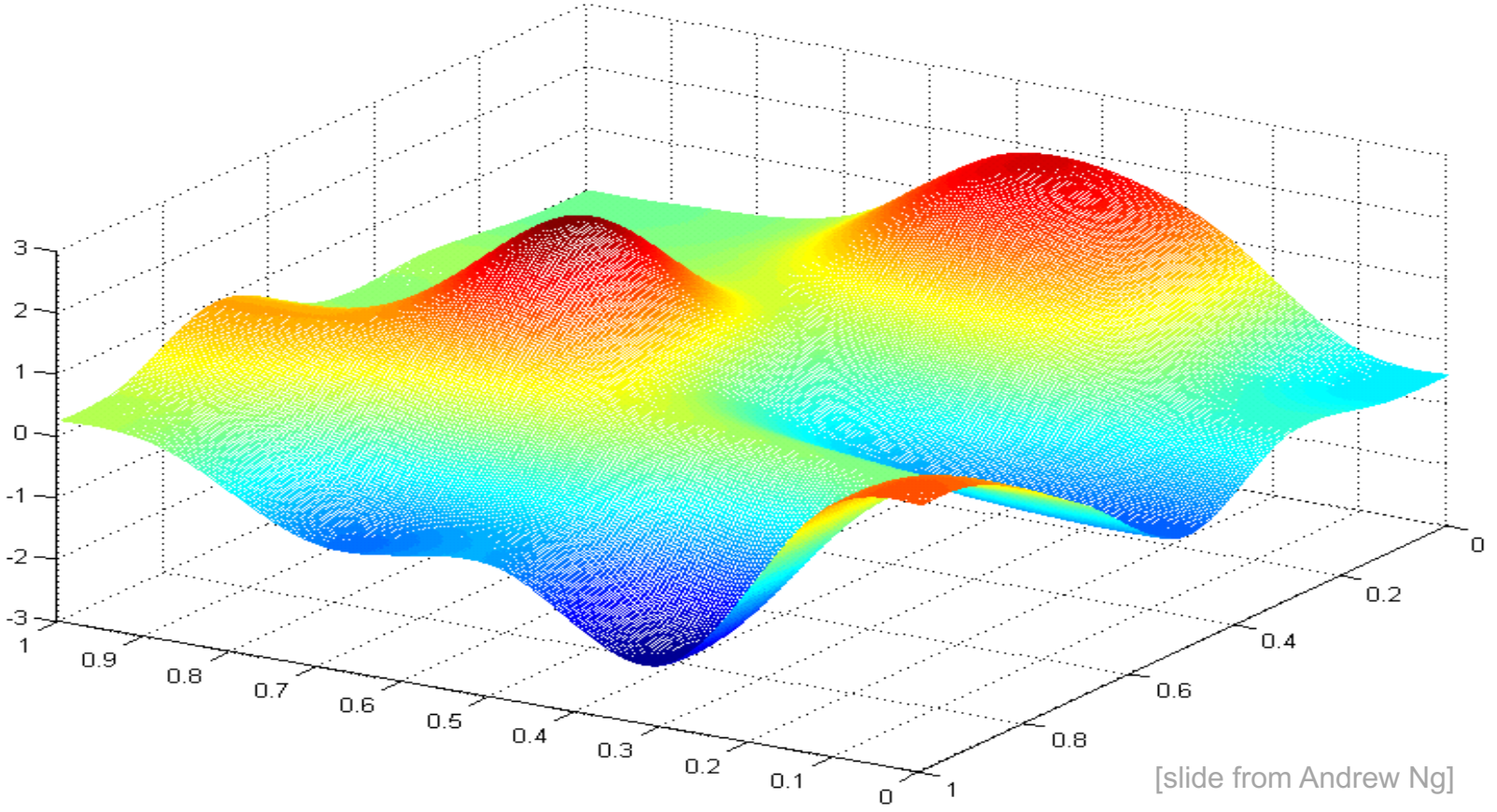
Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

- samples drawn i.i.d from data generating dist.
- guaranteed to converge if $\sum_{k=1}^{\infty} \epsilon_k = \infty$ and $\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$
- decrease learning rate over time

Gradientenabstieg

realistischeres Fehlergebirge (für 2 Gewichte)



[slide from Andrew Ng]

=> viele lokale Minima!
(u.a. Permutation der Neuronen)

Local Minima

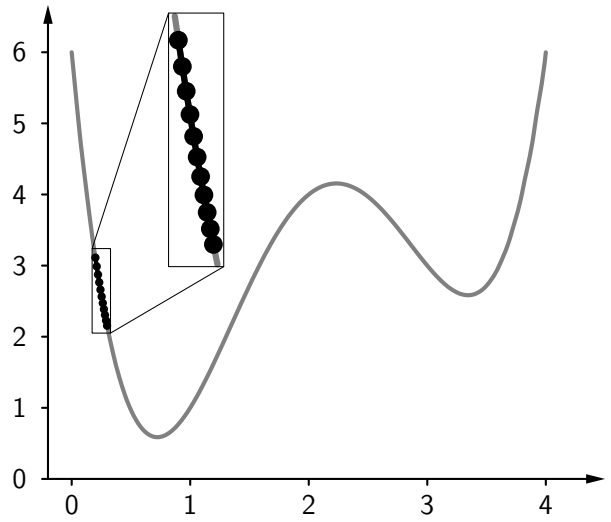
„For many years, most practitioners believed that local minima were a common problem plaguing neural network optimization. Today, that does not appear to be the case. The problem remains an active area of research, but experts now suspect that, for ***sufficiently large neural networks****, most local minima have a low cost function value, and that it is* ***not important to find a true global minimum*** *rather than to find a point in parameter space that has low but not minimal cost.“*

Plateaus, Saddle Points and Other Flat Regions

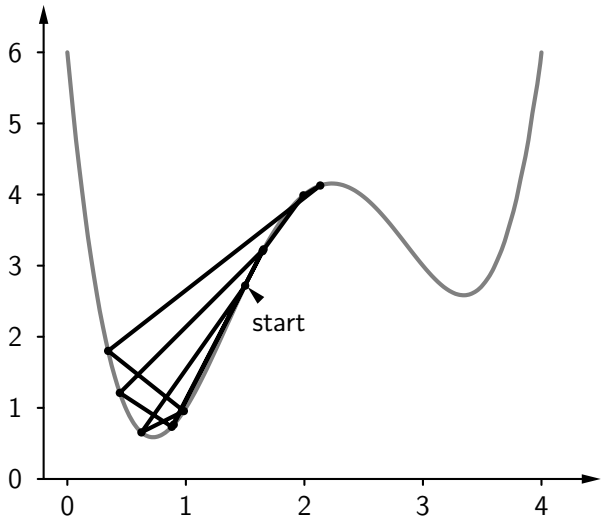
- saddle points
  - local min and local max w.r.t. different views
  - expected to be more common than local minima in high-dimensional cost functions
  - gradient descent empirically seems to be able to escape saddle points in many cases
- flat regions of constant value
  - very problematic for all numerical opt. methods

Einfluss der Lernrate

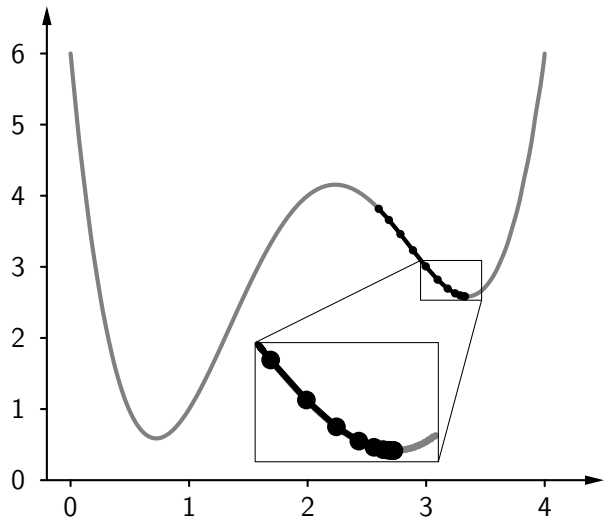
Beispielfunktion: $$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$



zu niedrig



zu hoch

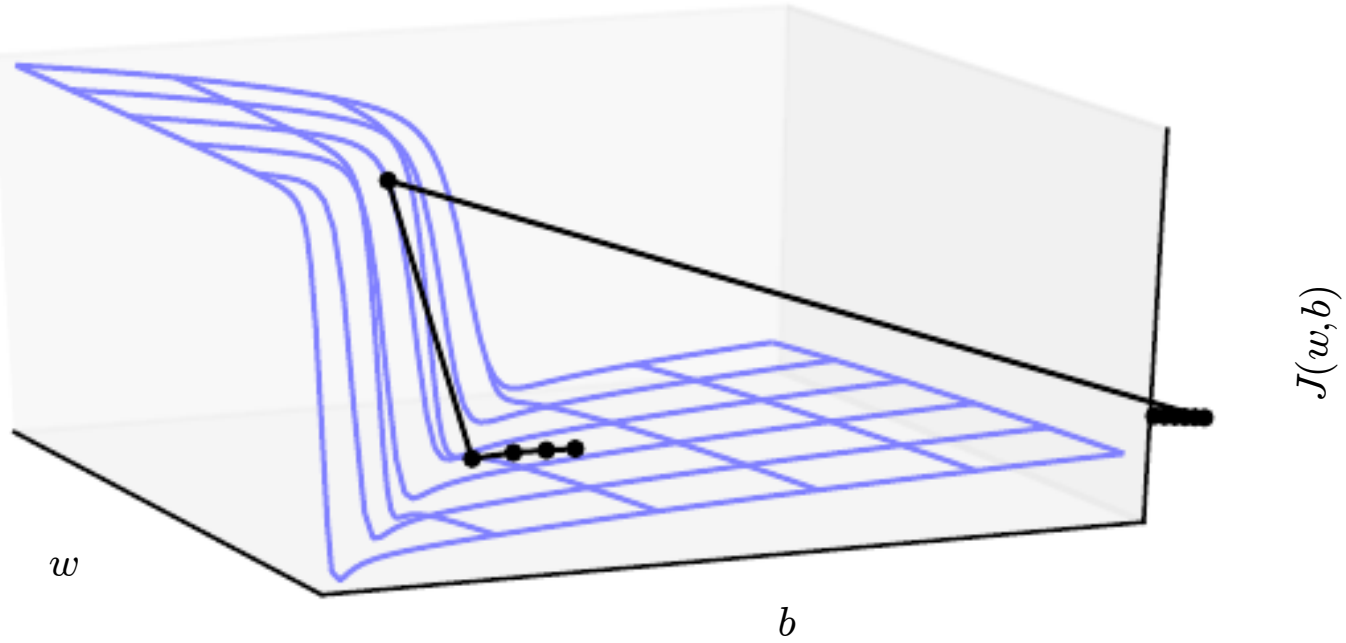


ungünstiger Startpunkt

Local vs. Global Landscape

- we can only make decisions based on local information
- problem: poor correspondence between local and global structure
  - direction that results in the most improvement locally does not point toward distant regions of much lower cost
  - remedy: initialize within well-behaved region

Exploding Gradients



- especially common in RNNs
- remedy: gradient clipping
  (keep direction but limit step size)

Varianten

Gewichts-Updateregel:

$$w(t+1) = w(t) + \Delta w(t)$$

Standard-Backpropagation:

$$\Delta w(t) = -\frac{\eta}{2} \nabla_w e(t)$$

Manhattan-Training:

$$\Delta w(t) = -\eta \, \mathrm{sgn}(\nabla_w e(t))$$

d.h. es wird nur die Richtung (Vorzeichen) der Änderung beachtet und eine feste Schrittweite gewählt

Moment-Term:

$$\Delta w(t) = -\frac{\eta}{2} \nabla_w e(t) + \beta \, \Delta w(t-1),$$

d.h. bei jedem Schritt wird noch ein gewisser Anteil des vorherigen Änderungsschritts mit berücksichtigt, was zu einer Beschleunigung führen kann

Beispiele

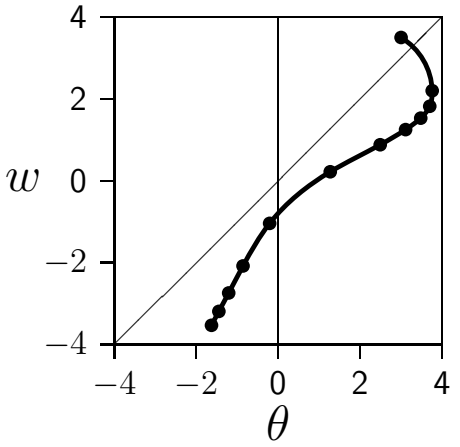
Epoche	θ	w	Fehler
0	3.00	3.50	1.295
20	3.76	2.20	0.985
40	3.70	1.82	0.970
60	3.48	1.53	0.957
80	3.11	1.25	0.934
100	2.49	0.88	0.880
120	1.27	0.22	0.676
140	-0.21	-1.04	0.292
160	-0.86	-2.08	0.140
180	-1.21	-2.74	0.084
200	-1.45	-3.19	0.058
220	-1.63	-3.53	0.044

ohne Momentterm

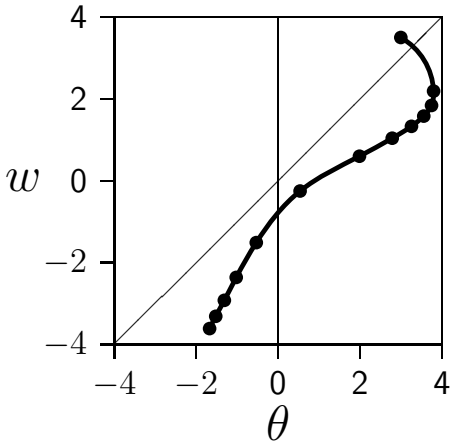
Epoche	θ	w	Fehler
0	3.00	3.50	1.295
10	3.80	2.19	0.984
20	3.75	1.84	0.971
30	3.56	1.58	0.960
40	3.26	1.33	0.943
50	2.79	1.04	0.910
60	1.99	0.60	0.814
70	0.54	-0.25	0.497
80	-0.53	-1.51	0.211
90	-1.02	-2.36	0.113
100	-1.31	-2.92	0.073
110	-1.52	-3.31	0.053
120	-1.67	-3.61	0.041

mit Momentterm

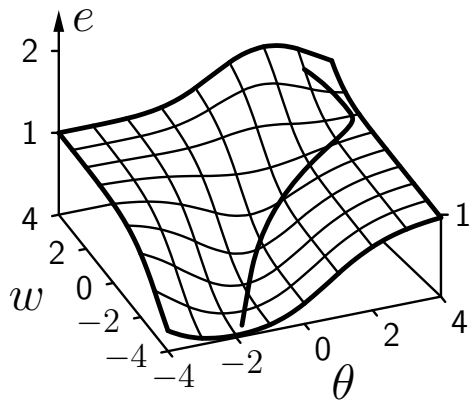
Beispiele



ohne Momentterm



mit Momentterm



mit Momentterm

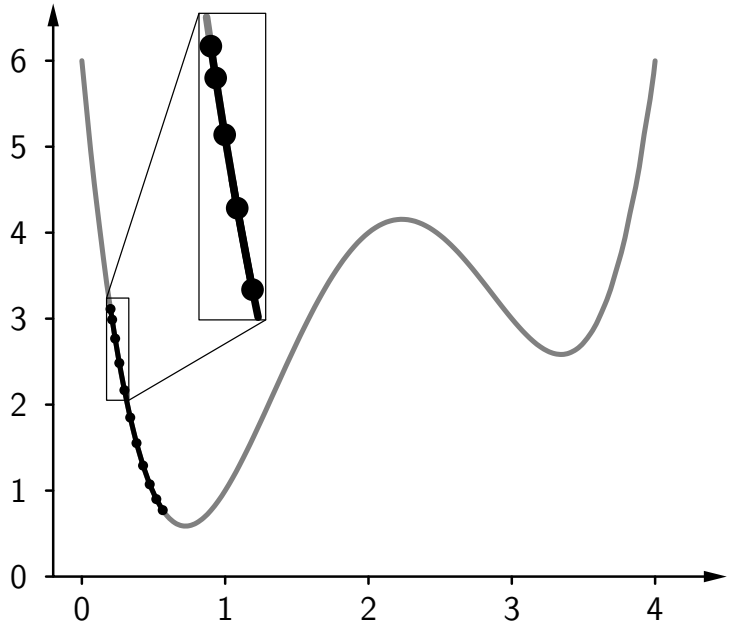
Punkte zeigen die Position alle 20 (ohne Momentterm) oder alle zehn Epochen (mit Momentterm).

Lernen mit Momentterm ist ungefähr doppelt so schnell.

Beispiele

Beispielfunktion: $$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

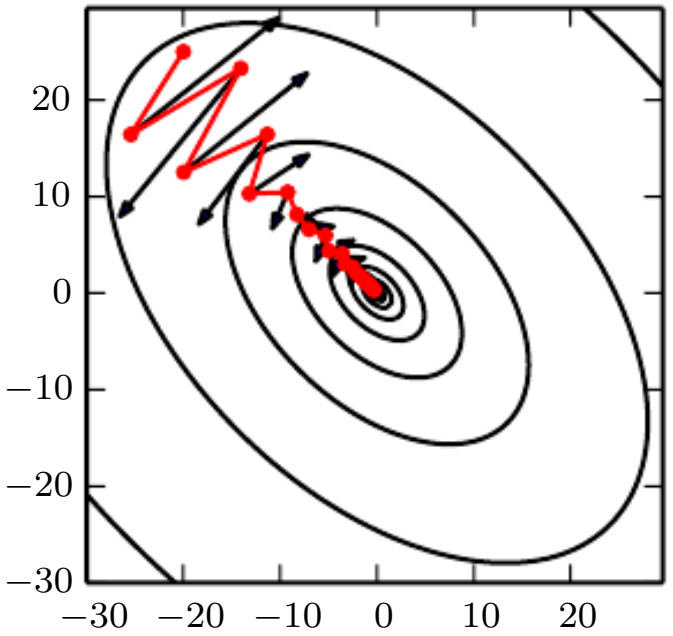
i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.112	−11.147	0.011
1	0.211	2.990	−10.811	0.021
2	0.232	2.771	−10.196	0.029
3	0.261	2.488	−9.368	0.035
4	0.296	2.173	−8.397	0.040
5	0.337	1.856	−7.348	0.044
6	0.380	1.559	−6.277	0.046
7	0.426	1.298	−5.228	0.046
8	0.472	1.079	−4.235	0.046
9	0.518	0.907	−3.319	0.045
10	0.562	0.777		



Gradientenabstieg mit Momentterm($\beta = 0.9$)

Momentum

accumulates an exponentially decaying moving average of past gradients and continues to move in their direction



Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

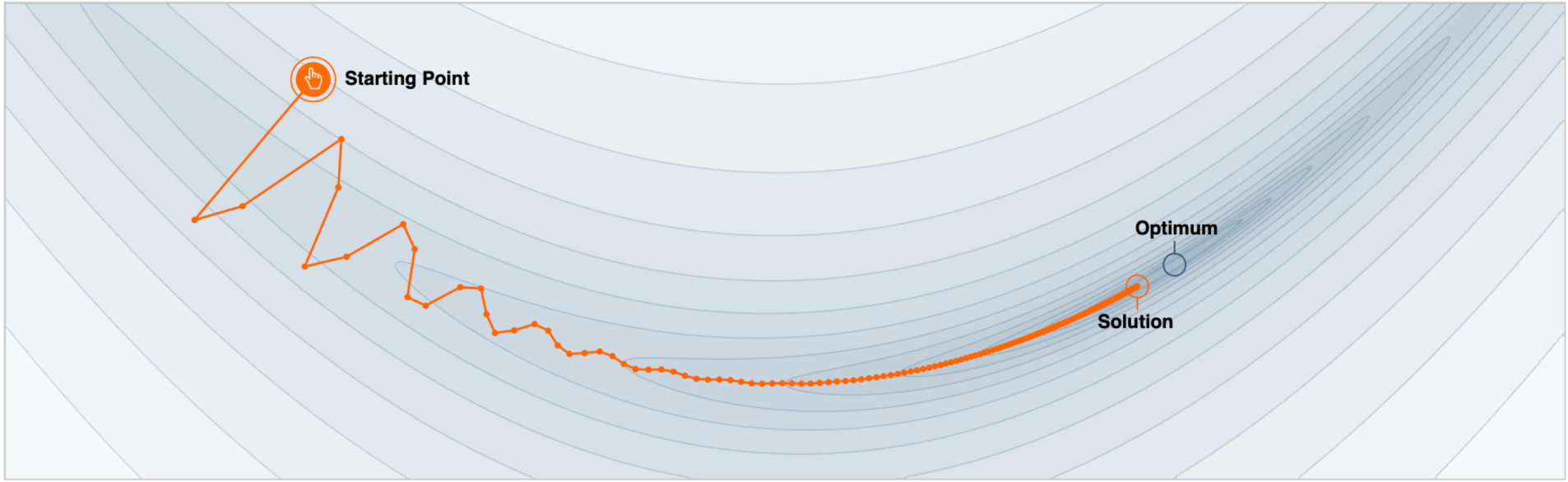
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

Momentum: Further Reading

Why Momentum Really Works



Step-size α = 0.02

0 0.003 0.006

Momentum β = 0.99

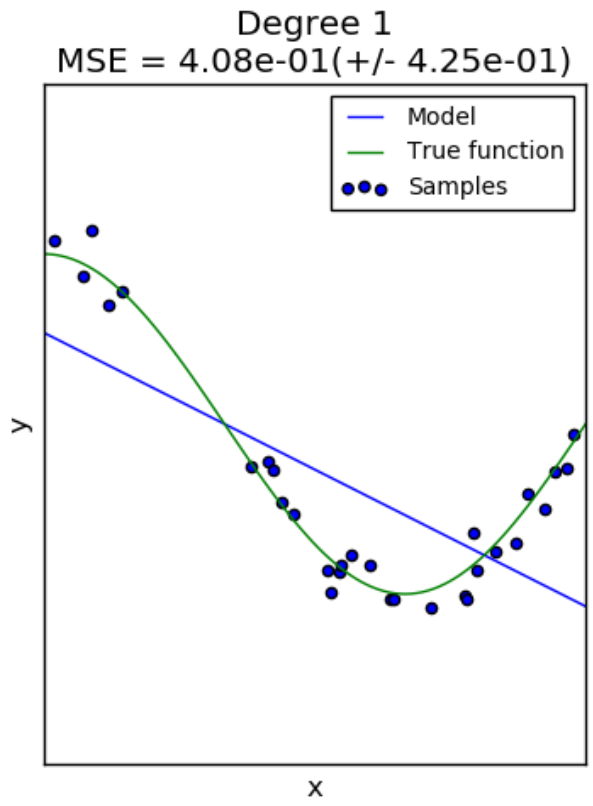
0.00 0.500 0.990

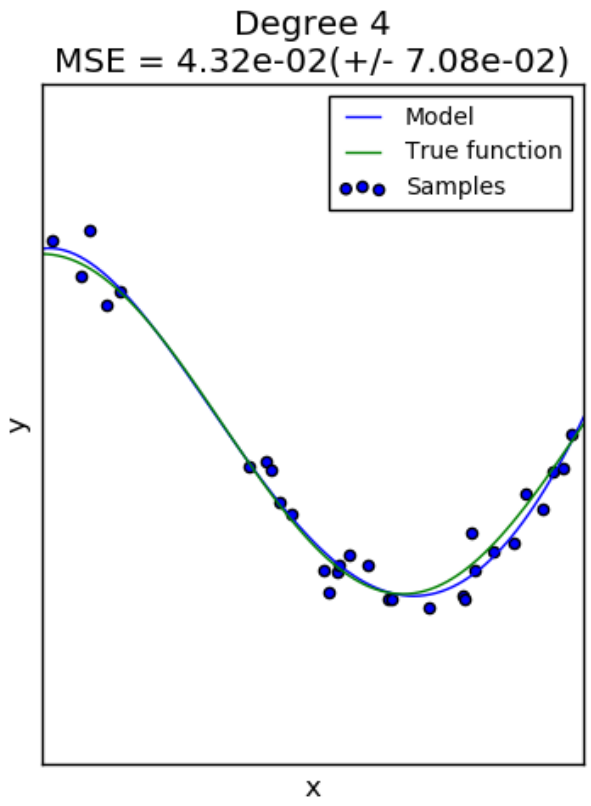
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

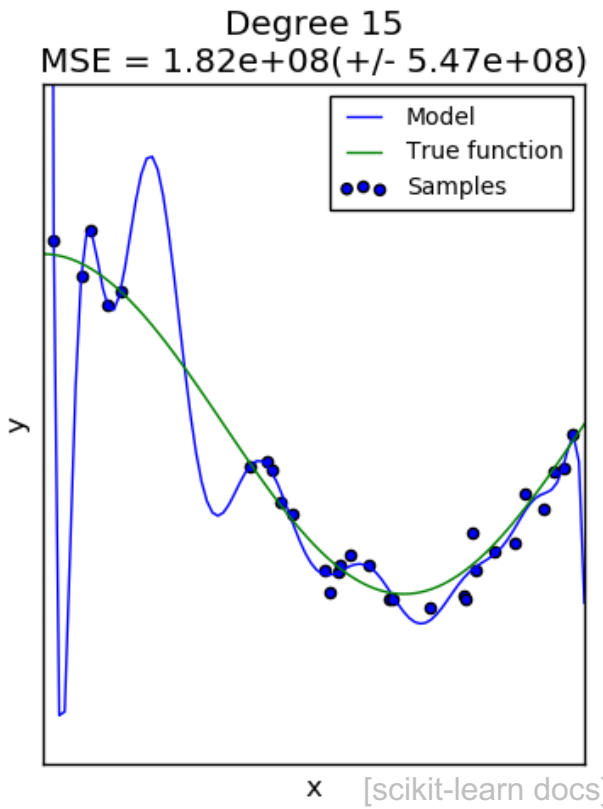
https://distill.pub/2017/momentum/

ML Grundlagen:
Bias, Variance & Regularisierung

Under- vs. Overfitting







[scikit-learn docs]

How to improve generalization performance?

- underfitting: more training or increase model *capacity*
- overfitting: less training or decrease model *capacity*

Under- vs. Overfitting

Prinzip des Trainings- und Validierungsdatenansatzes:

Underfitting: Ist die Anzahl an versteckten Neuronen zu gering, ist das MLP of nicht in der Lage, die Abhängigkeit zwischen Ein- und Ausgabedaten präzise genug darzustellen, da hierfür weitere Parameter benötigt werden.

Overfitting: Mit einer zu großen Anzahl an Neuronen (zur Verfügung stehenden Parametern) stellt das MLP nicht nur die Abhängigkeit zwischen Ein- und Ausgabedaten dar, sondern auch ungewünschte nebensächliche Zusammenhänge des Trainingsdatensatzes.

Overfitting führt meistens zu einem im Vergleich zum Trainingsdatensatz größeren Fehler auf dem Validierungsdaten. Der Grund hierfür sind die wahrscheinlich leicht anderen zufälligen Zusammenhänge im Validierungsdatensatz.

Overfitting kann durch die geeignete Wahl an versteckten Neuronen bezüglich eines minimalen Fehlers auf den Validierungsdaten vermieden werden.

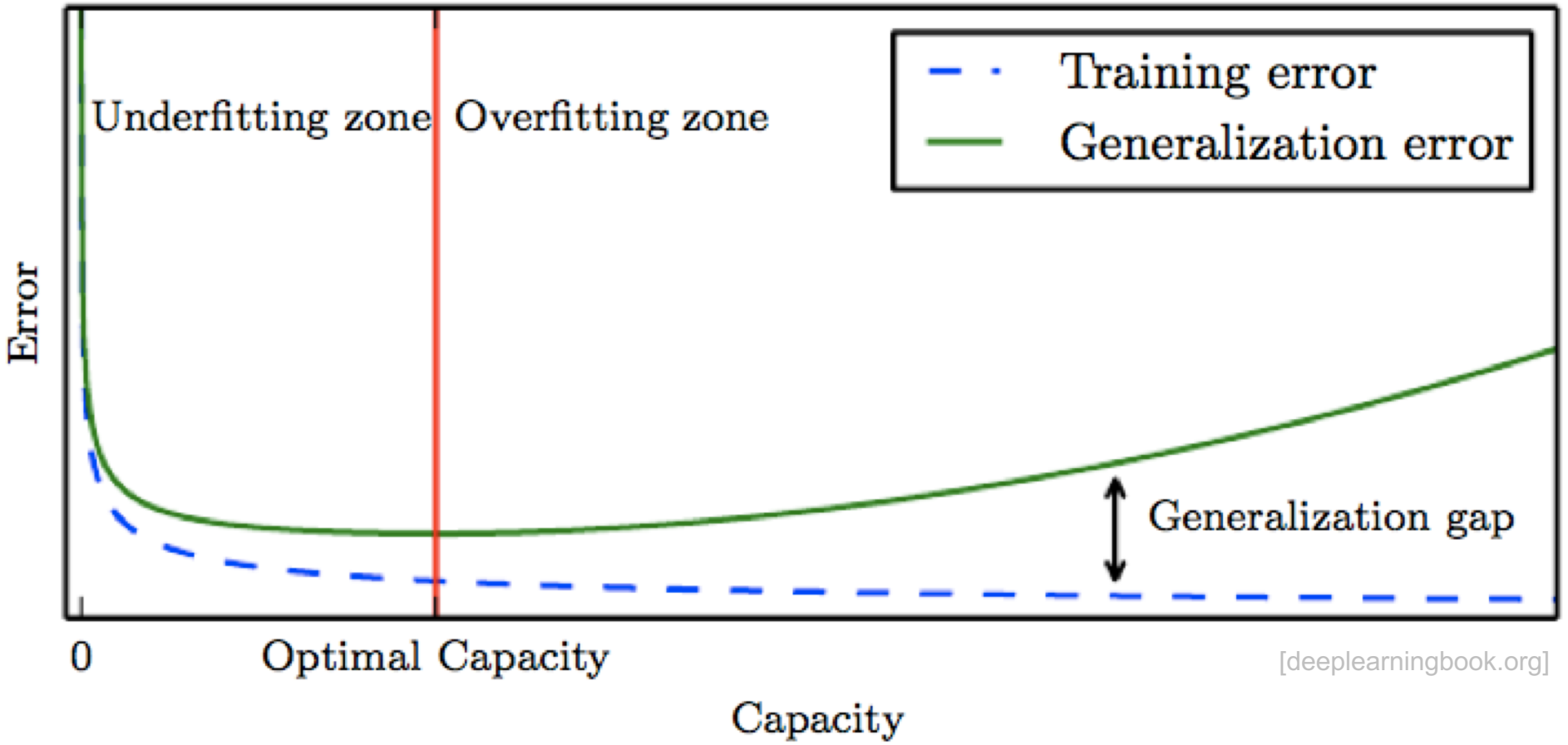
Kreuzvalidierung (Cross-Validation)

Die beschriebene Methode zur Aufteilung der Daten in Trainings- und Validierungsdaten wird manchmal auch Kreuzvalidierung genannt.

Typischer ist jedoch die folgende Beschreibung von Kreuzvalidierung:

> Der vorhandene Datensatz wird in n gleichgroße Teile aufgeteilt.
> (n-fache Kreuzvalidierung)
>
> Ist die Ausgabe nominal (auch symbolisch oder kategorisch genannt), wird die Aufteilung so vorgenommen, dass die relative Anzahl der Ausgabewerte konstant über die Anzahl der Teildatensätze ist.
> Dieses Vorgehen wird auch Stratifikation genannt.
>
> Aus diesen n Teildatensätzen werden n Trainings- und Validierungsdatenpaare geformt indem jeweils $n - 1$ Teildatensätze den Trainingsdatensatz bilden und der übrige Teildatensatz zur Evaluierung verwendet wird.

Modell-Kapazität



Bewertung & Selektierung von Modellen nur auf der Basis bisher ungesehener Daten!

Exception: 1st Epoch

- minibatch SGD follows the gradient of the **true generalization error** as long as no examples are repeated

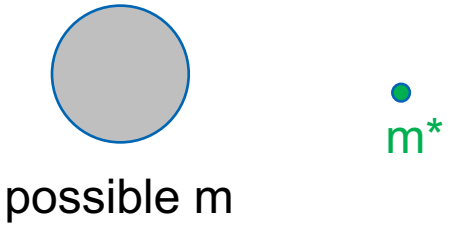
- i.e. if each training sample is used only once, there is no need for a validation set

Bias-Variance Trade-Off

- conflict: choose model that
  - accurately captures the training data regularities
  - generalizes well to unseen data
- bias
  - error from (implicit or explicit) assumption of the learning algorithm => underfitting
- variance
  - error from fitting to small fluctuations => overfitting

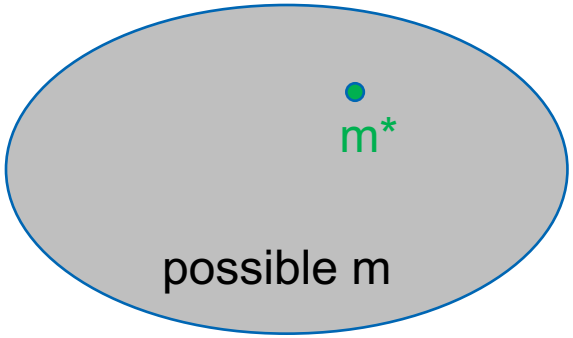
Bias-Variance Trade-Off

- high bias, low variance:

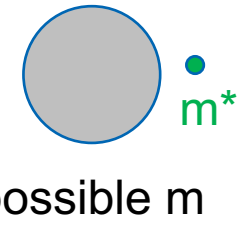


- low bias, high variance:

regularize!



- good trade-off:



Regularisierung

“In practical deep learning scenarios, we almost always do find—that the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately.”

[deeplearningbook, Ch7]

Parameter Norm Regularization

model parameters (in neural networks, these are typically the weights and biases)

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

training data with labels

general idea:

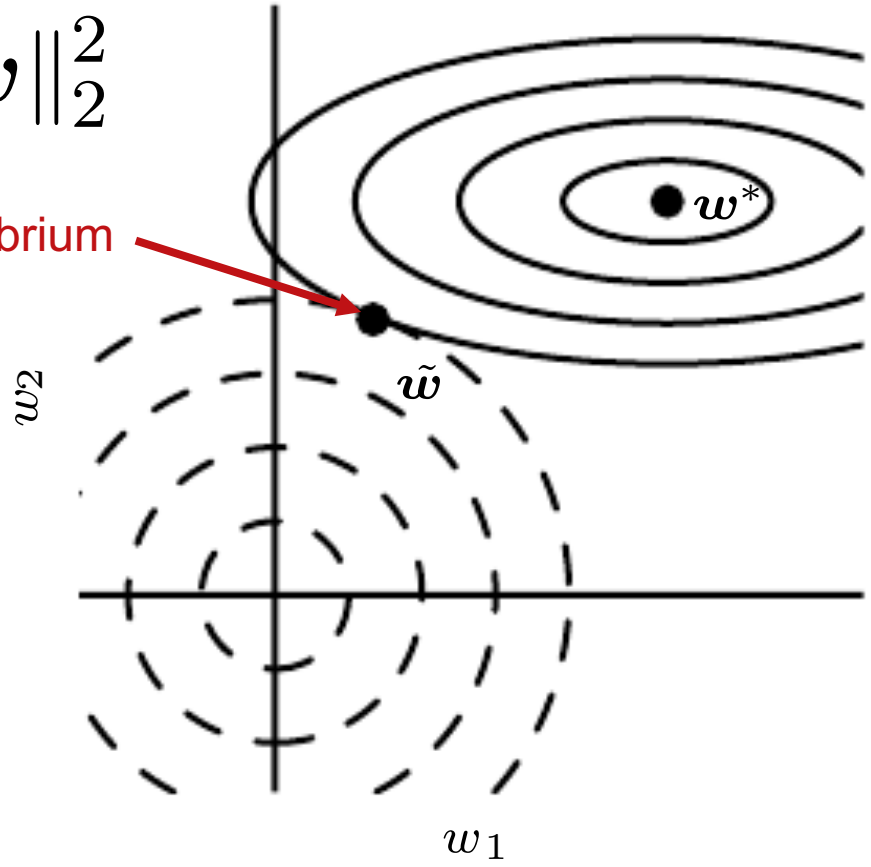
- add a regularization term to the loss function
- favours / penalizes certain parameter values
- typically combined with a factor α that balances the strength of the normalization
- typically only regularize weights

L2 Regularization a.k.a. weight decay

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

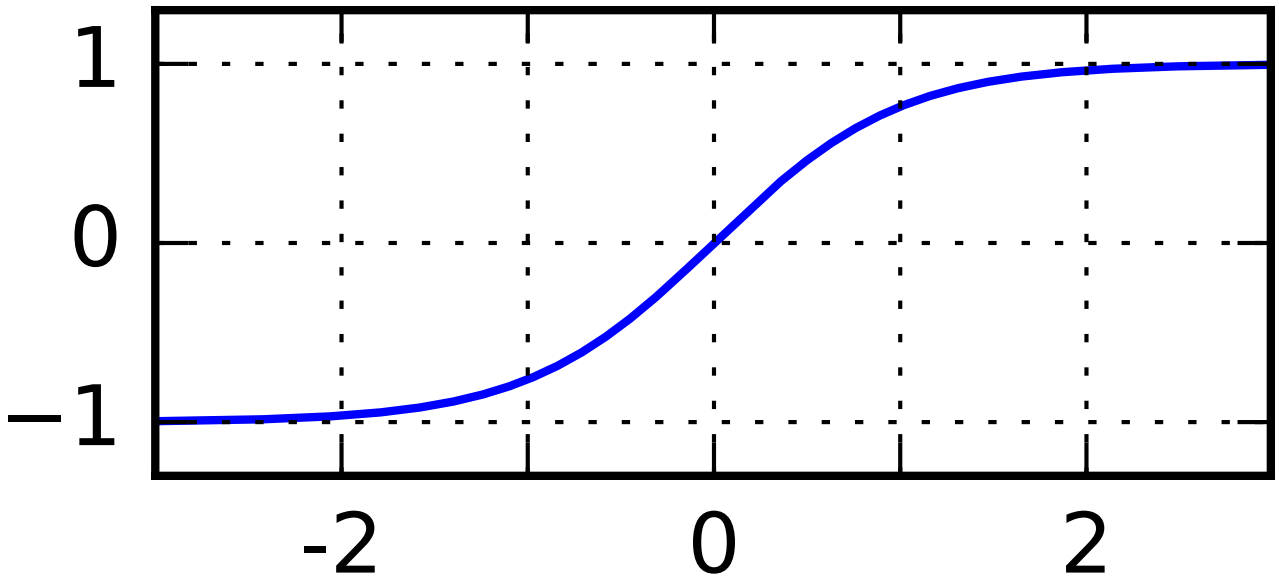
where $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$

- penalizes large weights
- weights are gradually driven towards the origin



Special Case: Sigmoids

(here: tanh)



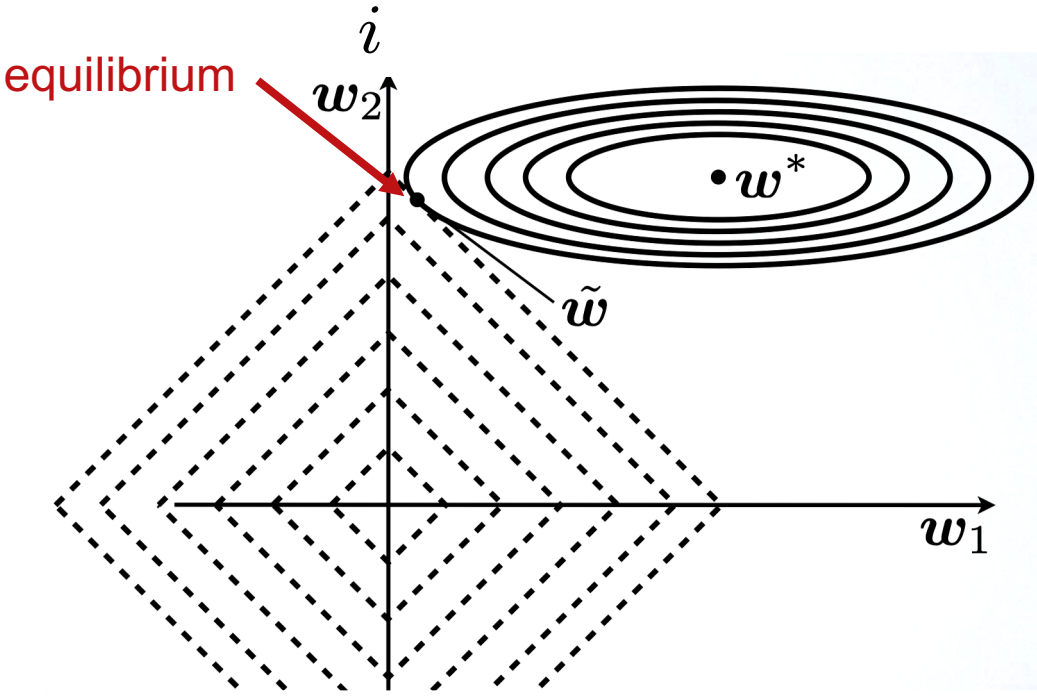
- near-linear behavior when input close to 0 (obtained if weights close to 0)

L1 Regularization

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

where $\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|$

- encourages sparsity
- feature selection mechanism (features with weight 0 can be discarded)



Early Stopping

idea: monitor performance on validation set and stop when peak is reached

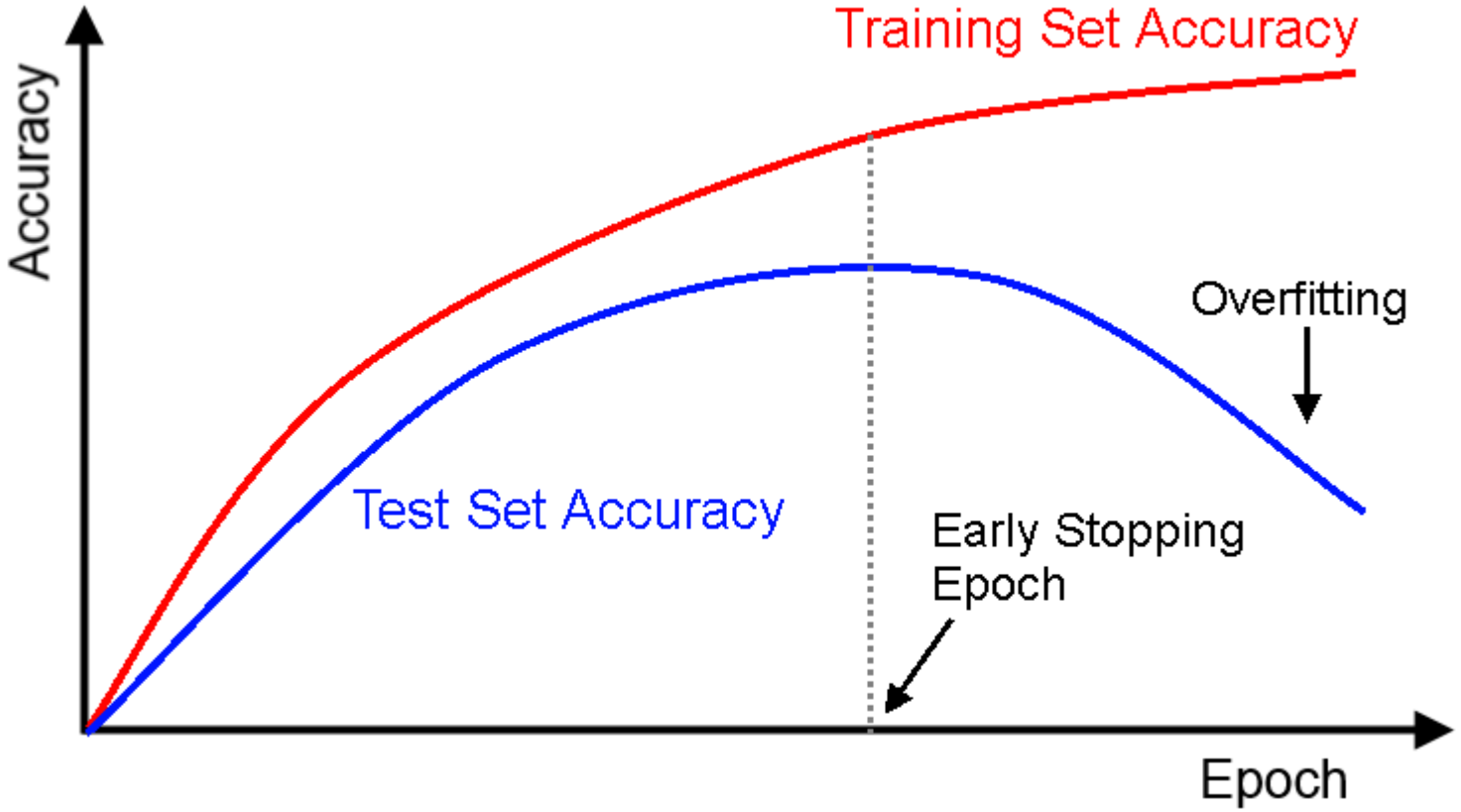
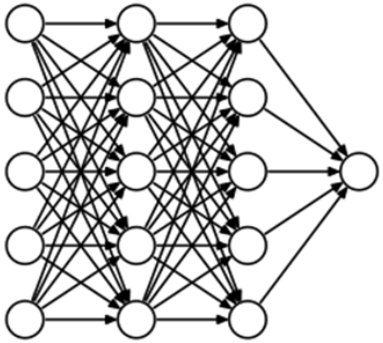
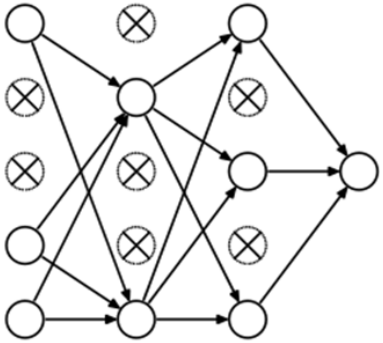


figure source: https://deeplearning4j.org/docs/latest/deeplearning4j-nn-early-stopping

Dropout

ohne Dropout





mit Dropout

Gewünschte Eigenschaft:
Robustheit bei Ausfall von Neuronen

Ansatz beim Lernen:
- Nutze nur p% der Neuronen ($p < 50$)
- Wähle diese zufällig

Ansatz beim Anwenden:
- Nutze 100% der Neuronen
- Multipliziere alle Gewichte mit p

Ergebnis:
- Robustere Repräsentation
- Verbesserte Generalisierung
- Verringerte Überanpassung

More Regularization Techniques

- more data / data augmentation
- adding noise / denoising
- semi-supervised learning
- multi-task learning
- parameter tying & sharing
- sparse representations
- bagging / ensembles
- DropConnect = randomly set weights to zero
- (layer-wise) unsupervised pretraining
- adversarial training
- …

Extra: Practical Methodology

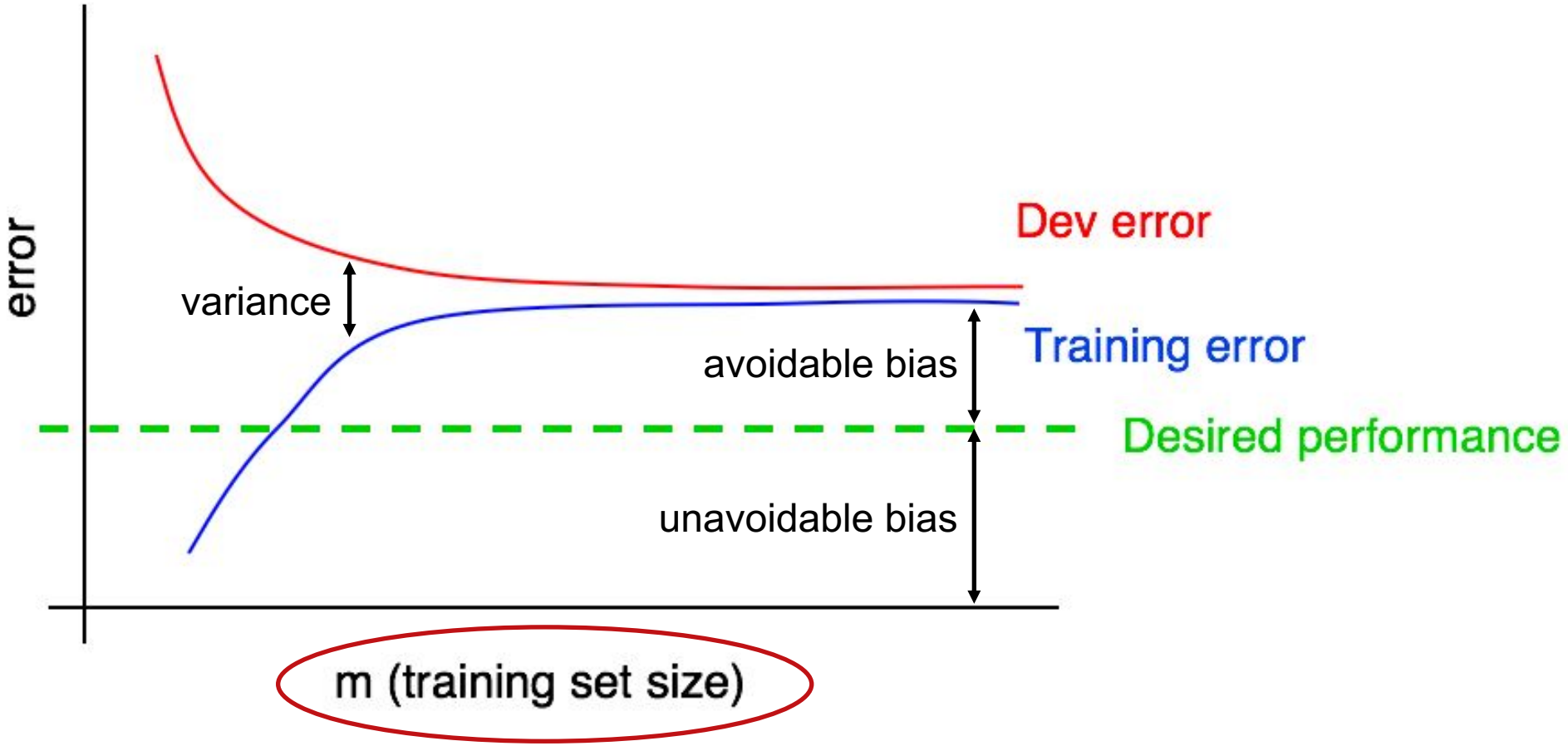
adapted from Andrew Ng. “Machine Learning Yearning” (draft), 2018

Bias & Variance (continued)

- optimal error rate (“unavoidable bias”)
  - needs to be estimated somehow (e.g. human error)
- avoidable bias (training error – optimal error rate)
- “variance” (generalization error)

- high avoidable bias (underfitting)
  - try to reduce training set error first: increase model size (capacity), modify input features, reduce regularization
- high variance (overfitting)
  - regularize, add more data, decrease model size, decrease number/type of input features (selection)
- both: modify model architecture

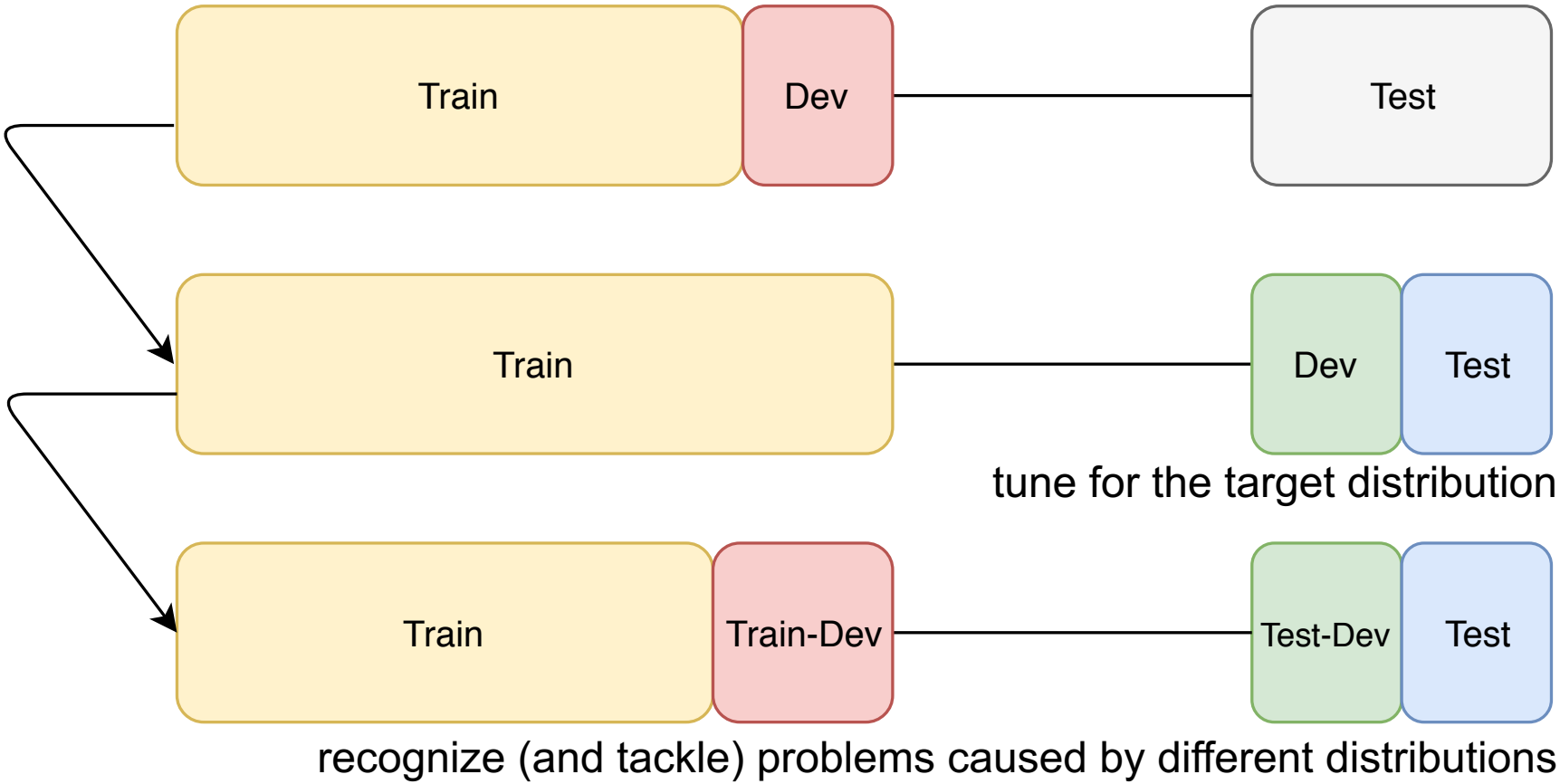
Learning Curves



adapted from A. Ng. “Machine Learning Yearning” (draft), 2018

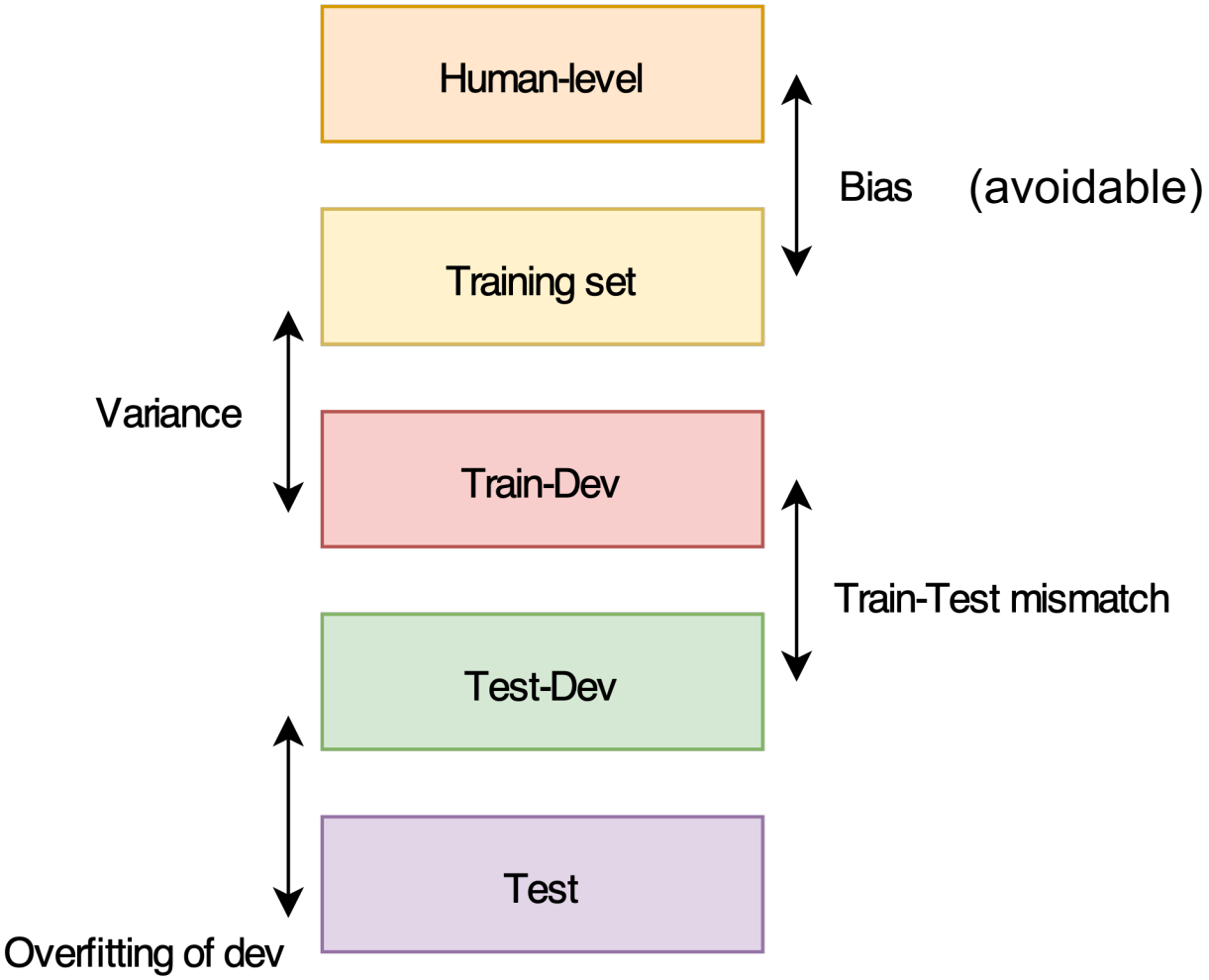
Data Splits for Different Distributions

=> Make dev and test sets come from the same distribution!



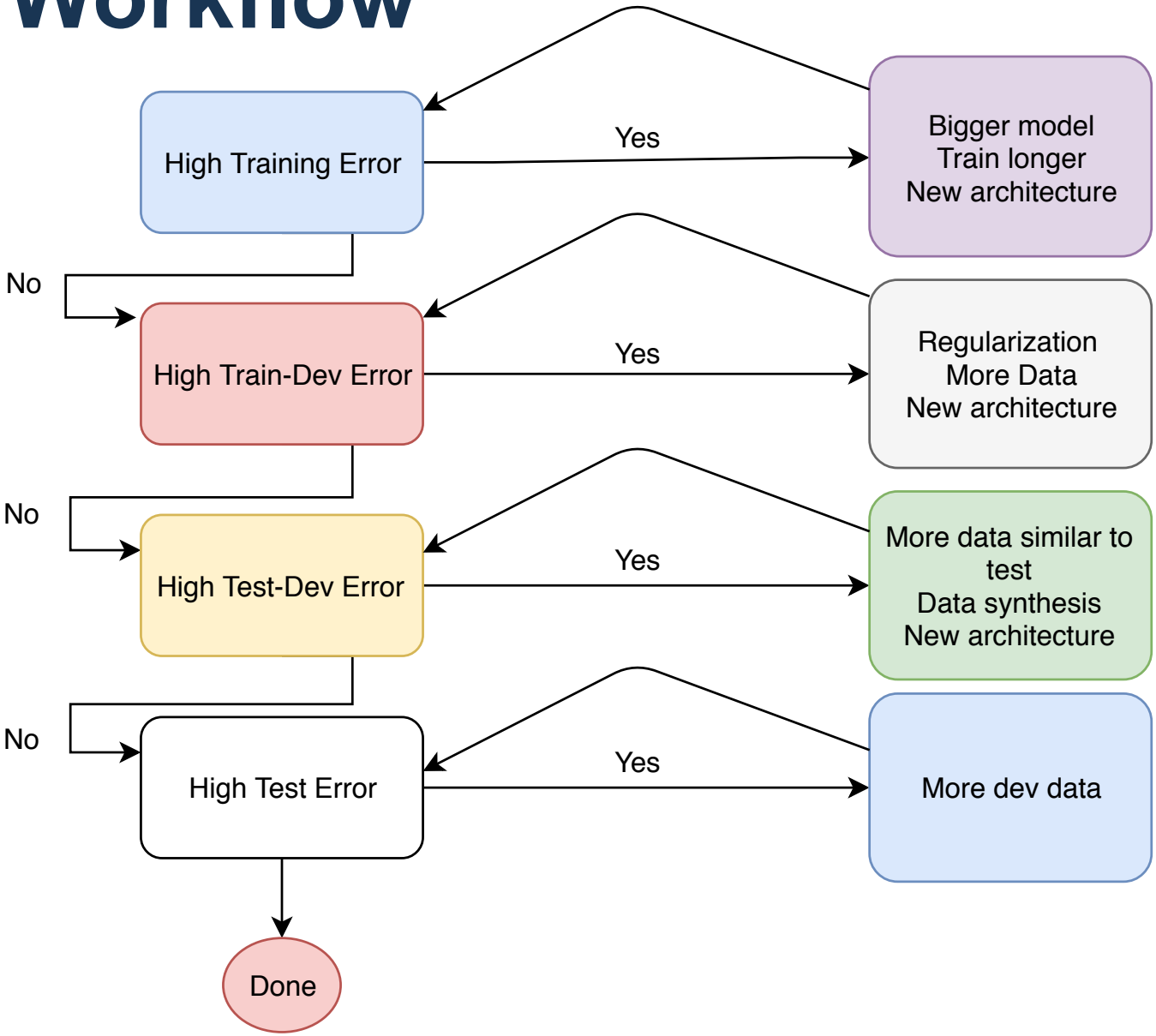
https://kevinzakka.github.io/2016/09/26/applying-deep-learning/

Error Factors



https://kevinzakka.github.io/2016/09/26/applying-deep-learning/

Workflow



(bias)

(variance)

(train-test data mismatch)

(overfit dev)

https://kevinzakka.github.io/2016/09/26/applying-deep-learning/

Further Reading

- http://mlyearning.org/

- http://mlexplained.com/2018/04/24/overfitting-isnt-simple-overfitting-re-explained-with-priors-biases-and-no-free-lunch/