

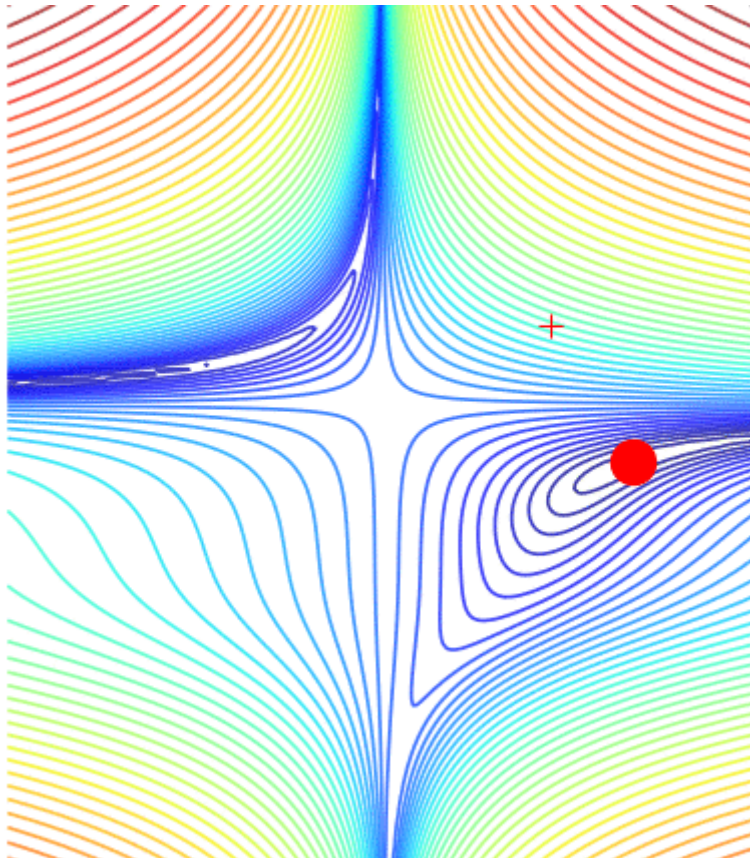
Einsatz von Gradientenabstiegsverfahren in Neuronalen Netzen

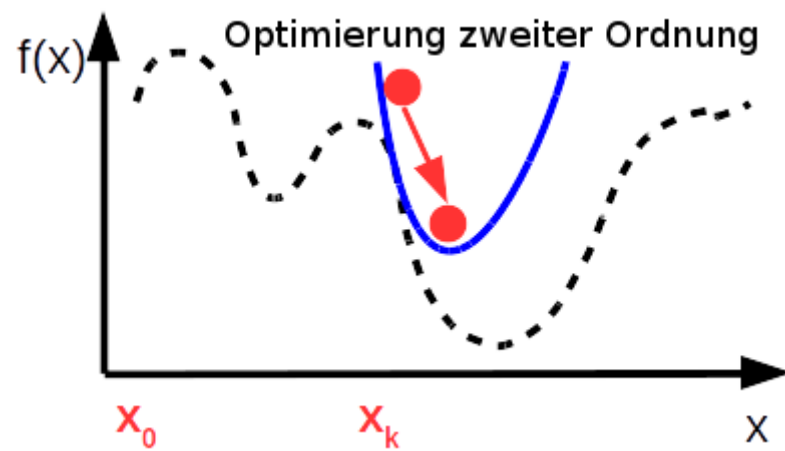
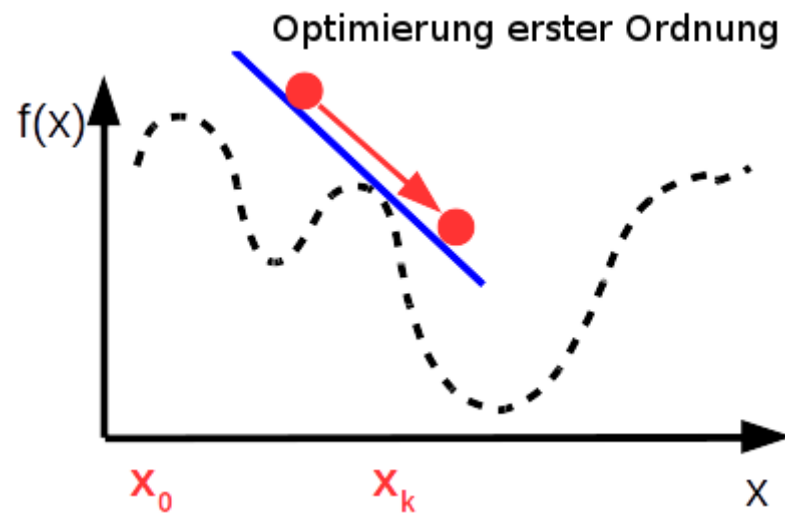
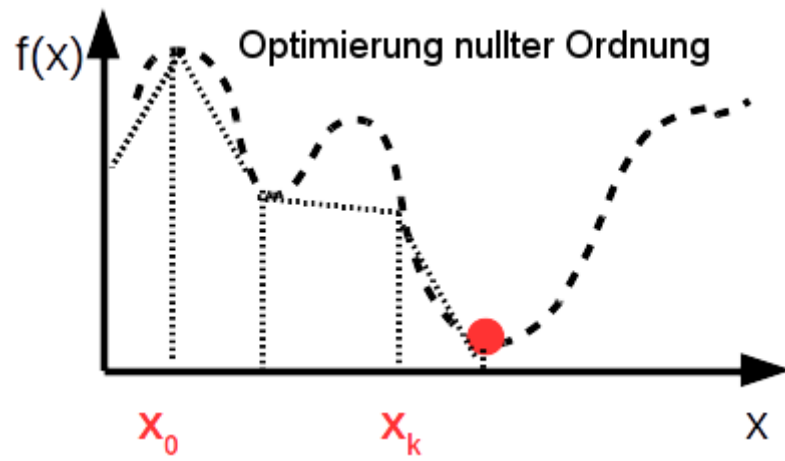
Inhalt

- Grundlagen der Optimierung
- Gradientenabstieg
- Lernen als Optimierung in Neuronalen Netzen
- Gradientenabstieg in Neuronalen Netzen
- Nachteile des Gradientenabstiegs
- Fazit

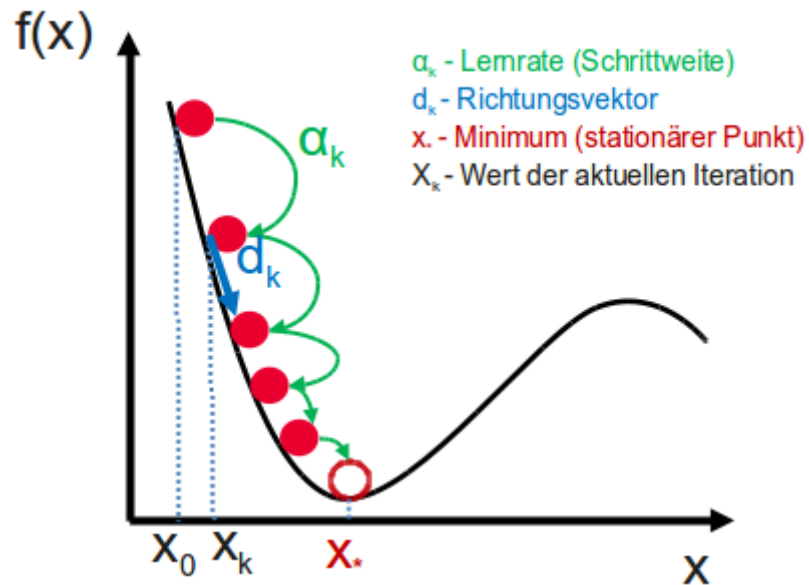
Optimierungsgrundlagen

Optimierungsalgorithmen sind in der Regel **iterative Verfahren**. Ausgehend von einem gegebenen Punkt $x_0(+)$ erzeugen sie eine Folge x_k **iterierten**, die zu einer Lösung (\bullet) **konvergieren** [2].





Gradientenabstieg

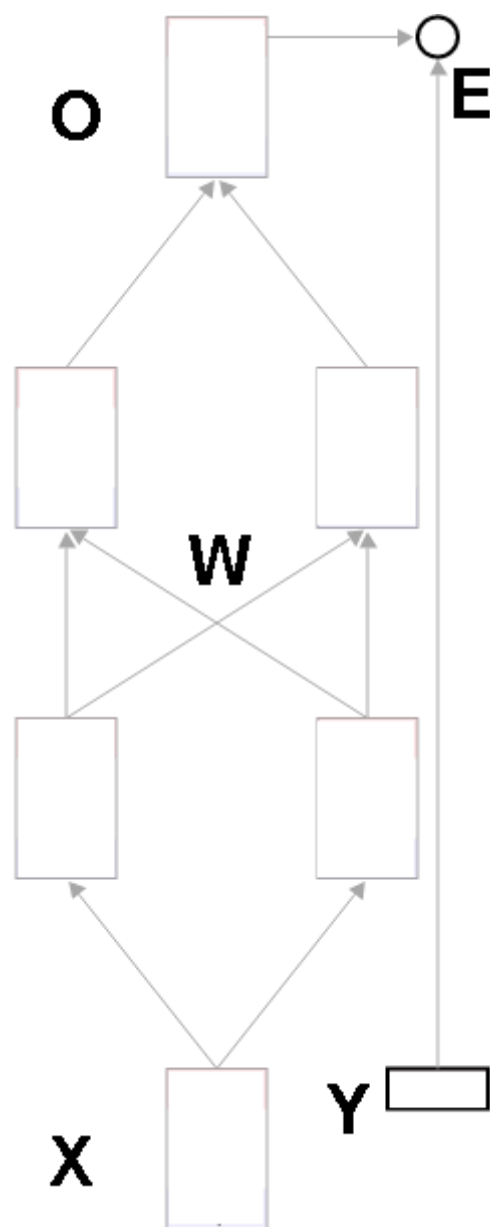


Lernen als Optimierung in Neuronale Netzwerk

Optimierungsalgorithmen helfen uns, eine **Zielfunktion zu minimieren (oder zu maximieren)**. Ein solches Zielfunktion ist in **neuronalen Netzen** einfach eine **mathematische Funktion (E)**, die von den **internen lernbaren Parametern (W)** des Modells abhängt [3].

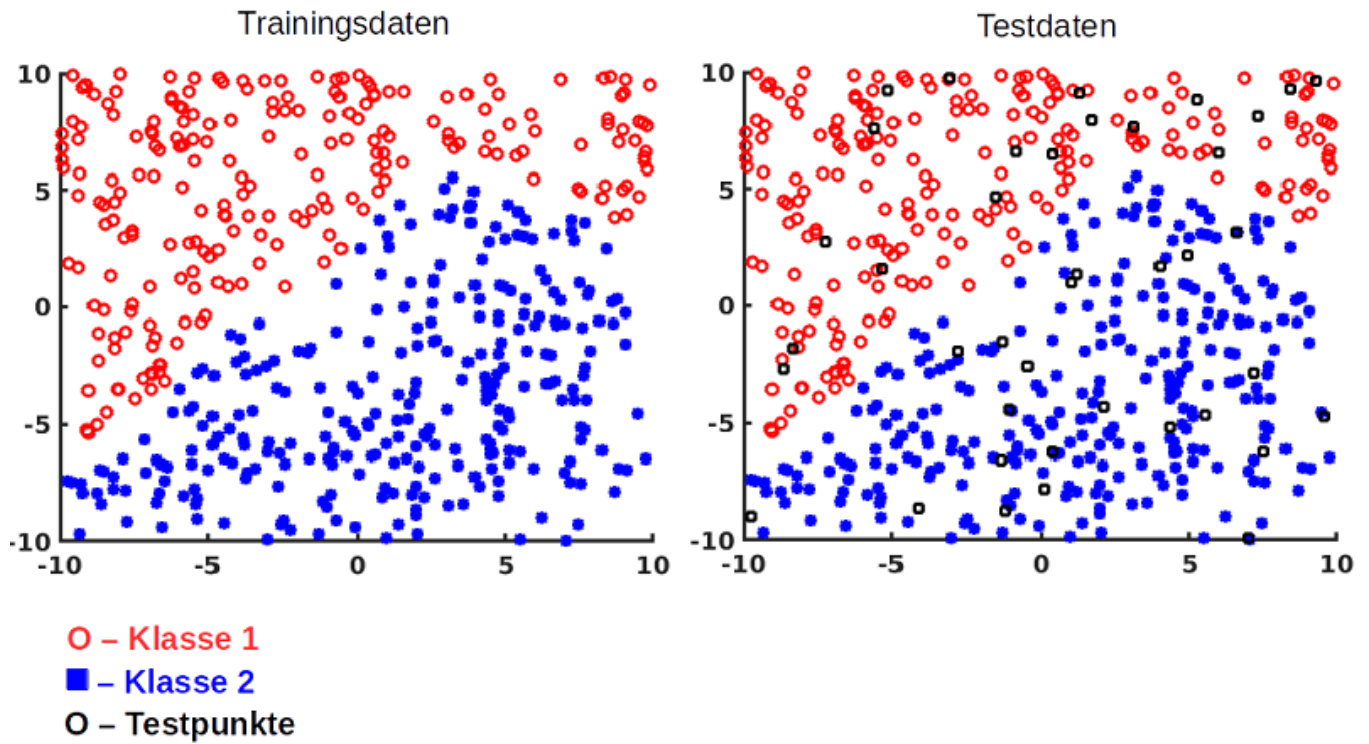
Diese **Parameter** werden bei der Berechnung der **erwarteten Werte (Y)** aus dem Satz von **Prädiktoren (X)** .

E beschreibt die Differenz zwischen dem erwarteten Wert und dem **tatsächlichen Netzwerkausgangswert (O)**.



Gradientenabstieg in Neuronale Netzwerk

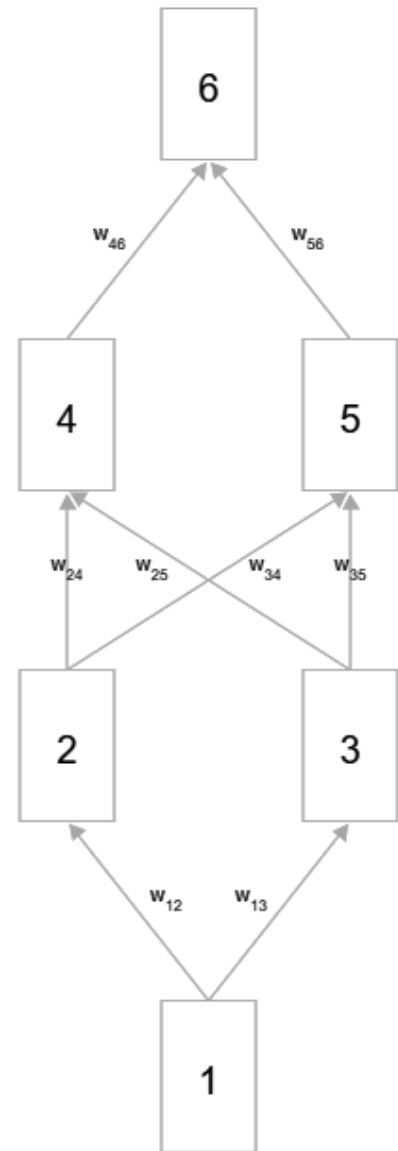
Binäre Klassifikation mit neuronalen Netzen



Neuronale Netzwerkstruktur

Neuronales Netzwerk zur Lösung der Binärklassifizierungsaufgabe: Ein Eingangsneuron der Dimension 2 ((x, y) - Koordinaten der Punkte), ein Ausgangsneuron und zwei versteckte Schichten mit jeweils zwei Neuronen.

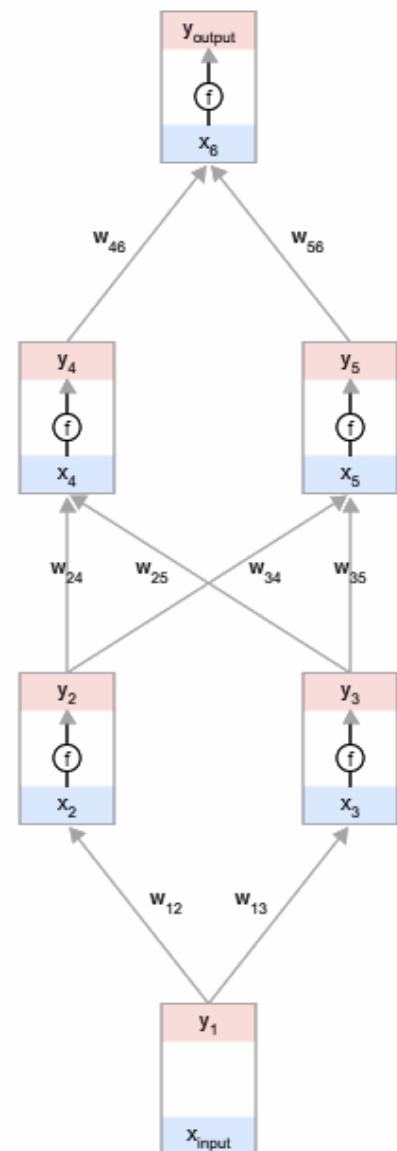
Neuronen in verknüpften Schichten sind über Kanten mit Gewichten w_{ij} verbunden.



Aktivierungsfunktion

Jedes Neuron hat einen Gesamteingabe x , eine Aktivierungsfunktion $f(x)$ und eine Ausgabe $y = f(x)$

$f(x)$ muss eine nichtlineare Funktion sein (z. B. tanh, ReLu, Sigmoid), sonst kann das neuronale Netzwerk nur lineare Modelle lernen.

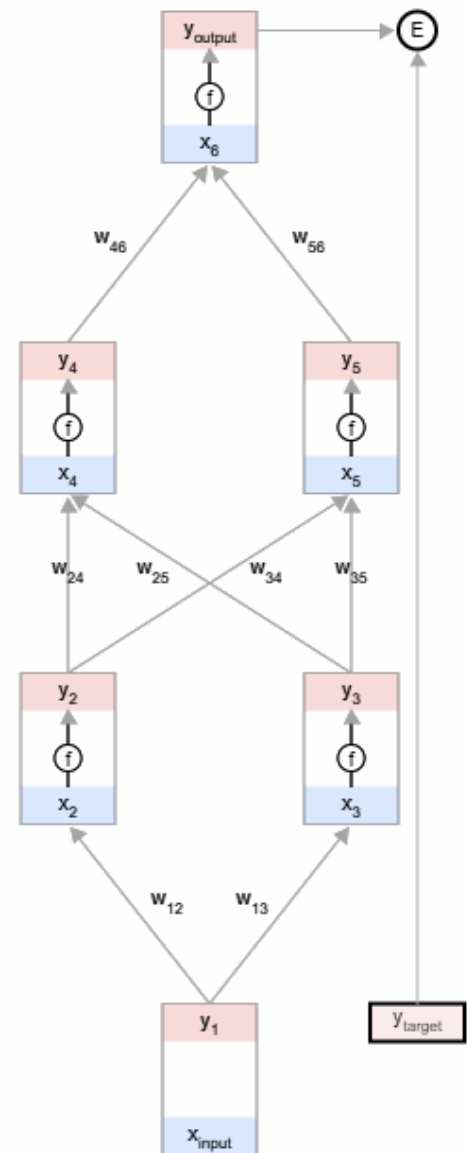


Fehler- / Verlustfunktion

Das Ziel ist es, die Gewichte des Netzwerks automatisch aus den Daten zu lernen, so dass für alle Eingänge x_{input} die vorhergesagte Ausgabe y_{output} nah am y_{target} ist.

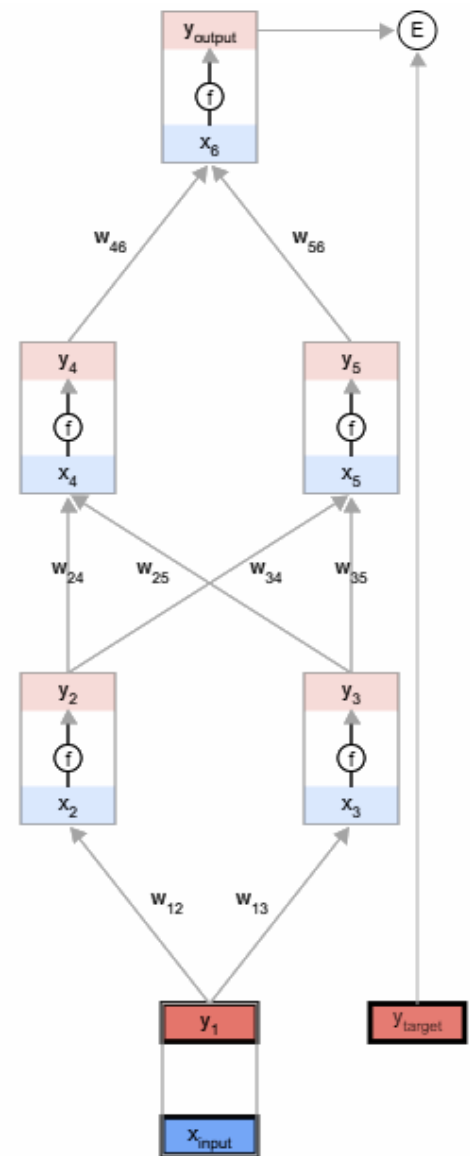
Eine häufig verwendete Fehlerfunktion ist der mittlere quadratische Fehler

$$E(y_{\text{output}}, y_{\text{target}}) = \frac{1}{2} (y_{\text{output}} - y_{\text{target}})^2$$



Vorwärtsausbreitung

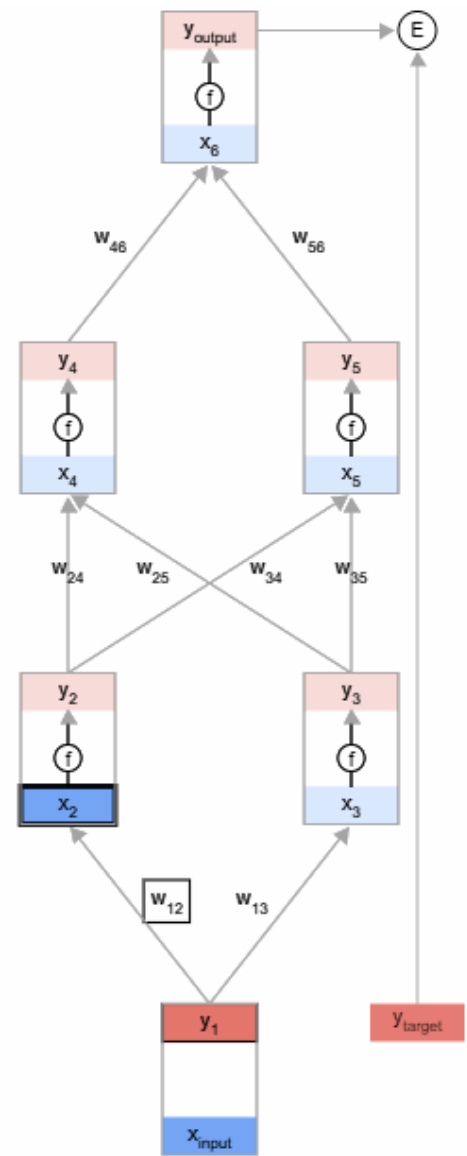
Das Netzwerk verwendet Eingabebeispiele (x_{input} , y_{target}), um die Neuronen zu aktualisieren.



Vorwärtsausbreitung

Um die verborgenen Schichten zu aktualisieren, verwendet das Netzwerk die Ausgaben **y** der vorherigen Schicht und berechnet mit den Gewichten die Eingabe **x** der Neuronen der nächsten Schicht.

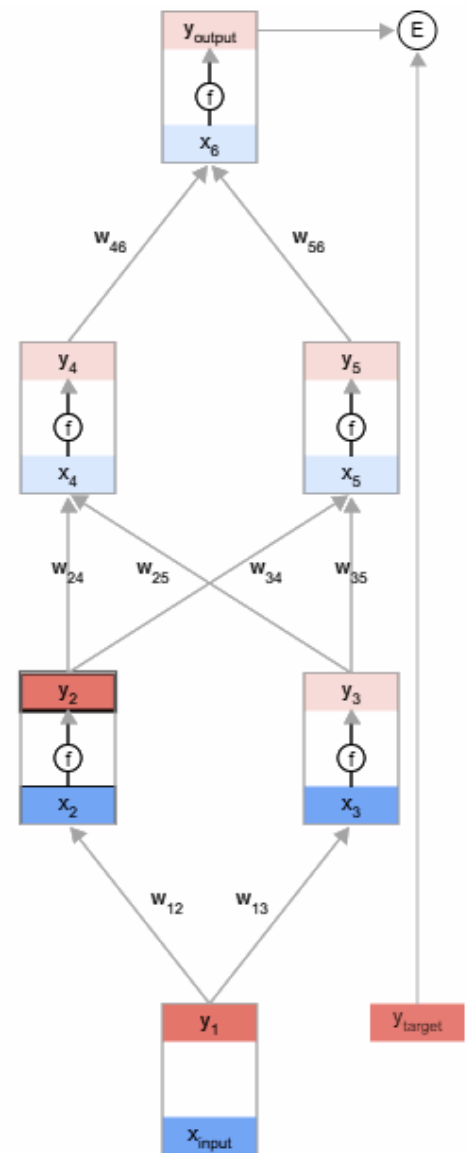
$$x_j = \sum_{i \in \text{in}(j)} w_{ij} y_i + b_j$$



Vorwärtsausbreitung

Dann aktualisieren wir die Ausgabe der Neuronen in den verborgenen Schichten. Hierfür verwenden wir die nichtlineare Aktivierungsfunktion, $f(x)$

$$y = f(x) = \tanh(x)$$

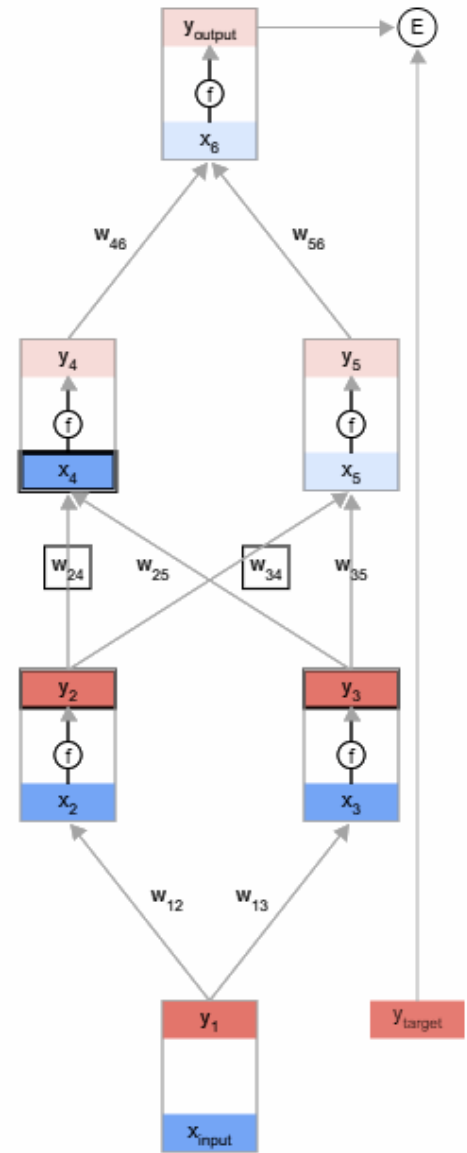


Vorwärtsausbreitung

Jedes Neuron im Netzwerk verbreitet die Eingabe im Rest des Netzwerks, um die Ausgabe des Netzwerks zu berechnen.

$$y = \tanh(x)$$

$$x_j = \sum_{i \in \text{in}(j)} w_{ij} y_i + b_j$$



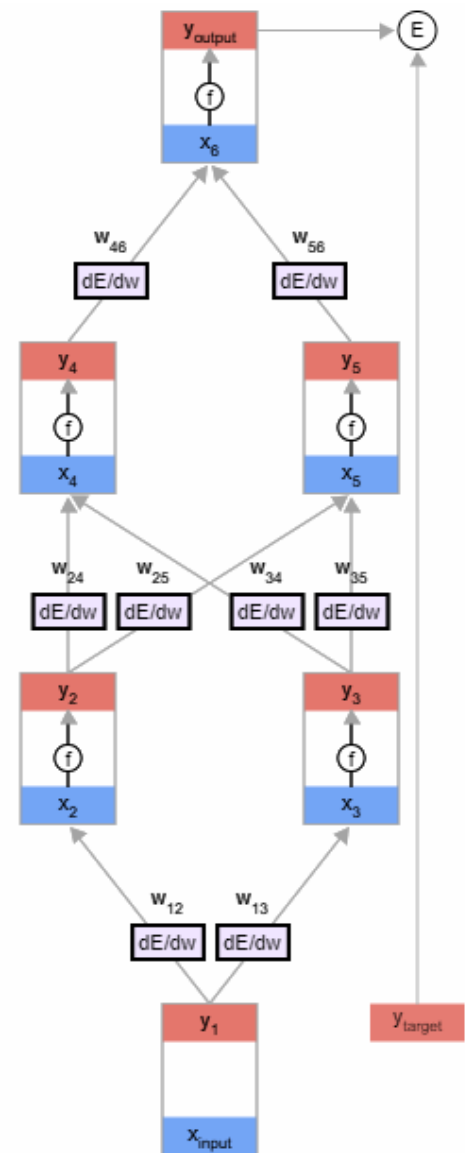
Gradientenabstieg

Der Backpropagation-Algorithmus entscheidet, wie oft jedes Gewicht des Netzwerks aktualisiert werden soll, nachdem die vorhergesagte Ausgabe mit der gewünschten Ausgabe für ein bestimmtes Beispiel verglichen wurde.

Dazu berechnet das Netzwerk, wie sich der Fehler E in Bezug auf jedes Gewicht ändert $\frac{dE}{dw_{ij}}$.

Sobald die Fehlerableitungen verfügbar sind, aktualisiert das Netzwerk die Gewichte mithilfe des Gradientenabstiegs:

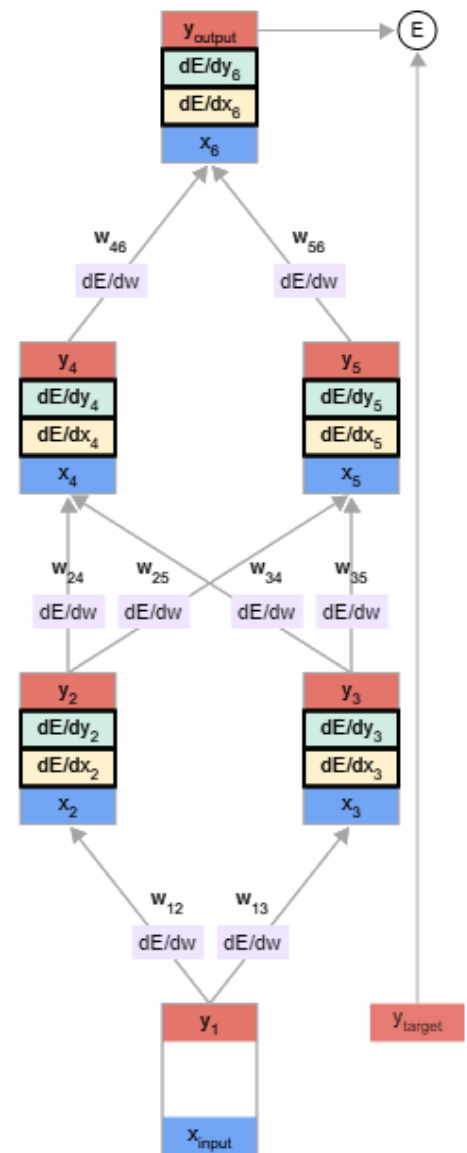
$$w_{ij} = w_{ij} - \alpha \frac{dE}{dw_{ij}}$$



Gradientenabstieg

Um $\frac{dE}{dw_{ij}}$ zu berechnen, müssen wir ermitteln, wie sich der Fehler in Abhängigkeit vom :

- Gesamteingang des Neurons $\frac{dE}{dx}$ und
- die Ausgabe des Neurons $\frac{dE}{dy}$ ändert.



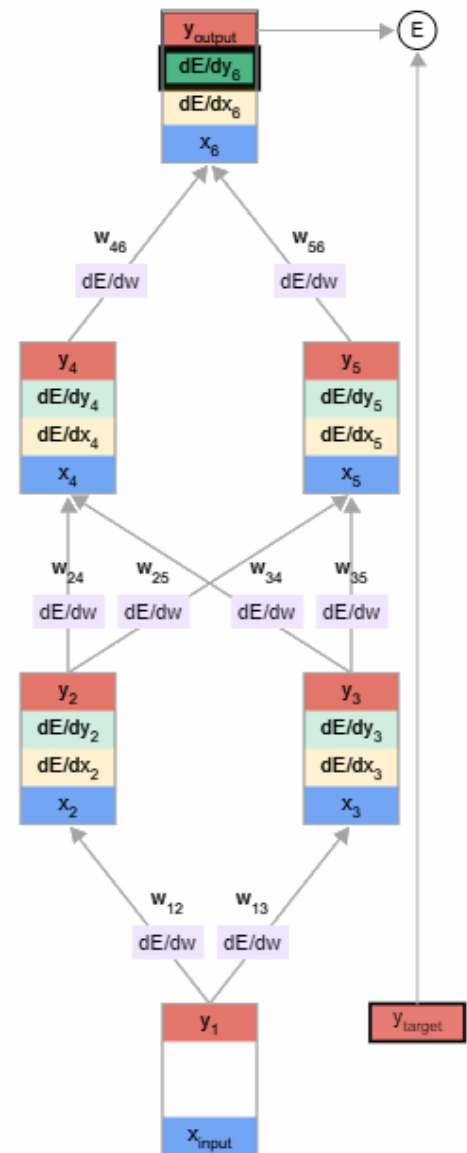
Backpropagation

Nach der Vorwärtsausbreitung der Eingabe kann das Netzwerk bereits die Ableitung in Bezug auf die Ausgabe berechnen.

Für die Fehlerfunktion

$E = \frac{1}{2} (y_{\text{output}} - y_{\text{target}})^2$ ist die Ableitung

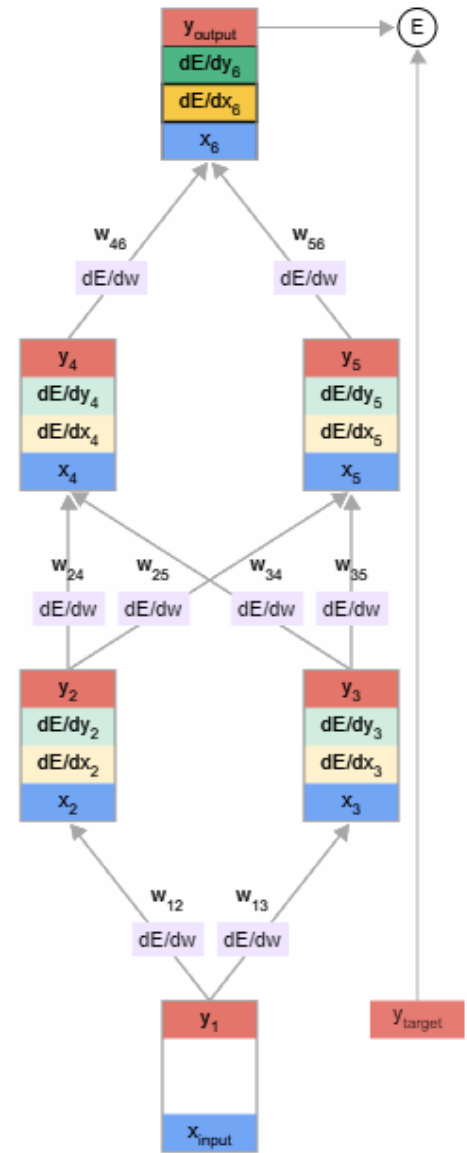
$$\frac{\partial E}{\partial y_{\text{output}}} = y_{\text{output}} - y_{\text{target}}$$



Backpropagation

Jetzt dass $\frac{dE}{dy}$ verfügbar ist, kann $\frac{dE}{dx}$ mit der Kettenregel aus der vorigen Ebene errechnet werden

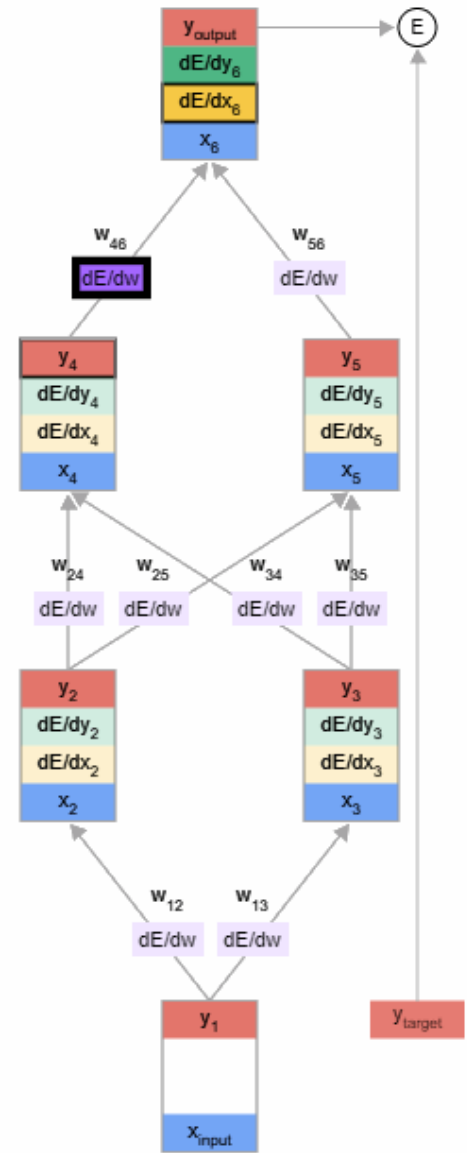
$$\frac{\partial E}{\partial x} = \frac{dy}{dx} \frac{\partial E}{\partial y} = \frac{d}{dx} f(x) \frac{\partial E}{\partial y}$$



Backpropagation

Sobald wir die Fehlerableitung in Bezug auf die Gesamteingabe eines Neurons haben, können wir die Fehlerableitung in Bezug auf die Gewichte erhalten, die in dieses Neuron kommen.

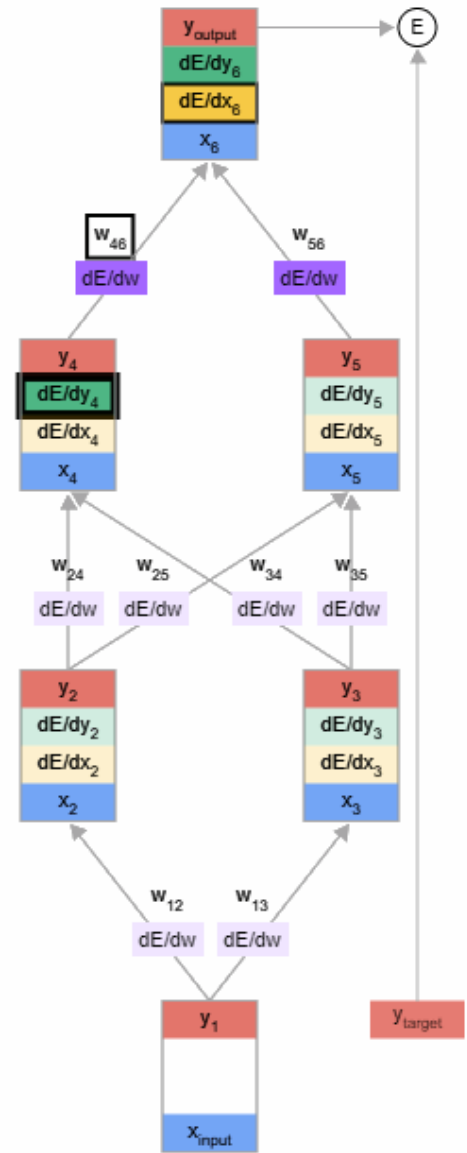
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}} \frac{\partial E}{\partial x_j} = y_i \frac{\partial E}{\partial x_j}$$



Backpropagation

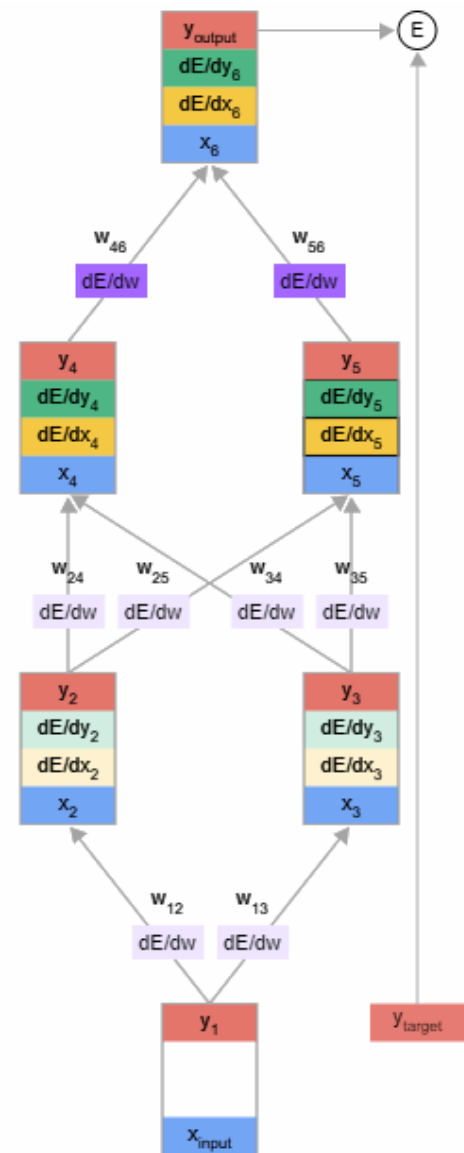
Ebenso kann $\frac{dE}{dy}$ mit der Kettenregel aus der vorigen Ebene errechnet werden:

$$\frac{\partial E}{\partial y_i} = \sum_{j \in \text{out}(i)} \frac{\partial x_j}{\partial y_i} \frac{\partial E}{\partial x_j} = \sum_{j \in \text{out}(i)} w_{ij} \frac{\partial E}{\partial x_j}$$



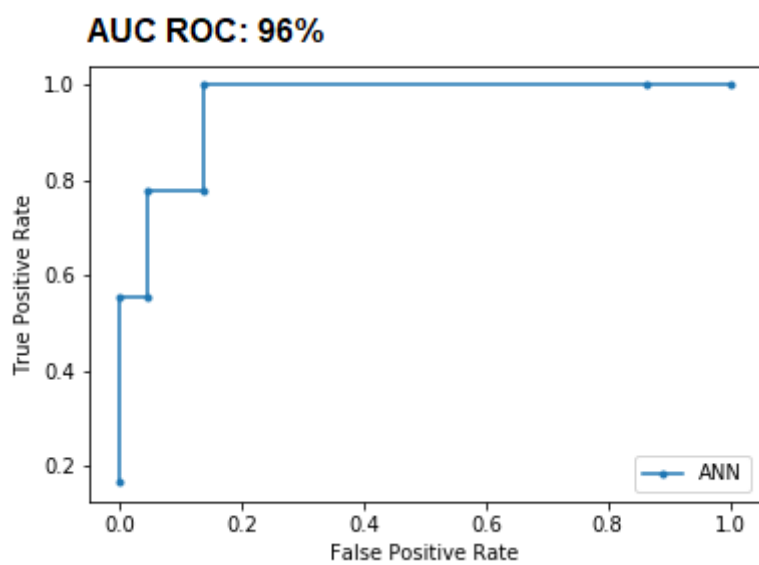
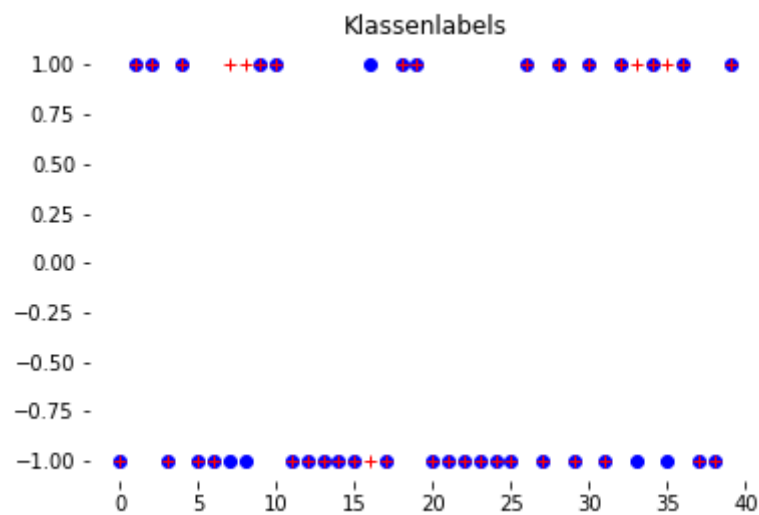
Backpropagation

Das Netzwerk wiederholt die Schritte für jedes Eingabebeispiel für eine bestimmte Anzahl von Epochen.

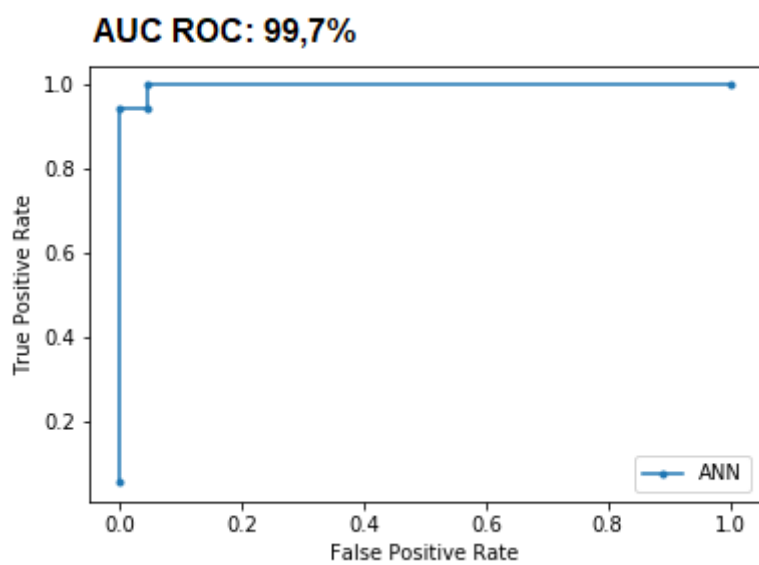
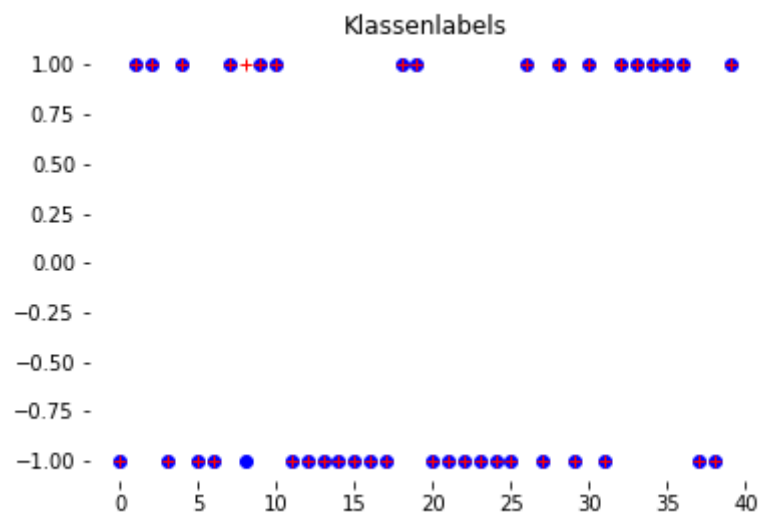


Ausführungsausgabe

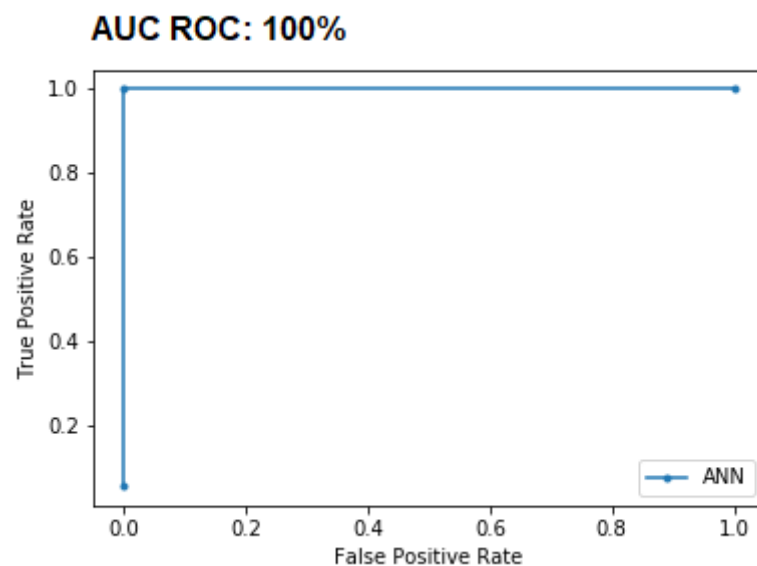
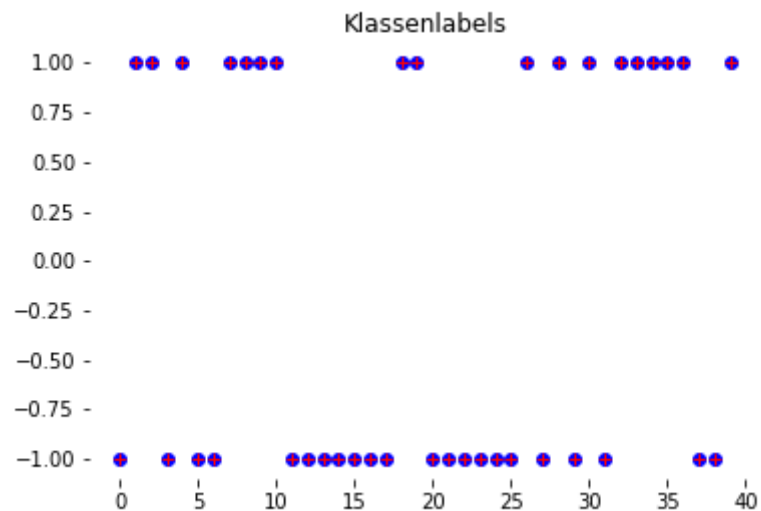
Erstaufführung



Zweite Aufführung

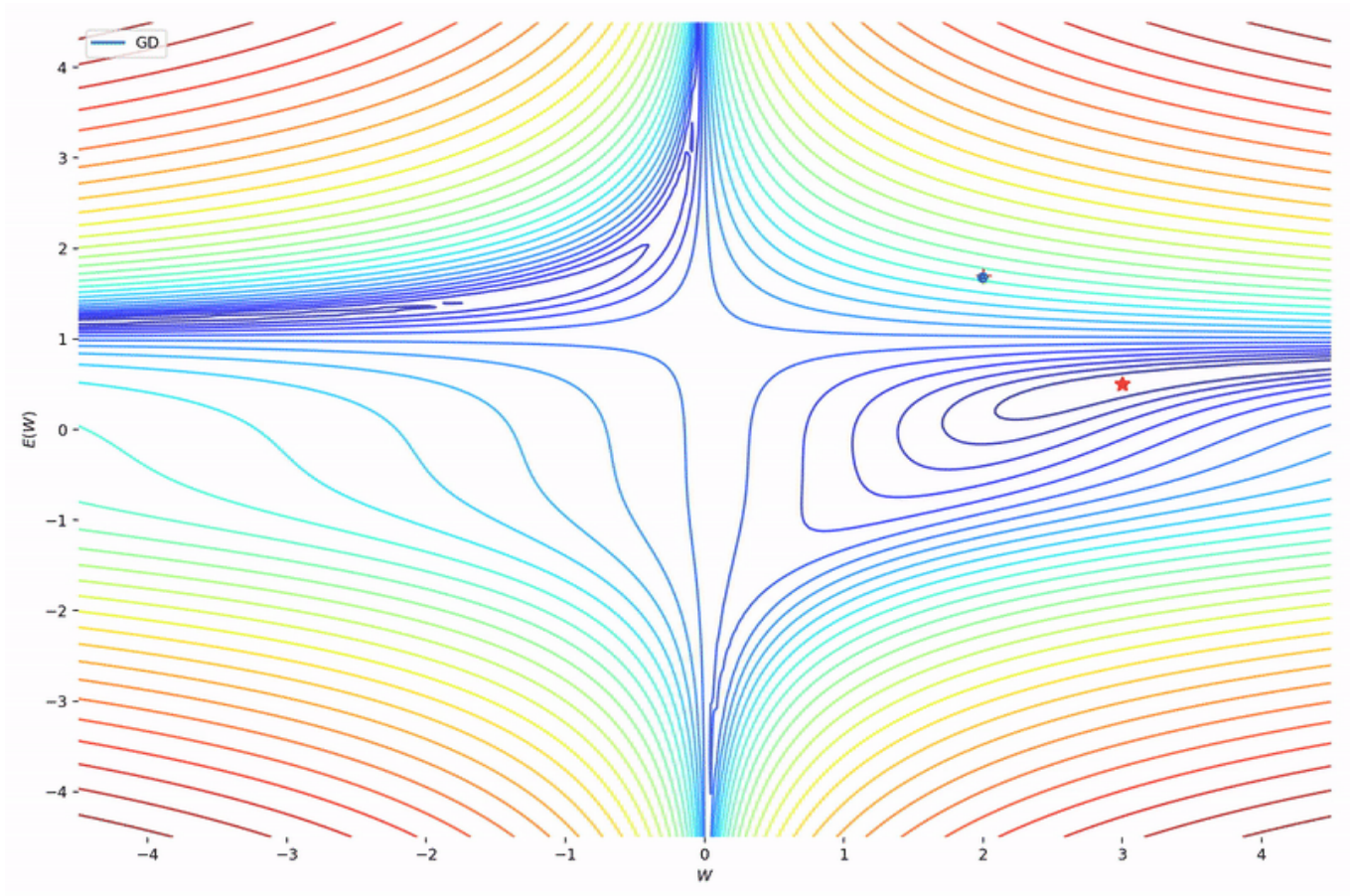


Dritte Aufführung



Nachteile des Gradientenabstiegs

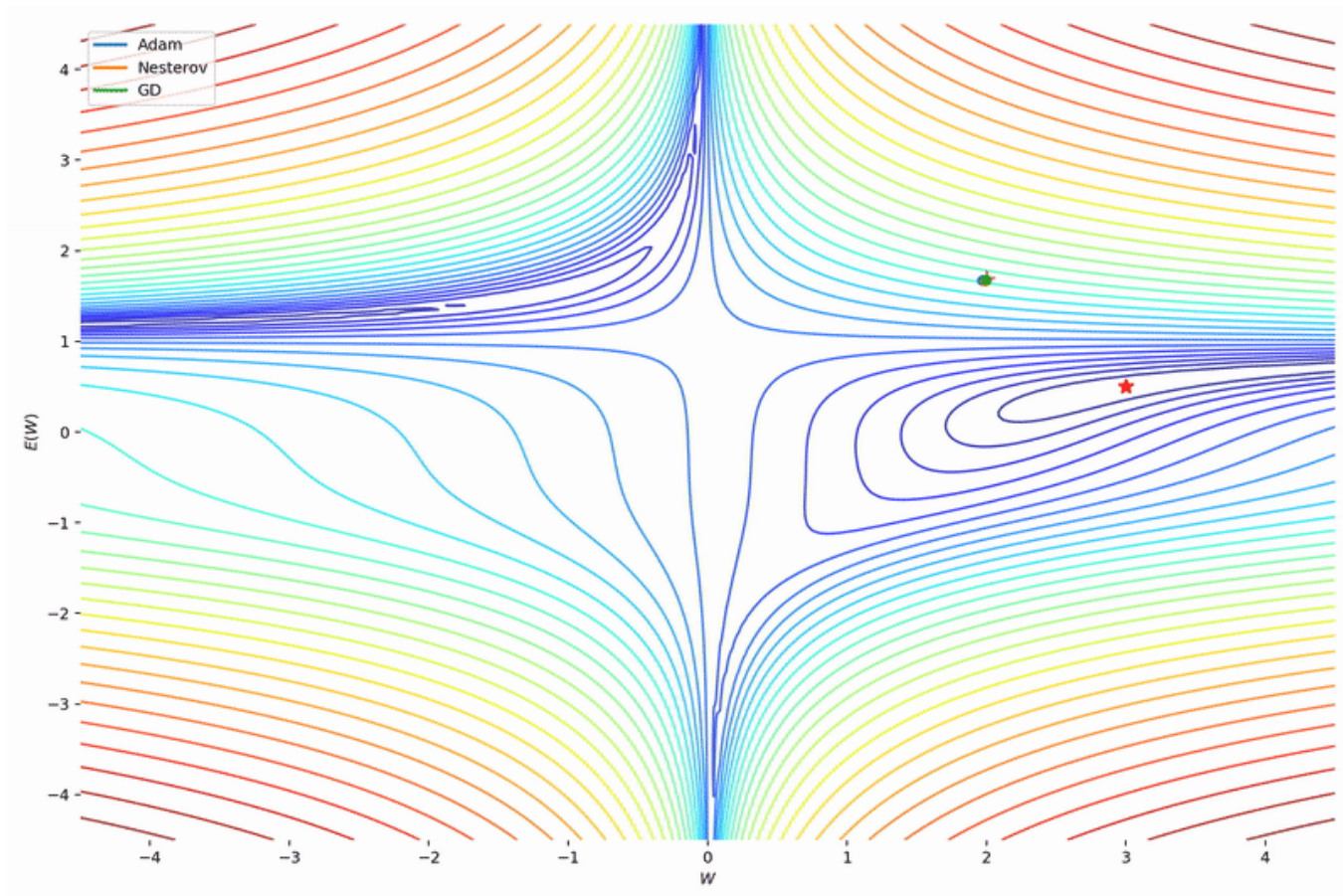
Abhängig von der Datenmenge machen wir einen **Kompromiss zwischen die Genauigkeit der Parameteraktualisierung und die Konvergenzgeschwindigkeit** [2, 4].

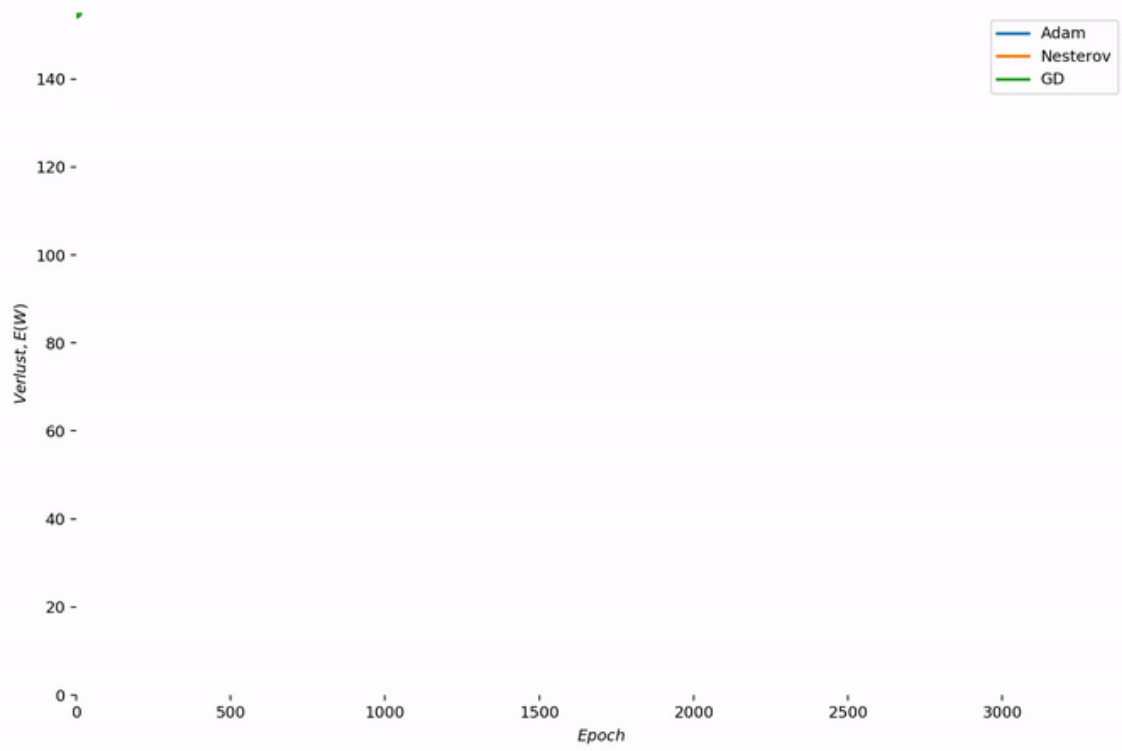


Einschränkungen der Gradientenabstieg

Es wurden **verschiedene Methoden entwickelt**, um diesen Einschränkungen auf unterschiedliche Weise **entgegenzuwirken** [3].

Gradient Descent	Nesterov Momentum	Adaptive Moment Estimation (Adam)
$w_{t+1} = w_t - \alpha \nabla E(w_t)$	$w_{t+1} = w_t - \alpha v_t$ $v_{t+1} = \frac{\gamma}{\alpha} v_t + \nabla E(w_t - \gamma v_t)$	$w_{t+1} = w_t - \alpha \frac{m_t}{\sqrt{v_t}}$ $m_{t+1} = f(\nabla E(w_t))$ $v_{t+1} = f(\nabla E(w_t)^2)$
<p>+ einfache Aktualisierungsregel (z. B. Summe, Produkt)</p> <p>— Konvergiert zum globalen Minimum für konvexe Fehleroberflächen und zu einem lokalen Minimum für nichtkonvexe Oberflächen</p>	<p>+ Beschleunigt in die jeweilige Steigungsrichtung und dämpft Schwingungen</p> <p>— funktioniert nicht gut, wenn die Kostenfunktionen stark konvex sind</p>	<p>+ speichert einen exponentiell abfallenden Durchschnitt vergangener Gradienten (glättender Momentum) mit adaptive Lernrate für jede parameter</p> <p>— hat den niedrigsten Trainingsfehler, aber nicht den niedrigsten Validierungsfehler, und der Validierungsfehler ist größer als der Trainingsfehler (d. h. leichte Überanpassung)</p>





Fazit

- **Gradientenabstieg** ist die **typische Optimierungsmethode** in neuronalen Netzen.
- **Lernen** in Neuronale Netzwerk ist ein **iterativer Prozess**.
- Bei der **Backpropagation** wird ein Gradientenabstieg verwendet, um zu einer **Lösung zu konvergieren** (d. h. die **Gewichte zu finden**), die die **Fehlerfunktion minimiert**.
- **Gradientenabstieg** ist jedoch **problematisch** (d. h. Konvergenz, Präzision), und dennoch wurden in der Praxis viele **verbesserte Verfahren** entwickelt.
- Das **Verständnis des Gradientenabstieg** bei der Diagnose der Backpropagation **ist für erfolgreiche Anwendungen erforderlich**.

Literaturverzeichnis

[1] <https://google-developers.appspot.com/machine-learning/> (<https://google-developers.appspot.com/machine-learning/>) (letzter Besuch, Dez 2019)

[2] Boyd, S., & Vandenberghe, L. (2004). Convex optimization. Cambridge university press.

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

[4] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Vorlesung Notebook herunterladen

