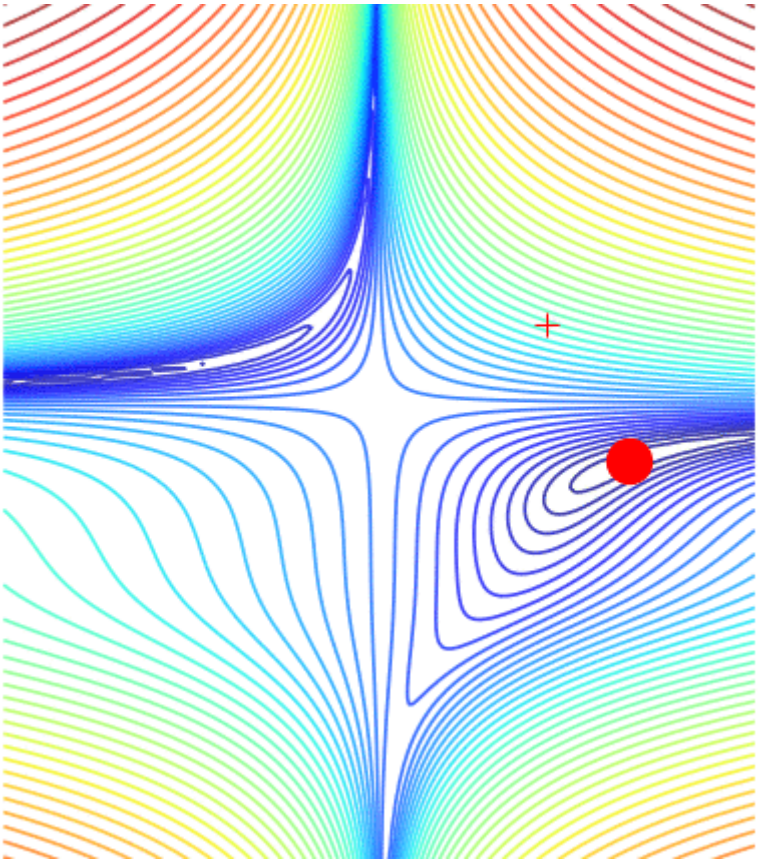
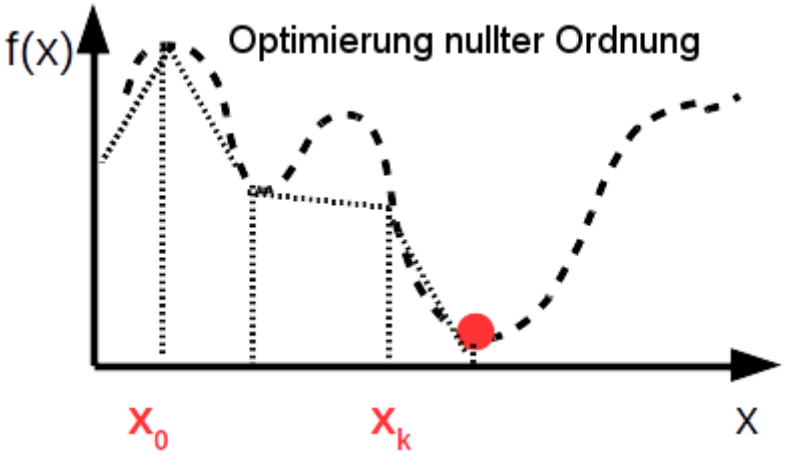
Einsatz von Gradientenabstiegsverfahren in Neuronalen Netzen

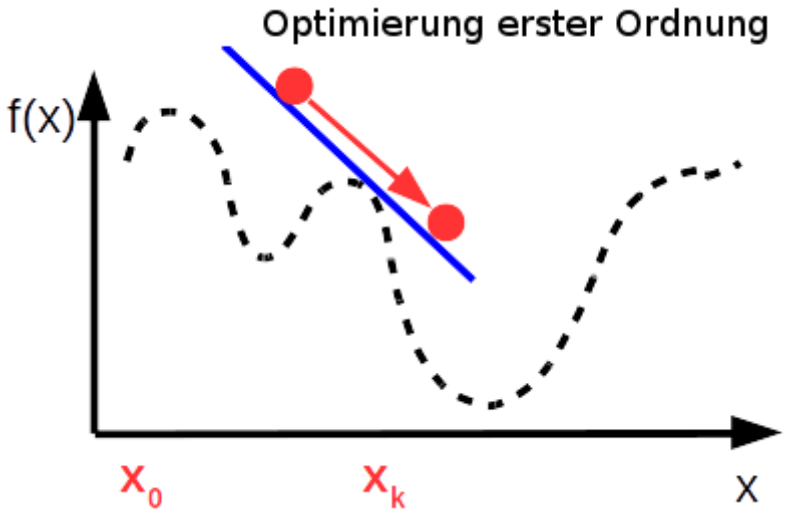
Inhalt

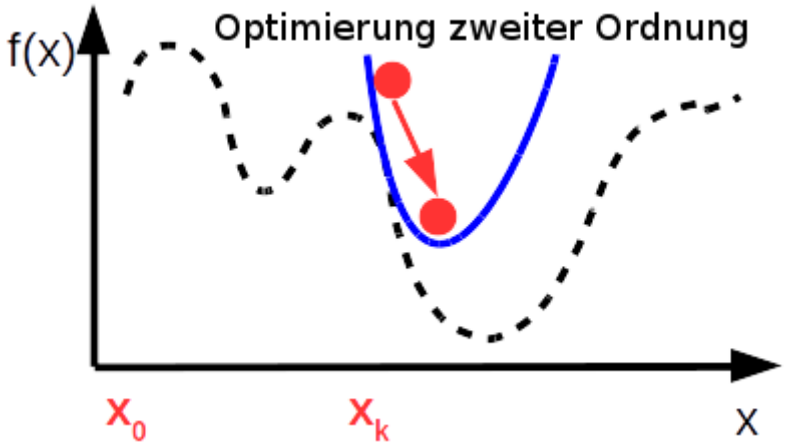
Optimierungsgrundlagen

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 (•) **konvergieren** [2].

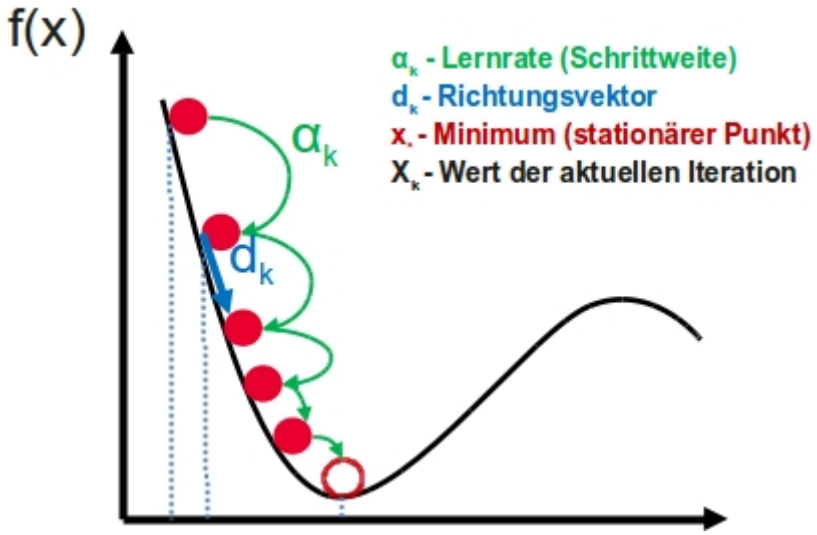








Gradientenabstieg

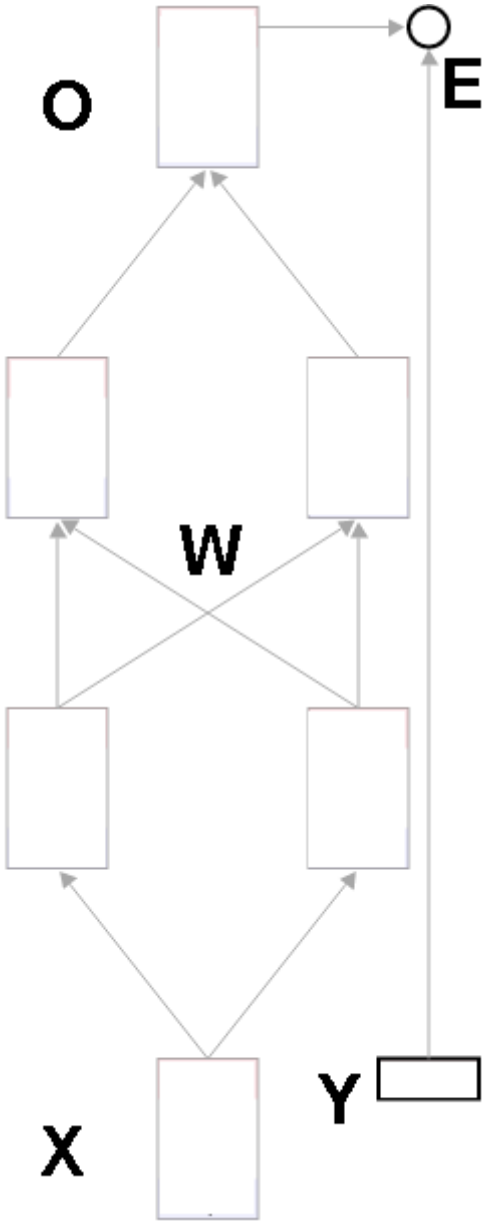


Lernen als Optimierung in Neuronale Netzwerk

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].

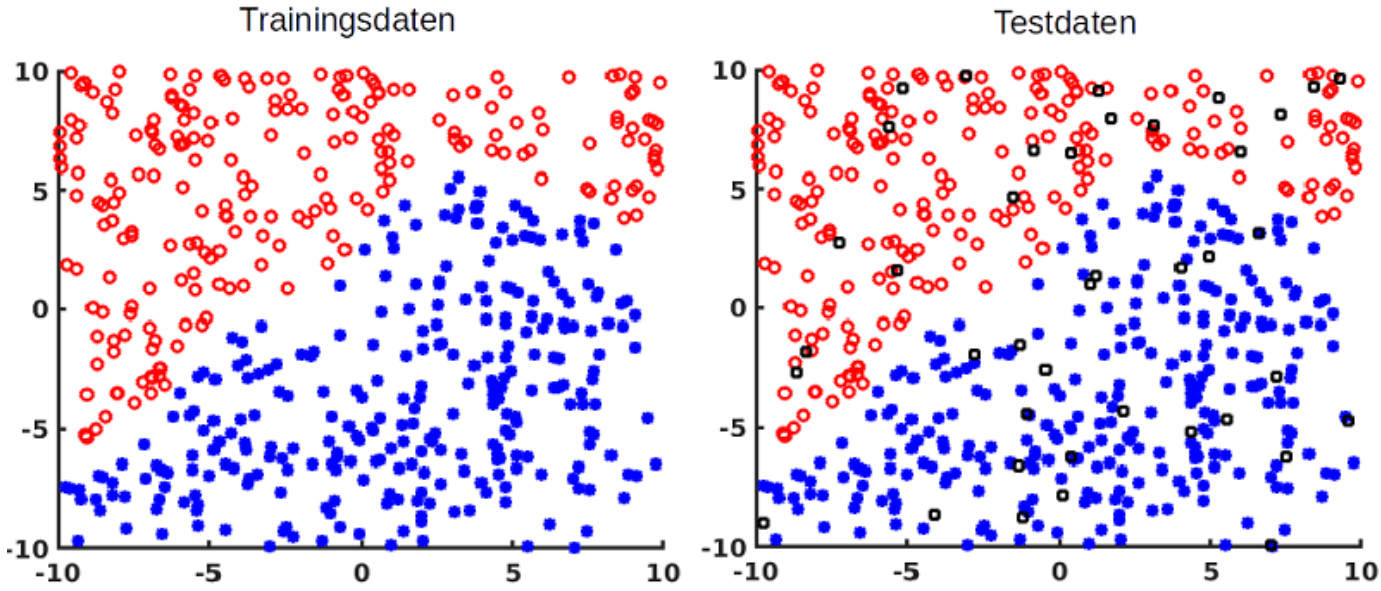
Diese **Parameter** werden bei der Berechnung der **erwarteten Werte (Y)** aus dem Satz von **Prädiktoren (X)** .

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



Gradientenabstieg in Neuronale Netzwerk

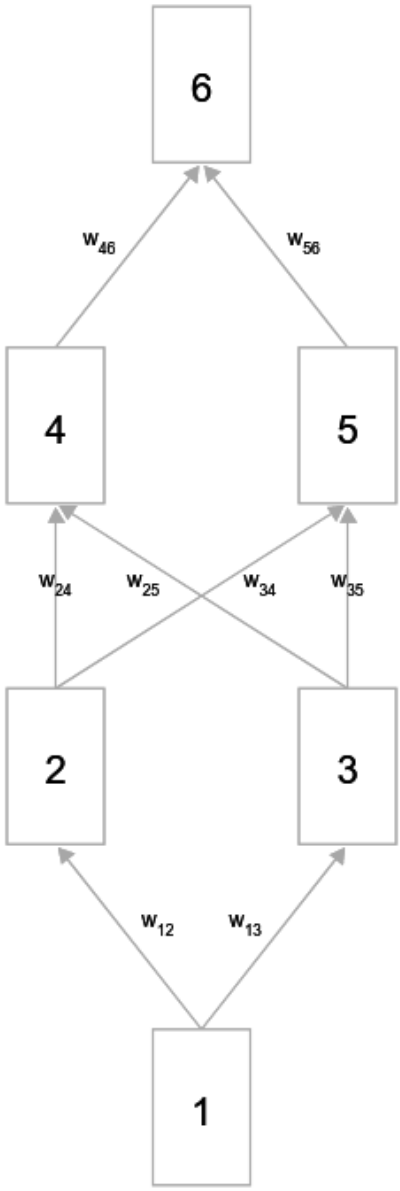
Binäre Klassifikation mit neuronalen Netzen



Einfaches neuronales Netz

Rechts sehen Sie das neuronale Netzwerk für die Klassifizierungsaufgabe mit einem Eingabe- (2D), einem Ausgabe-Neuron und zwei verborgenen Schichten mit jeweils zwei Neuronen.

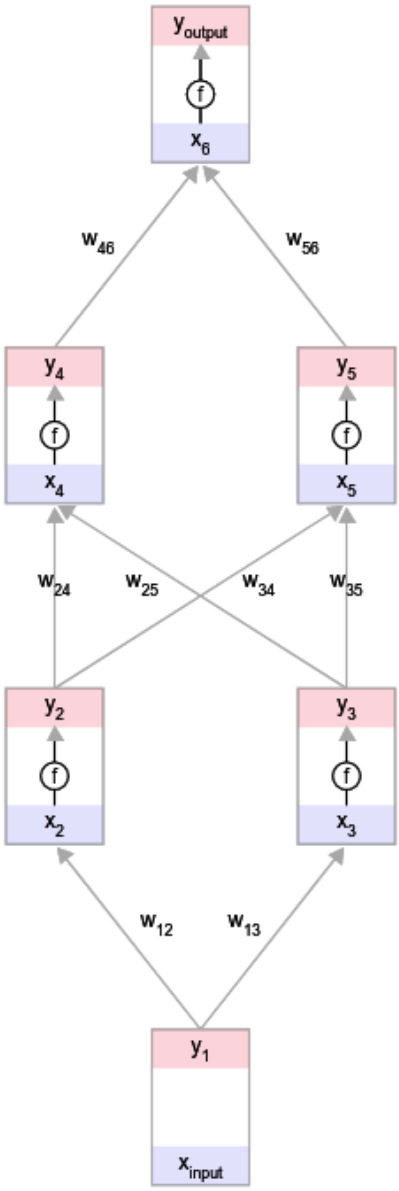
Neuronen in benachbarten Schichten sind mit Gewichten verbunden w_{ij} (d.h. welche sind die Netzwerkparameter).



Aktivierungsfunktion

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

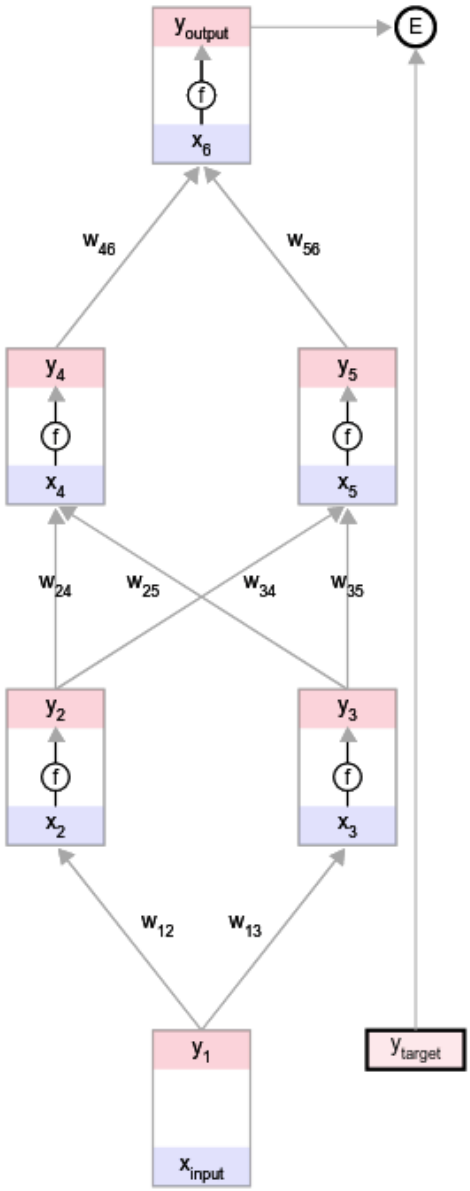
$f(x)$ muss eine nichtlineare Funktion sein (z. B. ReLu, tanh, sigmoid), sonst kann das neuronale Netzwerk nur lineare Modelle lernen.



Fehler- (Verlust-) Funktion

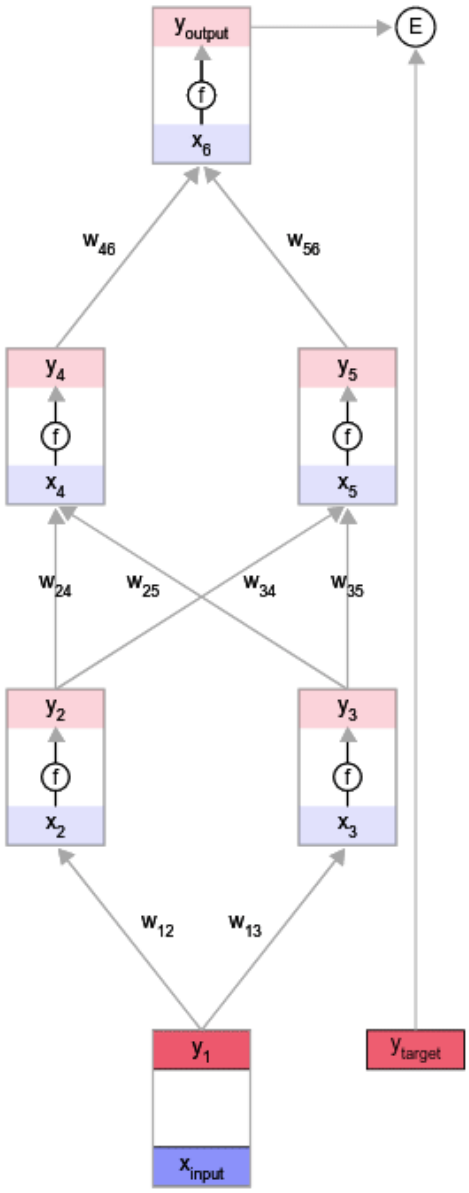
Das Ziel ist es, die Gewichte des Netzwerks automatisch aus Daten zu lernen, die die vorhergesagte Ausgabe ergeben y_{output} ist nah am Ziel y_{target} für alle Eingänge x_{input} .

Wir verwenden als Fehler eine typische Wahl, den mittleren quadratischen Fehler:
$E(y_{output}, y_{target}) = \frac{1}{2}(y_{output} - y_{target})^2$.



Vorwärtsausbreitung

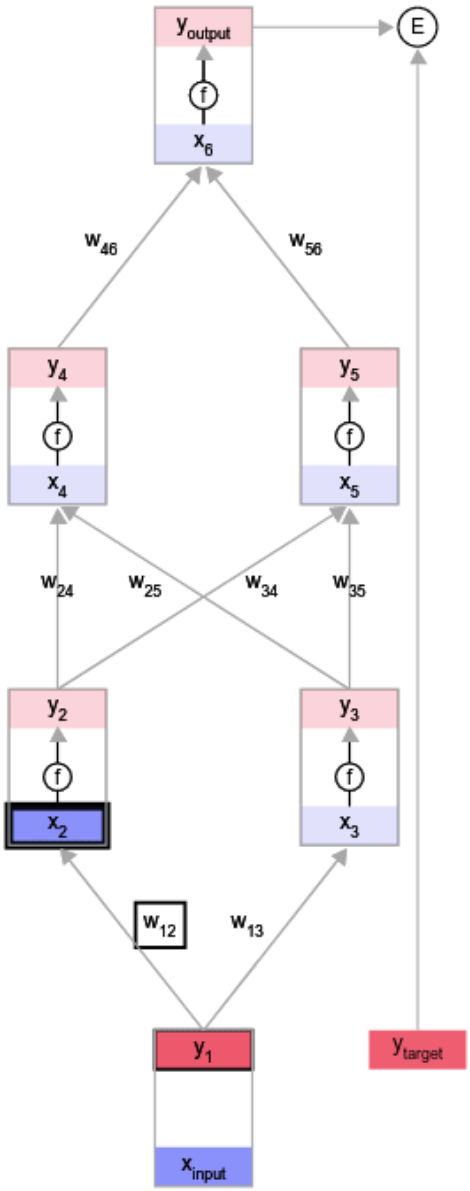
Das Netzwerk verwendet Eingabebeispiele (x_{input}, y_{target}) die Eingangsneuronen zu aktualisieren.



Vorwärtsausbreitung

Um die verborgenen Schichten zu aktualisieren, das Netzwerk verwendet die Ausgabe y der Neuronen in der vorherigen Schicht und verwenden Sie die Gewichte, um die Eingabe zu berechnen x der Neuronen in der nächsten Schicht.

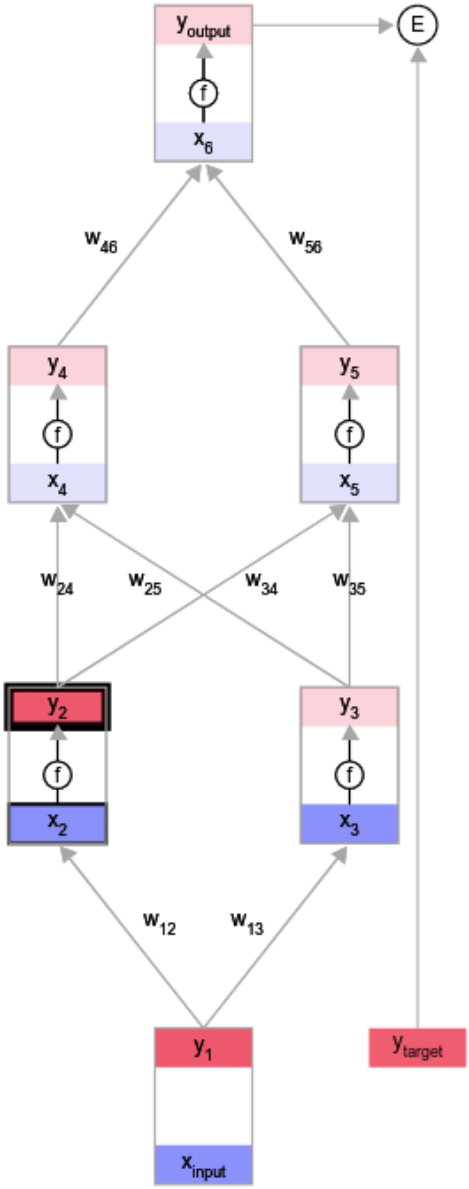
$$x_j = \sum_{i \in in(j)} w_{ij} y_i + b_j$$



Vorwärtsausbreitung

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

$$y = tanh(x)$$

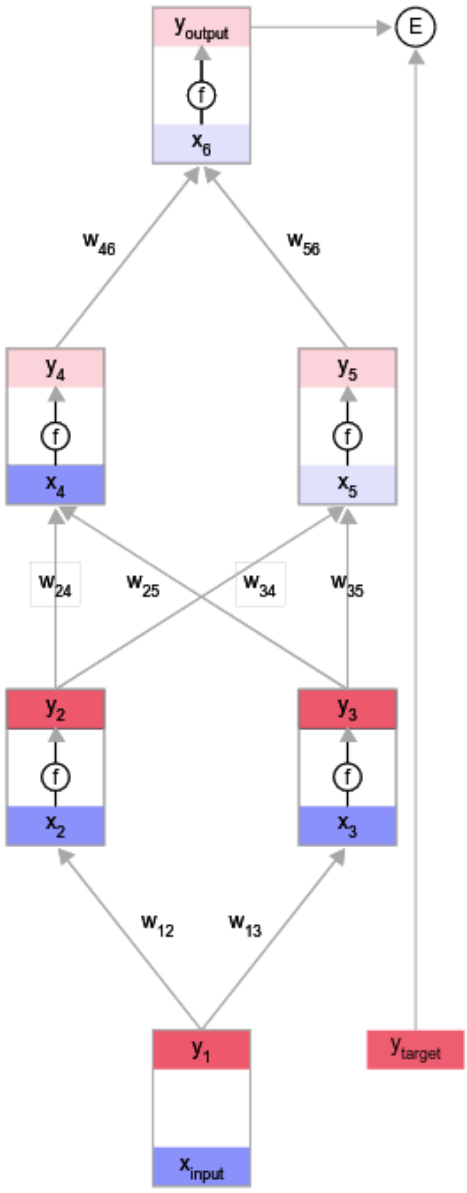


Vorwärtsausbreitung

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

$$y = tanh(x)$$

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

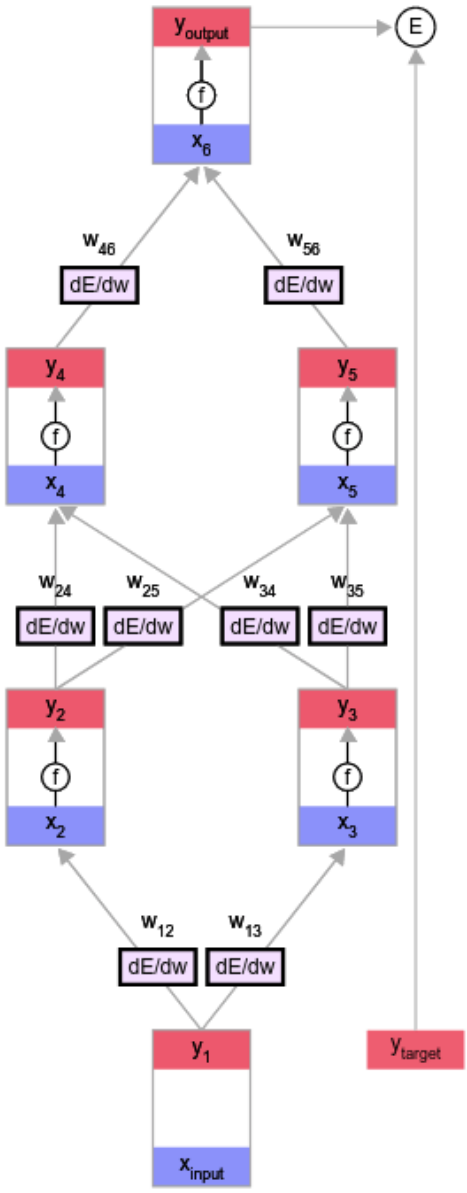


Gradientenabstieg

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.
Das Netzwerk muss berechnen, wie sich der Fehler in Bezug auf jedes Gewicht ändert $\frac{dE}{dw_{ij}}$.

Wenn die Fehlerableitungen verfügbar sind, aktualisiert das Netzwerk die Gewichte mithilfe des Gradientenabstieg:

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

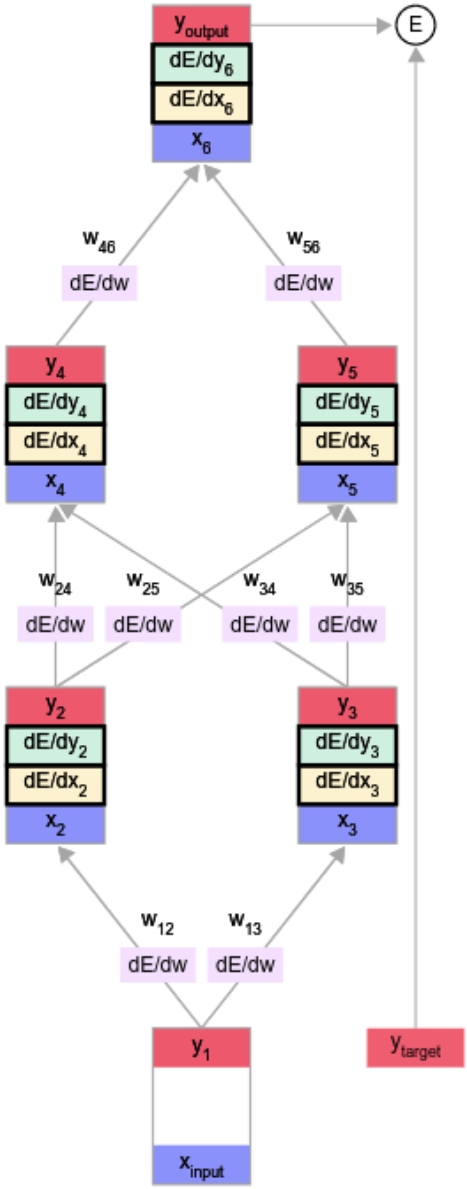


Gradientenabstieg

Um $\frac{dE}{dw_{ij}}$ zu berechnen Das Netzwerk muss berechnen, wie sich der Fehler in Bezug auf :

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

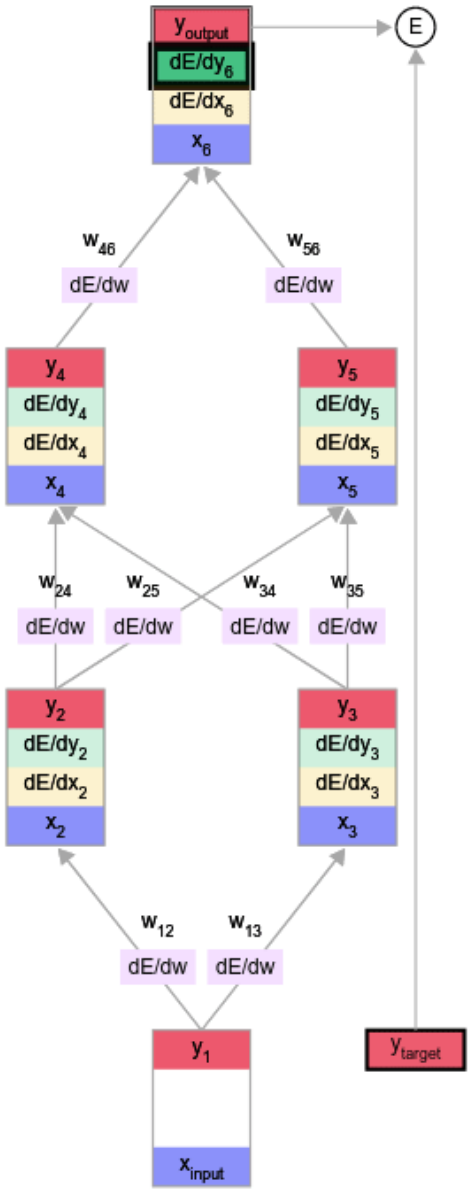
ändert.



Backpropagation

Nach Vorwärtsausbreitung der Eingabe kann man bereits die Ableitung des Fehlers in Bezug auf die Ausgabe berechnen. Angesichts der Fehlerfunktion $E = \frac{1}{2}(y_{output} - y_{target})^2$ das Derivat ist:

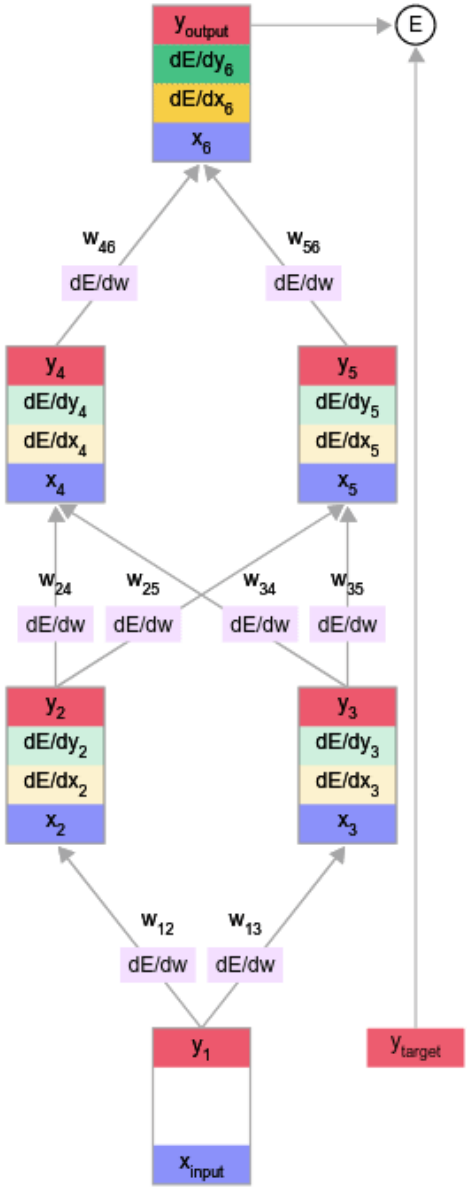
$$\frac{\partial E}{\partial y_{output}} = y_{output} - y_{target}$$



Backpropagation

Jetzt haben wir $\frac{dE}{dy}$ zur Verfügung und wir können $\frac{dE}{dx}$ mit der Kettenregel berechnen

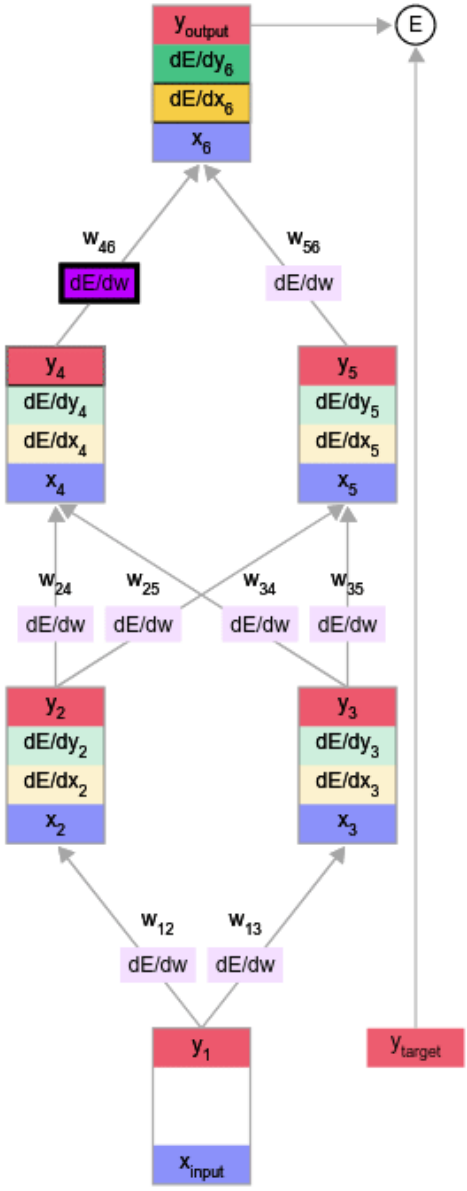
$$\frac{\partial E}{\partial x} = \frac{dy}{dx}\frac{\partial E}{\partial y} = \frac{d}{dx}f(x)\frac{\partial E}{\partial y}$$



Backpropagation

Sobald das Netzwerk die Fehlerableitung in Bezug auf die Gesamteingabe eines Neurons hat, kann es die Fehlerableitung in Bezug auf die Gewichte berechnen, die in dieses Neuron kommen:

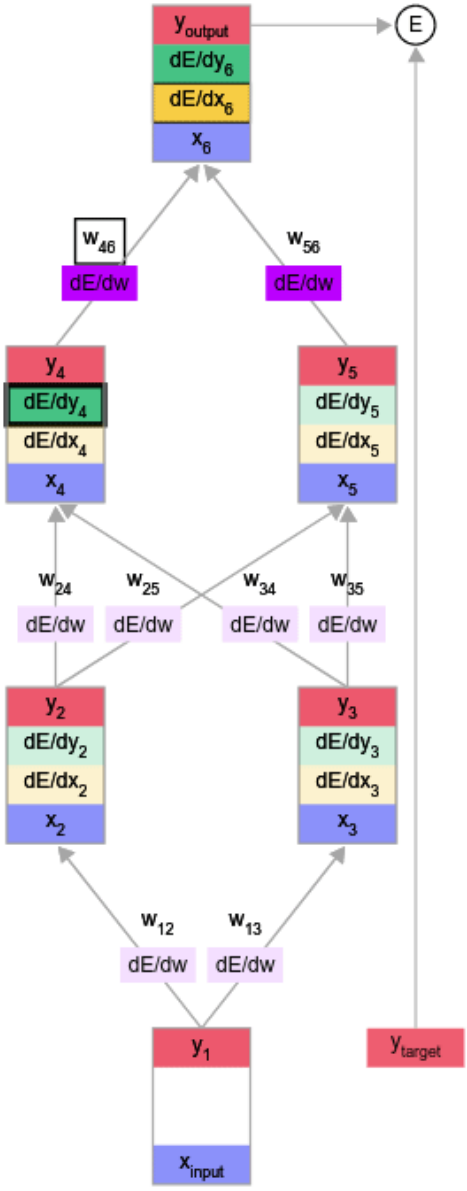
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}} \frac{\partial E}{\partial x_j} = y_i \frac{\partial E}{\partial x_j}$$



Backpropagation

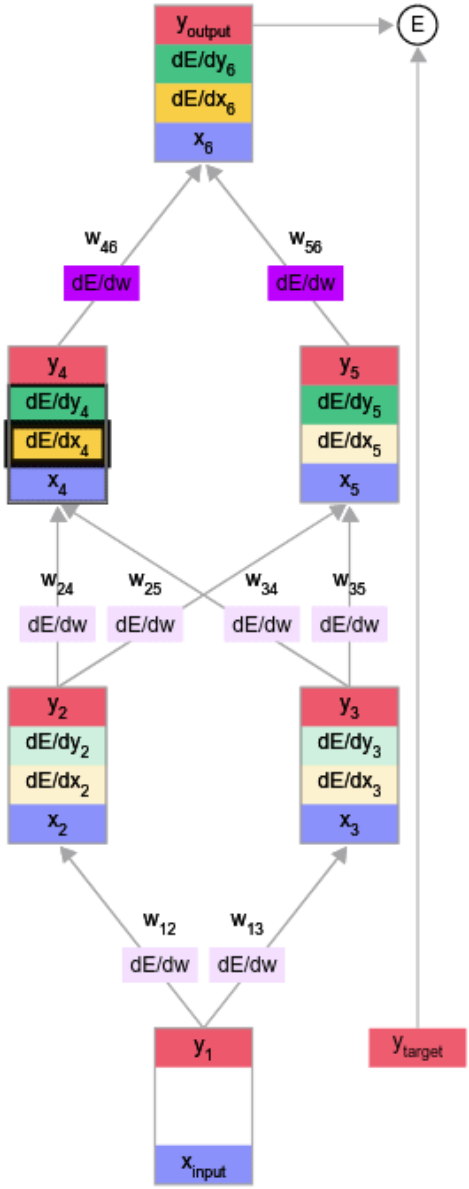
Ebenso kann mit der Kettenregel $\frac{dE}{dy}$ aus der vorherigen Ebene gerechnet werden:

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



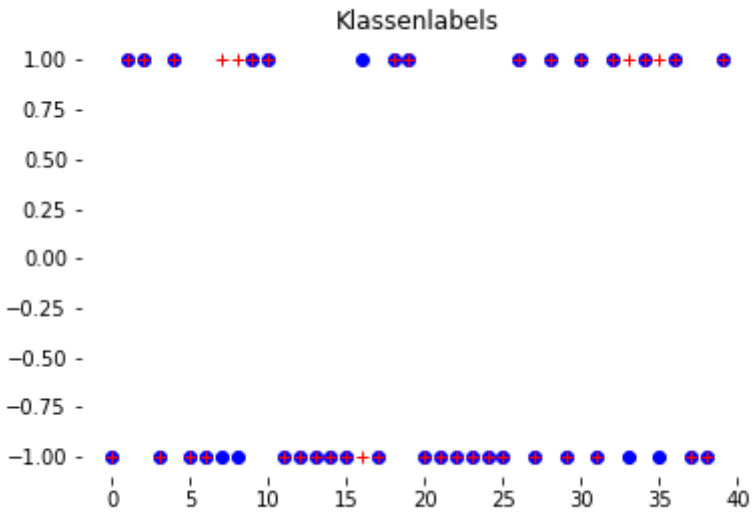
Backpropagation

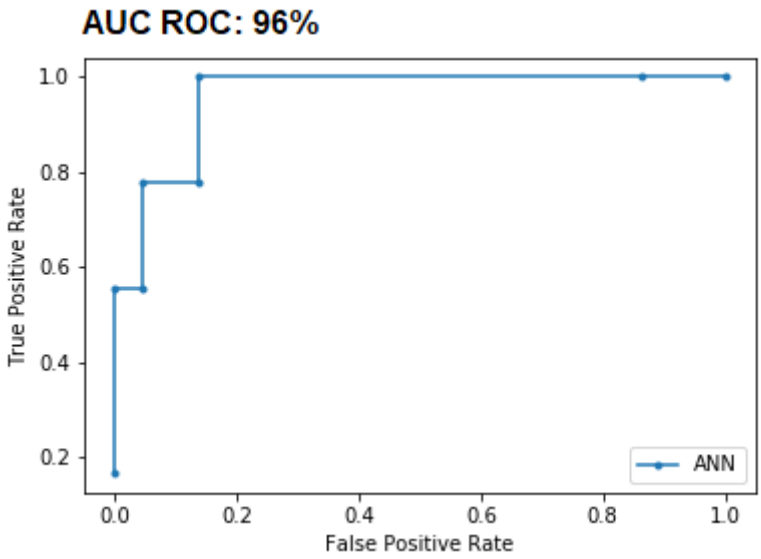
Die Schritte werden dann für jedes der Trainingseingabebeispiele mehrere Male (d. H. Epochen) wiederholt.



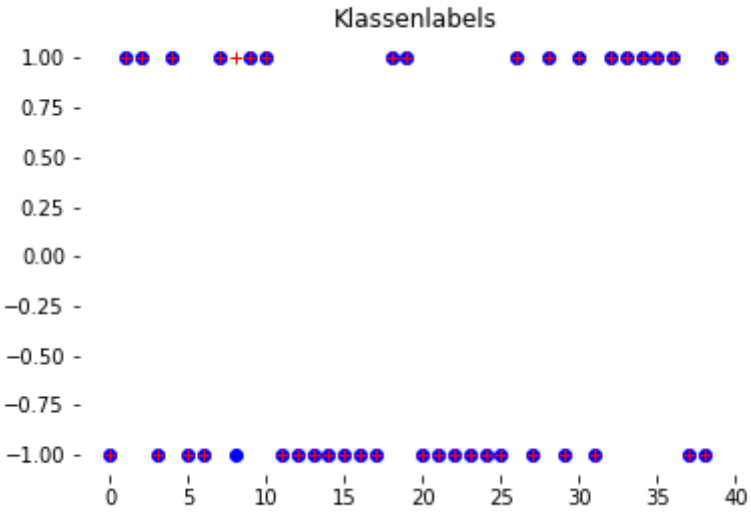
Ausführungsausgabe

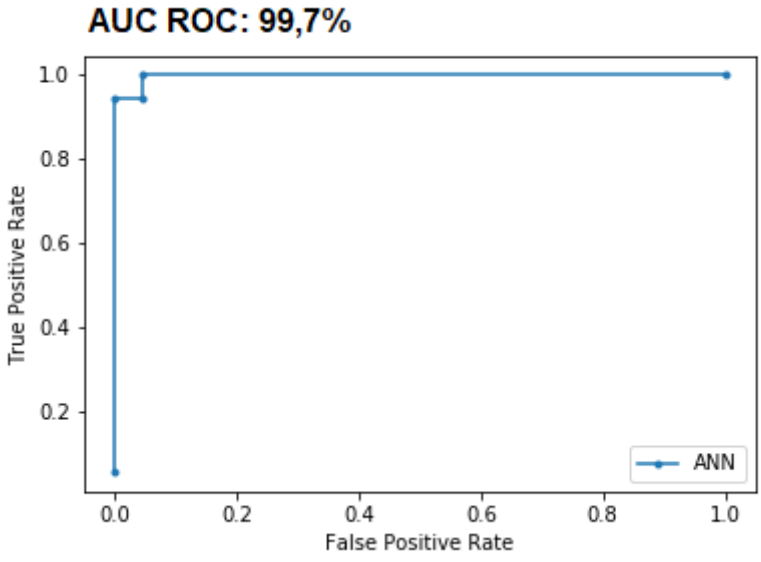
Erstaufführung



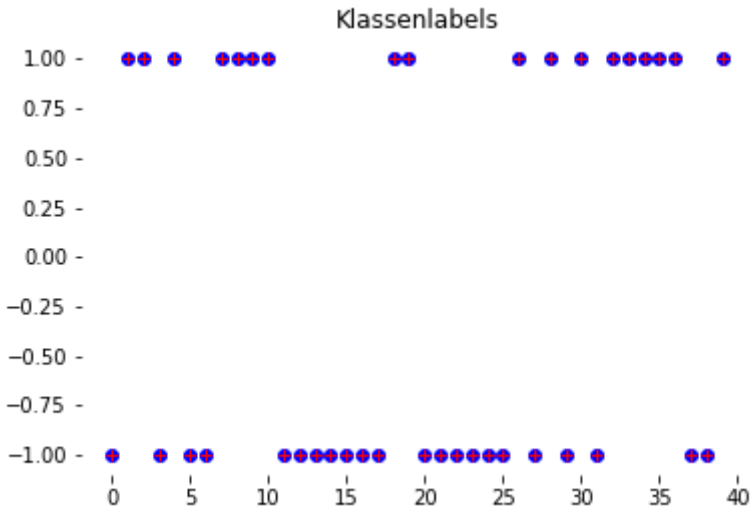


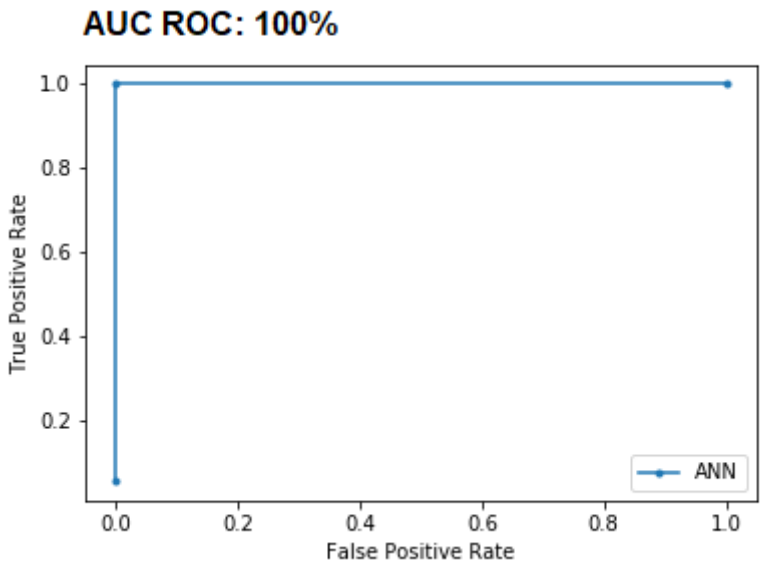
Zweite Aufführung





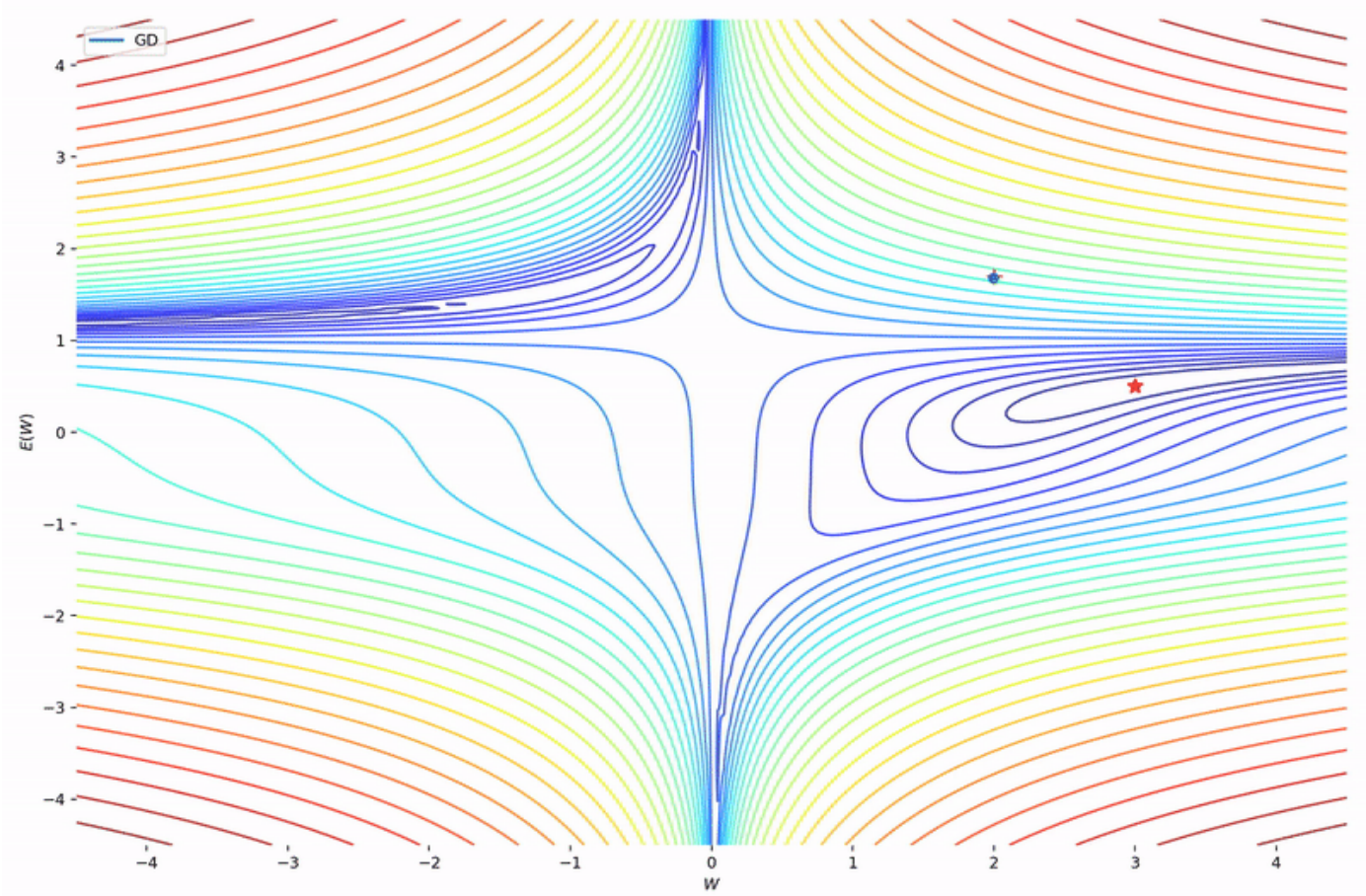
Dritte Aufführung





Einschränkungen der Gradientenabstieg

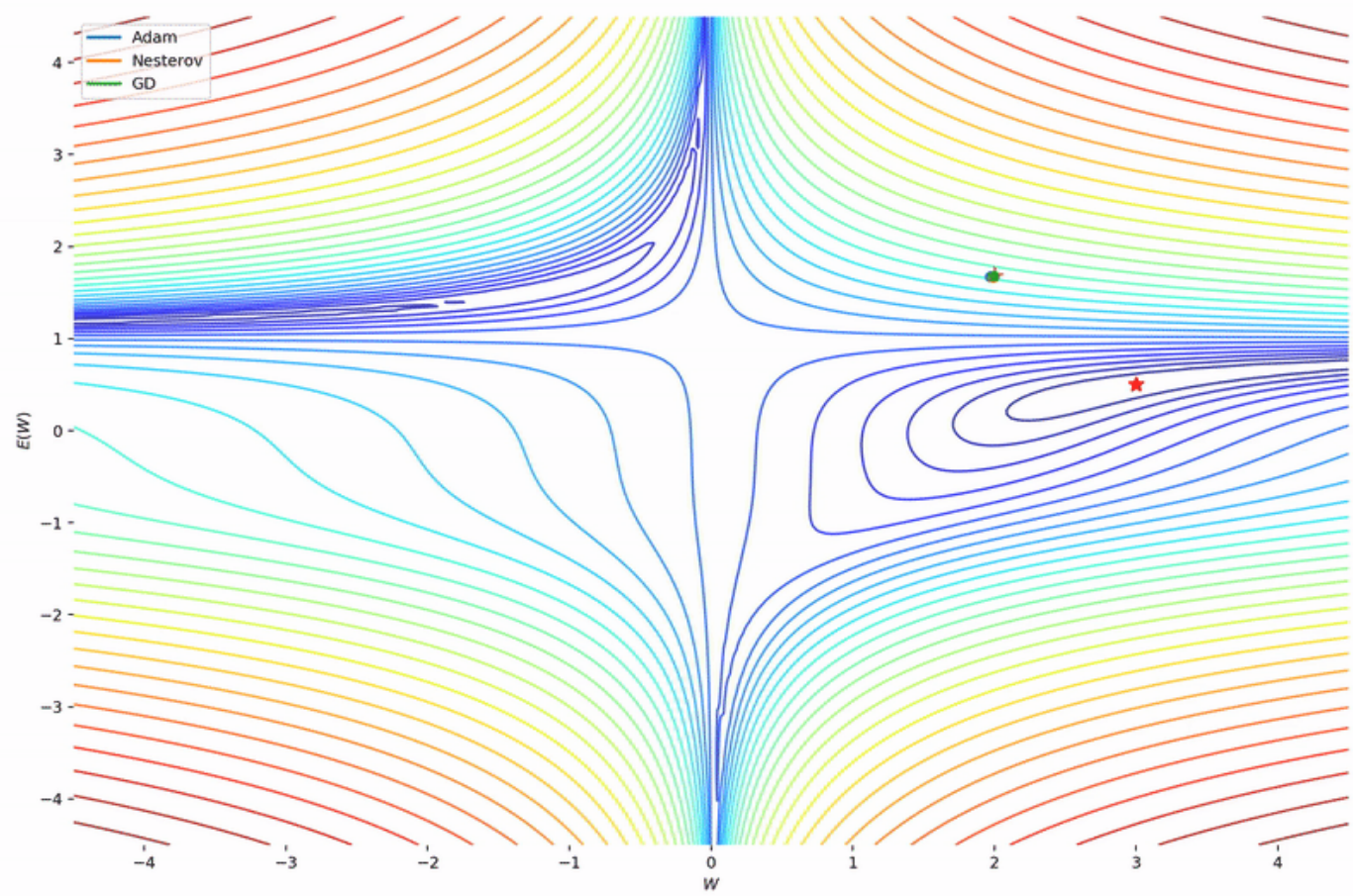
Abhängig von der Datenmenge machen wir einen **Kompromiss zwischen die Genauigkeit der Parameteraktualisierung und die Konvergenzzeit** [2, 4].

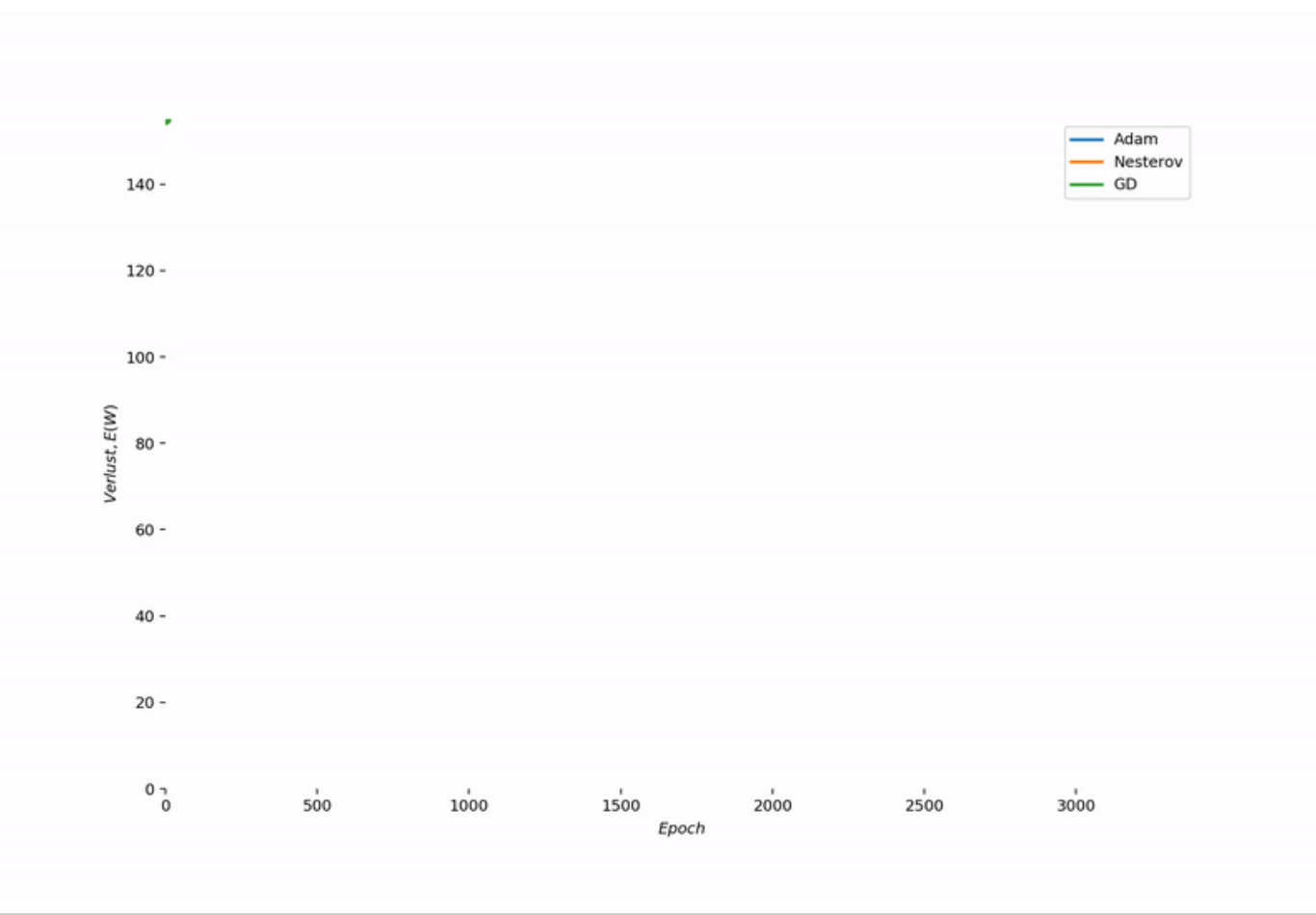


Einschränkungen der Gradientenabstieg

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

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





Fazit

- **Gradientenabstieg** ist die **typische Optimierungsmethode** in neuronalen Netzen.
- **Lernen** in Neuronale Netzwerk ist ein **iterativer Prozess**.
- Bei der **Rückwärtspropagierung (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.

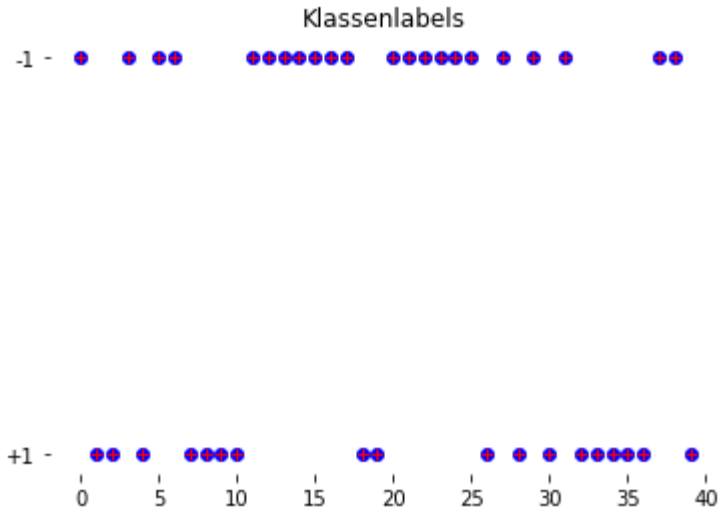
[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

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

Vorlessung Notebook herunterladen

In [4]:

```
# Einfaches neuronales Netz zur binären Klassifikation↔
```



NN AUC=1.000