Einführung in die Theorie der
Neuronalen Netze

Vorlesung 7

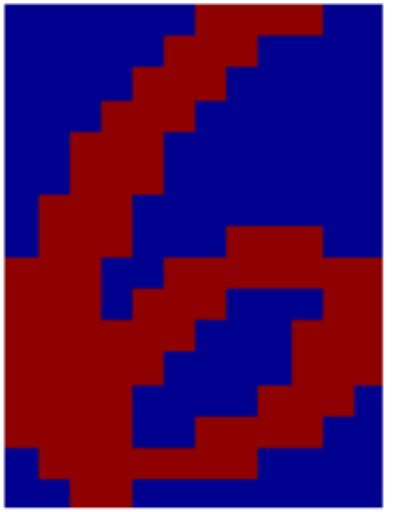
Backpropagation-Algorithmus und Beschleunigungen

Alexander Förster
Universität Bremen

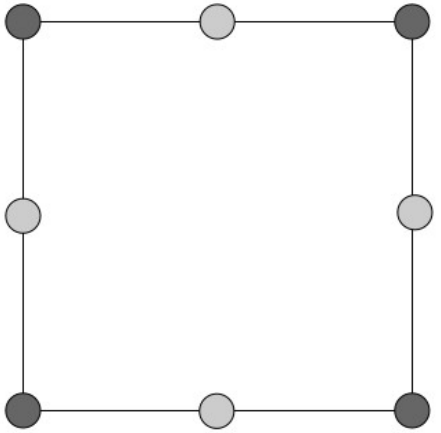
Wintersemester 2016 / 2017

Aufgabenbesprechung

1) Berechne für jede Ziffernklasse der letzten Aufgabe den Erwartungswert, um die Daten zu zentrieren. Berechne iterativ die ersten 9 Hauptkomponenten der jeweiligen Ziffernklasse. Visualisiere den Erwartungswert und die Hauptkomponenten als Bild.

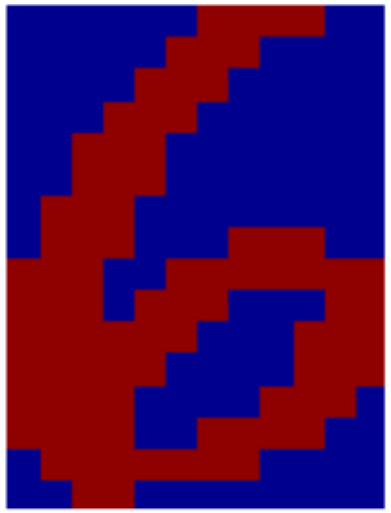


2) Verständnisfrage: Das rechte Bild zeigt 8 Punkte in der Ebene, die nicht durch zwei Hyperebenen linear getrennt werden können. Was ist die minimale Anzahl von Punkten im R2 welche nicht durch zwei lineare Trennungen separiert werden können.



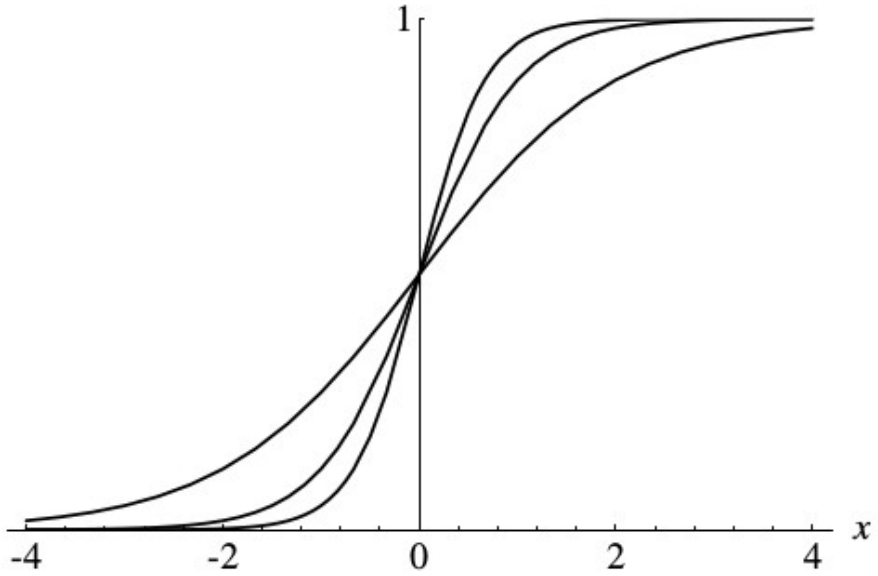
Zusatzaufgabenbesprechung

1) Berechne für alle Ziffern der letzten Aufgabe den Erwartungswert, um die Daten zu zentrieren. Berechne iterativ die ersten 15 Hauptkomponenten und visualisiere den Erwartungswert und die Hauptkomponenten als Bild.



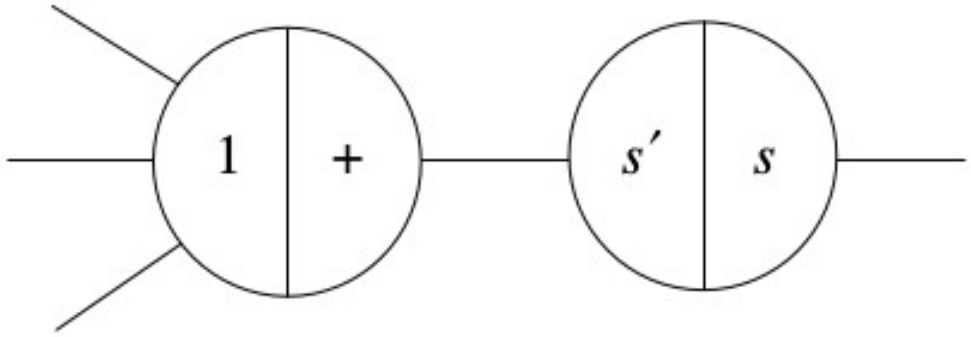
Wiederholung: BP Aktivierungsfunktion

- Stetig und differenzierbare Aktivierungsfunktion.
  Z.B. Sigmoide
  s(x) = 1/(1+e^(-cx))



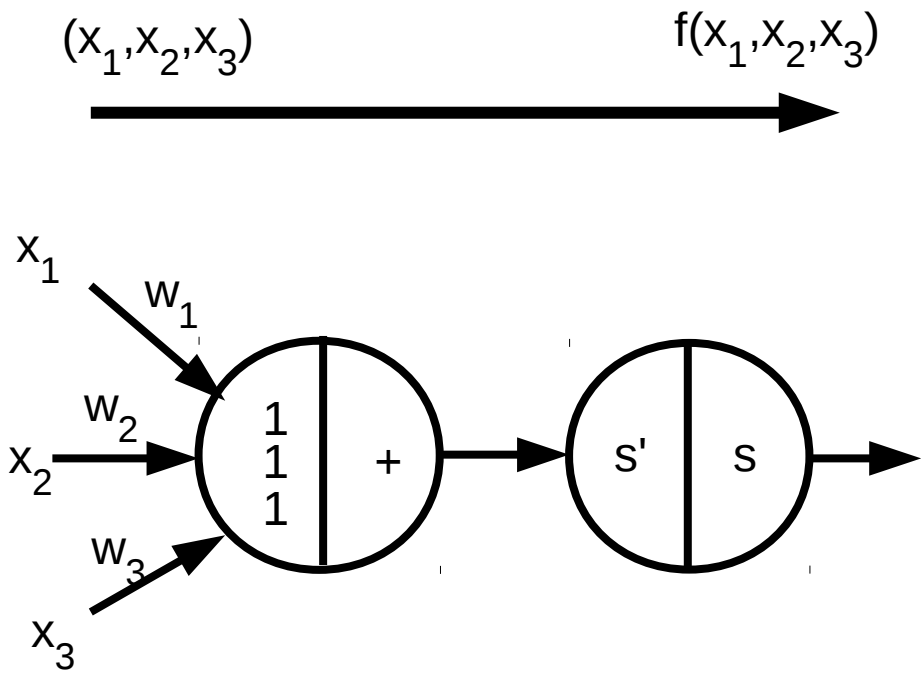
Wiederholung: BP Ableitungsspeicher

- Aufteilung von Knoten in Integrierungs- und Aktivierungsteil.
- Knoten mit Ableitungsspeicher. Z.B. für Sigmoide s'(x)=s(x)(1-s(x))



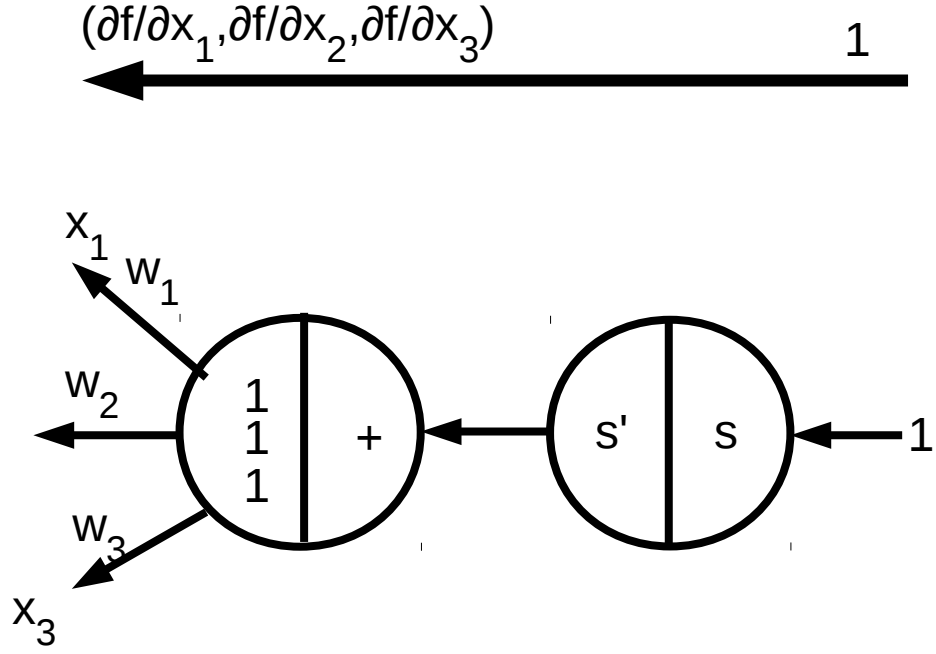
Wiederholung: BP Vorwärtsschritt

- Backpropagation Algorithmus **Vorwärtsschritt**:
  - x als Eingabe.
  - Ausgabe o und quadratischer Fehler zu gewünschte Ausgabe t wird berechnet.
  - Ableitungen werden in den Knoten gespeichert.

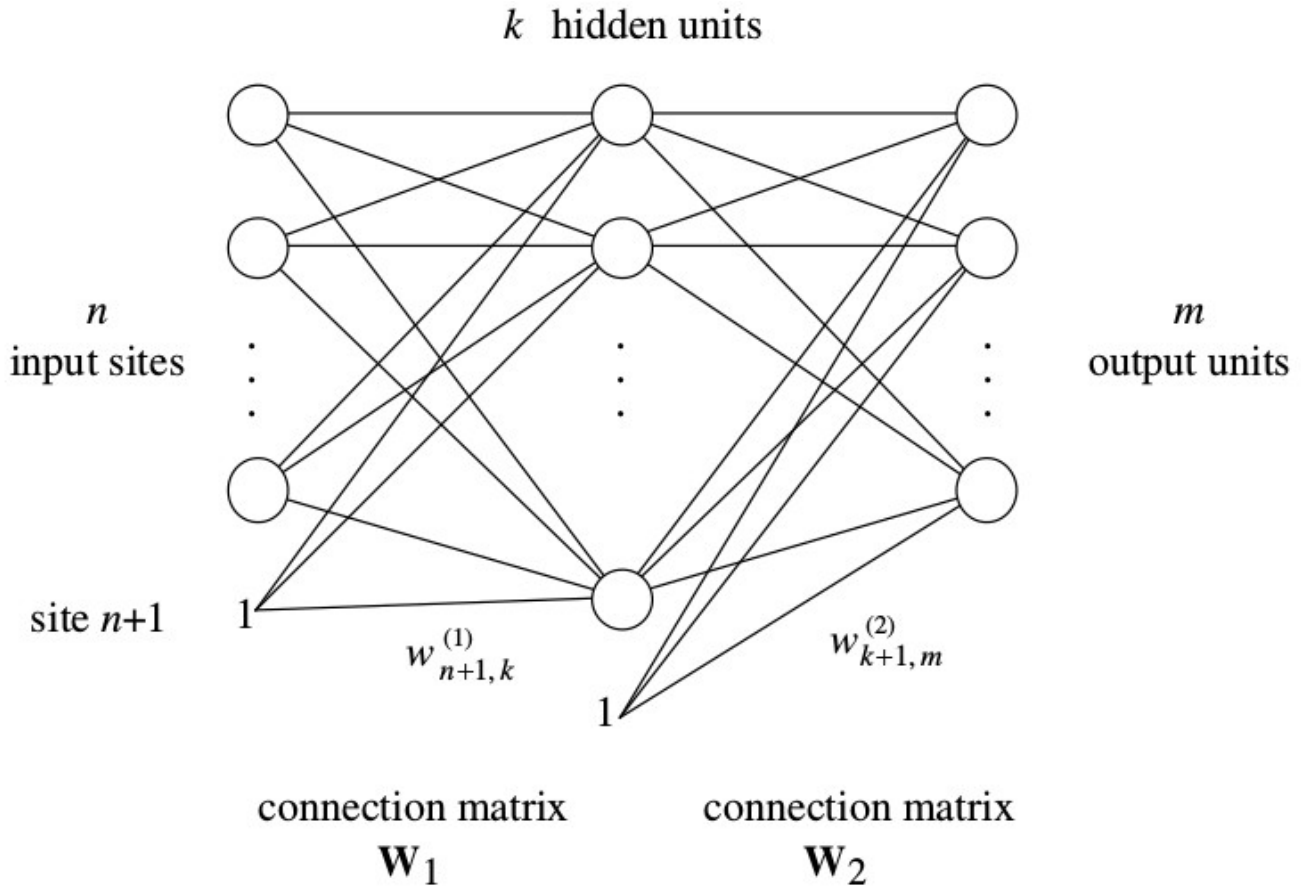


Wiederholung: BP Rückwärtsschritt

- Backpropagation Algorithmus **Rückwärtsschritt**:
  - 1 von rechts in das Netz. Werte werden ankommend summiert, mit der gespeicherten Ableitung multipliziert nach links weitergegeben.
  - Vorn kommt die Ableitung bzgl. x herraus.
  - Gleichzeitig wird bei jeder Kante die Ableitung bzgl. w berechnet und später korrigiert.



Backpropagation für geschichtete Netze



- Erweiterte Vektoren mit konstanter Eingabe (-θ als Gewicht).
- Gewichtsmatrizen zwischen den Schichten

BP Vorwärtss. für geschichtete Netze

- Ausgabe der letzten Schicht ist Eingabe der aktuellen Schicht mit der Erweiterung.
- Die Ausgabe für ein Knoten j der Schicht k-1 berchnet sich aus

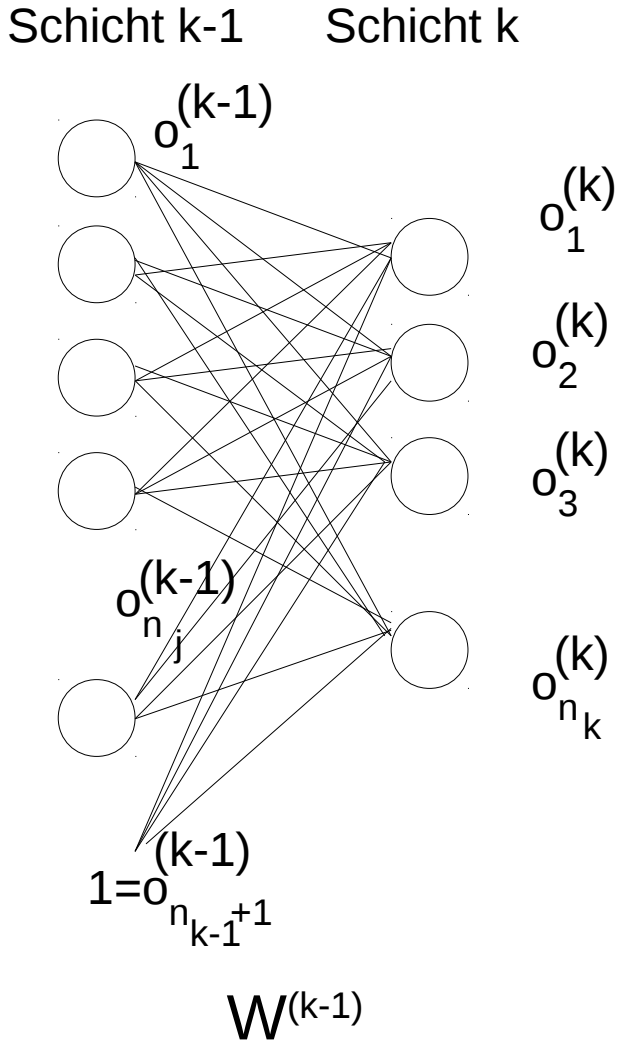
$$o_j^{(k)} = s\left(\sum_{i=1}^{n_{(k-1)}+1} w_{ij}^{(k-1)} o_i^{(k-1)}\right) \forall\ j = 1 \ldots n_k$$

$$o_{n_k+1}^{(k)} = 1$$

- In Matrixschreibweise

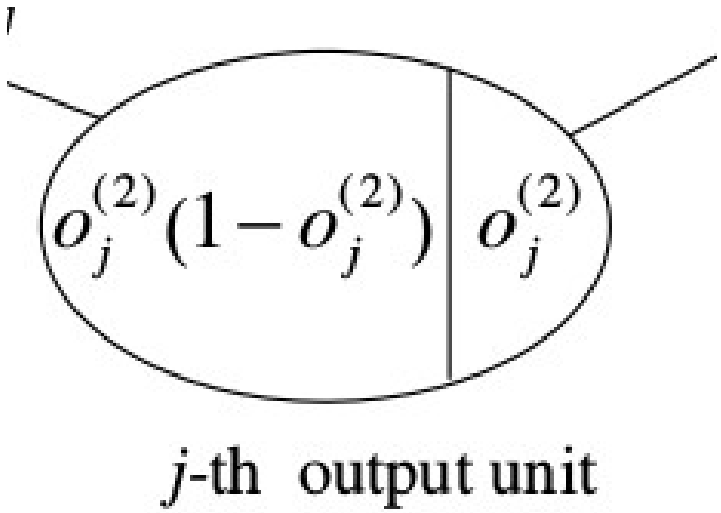
$$\hat{o}^{(k)} = s\left(o^{(k-1)} W^{(k-1)}\right)$$

$$o^{(k)} = \left(\hat{o}_1^{(k)}, \ldots, \hat{o}_{n_k}^{(k)}, 1\right)^t$$



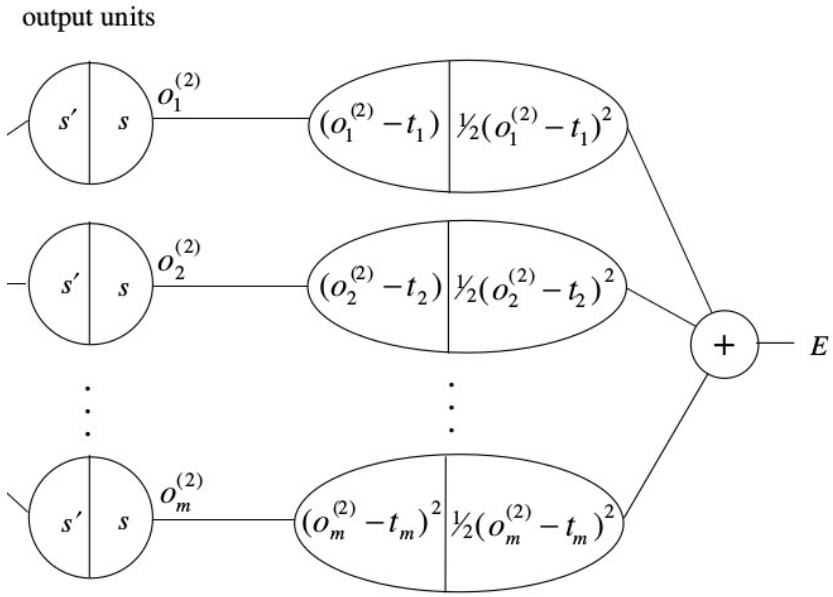
BP mit Ableitungen für geschichtete Netze

- Im Vorwärtsschritt wird nicht nur $o^{(k)}$ berechnet und gespeichert, sondern auch die Ableitungen der Sigmoiden $o^{(k)}(1-o^{(k)})$.
- In Matrixschreibweise eine Addition und punktweise Multiplikation

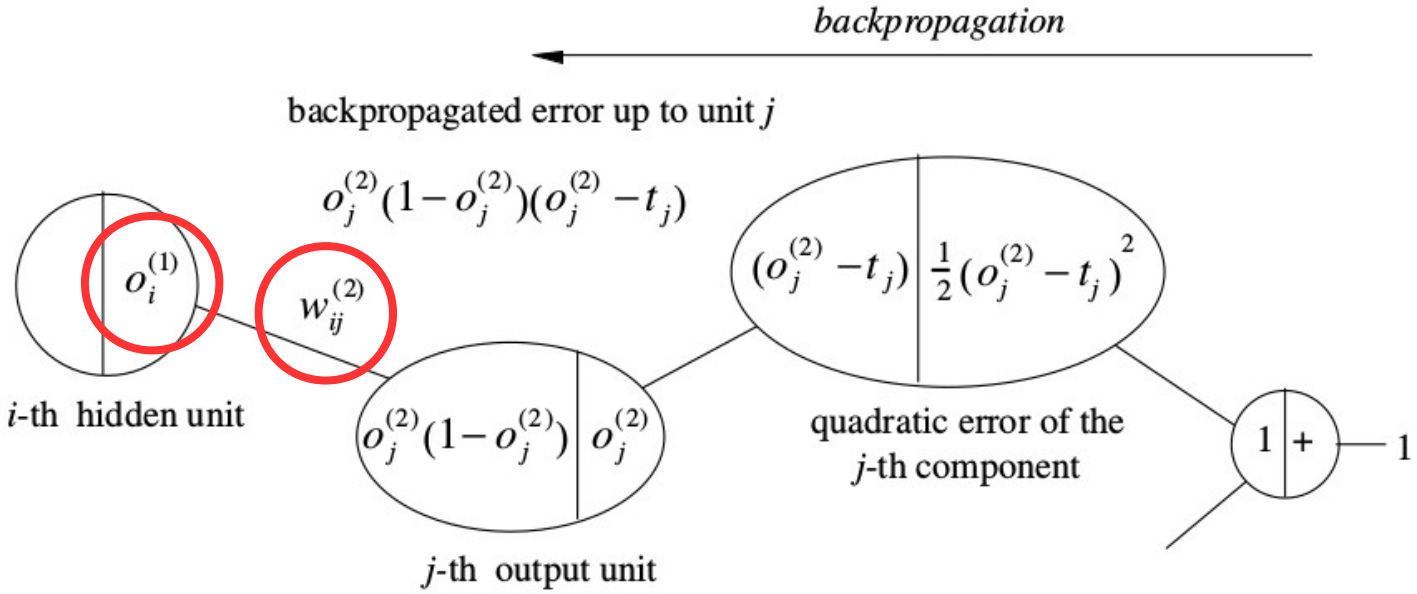


BP mit Fehlerberechnung für geschichtete Netze

- Außerdem wird noch der quadratische Fehler und dessen Ableitungen berechnet.
- Auch dies sind wieder punktweise Operationen



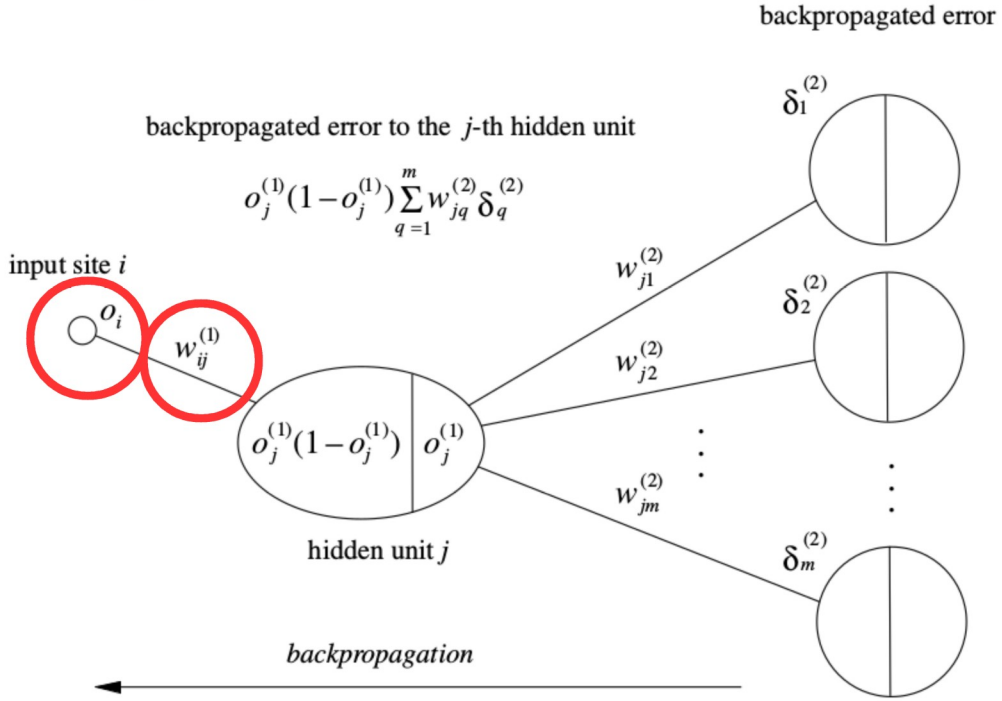
BP Rückwärtsschritt zur letzten verdeckten Schicht



- Zur Erinnerung: Um den Fehler bzgl. $w_{ij}^{(2)}$ zu berechnen wird $o_i^{(1)}$ als konstant angesehen und $w_{ij}^{(2)}$ als Variable.
- Mit der Definition $\delta_j^{(2)} = o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)}-t_j)$ gilt

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = [o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)}-t_j)]o_i^{(1)} = \delta_j^{(2)} o_i^{(1)}.$$

BP Rückwärtsschritt zur nächsten (linken) verdeckten Schicht



- Für die nächste Schicht gilt es alle von rechts kommenden Gewichte $w_{jk}^{(2)}$ und Fehlerableitungen $\delta_k^{(2)}$ zu berücksichtigen.
- Mit der Definition $$\delta_j^{(1)} = o_j^{(1)}(1-o_j^{(1)})\sum_{q=1}^{m} w_{jq}^{(2)}\delta_q^{(2)}$$
- gilt $$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_j^{(1)} o_i.$$

BP Gewichte aller Schichten anpassen

- Nach dem Berechnen aller partiellen Ableitungen des Fehlers bzgl. der Gewichte können wir die Gewichte anpassen, indem wir sie in Richtung des negativen Gradienten ändern. Die Schrittweite γ sollte dabei weder zu groß noch zu klein sein:
  - $\Delta w_{ij}^{(2)} = -\gamma o_i^{(1)} \delta_j^{(2)}$, für i=1,…,$n_2$ und j=1,…,n_3
  - $\Delta w_{ij}^{(1)} = -\gamma o_i \delta_j^{(1)}$, für o_i=x_i, i=1,…,n_1 und j=1,…,n_2

BP Gewichte unterschiedlich anpassen

- Die Gewichte können nach jedem Eingabemuster angepasst werden. Das heißt dann **Online Learning**.
- Wenn die Gewichte erst angepasst werden, nachdem die Gradienten der Fehler mehrerer Eingabemuster berechnet wurden, dann spricht man von **Batch Learning** oder **Offline Learning**.
- Beim Batch Learning müssen **nicht alle** Trainingsbeispiele vor der Anpassung der Gewichte ausgewertet werden (eine Epoche). Eine Teilmenge reicht. Wenn p Eingabevektoren für einen Stapellauf verwendet werden und für jedes Eingabe-Ausgabepaar die Korrekturen

$$\Delta_1 w_{ij}^{(1)}, \Delta_2 w_{ij}^{(1)}, \ldots, \Delta_p w_{ij}^{(1)}.$$

für das Gewicht wij(1) der ersten Schicht berechnet werden, dann wird das Gewicht mit der Summe der Korrekturen

$$\Delta w_{ij}^{(1)} = \Delta_1 w_{ij}^{(1)} + \Delta_2 w_{ij}^{(1)} + \cdots + \Delta_p w_{ij}^{(1)}.$$

angepasst.

- Für die Gewichte anderer Schichten gilt das entsprechende.

BP Rückwärtsschritt in Matrixform

- Den Vorwärtsschritt hatten wir schon in Matrixschreibweise: $\hat{o}^{(k)} = s\left(o^{(k-1)} W^{(k-1)}\right)$
- Die Ableitungen der Sigmoiden einer Schicht können wir in einer Diagonalmatrix speichern.
- Für die **zweite** Schicht

$$\mathbf{D}_2 = \begin{pmatrix} o_1^{(2)}(1-o_1^{(2)}) & 0 & \cdots & 0 \\ 0 & o_2^{(2)}(1-o_2^{(2)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_m^{(2)}(1-o_m^{(2)}) \end{pmatrix},$$

BP Rückwärtsschritt in Matrixform

- Den Vorwärtsschritt hatten wir schon in Matrixschreibweise: $\hat{o}^{(k)} = s\left(o^{(k-1)} W^{(k-1)}\right)$
- Die Ableitungen der Sigmoiden einer Schicht können wir in einer Diagonalmatrix speichern.
- Für die **erste** Schicht

$$\mathbf{D}_1 = \begin{pmatrix} o_1^{(1)}(1-o_1^{(1)}) & 0 & \cdots & 0 \\ 0 & o_2^{(1)}(1-o_2^{(1)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_k^{(1)}(1-o_k^{(1)}) \end{pmatrix}.$$

BP Rückwärtsschritt in Matrixform

- Den Vorwärtsschritt hatten wir schon in Matrixschreibweise: $\hat{o}^{(k)} = s\left(o^{(k-1)} W^{(k-1)}\right)$
- Die Ableitungen der **Fehlerberechnungen** des dreischichtigen Netzes können wir in einem Vektor speichern:
- $$\mathbf{e} = \begin{pmatrix} (o_1^{(2)} - t_1) \\ (o_2^{(2)} - t_2) \\ \vdots \\ (o_m^{(2)} - t_m) \end{pmatrix}$$
-
-
- Als Matrix: $e = o^{(2)} - t$

BP Rückwärtsschritt in Matrixform

$$\mathbf{D}_2 = \begin{pmatrix} o_1^{(2)}(1-o_1^{(2)}) & 0 & \cdots & 0 \\ 0 & o_2^{(2)}(1-o_2^{(2)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_m^{(2)}(1-o_m^{(2)}) \end{pmatrix}, \quad \mathbf{e} = \begin{pmatrix} (o_1^{(2)} - t_1) \\ (o_2^{(2)} - t_2) \\ \vdots \\ (o_m^{(2)} - t_m) \end{pmatrix} \qquad \mathbf{D}_1 = \begin{pmatrix} o_1^{(1)}(1-o_1^{(1)}) & 0 & \cdots & 0 \\ 0 & o_2^{(1)}(1-o_2^{(1)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_k^{(1)}(1-o_k^{(1)}) \end{pmatrix}.$$

- Für die propagierten Fehlervektoren $\delta^{(1)}$, $\delta^{(2)}$ des dreischichtigen Netzes ergibt sich dann in Matrixschreibweise:

$\boldsymbol{\delta}^{(2)} = \mathbf{D}_2\mathbf{e}$, denn wir wissen $\quad \delta_j^{(2)} = o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)} - t_j)$

$\boldsymbol{\delta}^{(1)} = \mathbf{D}_1\mathbf{W}_2\boldsymbol{\delta}^{(2)}$, denn wir wissen $\quad \delta_j^{(1)} = o_j^{(1)}(1-o_j^{(1)}) \sum_{q=1}^{m} w_{jq}^{(2)} \delta_q^{(2)}$

- Allgemein für ein l-schichtiges Netz:

$$\boldsymbol{\delta}^{(\ell)} = \mathbf{D}_\ell \mathbf{e}.$$

$$\boldsymbol{\delta}^{(i)} = \mathbf{D}_i \mathbf{W}_{i+1} \boldsymbol{\delta}^{(i+1)}, \qquad \text{for } i = 1, \ldots, \ell - 1.$$

BP Korrektur in Matrixform

- Die Gewichtsmatrixkorrekturen W_1 und W_2 sind

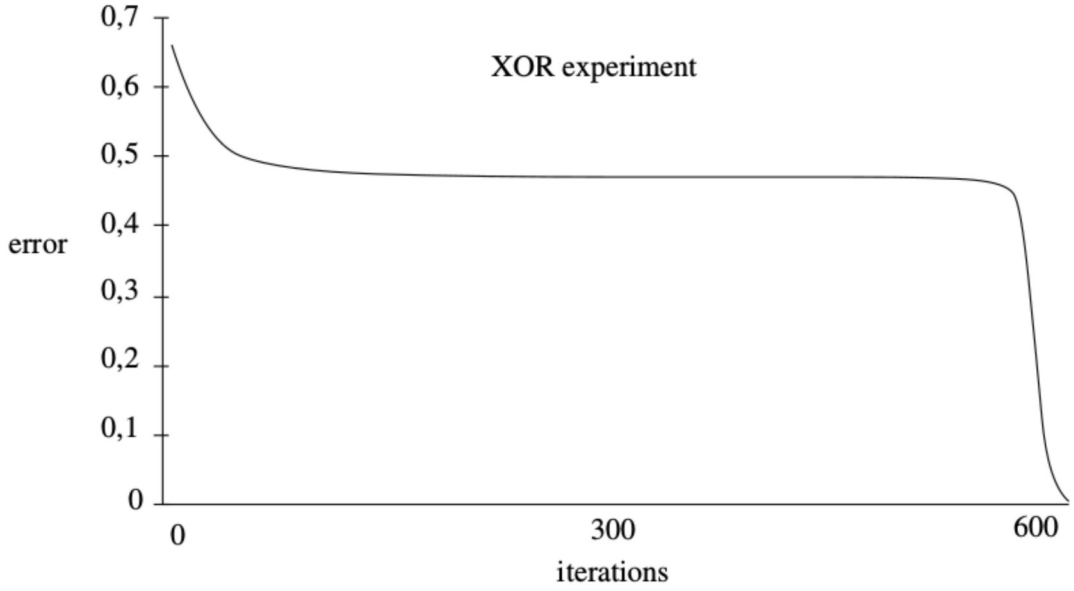
$\Delta\overline{\mathbf{W}}_2^{\mathrm{T}} = -\gamma\boldsymbol{\delta}^{(2)}\hat{\mathbf{o}}^{1\,\mathrm{T}}$, denn wir wissen $\Delta w_{ij}^{(2)} = -\gamma o_i^{(1)}\delta_j^{(2)}$,

$\Delta\overline{\mathbf{W}}_1^{\mathrm{T}} = -\gamma\boldsymbol{\delta}^{(1)}\hat{\mathbf{o}}^{\mathrm{T}}$, denn wir wissen $\Delta w_{ij}^{(1)} = -\gamma o_i\delta_j^{(1)}$,

- Dabei ist $\overline{W}$ die Matrix, zur erweiterten Eingabe ô und W die zur Eingabe o.

Lerngeschwindigkeit und Erfolg

- XOR mit drei Knoten.
- Ein langsamer Fortschritt.
- Durch den kleinen Gradienten der Sigmoide immerhin ein Fortschritt.
- Nach 600 Iterationen wurde das Problem gelernt.
- **Wie können wir das Lernen beschleunigen?**



Backpropagation Eigenschaften

- BP ist langsam
- Mit schlechter Parameterwahl ist PB noch langsamer
- Das Lernproblem für NNs ist NP-vollständig. Der Aufwand wächst also exponentiell mit steigender Parameterzahl
- On-line Lernen ist ab einer gewissen Problem- und Beispiel-größe meist besser als viele Veröffentlichungen über vermeintliche Beschleunigungen.

Backpropagation Eigenschaften

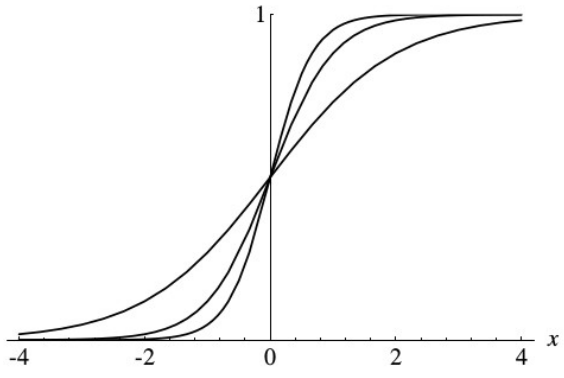
- BP ist langsam
- Mit schlechter Parameterwahl ist PB noch langsamer
- Das Lernproblem für NNs ist NP-vollständig. Der Aufwand wächst also exponentiell mit steigender Parameterzahl
- On-line Lernen ist ab einer gewissen Problem- und Beispiel-größe meist besser als viele Methoden in Veröffentlichungen über vermeintliche Beschleunigungen.
- **Trotzdem: Einige Methoden helfen.**

Online-Lernen

- Online Lernen geht oft in die falsche Richtung, denn es „optimiert“ nur auf ein Eingabemuster.
- Da die Eingaben **zufällig** wechseln sollten sich die Fehler mit der Zeit herausmitteln.
- Online-Lernen ist Lernen mit Rauschen.
- Durch das Rauschen werden lokale Minima verlassen, die bei der globalen Fehlerfunktion vorhanden sind.
- Es ist ein Monte-Carlo Verfahren.

Initiale Gewichte

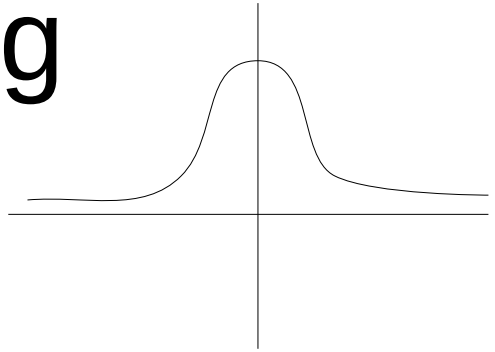
- Die Gewichte *w* sollten zufällig aus einem Intervall [-α,α] gewählt werden, sodass die Sigmoide nicht sofort in der Sättigung ist und somit die Ableitung sehr flach ist. Es hängt also von den Werten der Eingabe als auch von der Anzahl der Eingaben ab.
- Wenn die x und w unabhängig sind, und keinen Bias haben, dann liegt der Erwartungswert der Summe bei 0 und damit ist s nicht in der Sättigung.
- Mit steigendem α, bzw. größerem n steigt auch die Varianz und somit die Gefahr der Sättigung.



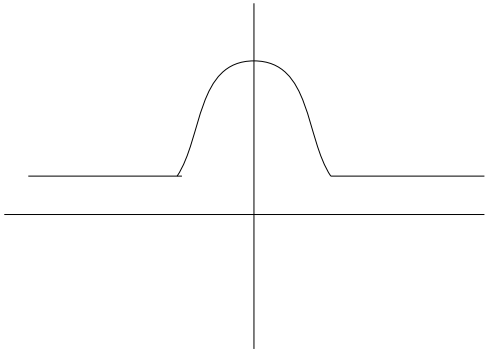
$$s\left(\sum_{i=1}^{n} w_i x_i\right)$$

Abschneiden der Sigmoidenableitung

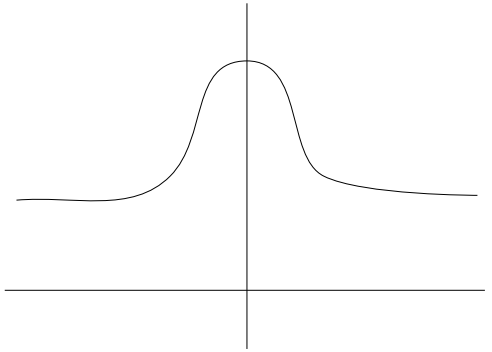
- Durch die geringe Steigung weiter von 0 entfernt ist das Lernen dort auch langsam.
- Wir können die Ableitung der Sigmoide nach unten beschränken, z.B. durch s'(x) ≥0,01
- Wir können eine Konstante ε zu s'(x) addieren.



s'(x)



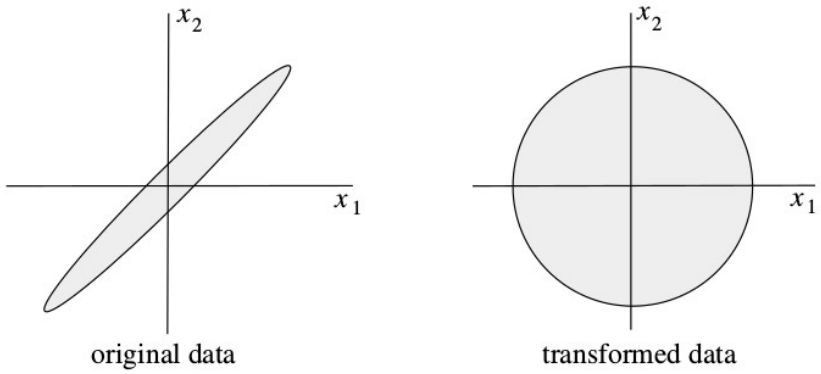
s'(x)≥0,01



s'(x)+ε

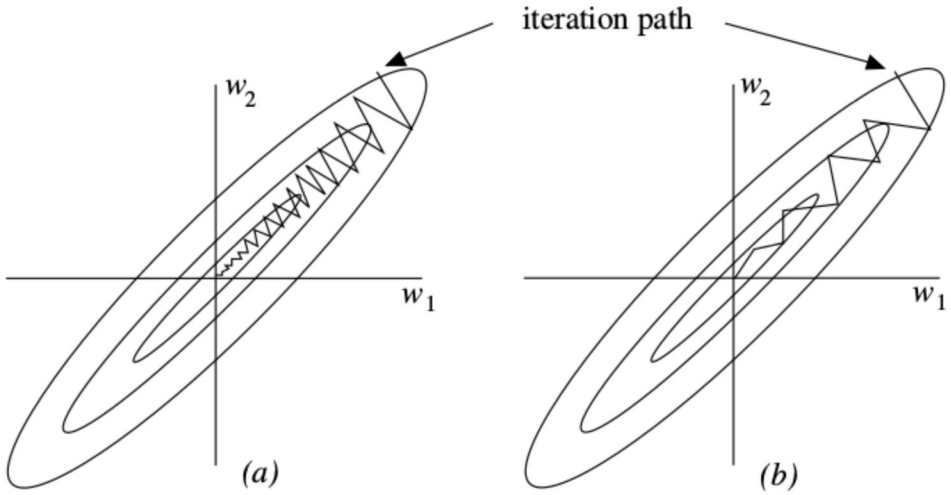
Optimieren der Eingabedaten

- Anwenden von PCA auf die Eingabedaten, um die Eingabedimension zu reduzieren.
- Verwenden von bipolaren Vektoren (nicht bei sparsamer Kodierung binäre Vektoren)
- Zentrieren der Daten um 0.
- Transformation der Daten in einer linearen Schicht, um die Daten zu dekorrlieren. Die Kovarianzmatrix ist möglichst eine Diagonalmatrix.



Backpropagation mit Impuls (Momentum)

- Wenn das Minimum in einem Schmalen Tal liegt (Abbildung links), dann osziliert der Gradientenabstieg fast orthogonal in Richtung des Minimums.
- Die Trägheit bildet ein gewichteten Mittelwert des aktuellen und vergangener Gradienten (Abbildung rechts)



- Für **ein** Gewicht wk(i) wird zum Zeitpunkt i die Korrektur mit

$$\Delta w_k(i) = -\gamma \frac{\partial E}{\partial w_k} + \alpha \Delta w_k(i-1)$$

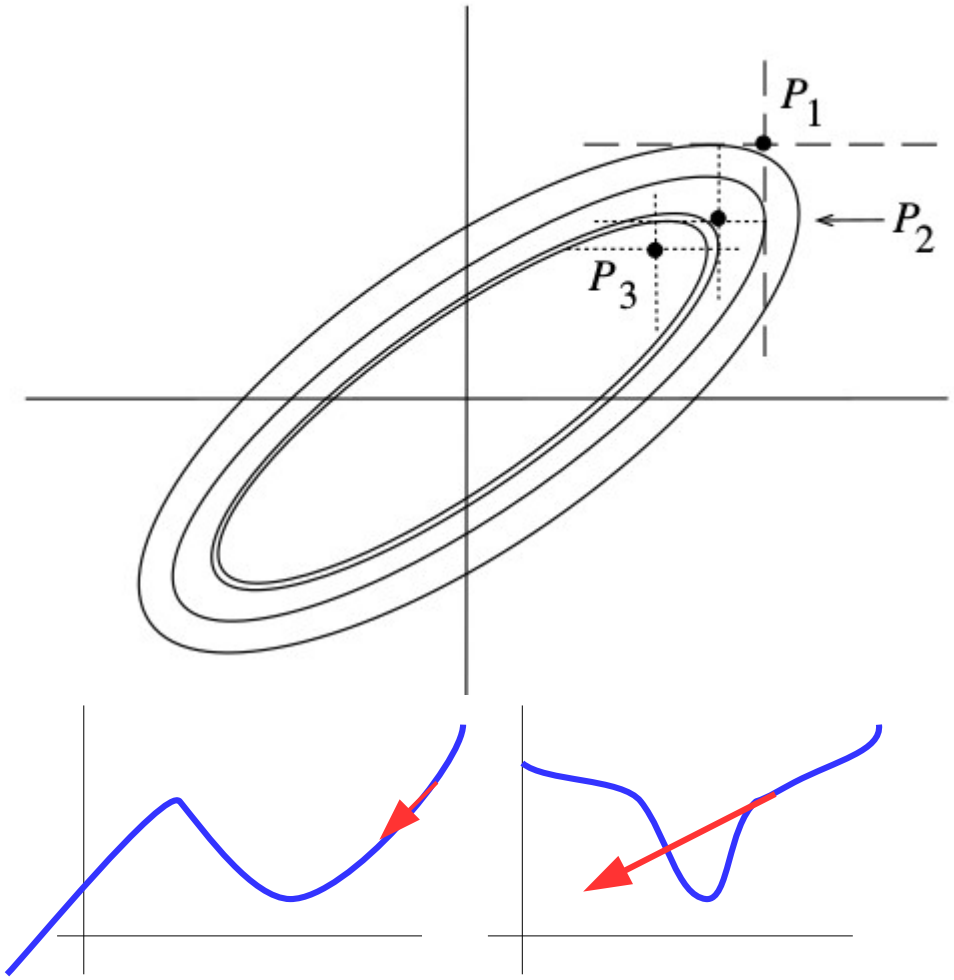
durchgeführt. Dabei sind γ und α Konstanten, die i.d.R. durch Versuch und Irrtum bestimmt werden.

Adaptive Schrittweiten

- Jedes Gewicht w_i bekommt seine eigene Lernrate γ_i.

$$\Delta w_i = -\gamma_i \frac{\partial E}{\partial w_i}.$$

- Idee: Die Fehlerfunktion für jede partielle Ableitung sieht anders aus. Daher sollte auch die Schrittweite individuell angepasst werden.
- Nur wie?

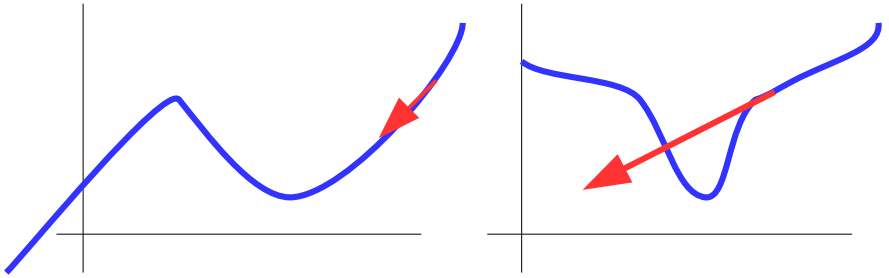


Adaptive Schrittweiten: Silva und Almeida Algorithmus

- Es werden die aufeinander folgenden partiellen Ableitungen nach w_i betrachtet. Wenn sich das Vorzeichen des Gradienten umdreht wird die Lernrate verringert. Wenn nicht wird sie vergrößert:

$$\gamma_i^{(k+1)} = \begin{cases} \gamma_i^{(k)} u \text{ if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} \geq 0 \\ \gamma_i^{(k)} d \text{ if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} < 0 \end{cases} \qquad \Delta^{(k)} w_i = -\gamma_i^{(k)} \nabla_i E^{(k)} .$$

- Die Konstanten *u* (up), *u*>1, und *d* (down) *d*<1 müssen geeignet gewählt werden.
- Vorsicht: γ wächst und schrumpft exponentiell, kann also leicht entarten.

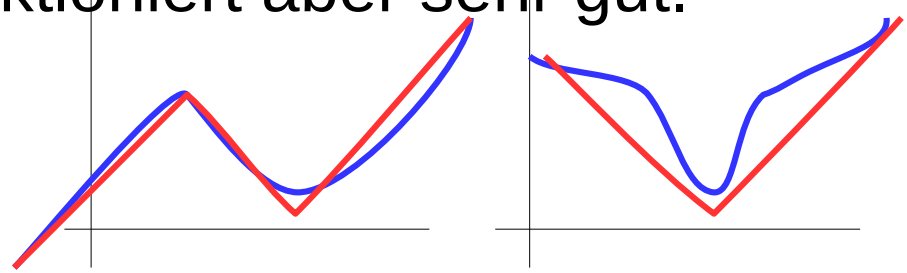


Adaptive Schrittweiten: **Rprop (!)**

- Bei Silva und Almeida gibt es zwei Probleme: γ ist nicht beschränkt und wenn der Gradient sehr flach ist, dann muss γ sehr groß sein, damit etwas passiert.
- Rprop beschränkt γ und betrachtet nicht den Gradienten, sondern nur dessen Vorzeichen:

$$\gamma_i^{(k+1)} = \begin{cases} \min(\gamma_i^{(k)} u, \gamma_{max}) & \text{if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} > 0 \\ \max(\gamma_i^{(k)} d, \gamma_{min}) & \text{if } \nabla_i E^{(k)} \cdot \nabla_i E^{(k-1)} < 0 \\ \gamma_i^{(k)} & \text{otherwise,} \end{cases} \qquad \Delta^{(k)} w_i = -\gamma_i^{(k)} \operatorname{sgn}(\nabla_i E^{(k)}),$$

- Die Konstanten *u* (up), *u*>1, und *d* (down) *d*<1 wie gehabt.
- Rprop sieht unpräzise aus. Funktioniert aber sehr gut.



Dynamische Anpassung

- Eine einfach zu implementierende Methode ist die globale, dynamische Anpassung der Lernrate γ. Die Methode ist aber nicht so gut wie die mit individuellen Lernraten.
- Idee: Es werden zwei Anpassungen berechnet:

$$\mathbf{w}^{(k_1)} = \mathbf{w}^{(k)} - \nabla E(\mathbf{w}^{(k)})\gamma^{(k-1)} \cdot \xi$$
$$\mathbf{w}^{(k_2)} = \mathbf{w}^{(k)} - \nabla E(\mathbf{w}^{(k)})\gamma^{(k-1)} / \xi$$

- Dabei ist ζ klein, z.B. ζ=1,7. Die Lernrate γ wird bezüglich des Fehlers der beiden Anpassungen gesetzt:

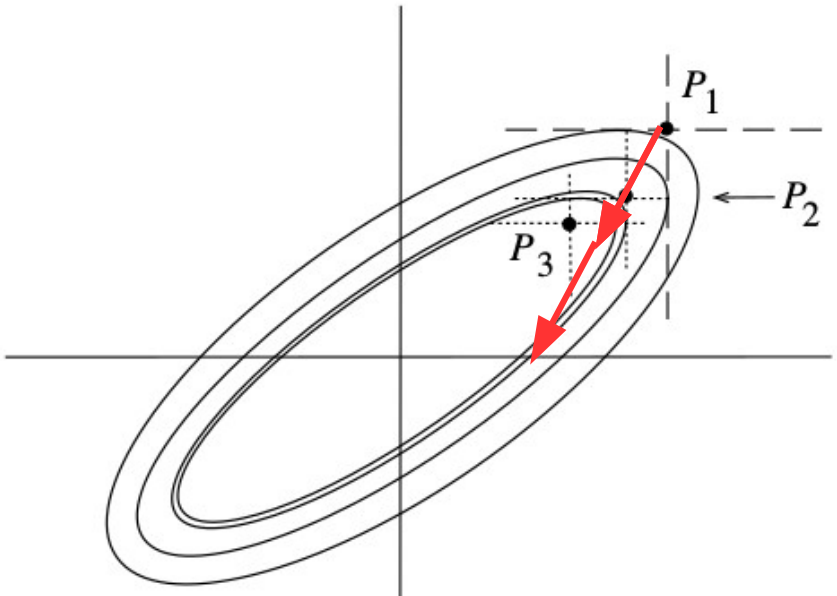
$$\gamma^{(k)} = \begin{cases} \gamma^{(k-1)} \cdot \xi \text{ if } E(\mathbf{w}^{(k_1)}) \leq E(\mathbf{w}^{(k_2)}) \\ \gamma^{(k-1)} / \xi \quad \text{otherwise.} \end{cases}$$

- Es wird der besser Gewichtesatz behalten:

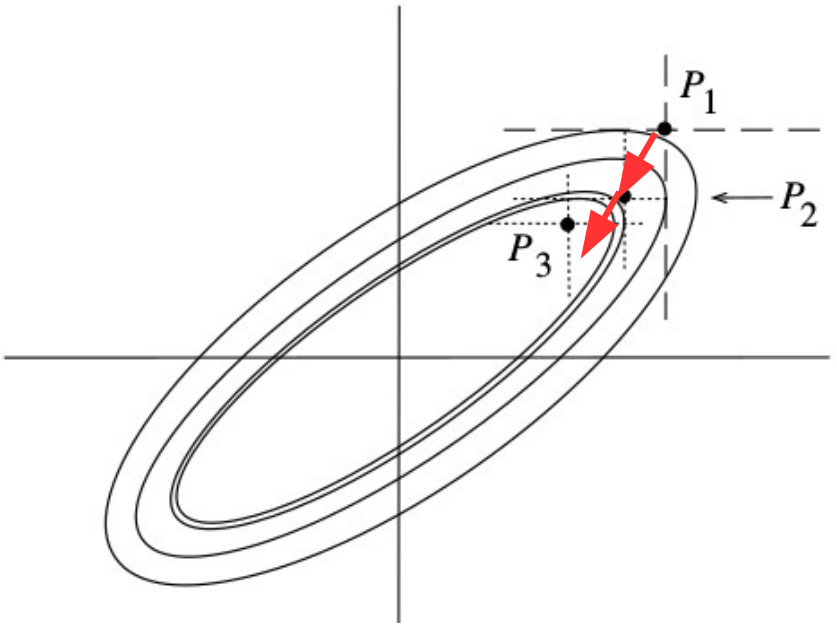
$$\mathbf{w}^{(k+1)} = \begin{cases} \mathbf{w}^{(k_1)} \text{ if } E(\mathbf{w}^{(k_1)}) \leq E(\mathbf{w}^{(k_2)}) \\ \mathbf{w}^{(k_2)} \quad \text{otherwise.} \end{cases}$$

Dynamische Anpassung

- Test mit größerer Lernrate: $\gamma^{(k-1)}\zeta$, wenn $\zeta>1$
- Test mit kleinerer Lernrate: $\gamma^{(k-1)}/\zeta$, wenn $\zeta>1$



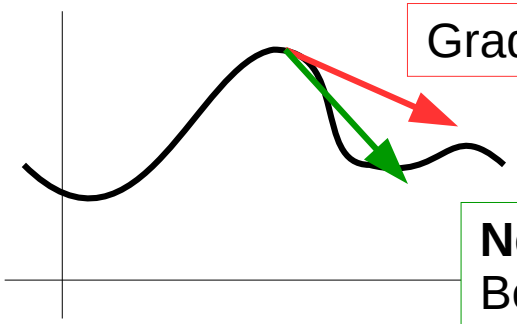
kleiner ist besser



größer ist besser

Algorithmen zweiter Ordnung

- Wenn wir nicht nur die Steigung, sondern auch die Änderung der Steigung, also die zweite Ableitung betrachten, dann hilft das eventuell bei der Fehlerminimierung.



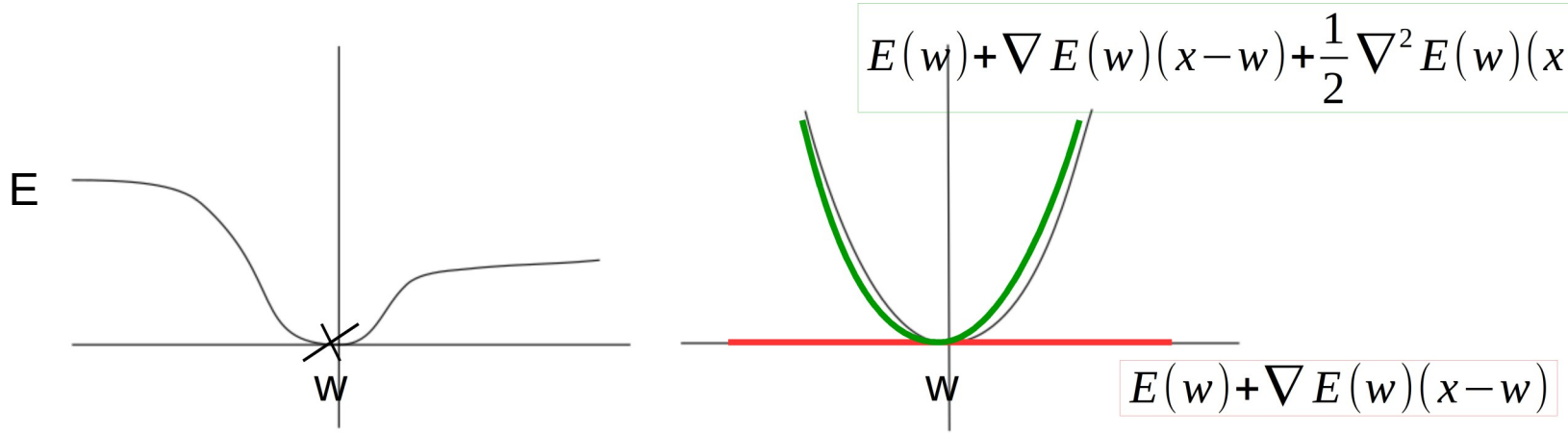
Gradientenabstieg: 1. Ableitung (Tangente)

Newton Methode:
Berücksichtigung des Verlaufs (2. Ableitung)

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\nabla_i E(\mathbf{w})}{\partial^2 E(\mathbf{w}) / \partial w_i^2}.$$

Algorithmen zweiter Ordnung

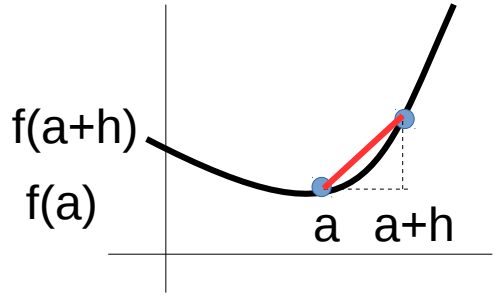
- Mathematisch haben wir eine quadratische Approximation der Funktion E an der Stelle w anstatt einer linearen, wenn wir nur die erste Ableitung betrachten.



- Wenn wir weiter Ableitungen betrachten, dann sind wie bei der Taylorapproximation mit Polynomen.

Algorithmen zweiter Ordnung

- Wir brauchen die zweite Ableitung nicht zu berechnen. Er reicht eine Approximation, wie man das auch mit der 1. Ableitung macht:
  - f'(a) = (f(a+h)-f(a))/h
  - f''(a) = (f'(a+h)-f'(a))/h
-



- Beispiele sind Quickprop und QRprop.

Quickprop

- Die Differenz zweier Ableitungen ist eine Näherung der zweiten Ableitung.
- Wir berechnen zum Zeitpunkt *k* die Korrektur des Gewichts w_i:

$$\Delta^{(k)} w_i = \frac{\nabla_i E^{(k)}}{\left(\nabla_i E^{(k)} - \nabla_i E^{(k-1)}\right) / \Delta^{(k-1)} w_i}$$

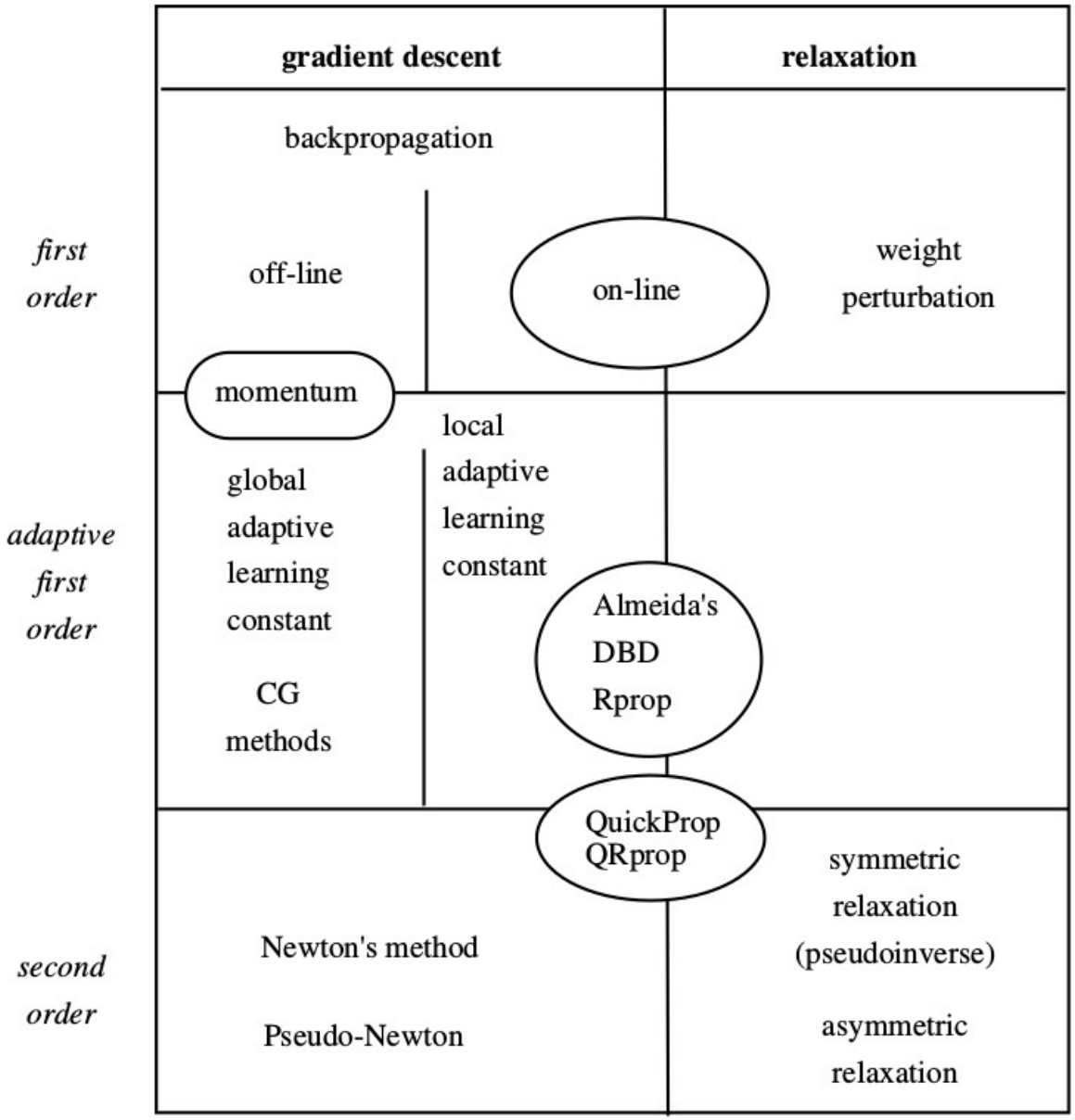
- Im Nenner steht die Approximation der zweiten Ableitung. Im Zähler die erste Ableitung.
- Wir haben hier keine explizite Lernrate.
- Die Korrektur kann sehr groß werden, deshalb sollte $\Delta^{(k)} w_i$ beschränkt werden.

Lernen ohne Backprop: Entspannungsverfahren

- Anstatt den Fehler zurückzurechnen können auch die Gewichte w bei gleicher Eingabe leicht variiert werden (Rauschen mit $w \pm \beta$). Das Gewicht wird dann in Richtung des kleineren Fehlers gezogen:

$$\Delta w_i = -\gamma \frac{E(w') - E(w)}{\beta}$$

Überblick Beschleunigunsverfahren



Aufgabe

1) Implementiere den Rprop Backpropagation-Algorithmus für ein Netz mit einer verdeckten Schicht.
2) Trainiere ein Netz zur Erkennung der handgeschriebener Ziffern mit dem R-Prop Algorithmus. Benutze die Trainingsdaten zum Lernen.
3) Wie hoch ist die Erkennungsrate der Trainingsdaten bei steigender Anzahl verdeckter Knoten?
4) Wie hoch ist die Erkennungsrate der Testdaten bei steigender Anzahl verdeckter Knoten?