VIII. LERNEN UND OPTIMIEREN

- In den meisten Lernverfahren für neuronale Netzwerke ist "lernen" gleichbedeutend mit "optimieren":

  Lernen = Prozess, welcher die Effizienz des Netzwerks (bezüglich einer vorgegebenen Aufgabe) optimiert.

- Wie die Effizienz eines Netzwerks gemessen oder bewertet wird, hängt von der Problemstellung ab.

  Ist z.B. der gewünschte Output $\{D_i^\nu\}$ für einen Satz von Lernbeispielen vorgegeben ($\nu = 1, \ldots, N$), so wählt man oft den quadratischen Outputfehler als Mass für die Effizienz:

$$F = \sum_{\nu=1}^{N} F^\nu \quad , \qquad F^\nu = \frac{1}{2} \sum_i (D_i^\nu - O_i^\nu)^2 \quad .$$

  Da $O_i^\nu = O_i^\nu(\underline{W})$, ergibt sich ein Optimierungskriterium der Form

$$F(\underline{W}) = \min \quad ,$$

  d.h. die Optimierung bezieht sich auf die Wahl der Verbindungsgewichte $\underline{W} = \{W_{ij}\}$.

- Die zwei Hauptfaktoren, die das Lernverhalten eines neuronalen Netzwerks bestimmen, sind

  1) die Topographie der "Fehlerlandschaft" über dem $\underline{W}$-Raum

  2) die gewählte Lernstrategie (bzw. Optimierungsstrategie).

VIII.1 Gradientenverfahren

Die meisten neuronalen Lernverfahren (siehe z.B. "Error Backpropagation", Kapitel VIII) beruhen auf einer Gradientenmethode zur Minimierung der Fehlerfunktion $F(\underline{W})$:

$$W_{ij} \longrightarrow W_{ij} + \Delta W_{ij}$$

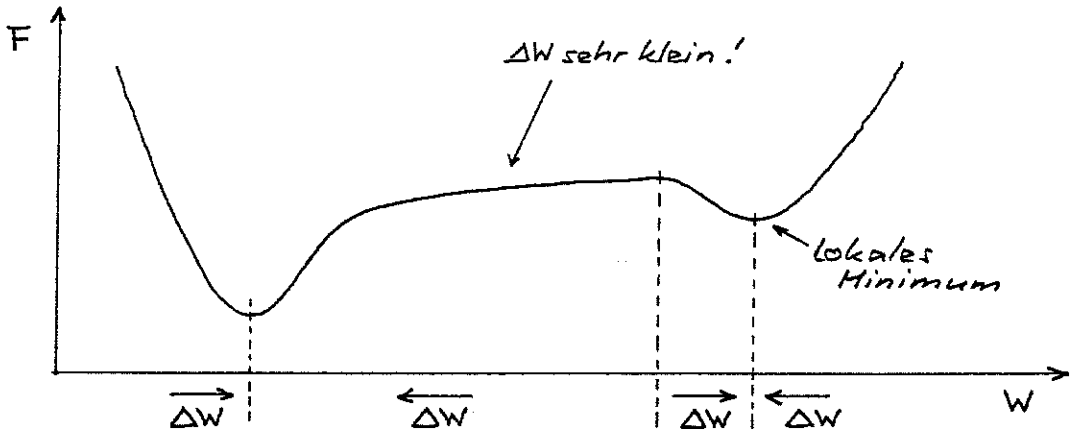
"Example-by-example" Version: $\Delta W_{ij} = -\eta \frac{\partial F^{\nu}}{\partial W_{ij}}$,

wobei die Lernbeispiele ν in zufälliger Reihenfolge immer wieder präsentiert werden.

"Batch"-Version: $\Delta W_{ij} = -\eta \frac{\partial F}{\partial W_{ij}}$,

d.h. die Fehler werden über alle Lernbeispiele summiert, bevor die Gewichte geändert werden.

- Probleme mit Gradientenverfahren bei komplizierten Fehlerlandschaften:
  - langsame Konvergenz (flache Täler u. Plateaus)
  - Steckenbleiben in schlechten lokalen Minima

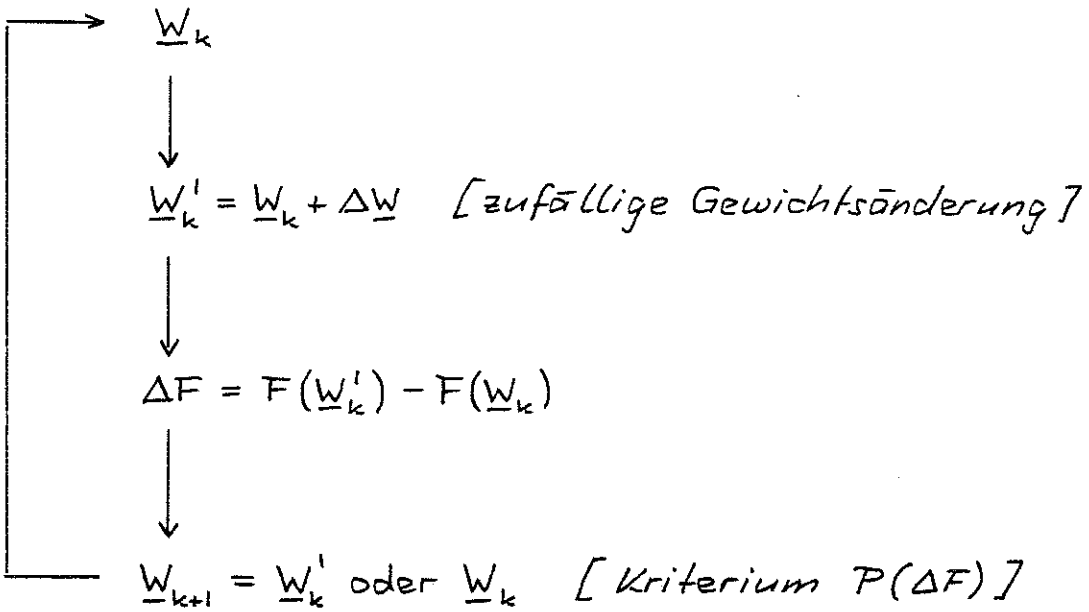


- Gradientenverfahren sind zudem nur anwendbar, wenn die Gewichte kontinuierlich variieren dürfen (d.h. $-\infty < W_{ij} < +\infty$) und wenn die Aktivierungsfunktion $f\left(\sum_j W_{ij} S_j\right)$ differenzierbar ist.

VIII.2 Stochastische Lernstrategien

Problem: $F(\underline{W}) = \sum_{\nu=1}^{N} F^{\nu}(\underline{W}) = \min$

- Stochastische Lernverfahren sind Strategien, die eine zufällige Suche im $\underline{W}$-Raum gegen ein gutes Minimum der Fehlerfunktion $F(\underline{W})$ steuern:



- Solche Verfahren sind auch dann anwendbar, wenn die Gewichte auf diskrete Werte beschränkt sind (z.B. $W_{ij} = \pm 1$) [→ Kombinatorische Optimierung], oder wenn die Aktivierungsfunktion f nicht differenzierbar ist.

- Implementierungsoptionen:
  - "Batch" oder "Example-by-Example"
  - Wahl der Wahrsch.keitsverteilung $\mu(\Delta\underline{W})$ für die zufälligen Gewichtsänderungen
  - Kriterium $P(\Delta F)$ für das Akzeptieren des zufällig erzeugten Gewichtsvektors $\underline{W}'_k$

VIII.3 "Iterative Improvement"

→ Kriterium $P(\Delta F)$:

$$\underline{W}_{k+1} = \begin{cases} \underline{W}_k' & \text{falls} \quad \Delta F \leq 0 \\ \underline{W}_k & \text{"} \quad \Delta F > 0 \end{cases}$$

d.h. die Gewichtsänderung wird nur akzeptiert, wenn die Fehlerfunktion abnimmt.

→ Stochastisches Äquivalent zur Gradientenmethode!

→ Ähnliche Probleme mit lokalen Minima, falls $|\Delta \underline{W}|$ beschränkt ist.

Aber Satz: *Falls* 1) *die Suche auf einen kompakten Unterraum S des $\underline{W}$-Raums beschränkt wird,*

2) *die Reichweite von $\mu(\Delta \underline{W})$ grösser als der Durchmesser von S ist,*

dann:

$$\lim_{k \to \infty} \text{Prob}\{ F(\underline{W}_k) < F_{min}^{(S)} + \varepsilon \} = 1$$

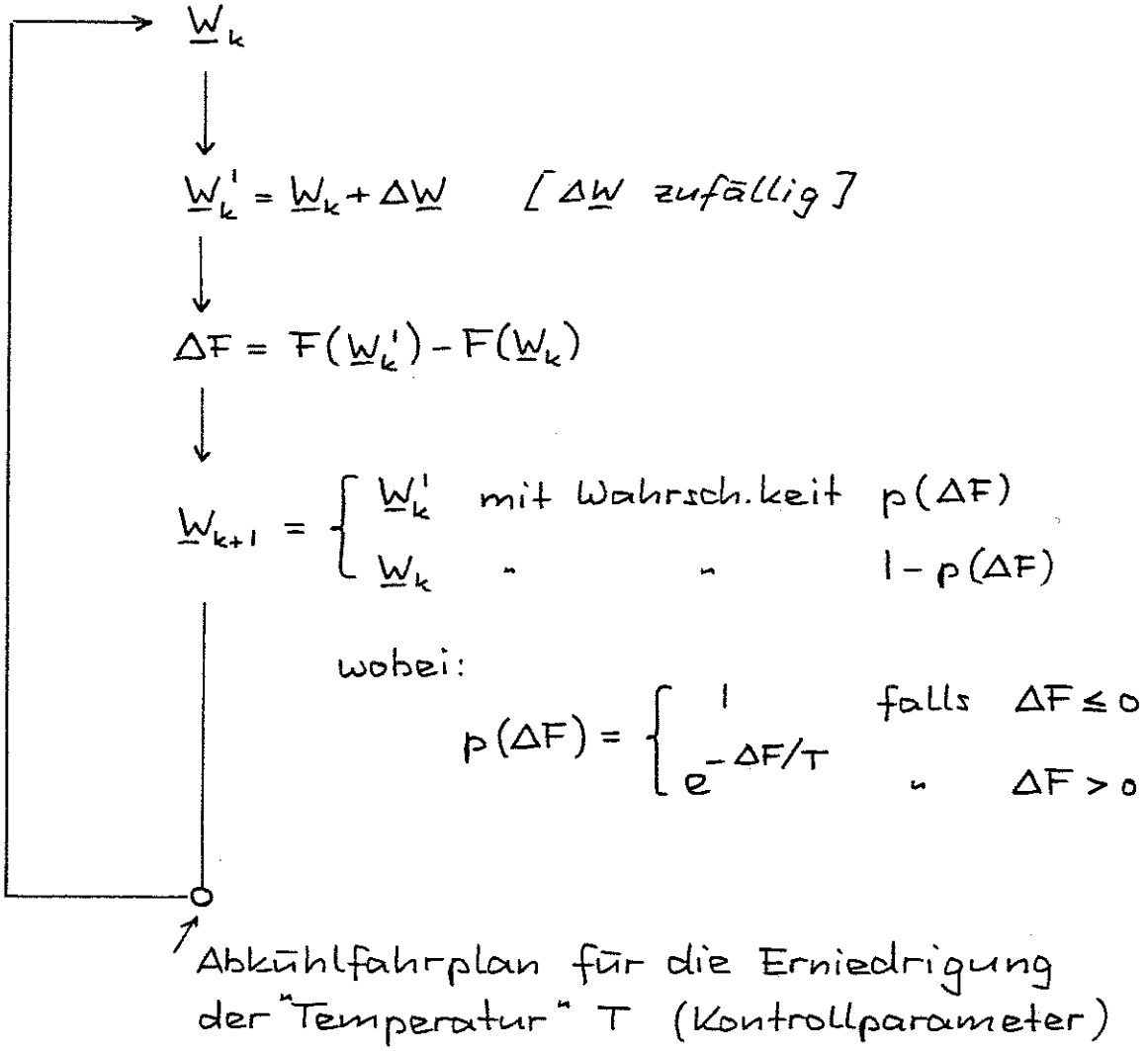
[F.J. Solis and R.J.-B. Wets, Math. of Op. Res. 6, 19 (1981)]

d.h. das "Iterative Improvement" Verfahren findet das Minimum von F im Unterraum S mit beliebiger Genauigkeit, falls genügend Iterationsschritte gemacht werden.

Frage: Verhalten des Algorithmus für beschränkte Zahl von Iterationsschritten?

VIII.4 "Simulated Annealing"

Der Simulated Annealing Algorithmus stammt aus der statistischen Physik [Kirkpatrick et al, Science 220, 671 (1983)] und kann wie folgt beschrieben werden:



Bemerkungen:

- Für jedes $T>0$ sind Gewichtsänderungen, die den Fehler vergrössern, mit einer gewissen Wahrsch.keit zugelassen. → Möglichkeit, aus schlechten lokalen Minima wieder herauszukommen.
- "Iterative Improvement" ist ein Spezialfall von "Simulated Annealing" ($T=0$ von Anfang an).

Warum funktioniert Simulated Annealing ?

Betrachte ein physikalisches System, dessen Zustand durch $\underline{W}$ charakterisiert ist und dessen Energiefunktion durch $F(\underline{W})$ gegeben ist.

Suche Zustände $\underline{W}$ mit möglichst tiefer "Energie" $F(\underline{W})$.

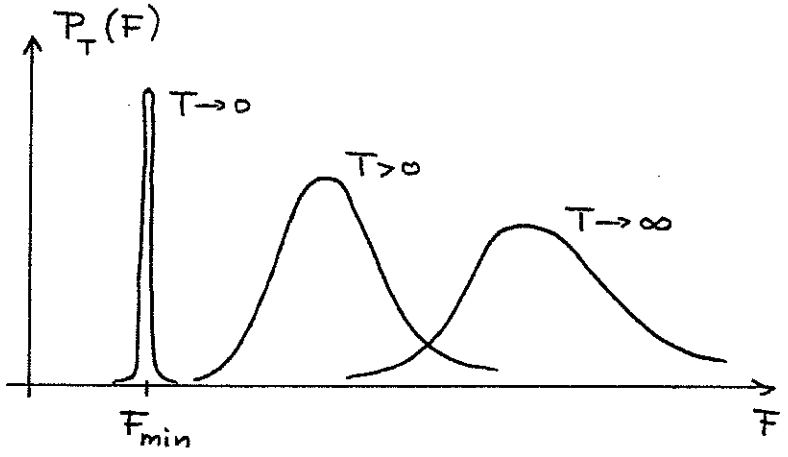
Statistische Mechanik:

→ Wahrsch.keit $P_T(\underline{W})$, das System bei einer Temperatur T im Zustand $\underline{W}$ zu finden:

$$P_T(\underline{W}) = \frac{1}{Z_T} e^{-F(\underline{W})/T} \quad , \quad Z_T = \sum_{\underline{W}} e^{-F(\underline{W})/T}$$

(Boltzmann-Verteilung) (Zustandssumme)

→



Mittlere "Energie": $\langle F \rangle (T) = - \frac{d \ln Z_T}{d(1/T)}$

Spezifische Wärme: $C(T) = \frac{d\langle F \rangle}{dT} = \frac{1}{T^2} [\langle F^2 \rangle - \langle F \rangle^2]$

Entropie: $S(T) = \frac{d}{dT} (T \ln Z_T)$

1) Das Annealing-Verfahren simuliert die Gleichgewichtsverteilung $P_T(\underline{W})$, falls es nur genügend lange bei einer konstanten Temperatur T iteriert wird. (Unter sehr allgemeinen Bedingungen an die zulässigen $\Delta\underline{W}$.)

2) Durch <u>langsames</u> Abkühlen verschiebt sich diese Verteilung dann zu immer tieferen F-Werten.

<u>Optimierung des Abkühlfahrplans ? :</u>

- Falls $T_k = \dfrac{A}{\log(1+k)}$, $k = 1, 2, \ldots\ldots$,

  so konvergiert der Simulated Annealing Algorithmus gegen das globale Minimum von F:

  $$\lim_{k\to\infty} F(\underline{W}_k) = F_{min} .$$

  [Mitra et al., Adv. Appl. Prob. <u>18</u>, 747 (1986)]

- Optimierung des Abkühlfahrplans für vorgegebenes $k_{max} < \infty$ (endliche CPU-Zeit !) ??

  → Anpassung an die "thermodynamischen Eigenschaften" des Problems !

  [v. Laarhoven and Aarts:
  Simulated Annealing – Theory and Applications
  Reidel Publishing (1987)]

  [Bernasconi: Optimization Problems and Statistical Mechanics
  in "Chaos and Complexity", World Scientific (1988)]

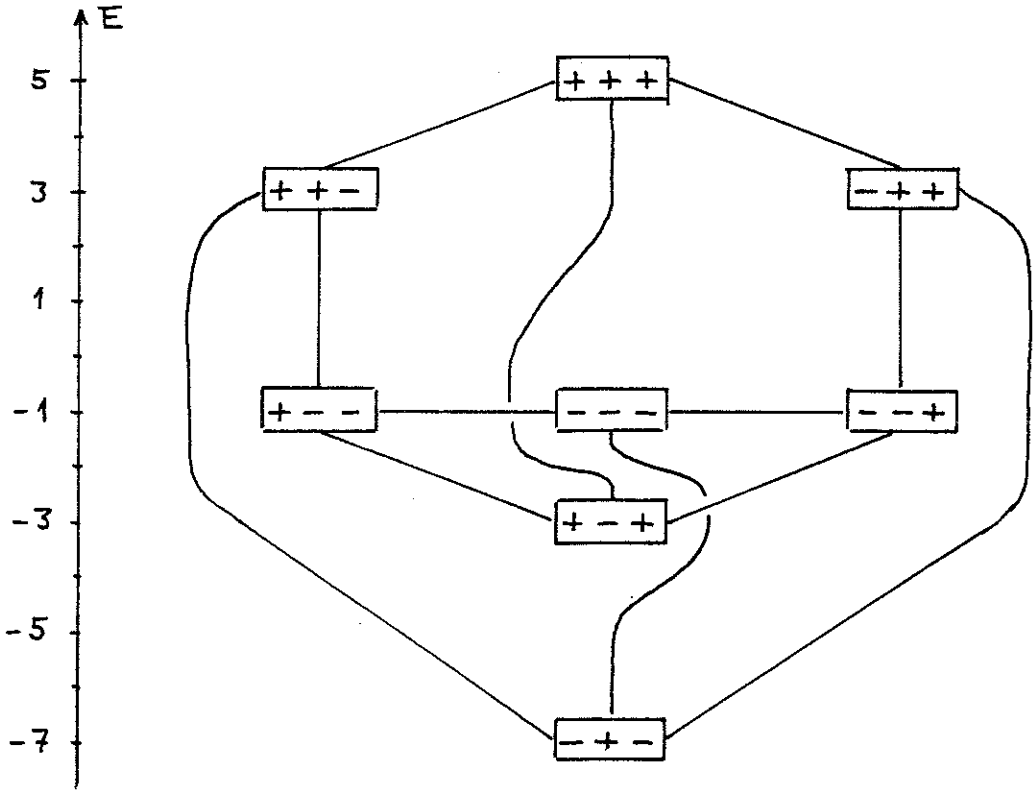
VIII.5 Beispiele für stochastische Optimierung

BEISPIEL 1:

$$E = W_1 + W_2 + W_3 + 2W_1W_2 - W_1W_3 + 2W_2W_3 - W_1W_2W_3 = \min$$

wobei W_1, W_2 und W_3 nur die Werte +1 oder -1 annehmen dürfen

Erlaubte zufällige Änderungen:
Vorzeichenwechsel eines einzelnen W_i

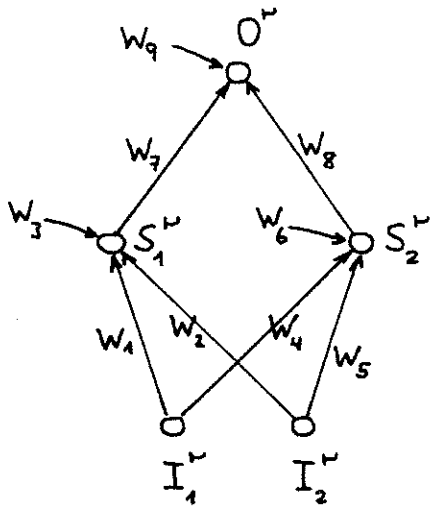


- Das "Iterative Improvement"-Verfahren kann im lokalen Minimum [+−+] steckenbleiben, da nur Übergänge zu Zuständen mit tieferem E zugelassen sind.

- Mit "Simulated Annealing" kann man aus diesem Minimum wieder herauskommen und in das globale Minimum [−+−] gelangen!

BEISPIEL 2:

XOR-Problem, 2 Hidden Units



$$O^\nu = f\left(W_7 S_1^\nu + W_8 S_2^\nu + W_9\right)$$

$$S_1^\nu = f\left(W_1 I_1^\nu + W_2 I_2^\nu + W_3\right)$$

$$S_2^\nu = f\left(W_4 I_1^\nu + W_5 I_2^\nu + W_6\right)$$

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

ν	I_1^ν	I_2^ν	D^ν (XOR)
1	1	1	−1
2	1	−1	1
3	−1	1	1
4	−1	−1	−1

$$E = \frac{1}{2} \sum_{\nu=1}^{4} \left(D^\nu - O^\nu\right)^2 = \min$$

Stochastische Optimierung:

→ Zufällige Änderungen:

$$W_i \longrightarrow W_i + \lambda \cdot RND \quad , \quad \lambda = 0.5$$

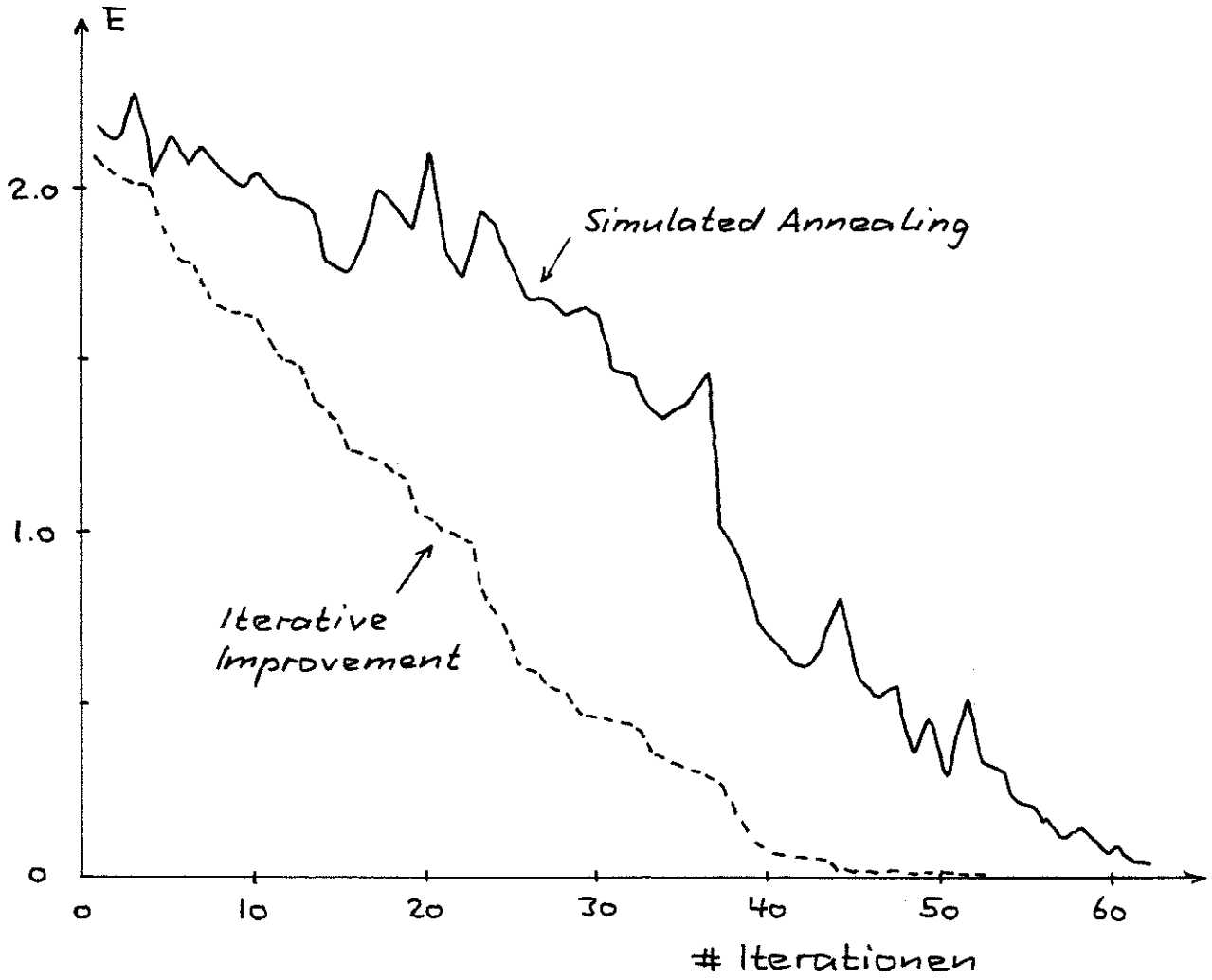
RND = random im Intervall (−1,+1)

[1 Iteration = alle Gewichte 1 mal zufällig geändert]

Resultate:

a) Wenn die Anfangsgewichte im Attraktionsgebiet des globalen Minimums liegen, konvergiert das "Iterative Improvement"-Verfahren schneller als der "Simulated Annealing"-Algorithmus.

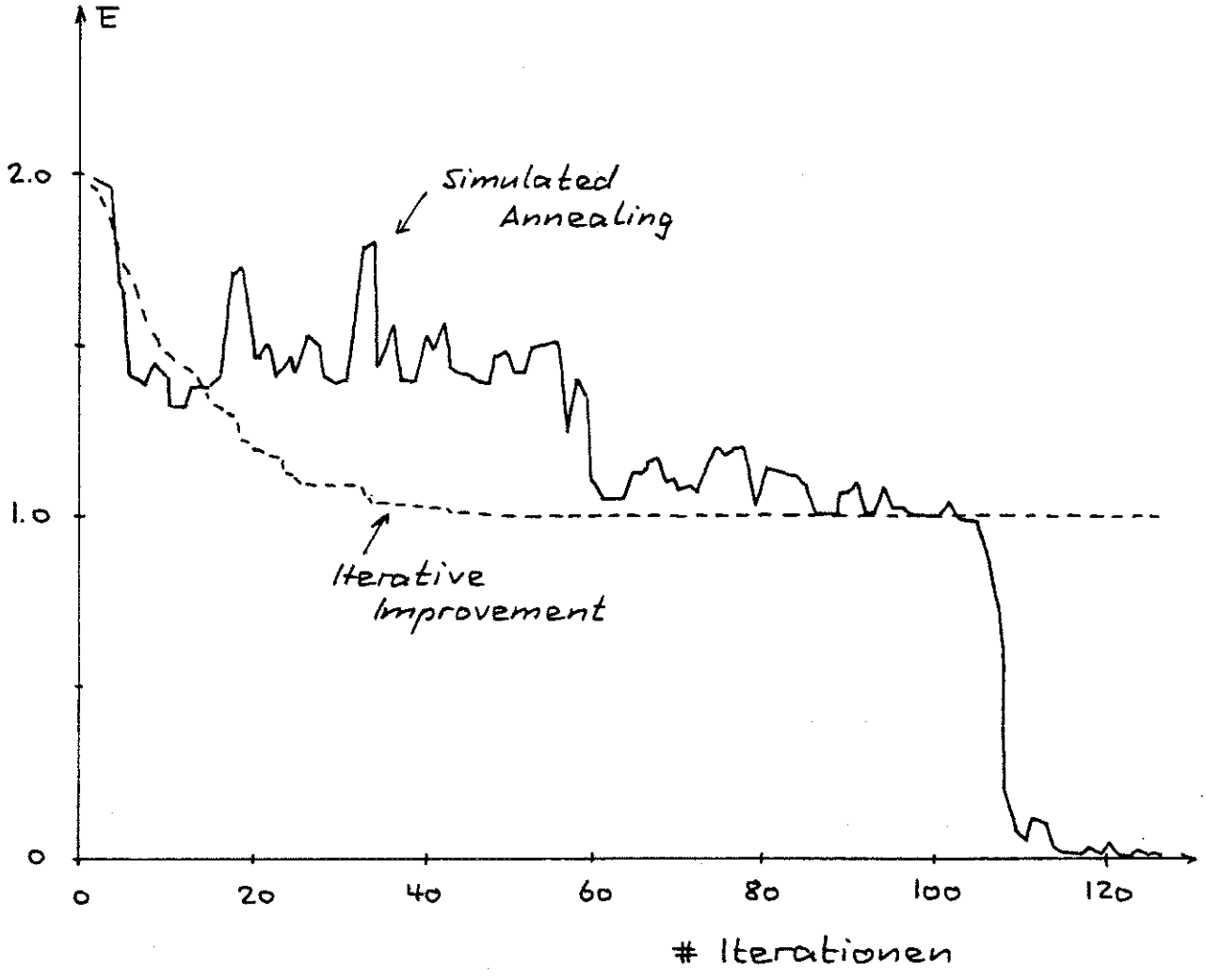
Beim "Iterative Improvement" nimmt E immer ab, während "Simulated Annealing" auch Übergänge zu Zuständen mit höherem E zulässt.



[XOR-Problem, 2 Hidden Units]

b) Oft bleibt das "Iterative Improvement" - Verfahren aber in einem schlechten lokalen Minimum stecken.

"Simulated Annealing" hingegen kann aus einem solchen Minimum wieder herauskommen. Nach genügend vielen Iterationen (theoretisch allerdings nur nach ∞ vielen) findet "Simulated Annealing" daher immer das globale Minimum.



[XOR-Problem, 2 Hidden Units]

VIII.6 Neuroevolution

Verwendung von genetischen Algorithmen (Evolutions-Strategien) zum Training von neuronalen Netzwerken.

Solche Lernverfahren operieren auf einer Population von verschiedenen neuronalen Netzwerken.

"Genotyp" eines neuronalen Netzwerks beschrieben durch Codierung der Gewichte oder/und der Netzwerk-Struktur:

(i) $\underline{X} = \underline{W}$ (Netzwerk-Struktur fest vorgegeben)

(ii) $\underline{X} = \underline{S}$ (Optimierung der Netzwerk-Struktur; Training der Gewichte z.B. mit Error-Backprop.)

(iii) $\underline{X} = \{\underline{S}, \underline{W}\}$ (Optimierung von Struktur und Gewichten)

Generation k: $\underline{X}_1^{(k)}, \underline{X}_2^{(k)}, \dots, \underline{X}_N^{(k)}$

↓

zufällige Mutationen: $\underline{X}_i \longrightarrow \underline{X}_i'$

Crossover: $\underline{X}_i, \underline{X}_j \longrightarrow \underline{X}_{(i,j)}$

Selektion: Nur $\underline{X}_i$ mit guter "Fitness" $F(\underline{X}_i)$ überleben

↓

Generation k+1: $\underline{X}_1^{(k+1)}, \underline{X}_2^{(k+1)}, \dots, \underline{X}_N^{(k+1)}$

→ Population entwickelt sich zu Individuen mit optimaler "Fitness" $F(\underline{X}_i)$

Beispiele:

- $F(\underline{X}_i)$ = Summe der quadr. Output-Fehler für Lernbeispiele
- $F(\underline{X}_i)$ = Aufprall-Geschwindigkeit (Mondlandung)
- $F(\underline{X}_i)$ = Zeit bis zum "Umfallen" (invertiertes Pendel)
- $F(\underline{X}_i)$ = Gewonnene Punkte bei Spielen

Literatur-Hinweise:

- G. Weiss, Towards the Synthesis of Neural and Evolutionary Learning in "Progress in Neural Networks" 5 (1997), 145-176
- M. Mandischer, Neurocomputing 42 (1-4), 2002, 87-117

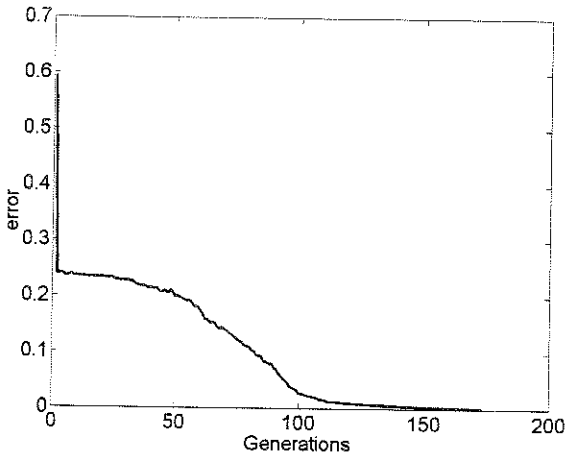
VIII.7 Beispiele für Neuroevolution

BEISPIEL 1: Symmetry-Detection

(Beat Gfeller, Semesterarbeit 2005)

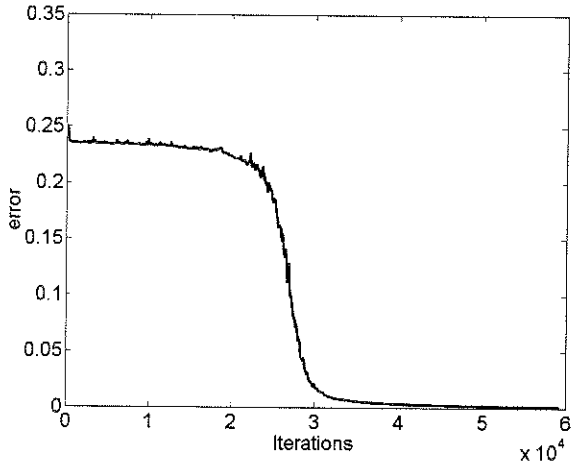
- 8-6-2-1 Multilayer-Perceptron
  → 71 Gewichte (inkl. Schwellen)
  Output = $\begin{cases} +1 & \text{falls binärer Input symmetrisch} \\ -1 & \text{sonst} \end{cases}$
- 15 Individuen / Generation
- 100 Netzwerk-Evaluationen / Generation

Evolutions-algorithmus:



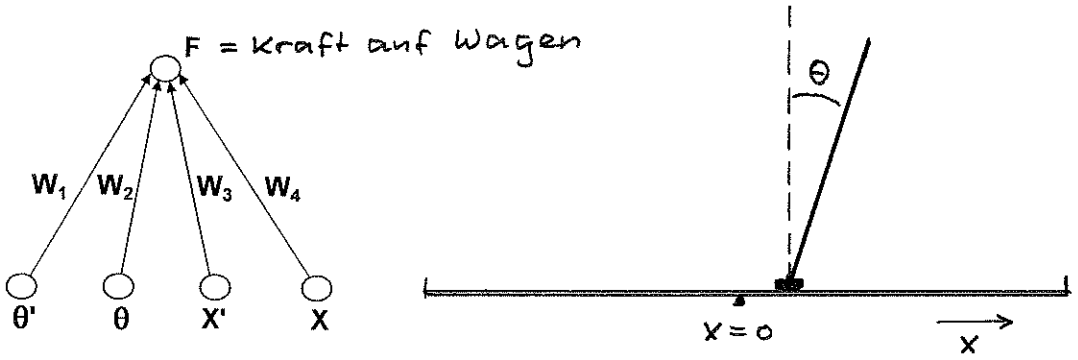
→ 17'000 Netzwerk-Evaluationen bis Fehler < 0.001

Error-Backpropagation:

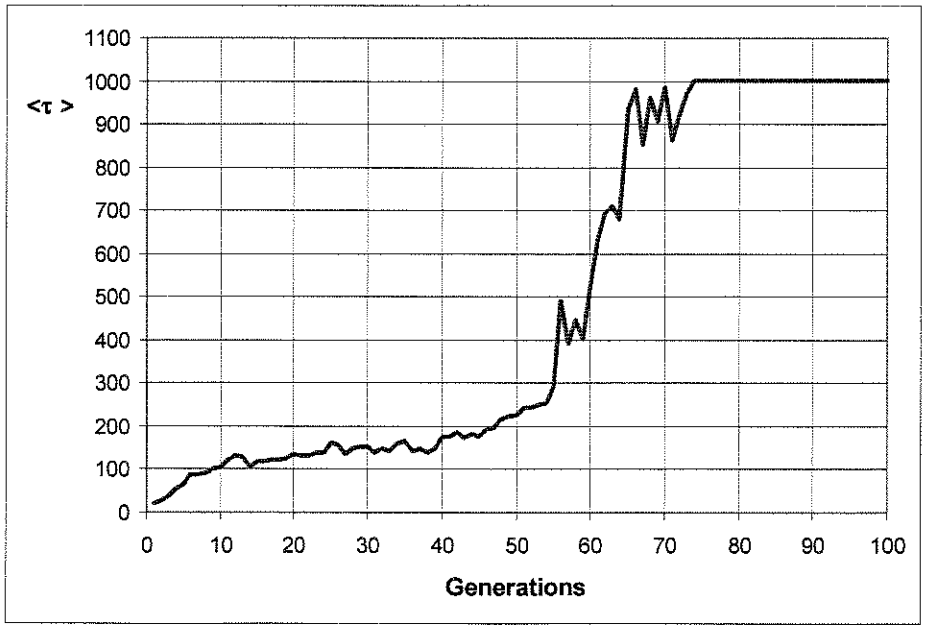


→ 60'000 Iterationen bis Fehler < 0.001

BEISPIEL 2 : Pole-Balancing



- Individuen = Perceptrons (4 Gewichte)
- 8 Individuen / Generation
- Fitness-Funktion = "Time to Failure"



<τ> = "Average Time to Failure"
für 4 beste Individuen / Generation

(τ = 1000 : → Ende der Episode)

→ ca. 70 Generationen
bis τ > 1000 sec