IX. Optimales Lernen und Verallgemeinern I

IX.1 Lernverhalten

- Lernen in neuronalen Netzwerken bezieht sich in den meisten Fällen auf einen Optimierungsprozess. Das Lernverhalten ist dann abhängig
  - von der Topographie der Fehlerlandschaft, auf welcher der Lernprozess stattfindet,
  - vom gewählten Lern- bzw. Optimierungsverfahren.

- Die Topographie der Fehlerlandschaft und die Effizienz einer Lernstrategie hängen von der Komplexität des Lernproblems ab, können aber durch verschiedene Implementierungsfaktoren beeinflusst werden:
  - Netzwerkgrösse, Netzwerkarchitektur
  - Darstellung der Input- und Outputdaten
  - Wahl der Aktivierungsfunktionen
  - Verteilung der Anfangsgewichte
  - Einstellung der Lernparameter
  - Präsentation (z.B. Reihenfolge) der Lernbeispiele
  - etc.

- Die Analyse des Lern- und Verallgemeinerungsverhaltens eines neuronalen Netzwerks ist daher ein sehr komplexes Problem:
  - Resultate für Netzwerke mit Hidden Units beziehen sich fast ausschliesslich auf empirische Studien.
  - Durch die optimale Wahl von Details bei der Implementation (Heuristiken, "Tricks") kann das Lern- und Verallgemeinerungsverhalten oft signifikant verbessert werden.

• Skalierung der Inputs

Bei realen Problemen haben die Inputdaten oft sehr unterschiedliche Wertebereiche, z.B.

Temperatur: $-20°C$ .... $+40°C$
Luftdruck: 900 mbar .... 1200 mbar

Wenn die Inputs alle gleich wichtig sind, müssten sich die entsprechenden Netzwerkgewichte dann in sehr verschiedenen Grössenordnungen bewegen, was zu einem schlechten Lernverhalten führen kann.

→ Regel für ein optimales Lernverhalten:

Die Inputs so skalieren, dass der Mittelwert über alle Lernbeispiele nahe bei Null liegt und dass ihr Wertebereich ungefähr gleich ist wie jener der Neuronenzustände (meistens also -1 bis +1).

• Darstellung der Input- und Outputdaten

- Lokale Darstellung:

  Jedes Konzept wird durch ein separates Neuron dargestellt:

  z.B. Buchstabe C: 0, 0, 1, 0, ..., 0

  [→ 26 Neuronen]

- Verteilte (codierte) Darstellung:

  z.B. Buchstabe C: 0, 0, 0, 1, 1
  (binäre Codierung)

  [→ 5 Neuronen]

→ *Codierte Darstellungen sind im allgemeinen effizienter, machen es aber schwierig, Strukturen klar darzustellen und weisen oft Unstetigkeiten in der Abbildungsfunktion auf (kleine Änderung am Eingang bewirkt grosse Änderung am Ausgang).*

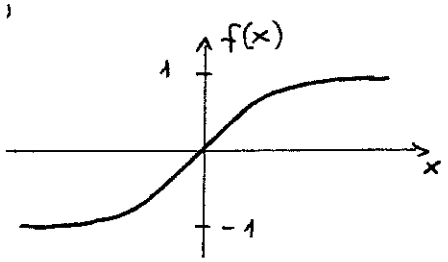
→ Bei der diskreten Darstellung treten weniger Unstetigkeiten auf. Zudem gibt die diskrete Darstellung eines Outputs Aufschluss über die Sicherheit der Netzwerkantwort:

- Wert des Outputs ($0 \leq O \leq 1$), oder Differenz zwischen grösstem und zweitgrösstem Output als Mass für die Zuverlässigkeit einer Zuordnung.
- Unklassifizierbare Inputs werden dadurch erkannt, dass alle Outputwerte klein sind.

- **Wahl der Aktivierungsfunktion**

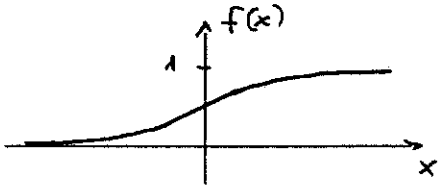
Aktivierungsfunktionen mit einem bezüglich Null symmetrischen Zielbereich,

z.B. $f(x) = \dfrac{1 - e^{-x}}{1 + e^{-x}}$



führen in den meisten Fällen zu einer schnelleren Konvergenz als solche, die nur positive Werte annehmen,

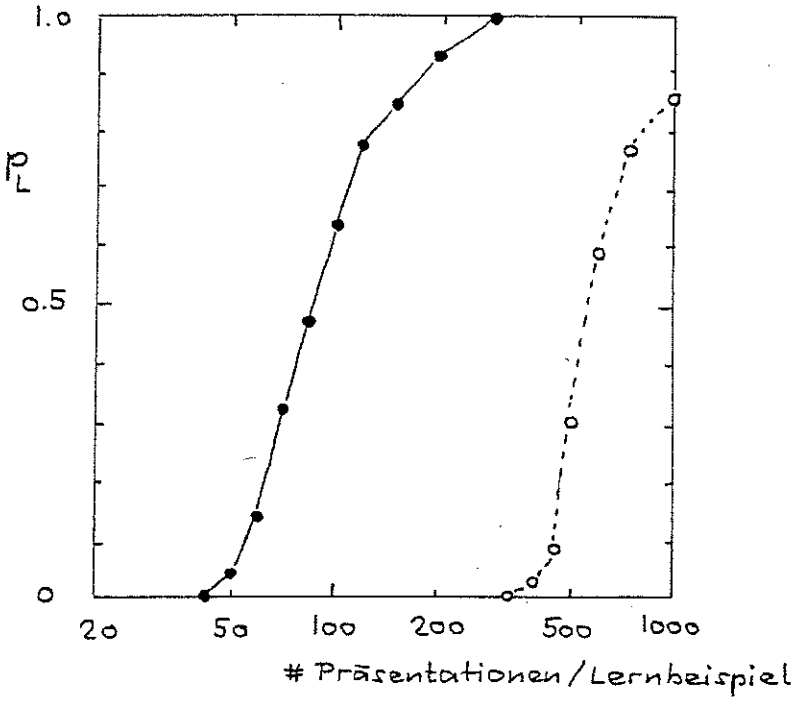
z.B. $f(x) = \dfrac{1}{1 + e^{-x}}$



Erklärung:

- Da $\Delta W_{ij} \propto S_j$ → $\Delta W_{ij} = 0$ falls $S_j = 0$
- Unterschiedliche Fehlerlandschaften

Beispiel: - XOR-Problem, 3 Hidden Units
- Error-Backpropagation mit Momentum Term
- η optimiert, $\alpha = 0.9$



p_L = Lernwahrscheinlichkeit

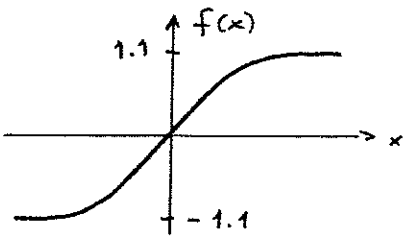
—●—● : $-1 \leq S_i \leq 1$ $\quad f(x) = \frac{1-e^{-x}}{1+e^{-x}}$

—o—o— : $0 \leq S_i \leq 1$ $\quad f(x) = \frac{1}{1+e^{-x}}$

Bemerkung:

Wenn die gewünschten Outputwerte -1 und +1 sind, ist es oft angebracht, die Aktivierungsfunktion so zu skalieren, dass sie einen etwas grösseren Zielbereich umfasst, z.B.

$f(x)$ für Outputneuron:



Damit wird vermieden, dass $\frac{df}{dx}$ gegen Null geht, wenn $0 = f(x)$ nahe bei 1 oder -1 ist, was sehr kleine Gradienten, d.h. sehr langsames Lernen zur Folge hätte.

- **Wahl der Anfangsgewichte**

Um ein optimales Lernverhalten zu erzielen, hat sich die folgende Faustregel bewährt:

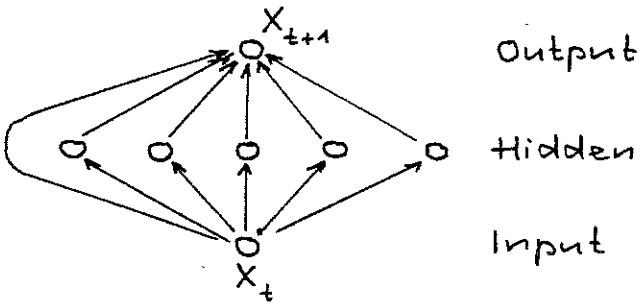
- Der Bereich $[-W_0, W_0]$, in dem die Gewichte W_{ij} eines Neurons i initialisiert werden, sollte so gewählt werden, dass

$$W_0 \approx \frac{2.4}{M}$$

wobei M = Anzahl Eingänge des Neurons i.

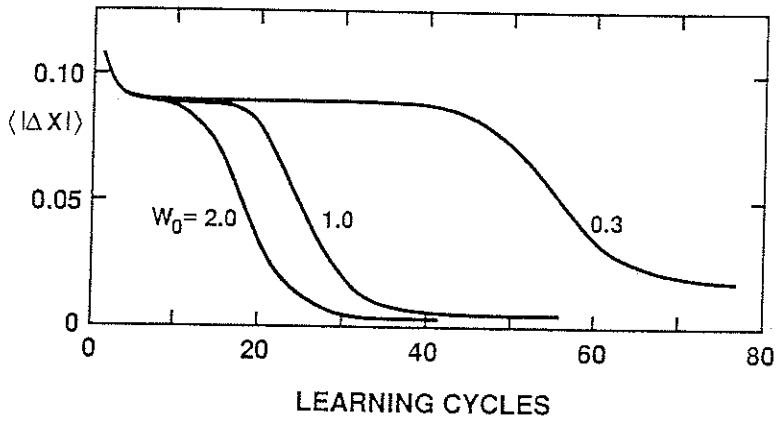
Beispiel: Vorhersage einer chaotischen Zeitreihe:

$$X_{t+1} = 4\lambda X_t (1 - X_t) \;, \quad 0 \le X_t \le 1 \;, \quad \lambda = 0.913$$



Error-Backpropagation mit Momentum Term

$$\eta = 0.05 \;, \quad \alpha = 0.9 \;, \quad f(x) = \frac{1}{1 + e^{-x}}$$



[$\langle |\Delta X| \rangle$ = mittl. absoluter Outputfehler]

• Präsentation der Lernbeispiele

- Konventionelle Lernstrategien:

  → Lernbeispiele zufällig ausgewählt, d.h. gleiche Präsentationshäufigkeit für alle Lernbeispiele.
  [→ UNIFORME STRATEGIEN]

- "Pädagogische" Lernstrategien:

  [Ch. Cachin, Semesterarbeit WS 1991/92]

  → Adaptive Erhöhung der Präsentationshäufigkeit für schlecht gelernte Beispiele.

  - Probabilistisch: Präsentationswahrsch.keit abhängig von Outputfehler
  - Deterministisch: Erzwungene Repetition schlecht gelernter Beispiele

- Analyse von Testproblemen:

  → Nur deterministische Strategien führen zu verbessertem Lernverhalten (Geschwindigkeit, Genauigkeit)!

z.B.
- FEHLERABHÄNGIGES WIEDERHOLEN (FAW)
- KARTEIKASTEN-SYSTEM (KKS)
- WIEDERHOLEN BIS GELERNT (WBG)

LERNGENAUIGKEIT NACH 10^4 BZW. 10^5 PRÄSENTATIONEN:

Strategie	Nachbarschaft-Probl.	Vorhersage-Probl.
UNIFORM	92.2 %	71.9 %
FAW	95.7 %	97.6 %
KKS	95.4 %	72.0 %
WBG	93.4 %	99.2 %

IX.2 VERALLGEMEINERUNGSVERHALTEN

Verallgemeinerung ist die Fähigkeit eines neuronalen Netzwerks, von gelernten Beispielen auf unbekannte Beispiele zu schliessen, d.h. einen generell gültigen Zusammenhang zwischen Input und Output zu erkennen.

Die Verallgemeinerungsfähigkeit wird meistens durch den Wert der Fehlerfunktion bzw. der Fehlerrate auf einem Testset gemessen. Im Testset dürfen nur unbekannte Inputmuster vorkommen, d.h. keine Beispiele aus dem Lernset.

- Einfluss von Netzarchitektur, Netzwerkgrösse und Lernsetgrösse

  - Bei zu kleiner Anzahl freier Parameter (Gewichte) oder bei einer zu einfachen Architektur kann das Netzwerk die gestellte Aufgabe nicht oder nur schlecht lösen.
  - Sind dagegen zu viele freie Parameter vorhanden, kann sich das Netzwerk sehr genau auf die Lernbeispiele spezialisieren. Dabei sinkt im allgemeinen jedoch die Verallgemeinerungsfähigkeit ("nur Auswendiglernen"!).

Da wir uns in praktischen Anwendungen vor allem für den Fehler auf dem Testset interessieren, gilt es, dasjenige Netzwerk zu finden, das für einen gegebenen Set von Lernbeispielen den kleinsten Fehler auf dem Testset aufweist.

[→ Optimale Verallgemeinerung]

Das Verallgemeinerungsverhalten eines neuronalen Netzwerks kann durch verschiedene Massnahmen oft beträchtlich gesteigert werden:

- Künstliche Vergrösserung des Lernsets, z.B. durch Verrauschen oder Verzerren der vorhandenen Lernbeispiele.

  [Verrauschte Lernbeispiele vermindern die Gefahr einer Spezialisierung auf besondere Eigenschaften der Lernbeispiele und führen deshalb oft zu einem signifikant besseren Verallgemeinerungsverhalten! ("Training with Noise")]

- Einbau von Vorwissen in die Darstellung der Lerndaten (Vorverarbeitung) oder in die Netzwerkstruktur (z.B. "Weight Sharing").

- Reduktion der freien Parameter durch Entfernung von einflussarmen Gewichten während des Lernprozesses (Pruning, Weight Decay).

- Vermeidung des Übertrainierens durch rechtzeitige Beendigung des Lernprozesses ("Stopped Training").

• Vorverarbeitungs-Techniken

Eine Vorverarbeitung der Inputdaten hat den Zweck, die Information in den Lernbeispielen so aufzubereiten, dass sie von einem neuronalen Netzwerk besser verarbeitet werden kann.

In den meisten Fällen will man damit auch die Anzahl der Inputs reduzieren, ohne wesentliche Information zu zerstören.

- Mit einer Vorverarbeitung kann die Verallgemeinerungsfähigkeit oft massiv gesteigert werden!

Mögliche Vorverarbeitungsmethoden reichen von einfachen Transformationen (z.B. Fouriertransformation) bis zu komplexen Signalverarbeitungstechniken. Welches die beste Technik ist, hängt von der Problemstellung ab.

Durch die Vorverarbeitung wird Wissen über die Problemstellung in das System eingebaut:

Beispiele:

- "Weight Sharing":

  Verbindungen zu Hidden Units, welche für verschiedene Orte im Inputmuster die gleiche Aufgabe lösen sollen (z.B. Kantendetektion bei der Mustererkennung), haben identische Gewichte.

  [Hertz, Krogh, and Palmer, p. 140]

- Optimale Wahl der Netzarchitektur:

  z.B. "Clumps-Problem"

  [Levin, Tishby, and Solla, Proc. IEEE, Vol. 78, No. 10, 1990, pp. 1568-1574]

- Erweiterung des Inputraums:

  z.B. durch polynomiale Entwicklung oder durch Fouriertransformation:

  Ursprünglicher Inputraum: X_i, $i = 1, \ldots, n$

  Erweiterter Inputraum: X_i, $X_i X_j$, $X_i X_j X_k$, ...

  oder X_i, $\sin(\pi X_i)$, $\cos(\pi X_i)$, $\sin(2\pi X_i)$, $\cos(2\pi X_i)$, ...

  → Dadurch können nichtlineare Trennflächen auch mit einem Perceptron realisiert werden.

  → Schnelleres Lernen!

  → "FUNCTIONAL LINK NETWORKS"
  [Y.-H. Pao, AI Expert (April 1989), pp. 60-68]

- "Pruning"-Techniken und "Weight Decay"

→ Elimination von Verbindungen mit kleinem Einfluss auf den Outputwert.

→ Verkleinerung der Anzahl freier Parameter, und damit bessere Verallgemeinerung.

Pruning:

1) Mit einem grossen Netzwerk starten und dieses mit den Lerndaten trainieren.

2) Verbindungen mit kleinstem Einfluss eliminieren (z.B. Verbindungen mit den kleinsten Gewichten).

3) Reduziertes Netzwerk nachtrainieren.

[Schritte 2 und 3 ev. mehrmals wiederholen.]

Weight Decay:

Zusatzterm in der Fehlerfunktion:

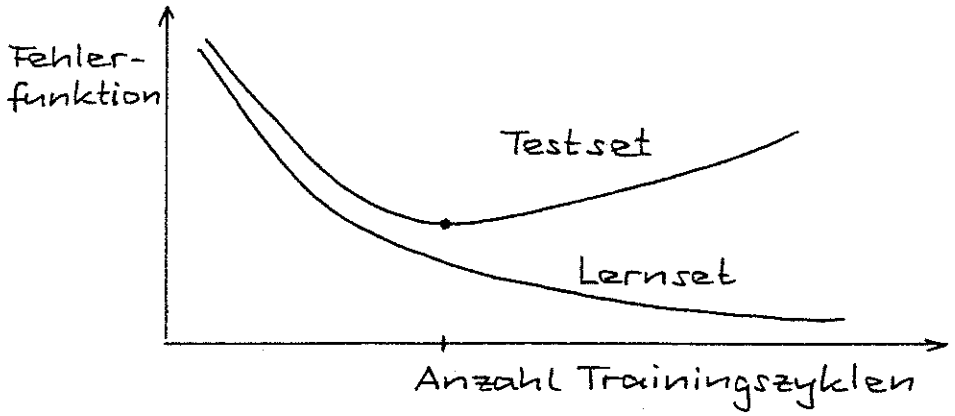
z.B. $$F = \sum_{\mu} (D^{\mu} - O^{\mu})^2 + \lambda \sum_{(ij)} W_{ij}^2 = \min$$

→ Unwichtige Gewichte (d.h. Gewichte, die durch Lernbeispiele nicht ständig verstärkt werden) konvergieren automatisch gegen Null und werden so eliminiert.

[Hertz, Krogh, and Palmer, pp. 157-158]

- **Übertrainieren, "Stopped Training"**

Während des Trainings kann es vorkommen, dass der Fehler auf dem Testset nach Erreichen eines Minimums wieder ansteigt:



→ Zu langes Lernen kann der Verallgemeinerungsfähigkeit schaden!

Begründung:

Gewichte, die eine wichtige Information beinhalten, haben grosse Fehlergradienten (eine kleine Gewichtsänderung bewirkt einen grossen Fehler).

Die "wichtigen" Gewichte werden deshalb zuerst trainiert und bringen das Netzwerk in einen Zustand guter Verallgemeinerungsfähigkeit. Danach werden die "unwichtigen" Gewichte weitertrainiert und bewirken, dass sich das Netzwerk immer mehr auf das vorgegebene Lernset spezialisiert, was meistens mit einem Verlust an Verallgemeinerungsfähigkeit verbunden ist.

→ Um den Punkt optimaler Verallgemeinerung zu detektieren, wird deshalb oft nach jedem Lernzyklus der Fehler auf einem Testset gemessen.
[→ "STOPPED TRAINING"]

• Crossvalidation

In praktischen Anwendungen ist die Anzahl der Lerndaten oft sehr beschränkt. Um einen effizienteren Nutzen aus den zur Verfügung stehenden Daten zu ziehen als bei einer festen Aufteilung in ein Lernset und ein Testset, werden deshalb oft sogenannte CROSSVALIDATION-Techniken angewandt:

- Aufteilung der Trainingsdaten in k Subsets.
- Jedes Subset wird einmal als Testset verwendet, wobei das Netz jeweils mit den Daten aus den verbleibenden k-1 Subsets trainiert wird.
- Die quadratischen Fehler auf den k Testsets werden gemittelt (→ CROSSVALIDATION-FEHLER).

→ *Der Crossvalidation-Fehler liefert eine gute Abschätzung des Verallgemeinerungsfehlers!*

Crossvalidation kann zur (empirischen) Bestimmung einer optimalen Netzwerkarchitektur (z.B. Anzahl der Hidden Units) verwendet werden:

- Das Crossvalidation-Verfahren wird für eine Reihe von Netzarchitekturen durchgeführt, und die optimale Architektur ist dann diejenige mit dem kleinsten Crossvalidation-Fehler.

  [siehe folgendes Beispiel]

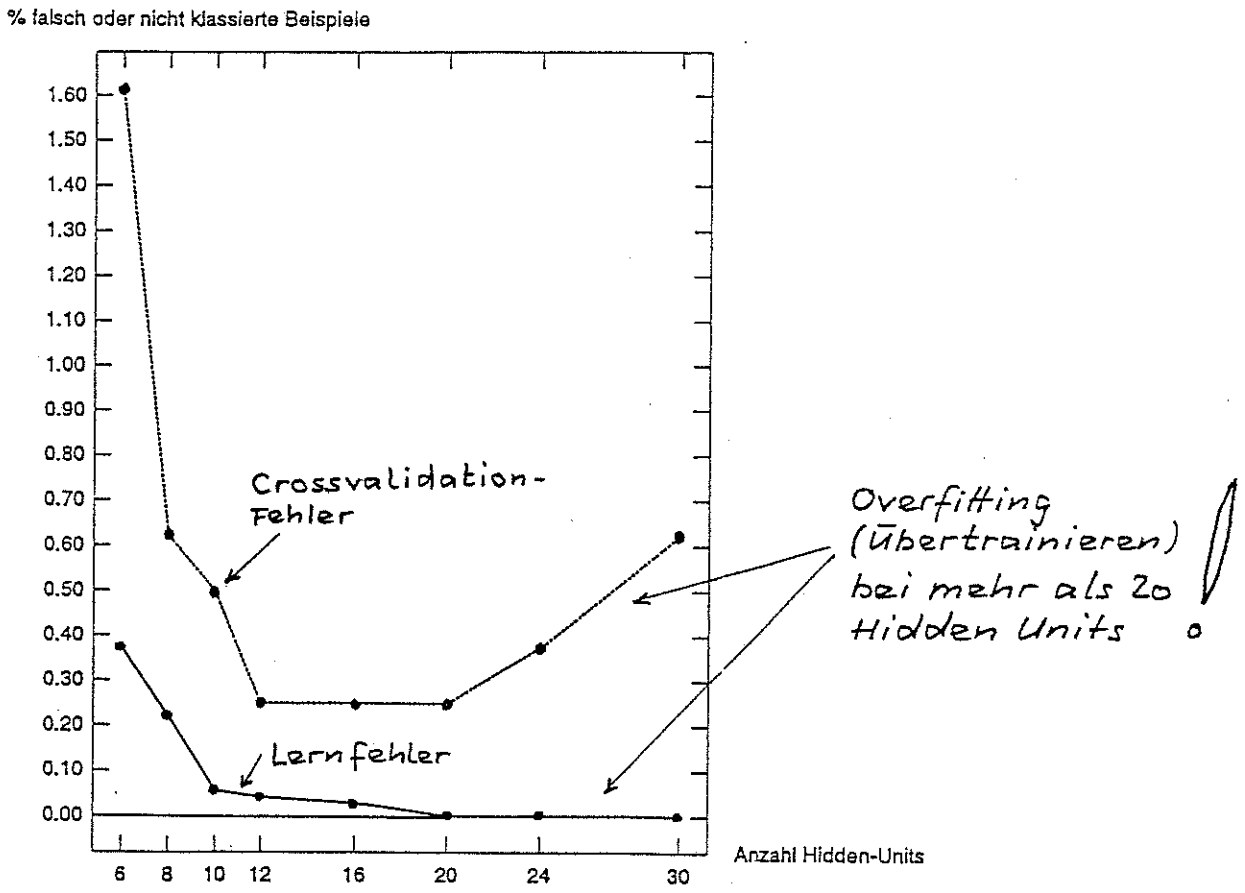
- Beispiel:

KLASSIFIZIERUNG VON ISOLATIONSDEFEKTEN AUFGRUND VON GEMESSENEN TEILENTLADUNGSMUSTERN

(Diplomarbeit Ch. Cachin, WS 1992/93)

→ Feedforward-Netzwerk mit 24 Inputs, 8 Outputs und einer Schicht von Hidden Units

→ Abhängigkeit der Lern- und Verallgemeinerungseigenschaften von der Anzahl Hidden Units:



→ Optimale Anzahl Hidden Units: 12 bis 20