MASCHINELLES LERNEN UND BIO-INSPIRIERTE OPTIMIERUNG

VORLÄUFIGE, UNVOLLSTÄNDIGE
BEILAGE ZU TEILEN DER VORLESUNG
WS-2004/2005 und SS 2005
(Zusätzliches jüngeres Material
in der WWW-Seite zur Vorlesung)

DOZENT:
Prof. Jürgen Schmidhuber
Fakultät für Informatik, TUM, Germany
& IDSIA, Switzerland

Kapitel 1

ÜBERBLICK

- GEBIET: Informatik VI, prüfbare Vorlesung
- HÖRERKREIS: nach DVP, Ausnahmen möglich
- VORAUSSETZUNGEN: Grundkenntnisse in Analysis, Lineare Algebra und Statistik (im Vordiplomwissen enthalten, werden aber nochmals kurz aufgefrischt).
- GEEIGNET für alle, die Fopra, Hauptseminar, Diplomprüfung oder Diplomarbeit im Gebiet der adaptiven Robotik machen wollen.

WARNUNG: *Dieses Skript bietet Ihnen lediglich ein skelettartiges Gerüst einiger weniger Kapitel des Vorlesungsstoffes, mit dem Sie Ihre eigene, hoffentlich detailliertere Mitschrift vergleichen können. Insbesondere fehlen Ausführungen zu den Bereichen: Support Vector Machines, Bayes Networks, Universal Learners, Optimal Search. Das zugehörige sowie vielfältiges weiteres Material ist auf der Kurswebsite zu finden!*

Bitte überprüfen Sie selbst noch einmal alle mathematischen Herleitungen. Der Dozent freut sich über jeden Fehler, den Sie entdecken. Leider gibt es noch kein Buch (und schon gar kein deutsches Buch), welches all die wichtigen Themen anspricht, die in der Vorlesung behandelt werden.

1.1 INHALT

Einige der wichtigsten Lernalgorithmen aus den Bereichen überwachtes Lernen, unüberwachtes Lernen, und "Reinforcement-Lernen" werden a) theoretisch begründet und b) in Anwendungen vorgestellt (vorzugsweise in Anwendungen, bei denen sie zu besseren Ergebnissen führen als konventionelle Verfahren).

- Einführung zur Bayes Induktion (separates Material)
- Universelle Lernmaschinen (unberechenbar, aber optimal) (separates Material)
- Asymptotisch optimale Suchalgorithmen (separates Material)
- Support Vector Machines (separates Material)
- Gradientenbasierte Verfahren für

  a) "Feedforward"-Netze (Backprop). Anwendungen: Mustererkennung, Zeitreihenvorhersage, überwachte Robotersteuerung u.v.m.

  b) Rekurrente Netze mit zeitlich variierenden Eingaben. Anwendungen: Spracherkennung, Grammatik-Lernen, alle Probleme mit teilweise sequentieller Lösungsstruktur.

  c) Kompositionen mehrerer Netze mit unterschiedlichen Aufgaben (Weltmodellbauer etc.). Anwendungen: Robotersteuerung etc..

- Generalisierungsfähigkeit – was ist das?

  a) Fundamentale Schranken des Lernbaren

  b) Ockhams Rasiermesser: bevorzuge einfache Datenmodelle.

  c) Verfahren zur Bestrafung überkomplexer Datenmodelle. Anwendungen: überall

  d) Der Bayes-Standpunkt

- Konzepte für "Reinforcement"-Lernen

  a) "Temporal Difference" Algorithmen. Hauptanwendung: Vorhersage in Markovumgebungen, Anwendungsvariante: Brettspiele wie z.B. Backgammon.

  b) Q-Lernen. Anwendungen: Agenten- und Robotersteuerung.

  c) Erweiterungen: Weltmodelle.

  d) Bezug zur dynamischen Programmierung

- Algorithmen für unüberwachtes Lernen

  a) Einführung in relevante Konzepte der Informationstheorie: Entropie, Kullback-Leiber Distanz, wechselseitige Information.

  b) Biologisch plausible Algorithmen für automatische Dekorrelation und Maximierung des Informationsflusses.

  c) Informationstheorie und topologieerhaltende Abbildungen. Anwendung: Optimierungsprobleme

- Evolutionäre Lernstrategien

  a) Genetische Algorithmen

  b) "Bucket Brigade"

  c) "Artificial Life"

1.2 TYPISCHE KÜNSTL. NEURONALE NETZE (KNN)

- Immer noch bestes Buch: [9]
- KNN: Paar (V, E): Menge von Knoten $V = \{1, 2, ..., n_V\}$ ("Neuronen"). $E \subseteq V \times V$ gerichtete Kanten, Verbindungen. Gewicht ("Synapse") von $(i, j) \in E$ ist $w_{ji} \in \mathbf{R}$. Biologieanalogie!
- Lagen: 1. Lage: Menge aller Eingabeknoten. n-te Lage: Menge aller Knoten, zu denen ausgehend von der ersten Lage mindestens ein Kantenpfad der Länge $n-1$ führt, aber kein Kantenpfad der Länge $m \geq n$. Dabei ist Kantenpfad der Länge $n-1$ Liste:

  $((k_1, k_2), (k_2, k_3), \ldots, (k_{n-1}, k_n))$ mit $k_i \in V, i = 1, \ldots, n$ und $(k_i, k_{i+1}) \in E, i = 1, \ldots, n-1$.

- Vorwärtsgerichtete Netze: Keine Zyklen.
- Typisch: Knoten beliebig durchnumeriert. Aktivation des k-ten Knotens in Antwort auf einen an der Eingabelage anliegenden Eingabevektor x (mit i-ter Komponente x_i) sei o_k, wobei $o_i = x_i$, falls i Eingabeknoten ist. In der r-ten Lage ($r > 1$):

$$net_k = \sum_{l \in Lagen < r} w_{kl} o_l, \quad o_k = f(net_k), \tag{1.1}$$

  mit f Aktivierungsfunktion, z.B.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

  oder

$$f(x) = 1 \ \ falls \ \ x > 0.5, \ \ f(x) = 0 \ \ sonst. \tag{1.3}$$

1.3 GRADIENTENBASIERTE LERNALGORITHMEN

- Zielfunktion: z.B. durch einzelnes Muster x verursachter Fehler J:

$$J = \frac{1}{2} \sum_{i \ in \ Ausgabelage} (o_i - d_i)^2$$

  wobei d_i gewünschte Ausgabe des i-ten Ausgabeknotens.

- Minimiere Fehlerfunktion! Gradientenabstieg. Ein Iterationsschritt: für alle Gewichte w_{ij}:

$$w_{ij} = w_{ij} + \triangle w_{ij} = w_{ij} - \alpha \frac{\partial J}{\partial w_{ij}}$$

  wobei $\alpha > 0$ Lernrate.

- Zur Gradientenberechnung: Kettenregel: Gegeben diffbare Funktion $f : R^n \to R$

$$f(g_1(o_1, ..., o_m), g_2(o_1, ..., o_m), \ldots, g_n(o_1, ...o_m)) \tag{1.4}$$

  wobei alle $g_i : R^m \to R$ mit $i = 1 \ldots n$ ihrerseits nach allen o_i ableitbar sind. Dann gilt

$$\frac{\partial f(g_1(.), g_2(.), \ldots, g_n(.))}{\partial o_i} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial o_i}. \tag{1.5}$$

1.4 BEISPIEL: BACKPROP (BP)

Gebräuchlichster Lernalg. f. überw. Lernen. Unabhängig entwickelt von [74][30][42][53].

- Gradient:

$$-\frac{\partial J}{\partial w_{ij}} = -\frac{\partial J}{\partial o_i} \frac{\partial o_i}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} = \delta_i o_j \tag{1.6}$$

  mit

$$\delta_i = -\frac{\partial J}{\partial net_i} = -\frac{\partial J}{\partial o_i} f'(net_i). \tag{1.7}$$

- 1. Faktor berechnen: falls i Ausgabeknoten: bei J wie oben trivial: $(o_i - d_i)$. Falls i kein Ausgabeknoten und in Lage r: Kettenregel! 1. Faktor:

$$\frac{\partial J}{\partial o_i} = \sum_{k \ in \ Lagen > r} \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial o_i} = - \sum_{k \ in \ Lagen > r} \delta_k w_{ki}.$$

- Also **Fehlerpropagierungsalgorithmus** symmetrisch! Vorwärts: Input anlegen, dann lagenweise

$$net_k = \sum_{l \in \ Lagen < r} w_{kl} f(net_l), \tag{1.8}$$

  Rückwärts: Fehler $(d_i - o_i) f'(net_i)$ an den Ausgabeknoten anlegen, dann lagenweise

$$\delta_i = f'(net_i) \sum_{k \in \ Lagen > r} w_{ki} \delta_k.$$

  Schließlich Gewichtsänderungen f. alle w_{ij}:

$$w_{ij} = w_{ij} + \alpha \delta_i o_j$$

  Falls dabei $f(x) = \frac{1}{1+e^{-x}}$, dann $f'(x) = f(x)(1 - f(x))$.

- Normalerweise mehr als ein Eingabemuster: der Gradient der Summe ist die Summe der Gradienten.
- *on-line* (Gewichtsänderung nach jeder Musterpräsentation) versus *off-line* (Gewichtsänderung erst nach Präsentation aller Muster).
- Gradientenabstiegsbeschleunigungsverfahren. Z.B. Momentum, konjugierte Gradientenverfahren, Methoden 2. Ordnung, effiziente Approximationen von Methoden zweiter Ordnung [12][55][43][65].
- Allgemeinere, probabilistische Interpretation: d_j ist Zufallsvariable. Minimierung von quadratischer Fehlersumme (QFS) dann erreicht, wenn o_j stets gleich dem bedingten Erwartungswert $E(d_j \mid x)$ ist. Z.B. Lokale Repräsentation: $dim(d)$ mögl. gewünschte binäre Ausgabevektoren (der p-te heißt d^p) haben nur je eine Nichtnull-Komponente, nämlich $d_p^p = 1$. Dann führt Minimierung von QFS zur Schätzung der bedingten Wahrscheinlichkeit $P(gewünschte\ Ausgabe = d^p \mid x)$ in o_p.
- Andere Zielfunktionen. Alles, was diffbar ist und Sinn macht!
- Einschränkungen für die Gewichte.

1.5 ANWENDUNGEN

- Trivialbeispiele: XOR etc.
- frühes Beispiel: NETTALK
- Proteinstrukturvorhersage [13][67].
- Mustererkennung (z.B. handgeschriebene Ziffern) [31][71]).
- Aktienkursvorhersage, Zeitreihenvorhersage (e.g. [73]).
- überwachte Motoriksteuerung [52], überwachtes Backgammon-Lernen [70] (Gewinn der Backgammon Computer Olympiade).
- Datenkompression: a) Autoassoziator. b) Textkompression (probabilistische Interpretation).
- **HAUSAUFGABE:** Implementiere BP und teste es.

Kapitel 2

REKURRENTE NETZE

- Rekurrentes Netz: Paar (V, E): V Knoten, Akt.fn. wie früher. $E \subseteq V \times V$ Verbindungen, jetzt aber u.U. zyklisch. Gewicht von (i, j) ist wieder w_{ji}.
- 2 Arten: a) Stationäre Eingaben – Equilibrium [25] [17] [63] [1] [46]. Zu uninteressant, wir machen gleich b) Zeitveränderliche Eingaben.
- Warum? Beliebige (teilweise) sequentielle Algorithmen statt bloße Musterassoziationen!
- Mächtigkeit: Wie herkömmlicher Rechner (wie Turingmaschine mit begrenztem Speicher). Warum? Herkömmlicher Rechner ist auf rekurrentem Netz nachbaubar, da UND- und ODER- Schaltungen etc. nachbaubar.
- Um Indizes zu sparen: betrachte *einzelne*, geordnete Sequenz von Eingabevektoren mit s diskreten Zeitschritten $0 \leq t \leq s$: Knoten eindeutig durchnumeriert. U: Menge der Indizes k: $x_k(t)$ ist Ausgabe des Nichteingabeknotens mit Nummer k zum Zeitpunkt t. I: Menge der Indizes k: $x_k(t)$ ist Aktivation des Eingabeknotens mit Nummer k zum Zeitpunkt t.
- Ablaufdynamik: Sequentiell Eingabevektoren anlegen, dabei für $k \in U$:

$$net_k(0) = 0, \;\; \forall t \geq 0 : x_k(t) = f_k(net_k(t)),$$

$$\forall t \geq 0 : \;\; net_k(t+1) = \sum_{l \in U \cup I} w_{kl} x_l(t). \tag{2.1}$$

  f_k diffbare Aktivierungsfunktion, wie früher.
- $T(t)$: Menge von Indizes $k \in U$, für die ein von einem externen Lehrer definierter Zielwert $d_k(t)$ zum Zeitpunkt t existiert.
- Zielfunktion: Definiere

$$e_k(t) = d_k(t) - x_k(t), \;\; falls \; k \in T(t),$$

$$e_k(t) = 0 \;\; sonst,$$

$$E(t) = \frac{1}{2} \sum_{k \in U} (e_k(t))^2, \;\; E^{total}(t', t) = \sum_{\tau = t'+1}^{t} E(\tau). \tag{2.2}$$

  Minimiere:

$$E^{total}(0, s) \tag{2.3}$$

  Erfordert i.a. die zeitweise Speicherung von vergangenen Ereignissen! *"Temporal credit assignment problem"*.

2.1 BACKPROP THROUGH TIME (BPTT)

Siehe z.B. [75], auch [37].

- Entfalte den dyn. Ablauf des rekurrenten Netzes im Raum → Zurückführung auf Vorwärtsnetz.
- **Algorithmus:**

$$\delta_i(\tau) = -\frac{\partial E^{total}(0,s)}{\partial net_i(\tau)}.$$

$\delta_i(\tau)$ kann für alle $i \in U, 0 \leq \tau \leq s$ in einem einzigen Pass berechnet werden:

$$\delta_i(\tau) = f_i'(net_i(\tau))e_i(\tau) \;\; falls \;\; \tau = s$$

$$\delta_i(\tau) = f_i'(net_i(\tau))(e_i(\tau) + \sum_{l \in U} w_{li}\delta_l(\tau+1)) \;\; falls \;\; 1 \leq \tau < s.$$

Endgültige Gewichtsänderungen:

$$\triangle w_{ij}(s) = -\alpha \frac{\partial E^{total}(0,s)}{\partial w_{ij}} = \alpha \sum_{\tau=1}^{s} \delta_i(\tau) x_j(\tau - 1).$$

- Zeitkomplexität (Anzahl der Additionen und Multiplikationen): $O(sn^2)$. Speicherkomplexität: $O(sn)$.
- Unexakte, aber praktisch brauchbare Variante: *"Abgeschnittenes" BPTT* [75]. Nur k Zeitschritte zurück in die Vggh. propagieren!

2.2 SENSITIVITÄTSANALYSE (RTRL-ALGORITHMUS)

Siehe [51] [76] [44] [39].

- Einfacher Algorithmus mit konstantem Speicher. Gleicher Gradient wird anders (ohne Rückpropagieren) berechnet.
- Gesucht wieder

$$\triangle w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}} = -\alpha \sum_{k \in T(t)} (x_k(t) - d_k(t)) \frac{\partial x_k(t)}{\partial w_{ij}}.$$

- Kettenregel: für alle $k \in U, t > 1$:

$$\frac{\partial x_k(t+1)}{\partial w_{ij}} = f_k'(net_k(t+1)) \left[\sum_{l \in U} w_{kl} \frac{\partial x_l(t)}{\partial w_{ij}} + \delta_{ik} x_j(t) \right].$$

δ_{ik} bezeichnet hier das Kroneckersche Delta und ist 1 für $i = k$ und 0 sonst.

- Also **Kern des ALGORITHMUS:**

$$\forall i,j,k: \; p^k_{ij_{new}} \leftarrow f_k'(net_k(t)) \left[\sum_{l \in U} w_{kl} p^l_{ij_{old}} + \delta_{ik} x_j(t) \right] \; ; \; \forall i,j,k: \; p^k_{ij_{old}} \leftarrow p^k_{ij_{new}}$$

Jetzt noch Gewichtsänderung wie oben.

- *Komplexität.* Speicher: $O(n^3)$, Berechnungskomplexität pro Zeitschritt: $O(n^4)$.
- Es gibt einen Alg. von Schmidhuber [57] mit Speicher: $O(n^3)$, Zeit: $O(n^3)$.
- Alternative: "Schnelle" Gewichte statt rekurrente Verbindungen: [60].

2.3 GROSSES PROBLEM UND SEINE LÖSUNG

Schwindender Fehlerfluss (separates Material) [19, 20]. Lösung: Long Short-Term Memory LSTM [22, 15].

2.4 TRADITIONELLE RNN: ANWENDUNGEN

- Trivialbeispiele: Flipflop etc. [76]
- Spracherkennung
- Zeitreihenvorhersage
- Grammatik-Lernen [16]
- Simulation echter biologischer Netzwerke [77]
- Probleme mit langen Zeitlücken [58]!
- **HAUSAUFGABE:** Implementiere RTRL und teste es.

2.5 LSTM RNN: ANWENDUNGEN

Separates Material, von Spracherkennung bis zur Musikkomposition und Robotersteuerung (1997-2004): http://www.idsia.ch/juergen/rnn.html

Kapitel 3

MODELLBAUER

3.1 PROBLEMSTELLUNG

- *"Reinforcement"*-Lernen (RL). Beispiel: KNN steuert Agent in Umgebung. Gewünschte Netzausgaben sind *nicht* von vornherein bekannt: Evaluierungsfunktion bewertet vom Agenten verursachte Umgebungszustände: "gut", "schlecht"; "Ziel erreicht", "Ziel verfehlt"; "Lust", "Schmerz". I.a. skalares zeitabh. Bewertungssignal.
- Ziel: Finde "gute" Steuersignale zum richtigen Zeitpunkt.
- Hoffnung: Abbildung von Netzwerkausgaben auf Performanzmaß ist diffbar [74] [26].
- *Steuernetzwerk* C (BP-Netzwerk) beantwortet p-ten Zustandsvektor x^p aus der Umgebung mit "Steuervektor" a^p. Umgebung berechnet aus x^p und einem a^s mittels Evaluierungsfunktion $eval$ ein Resultat (z.B. neuen Zustand, *Reinforcement*-Wert, oder beides)

$$u^{p,s} = eval(x^p, a^s).$$

- *erwünschte* Zustandsvektoren d^p. Ziel: Minimiere

$$E_C = \sum_p \frac{1}{2} \sum_i (e_i^p)^2,$$

mit

$$e_i^p = d_i^p - u_i^{p,p}.$$

Problem: Umgebungsdynamik unbekannt. Lehrer weiß auch nicht, wie's geht.

- Lösung: Lerne erst *Weltmodell* M (zweites BP-Netzwerk). M bildet $x^p \circ a^s$ auf "Prediktionsvektor" $y^{p,s}$ ab. Soll zu erwartenden geänderten Umgebungszustand prophezeihen. Für M zweites Hilfs-Performanzmaß :

$$E_M = \sum_{p,s} \frac{1}{2} \sum_i (y_i^{p,s} - u_i^{p,s})^2.$$

- Nachdem M trainiert: Friere M's Gewichte ein. M dient als Approximation von $eval$ zur Berechnung von Fehlergradienten für C. Sei c_j^p die Aktivation des j-ten Knoten in C bei Eingabe x^p. Jedes Gewicht w_{ij} in C wird nun gemäß

$$\triangle w_{ij} \sim -\frac{\partial \sum_p \frac{1}{2} \sum_i (y_i^{p,p} - d_i^p)^2}{\partial w_{ij}} = \sum_p \delta_i^p c_j^p \tag{3.1}$$

geändert, wobei

$$\delta_i^p = -\frac{\partial \frac{1}{2} \sum_i (y_i^{p,p} - d_i^p)^2}{\partial net_i^p}$$

gilt. Dazu berechne für alle p zunächst durch BP in M die Werte

$$\kappa_i^p = -\frac{\partial \frac{1}{2} \sum_i (y_i^{p,p} - d_i^p)^2}{\partial a_i^p}. \tag{3.2}$$

Ist i Ausgabeknoten von C :

$$\delta_i^p = f_i'(net_i^p)\kappa_i^p$$

Ist i versteckter Knoten der r-ten Lage von C:

$$\delta_i^p = f_i'(net_i^p) \sum_{l \in \, Lagen \, > r} w_{li}\delta_l^p.$$

- **ALGORITHMUS** also ganz simpel: C und M bilden zusammen Gesamtnetz, M hängt an C's Ausgabe, einfach durch M durchpropagieren, ohne M zu ändern, Fehler durch M's Eingabeknoten rein in C (durch C's Ausgabeknoten), in C Gewichtsänderungen wie gewohnt!
- Erweiterungen: Dasselbe mit rekurrenten Netzen.

3.2 ANWENDUNGEN

- Simulierten Lastwagen mit Anhänger rückwärts an Laderampe andocken [40].
- Erlernen zielgerichteter Retinatrajektorien zur selektiven Aufmerksamkeitssteuerung [61].

Kapitel 4

GENERALISIERUNG

4.1 "EINFACHE" MODELLE

- Gegeben Reihe: 2, 4, 6, 8, ... Wie heißt nächste Zahl? Antwort: 34. Warum? Weil Gesetz:

$$y = x^4 - 10x^3 + 35x^2 - 48x + 24.$$

  Aber Intelligenztest verlangt: 10 statt 34. Warum? "Einfache" Lösungen bevorzugt. Ockhams Messer! Gute Lernalgorithmen finden "einfache" Lösungen.

  Allgemein aber fundamentale Schranke des Lernbaren: Kein *a priori* Wissen $\rightarrow$ es bleibt nichts als exhaustive Suche.

- Was ist "einfache" Struktur? Was ist "komplexe" Struktur? Kolmogorov-Komplexität: Länge des kürzesten Programms, das Struktur generiert [28] [66] [10]. Eigentlich maschinenabhängig. Bei großen Strukturen: Maschine unwichtig:

- Turing Maschine (TM) mit Eingabeband, Rechenband, Ausgabeband.

  TM C: partielle Fn.: $f_C : \{0,1\}^* \rightarrow \{0,1\}^*$

  Es ex. univ. TM U mit f.a. C, p ex. $\mu_C : f_C(p) = f_U(\mu_C p)$, μ_C konstanter Prefix.

  Sei $\mid s \mid :=$ Zahl der bits in s.

  p ordentlich <> U liest alle $\mid p \mid$ bits und hält (trägt also Information über seine eigene Länge). Kein ord. Prog. Prefix eines anderen [78] [32] [11]!

  Kolmogorov-Komplexität oder alg. Komplexität, alg. Info:

$$I(s) = min\{\mid p \mid\mid p\, ord.,\ f_U(p) = s\}.$$

- I.a. ist $I(s)$ nicht berechenbar – das kürzeste Prog. läßt sich nicht finden. Und unbeweisbar, daß eine Struktur besonders komplex ist:

  Formales System: Axiome, fixe Schlussregeln $\rightarrow$ syst. Aufz. Theoreme.

  Barzdin, Chaitin: Es ex. k^* so daß unbeweisbar: $I(s) > I(axiome) + k^*$:

  Sei s^k das s im ersten Theorem "$I(s) > I(axiome) + k$".

  $I(axiome) + k < I(s^k)$

  $I(s^k) \leq I(axiome) + I(k) + O(1)$

  $k \leq I(k) + O(1)$. FALSCH f. $k > k^*$, da

  $I(k) \leq O(log_2 k)$. $\rightarrow$ GÖDEL!

- Was hat das mit Lernen zu tun? $\rightarrow$

- Ockham und Bayes (1):

$$P(w \mid D) = \frac{P(D \mid w)P(w)}{P(D)}$$

OPTIMALES Lernen: Finde w mit $P(w \mid D)$ maximal. Proportional zu $P(w)$! Hat man *a priori* Wahrsch. so daß $P(w)$ groß für einfache w: Ockham automatisch. Sei w Programm für univ. TM. Es ex. natürliche "universelle" a priori W-verteilung auf allen haltenden Programmen: Die kurzen sind wahrscheinlicher, also "besser". Ein guter Lernalgorithmus findet die kurzen!

4.2 "EINFACHE" KNN

- **Lernfehler versus Generalisierungsfehler**: viele Gewichte im NN: → Trainingset wird beleibig genau angenähert → wie look-up table. Aber irgendwann *steigt Generalisierungsfehler* wieder! → "Overfitting": Netz konzentriert sich auf Besonderheiten der Trainingsmenge statt der zugrundeliegenden Regularitäten. Netz zu komplex.
- Wie ist Overfitting vermeidbar?
- Wenn auch kürzeste Programme i.a. nicht auffindbar, so doch oft relativ kurze Programme → Hoffnung: Generalisierungsverbesserung. NN: Netze mit kurzer Beschreibung gesucht. Daher: bestrafe überkomplexe NN: Möglichst wenig versteckte Knoten, Gewichte, etc.
- Heuristik: Möglichst kleine Gewichte durch Zusatzfehlerterm:

$$E_{total} = E_{supervised} + \sum_{i,j} w_{ij}^2 \tag{4.1}$$

E_{total} ableiten: "Weight Decay". Schließlich "pruning" der Gewichte nahe Null (destruktive Methoden).

"Weight Decay" wie oben ist Standard. Aber eine befriedigende Lösung für die Suche nach "einfachen" Netzen existiert zur Zeit noch nicht.

- Alternative: statt von großen NN zu kleineren, von kleinen zu größeren (konstruktive Methoden).
- Sonstige Ansätze : "Soft Weight Sharing" [41]: Fehlerterm erzwingt ähnliche Gewichte, verteilt nach Gaussmixturmodellen → simple Beschreibung.
- Im Prinzip besser: **"MINIMAL DESCRIPTION LENGTH"(MDL)-Verfahren** [66] [49][50]: **Minimiere Information des Modells plus Information der Rekonstruktionsfehler!** Hinton [18]: Rauschen auf den Gewichten, je mehr Rauschen, desto weniger Information. Wieder: Minimiere Information der Gewichte plus Information der Rekonstruktionsfehler !
- Funktioniert sehr gut: **"FLAT MINIMUM SEARCH"** (separates Material) [21]

Kapitel 5

REINFORCEMENT LERNEN

5.1 TEMPORAL DIFFERENCES

- Problembeispiel: Backgammonspielen. Unterschied zu überwachtem Lernen: Kein Lehrer sagt dem System, welcher Zug zu welchem Zeitpunkt auszugeben ist.
- Ansatz: Vorhersagen lernen: was passiert langfristig, wenn man jetzt diesen Zug macht? Dazu
- "Temporal Difference" Methoden (Samuel, Sutton): Vorhersage in Markovumgebungen [68][54].

  Sequenz von Beobachtungen $x_t, t \in \{1 \dots n\}$ eines dyn. Systems. Problem: zu jedem Zeitpunkt $t \leq n$ Vorhersage P_t über den finalen Zustand $x_n = P_n$ machen. Purer Gradientenabstieg:

$$\triangle w^T = -\alpha \sum_{t=1}^{n-1} (P_n - P_t) \frac{\partial P_t}{\partial w}.$$

  Statt dessen TD(λ)-Methoden:

$$\triangle w^T = -\alpha \sum_{t=1}^{n-1} (P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial P_k}{\partial w}.$$

  ($0 \leq \lambda \leq 1$ ist 'Schwundfaktor').
- Für $\lambda = 1$ ergibt sich exakt der pure Gradientenabstieg.

  Manchmal ist TD(λ) aber für $\lambda \neq 1$ besser. Bei absorbierenden Markov-Prozessen (z.B. Brettspielen) konvergiert *lineares* TD(0) zu den optimalen Voraussagen ('maximum likelihood estimates'). (Markov-Prozeß: Prozeß, bei dem die Weiterevolution von einem gegebenen Zustand zur Zeit t nicht von früheren Zuständen abhängt. Absorbierender Markov-Prozeß: Wahrscheinlichkeit der Terminierung ist 1.)
- Ab jetzt: TD(0). Inkrementeller Algorithmus:

$$\triangle w_t^T = -\alpha (P_{t+1} - P_t) \frac{\partial P_t}{\partial w}.$$

  Vorhersage der eigenen Vorhersage! Z.B. mit Backprop-Netzen oder sonstigen Funktionsapproximatoren.
- *kumulative Voraussagen*: zu jedem Zeitpunkt t sei die Summe noch ausstehender Reinforcementsignale R_k zu allen späteren Zeitpunkten k vorherzusagen. Nach der Lernphase soll für alle t gelten:

$$P_t = \sum_{k=0}^{m} \gamma^k R_{t+k+1}.$$

$0 < \gamma \leq 1$ 'Discountrate', welche bestimmt, wie stark *Erwartungen* über verschieden weit in der Zukunft liegende Ereignisse die Gewichtsänderung mit beeinflussen sollen. Nach obiger Gleichung soll gelten:

$$P_t = R_{t+1} + \gamma P_{t+1}.$$

Gilt aber in der Regel noch nicht! Also Gewichtsänderung

$$\triangle w_t^T = -\alpha(R_{t+1} + \gamma P_{t+1} - P_t)\frac{\partial P_t}{\partial w}.$$

- Wieder Beispiel: Backgammonspielen. TD-Lernen mit Backprop-Netz. Eingabe: Repräsentation der Spielstellung Ausgabe: Bewertung. Gelernt wie oben: R_t ist 1 falls Sieg für Schwarz, 0 sonst. Gewählt wird der Zug, der in den Nachfolgezustand mit der höchsten Bewertung führt. System spielt ca. 10^6 mal gegen sich selbst. Ist schließlich besser als das beste handverfertigte Programm, und so gut wie die drei besten menschl. Spieler der Welt [70].
- Warum funktioniert's? U.a., weil Backgammon Zufallskomponente hat. Allgemein aber lieber statt dessen:

5.2 Q-LERNEN

- n mögliche Zustände $x_1, x_2, ..., x_n$. m mögliche Aktionen $a_1, a_2, ..., a_m$. Zur Zeit t: $x(t)$, $a(t)$, Reinforcement $r(t)$, Q-Tabelle Q: $n \times m$-Matrix.
- Algorithmus [72]: Q-Tabelle zufällig initialisieren. Dann Kernschleife:

  1. Beobachte jetzigen Zustand $x(t)$. Wähle zufällig $p \in [0, \ldots, 1]$. Falls $p \leq \mu \in [0, \ldots, 1]$ (z.B. $\mu = 0.1$): Wähle $a(t)$ zufällig. Sonst wähle $a(t)$ so, daß $Q(x(t), a(t))$ maximal.

  2. Führe $a(t)$ aus, erhalte $x(t+1)$ und $r(t)$.

  3. Setze

  $$Q(x(t), a(t)) \leftarrow (1-\alpha)Q(x(t), a(t)) + \alpha(r(t) + \gamma\, max_b Q(x(t+1), b)),$$

  wobei α Lernrate und $0 \leq \gamma \leq 1$.
- Es existiert Beweis, daß Q-Lernen mit look-up tables konvergiert.
- Mit Backprop-NN statt Q-table: Kein Konvergenzbeweis mehr, aber gute praktische Resultate. Zustände durch Eingabevektoren $x(t)$ repräsentiert. Aktionen durch Eingabevektoren $a(t)$ repräsentiert. Netz sieht Zustands/Aktionspaare wie $(x(t), a(t))$. Ein Ausgabeknoten. Ausgabe heißt wieder $Q(x(t), a(t))$. Modifikationen zu oben:

  Schritt 3. Tatsächliche Ausgabe des NN: $Q(x(t), a(t))$. Gewünschter Wert aber:

  $$r(t) + \gamma\, max_b Q(x(t+1), b).$$

  Trainiere Netz mit backprop und $\eta > 0$ als Lernrate.
- Anwendungen: Brettspiele, Agenten- und Robotersteuerung, z.B. [33].
- Erweiterungen: Weltmodelle ("DYNA-Q"). Modellnetz lernt, Zustandsübergänge vorherzusagen. System macht statt wirklicher Q-Lern-Experimente manchmal (oft) "mentale Experimente" mit seinem Modell. Gut, falls Experimente teuer (Flugzeugabstürze etc.). Anwendungen: Hindernisvermeidung [69] [38].

5.3 BEZUG ZUR DYNAMISCHEN PROGRAMMIERUNG

- Am einfachsten: Diskreter deterministischer mehrstufiger Entscheidungsprozeß: höchstens abzählbar viele Transformationen bestimmen Prozeßverlauf. Prozeß startet im Zustand p_1, für $t > 1$:

  $$p_t = T(p_{t-1}, q_{t-1}).$$

  Zu maximieren sei für n-stufigen Markov-Prozeß Kostenfunktion $U(q_1, q_2, \ldots, q_n)$.

  Hat U z.B. die Form

  $$U = u(p_1, q_1) + u(p_2, q_2) + \ldots + u(p_n, q_n),$$

  so ist Markov-Eigenschaft gewährleistet.

- Bellmans Optimalitätsprinzip [8]: *Eine optimale Folge von sukzessiven Entscheidungen (eine* optimale Strategie*) hat die Eigenschaft, daß unabhängig vom Initialzustand des Prozesses und der ersten Entscheidung die noch ausstehenden Entscheidungen eine optimale Strategie bezüglich des Zustands darstellen, der aus der ersten Entscheidung resultiert.*

  Daher für Markov-Prozesse der Ansatz der *dynamischen Programmierung*: Rekursiv definiert man:

  $$f_1(p_1) = max_{q_1} u(p_1, q_1)$$

  und

  $$f_t(p_1) = max_{q_1}(u(p_1, q_1) + f_{t-1}(T(p_1, q_1)))$$

  Durch das Optimalitätsprinzip ist der Beitrag der letzten $n - 1$ Schritte gleich $f_{n-1}(T(p_1, q_1))$.

- Sei p die Zahl der Dimensionen, die zur Beschreibung der Prozeßzustände notwendig sind. Die Auswahloperation eines Punktes im pn-dimensionalen Raum wird reduziert auf n Auswahloperationen im p-dimensionalen Raum.

- Q-Lernen ist nur eine inkrementelle on-line Variante der dyn. Prog.!

- **HAUSAUFGABE:** Implementiere Q-Lernen und verwende es zum Finden von Wegen in einem Labyrinth.

Kapitel 6

UNÜBERWACHTES LERNEN

6.1 WARUM?

- interne Repräsentation des p-ten Eingabevektors x^p sei n-dimensionaler Vektor y^p . Finde nützliche, weitgehend zielunabhängige Rep.! Nützlich? $\rightarrow$
- Finde *"Regularitäten"* in den Eingaben so daß z.B. ohne detailliertes Vorwissen über später zu lösende Aufgaben Effizienzgewinne erwartet werden können. Im folgenden: *"Regularit äten"* = statistische Redundanz.
- *(1) Redundanzminimierung* für kompakte Repräsentationen, effiziente Klassifikationsverfahren und bessere Generalisierungsfähigkeit.
- *(2) Informationstransmissionsmaximierung.*
- *(3) Dekorrelation zur Beschleunigung überwachter Lernvorgänge.* z.B.: Überwacht lernendes lineares Netz L, Eingaben mit unkorrelierten Komponenten, $\rightarrow$ Hessematrix

  $$\nabla_{w_L} \nabla_{w_L} E_L$$

  von L's Fehlerfunktion E_L diagonal (w_L: L's Gewichtsvektor). $\rightarrow$ 2. Ableitung läßt sich billig berechnen $\rightarrow$ effiziente Methoden 2. Ordnung zur Lernbeschleunigung
- *(3b) Kausaldetektion in Eingabeströmen.*
- *(4) Extraktion vorhersagbarer nicht-trivialer Mustereigenschaften.*
- Problem: Aneignung von Information über die Umgebung – aber kein Lehrer! Was ist Information überhaupt? Kolmogorov Information kennen wir schon. Meist ist jedoch statistische Information gemeint (Shannon) [64].

6.2 INFORMATIONSTHEORIE

- Informationsmaß H: n Ereignisse mit Wahrsch. $p_1, p_2, ... p_n$. $\sum_i p_i = 1$. Entropie $H(p_1, p_2, \ldots, p_n)$ soll sein: Erwartungswert des Informationsgehaltes eines Zeichens.

  $$H = -K \sum_i p_i log p_i, \quad (K\ const. > 0).$$

  Ist additiv: $H(X) + H(Y) = H(X, Y)$ falls Zv. X,Y unabh.
- bedingte Entropie:

  $$H(Y \mid X) = -\sum_{i,j} p(i,j) log p(j \mid i)$$

- Wechselseitige Information zw. X und Y:

$$I(X,Y) = H(X) - H(X \mid Y) = H(X) + H(Y) - H(X,Y).$$

- Kullback-Leiber Distanz [29]: Zwei Verteilungen, T und P. α indiziert die mögl. Fälle.

$$G(T;P) = \sum_{\alpha} T_\alpha log \frac{T_\alpha}{P_\alpha}$$

Auch für "Boltzmann Maschine" [17]. Interessant:

$$I(X,Y) = \sum_{x,y} p(x,y) log \frac{p(x,y)}{p(x)p(y)} = G(gemeinsame\ Vert.v.X,Y; unabh.\ Vert.v.X,Y)$$

G-MAX-algorithmus [45].

6.3 ALGORITHMEN FÜR DEKORRELATION, INFOMAX

Siehe auch [5].

- Wechselseitige Information und NN:

1. X Eingabe, Y Ausgabe. Maximiere Informationstransmission $I(X,Y) = H(X) + H(Y) - H(X,Y) = H(Y)$ (da Eingaben feststehen).

2. X Ausgabe von Modul1 , Y Ausgabe von Modul2. Beide Module sehen verwandte, aber ungleiche Eingaben (Beispiel: Stereogramm). Maximiere $H(X) + H(Y) - H(X,Y) \rightarrow$ was haben die Eingaben gemeinsam? (Im Beispiel: die Tiefe). Siehe auch [6] [62] [7].

- Zu 1: Bei linearem Netz mit nur einer Ausgabe, Gewichte w, Rauschterm n: $y_1^p = wx^p + n$:

$$I = \frac{1}{2} ln(\frac{VAR(y_1)}{VAR(n)}). \tag{6.1}$$

Ableiten: $\rightarrow$ Hebbregelvariante [35][36].

Wird nun zu jedem Eingabesignal Rauschen mit Varianz $V(n)$ addiert:

$$I = \frac{1}{2} ln(\frac{VAR(y_1)}{VAR(n) \sum_i w_i^2}), \tag{6.2}$$

(w_i Gewicht vom i-ten Eingabeknoten). Der resultierende Lernalgorithmus ist äquivalent zur Maximierung der Ausgabevarianz unter gleichzeitiger Minimierung der Länge des Gewichsvektors, was der Analyse der prinzipiellen Komponenten des Eingabeensembles enstpricht.

- Hebbvarianten sind biologisch plausible Algorithmen für automatische Dekorrelation. Experimente liefern *"On-Center-Off-Surround Strukturen"*.

- Nun mehr als ein Ausgabeknoten: Maximiere

$$I = \frac{1}{2} ln(\frac{Det(Q(y))}{VAR(n)}), \tag{6.3}$$

wobei $Q(y)$ Kovarianzmatrix. *Dekorrelation* der Ausgaben, $VAR(n)$ klein. Sonst erhöht sich die Redundanz der y-Knoten.

Partielle Abl. von $Det(Q(y))$ bezüglich aller y_i berechnen! Biologisch unplausibel.

- Experimente: Topologieerhaltende Abbildungen wurden auf unüberwachte Weise gefunden! $\rightarrow$ nächster Punkt.

- Computerangepaßter Alg. von Kohonen [27] zur Generierung topologieerhaltender Abbildungen. (Anwendung: z.B. Optimierungsprobleme). Aber Kohonen-Algorithmus scheint in gewissem Sinne nur Konsequenz eines allg. Prinzips zu sein. Siehe aber nächstes Kapitel.

6.4 FAKTORIELLE CODES

- Det. Abb. von Eingabemustern x^p auf Ausgabevektoren y^p. Jede Komponente y_i von y nimmt bei mit Wahrsch. $P(y_i = a)$ den Wert a an. Faktorielle Codes definiert durch:

$$\forall p: \quad P(x^p) = \prod_i P(y_i = y_i^p). \tag{6.4}$$

  Ist Code reversibel (informationserhaltend), so ist Abb. von x^p auf y^p bijektiv und es gilt $\forall p: \quad P(x^p) = P(y^p)$. Dann faktorielle Codes [2]:

$$\forall p: \quad P(y^p) = \prod_k P(y_k = y_k^p). \tag{6.5}$$

  Die einzelnen Komponenten von y müssen also statistisch voneinander unabhängig sein.

- Warum faktorielle Codes?

  (1) *Verbesserung traditioneller statistischer Klassifikationsmethoden.*

  (2) *Eingabesegmentierung, Kompression, Datenreduktion.*

  (3) *Occam's Rasiermesser*.

  (4) *Schnelles Lernen.*

  (5) *Neuigkeitsentdeckung*.

- Binäre Faktorielle Codes:

$$E(y_i \mid \{y_k, k \neq i\}) = E(y_i)$$

  Wie kriegt man sie?

  1. Minimierung der Bitentropiesumme [3]:

$$-\sum_i P(y_i = 1) log P(y_i = 1) - \sum_i P(y_i = 0) log P(y_i = 0). \tag{6.6}$$

  2. Vorhersagbarkeitsminimierung [59]. Netz mit m Eingabeknoten und n Ausgabeknoten oder Codeknoten. i-ter Codeknoten liefert Ausgabe $y_i^p \in [0, 1]$ in Antwort auf Eingabevektor x^p. **Für jeden Codeknoten existiert Prediktornetz, welches versucht, den Codeknoten aus den verbleibenden $n - 1$ Codeknoten vorherzusagen. Doch jeder Codeknoten wehrt sich, indem er versucht, so unvorhersagbar wie möglich zu werden.** Dazu muß er lernen, abstrakte "features" der Eingabedaten zu codieren, so daß er statistisch von den anderen Codeknote unabh. wird. Wie? Prediktornetz für Codeknoten i heißt P_i. Seine Eingabe sind die $\{y_k^p, \ k \neq i\}$, seine Ausgabe heißt P_i^p. P_i *minimiert*

$$\sum_p (P_i^p - y_i^p)^2 \tag{6.7}$$

  und lernt damit den bedingten Erwartungswert $E(y_i \mid \{y_k, k \neq i\})$. Doch die Codeknoten *maximieren* dieselbe Zielfunktion:

$$V_C = \sum_{i,p} (P_i^p - y_i^p)^2. \tag{6.8}$$

  Prediktoren und Codeknoten co-evolvieren im Kampf gegeneinander. V_C ist im wesentlichen äquivalent zu

$$\sum_i VAR(y_i) - \sum_{i,p} (P_i^p - \bar{y_i})^2, \tag{6.9}$$

  wobei $\bar{y_i}$ Mittelwert der Ausgabe von i, VAR ist Varianzoperator.

  Anwendung: Buchstabenbilder mit Auftretenswahrscheinlichkeiten gemäß engl. Sprache [34].

  (Mit) am besten funktioniert: **"FLAT MINIMUM SEARCH"** (separates Material) [21]

Kapitel 7

SELBSTORGANISIERENDE KARTEN

7.1 Neurophysiologische Motivation

im Cortex 2-dim. Schicht (Karte) von Neuronen;
durch laterale Inhibition (seitl. Hemmung) untereinander verbunden
Eingangssignal v aktiviert Neurone gemäß
$f_r = \sigma(\sum_l w_{rl} v_l + \sum_{r'} g_{rr'} f_{r'} - \Theta)$
f: Aktivierung eines Neurons
σ: sigmoide Funktion
$g_{rr'}$: Rückkopplung vom Typ laterale Inhibition; für kurze Distanzen erregend; für lange Distanzen hemmend
Durch die laterale Inhibition entsteht in der Karte ein einziger, zusammenhängender Cluster aktivierter Neurone. Die Lage des Clusterzentrums ist abh. vom Eingangssignal

7.2 Kohonen's Modell

Näherungsannahme: Der Ort r' maximaler Erregung ist vom Eingangssignal v allein abhängig.
Bedingungen:
1. $\sum_l w_{rl}^2$ konst., für alle Neurone gleich
2. $||v|| = 1$
dann gilt: $||w_{r'} - v|| = min||w_r - v||$; $w_r := [w_{r1}, ..., w_{rd}](I)$
dies definiert für feste w eine nichtlin. Projektion des Raumes der Eingangssignale auf die 2-dim. Neuronenschicht.
Neurone reagieren durch Anpassen von w

Kohonen's Algorithmus (1982)

0) 'Initialisierung': Starte mit geeignetem Anfangswerten w_{rl}, z.B. zufällig
1) 'Stimuluswahl': Wähle entsprechend der Wahrscheinlichkeitsdichte P(v) einen zufälligen Vektor v (Sensorsignal)
2) 'Response': Bestimme Erregungszentrum r' aus (I)
$||v - w_{r'}|| \leq ||v - w_r|| \forall r \in A$
3) 'Adaptionsschritt': $w_r^{neu} = w_r^{alt} + \epsilon h_{rr'}(v - w_r^{alt})$
4) verringere Lernrate ϵ und $h_{rr'}$.
gehe zu 1)
sowohl die Lernrate ϵ als auch die Nachbarschaftsfunktion $h_{rr'}$ müssen im Laufe der Zeit gegen Null

konvergieren.

7.3 Vergleich zur Datenkompression

$E[W]$: Erwartungswert des quadratischen Rekonstruktionsfehlers
$E[W] = \int ||v - w_{r'}(v)||^k P(v)dv$
$k = \frac{1}{2} + \frac{3}{2(2n+1)^2}$
n definiert über Nachbarschaftsfunktion $h_{rr'} = 1$ für $||r - r'|| \leq n; 0$ sonst

bei Vektorquantisierung (allgem. Ansatz zur Datenkompression) gilt: k = 2

7.4 Vergleich zur Varianzanalyse

bei Varianzanalyse sind w^i die D normierten, eigenwertgrößten Eigenvektoren der Korrelationsmatrix C der Daten (w^i: Hauptachsen)
bei Kohonen: nichtlinearer Zusammenhang zwischen r_i und v_i. Hauptachse entspricht der Hauptkurve, Hauptmannigfaltigkeit.

Kapitel 8

EVOLUTIONÄRE STRATEGIEN

8.1 GENETISCHE ALGORITHMEN

Siehe [23].

- *"Pool"* von Bitsequenzen fester Länge n. Jede Bitsequenz wird an zu lösender Aufgabe getestet. Güte der Performanz: reelle Zahl (*Fitness*).
- Je höher die *Fitness* eines "Genotyps", desto höher die Wahrscheinlichkeit, daß er sich "fortpflanzen" darf: Zwei Genotypen (erfolgreiche bevorzugt) tauschen "genetisches Material" in Form von Bitsubsequenzen aus. Gelegentlich auch "Mutationen" (zufällig ausgewählte Bits werden durch Komplement ersetzt).
- Genotypen mit hoher *Fitness* verdrängen nun aus dem *Pool* solche mit niedriger *Fitness*.
- Iteriere Verfahren, bis Genotyp mit genügend hoher *Fitness* entstanden ist.
- Auf neuronale Netze mit binären Gewichten läßt sich das Verfahren ohne große Umschweife anwenden. Scheint aber nicht allzuviel zu bringen.

8.2 EIMERKETTE

Siehe [24].

- Botschaften in Form von Bitsequenzen der Länge n können von der Umgebung oder von "Klassifikatoren" auf der globalen *Botschaftentafel* plaziert werden. Jeder Klassifikator besteht aus "Bedingungssequenz" und "Aktionssequenz". Beide Teile sind Sequenzen aus $\{0, 1, \_\}^n$, wobei "_" das "dont care symbol" ist.
- Jeder Klassifikator hat positive reelle Variable die *"Stärke"*.
- Bestimmte Botschaften auf der Botschaftentafel veranlassen Agenten, bestimmte Operationen in der Umgebung auszuführen. Externer Kritiker verteilt u.U. Belohnung (Auszahlung) an die gegenwärtig aktiven Klassifikatoren, deren Stärke dadurch wächst.
- Zyklus : Vergleiche alle Botschaften auf der Botschaftentafel mit den Bedingungssequenzen aller Klassifikatoren. Jeder Klassifikator, dessen Bedingungsteil mit wenigstens einer Botschaft übereinstimmt, berechnet seinen *"Wetteinsatz"*, indem er seine *Spezifizität* (die Anzahl der Bits in seinem Bedingungsteil, die keine "_"s sind) mit dem Produkt seiner Stärke und einer kleinen Konstanten multipliziert.

  Die höchstbietenden Klassifikatoren dürfen für den nächsten Zyklus nun ihren *Aktionsteil* auf die Botschaftentafel schreiben. (Tritt dabei das "_" in einer Aktionssequenz auf, so wird das entsprechende Bit der "triggernden" Botschaft übernommen.)

Für das Schreibeprivileg müssen die Gewinner allerdings mit ihrem *Wetteinsatz* bezahlen, welcher unter denjenigen Klassifikatoren verteilt wird, die während des letzten Zyklus die Vorraussetzungen für die Schreiboperationen der Gewinner schufen.

- Jeder in einer bestimmten Situation "aktive" Klassifikator wird also schwächer werden, wenn er seine Verluste im nächsten Zyklus nicht wieder ausgleichen oder gar mehr als wettmachen kann. Der Erfolg eines zu einem gegebenen Zeitpunkt aktiven Klassifikators hängt rekursiv von den Erfolgen seiner Nachfolger ab. (ein Ansatz für "temporal credit assignment", wie Eimerkette zum Feuerlöschen).
- Variante für KNN: [56].

8.3 "ARTIFICIAL LIFE"

Siehe z.B. [48] [47] [14].

- Was ist Leben? Biologe: Es reproduziert sich, und evolviert.
- Dies trifft auf "künstliches Silikonleben" auf Digitalrechnern zu. Beispiel [48]:
- Orginalassemblersequenz (ca. 80 bit) besteht im wesentlichen aus Code zur Selbstreproduktion. Gelegentliche Mutationen. Limitierte Ressourcen: "Bitsuppe" umfaßt nur 60000 bits. Kopien der Orginalassemblersequenz füllen rasch die Bitsuppe, ist diese voll, beginnt eingebautes Absterben und Konkurrenzkampf, ab jetzt überleben die Sequenzen, die sich mit dem geringsten Aufwand am besten vervielfältigen können. Es entstehen zahlreiche neue "Arten", "Parasiten", etc.
- Interessant für Studium der Evolution, praktische Verwertbarkeit für zielgerichtetes Lernen jedoch noch unklar.

Literaturverzeichnis

[1] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 609–618, 1987.

[2] H. B. Barlow. Unsupervised learning. *Neural Computation*, 1(3):295–311, 1989.

[3] H. B. Barlow, T. P. Kaushal, and G. J. Mitchison. Finding minimum entropy codes. *Neural Computation*, 1(3):412–423, 1989.

[4] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[5] S. Becker. Unsupervised learning procedures for neural networks. *International Journal of Neural Systems*, 2(1 & 2):17–33, 1991.

[6] S. Becker and G. E. Hinton. A self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355:161–163, 1992.

[7] S. Becker and G. E. Hinton. Learning mixture models of spatial coherence. *Neural Computation*, 5(2):267–277, 1993.

[8] R. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961.

[9] C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[10] G.J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:547–569, 1966.

[11] G.J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22:329–340, 1975.

[12] S. E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie-Mellon Univ., 1988.

[13] R. Farber, A. Lapedes, and K. Sirotkin. Determination of eukaryotic protein coding regions using neural networks and information theory. *Journal of Molecular Biology*, 226:471–479, 1992.

[14] W. Fontana. Algorithmic chemistry. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 159–210. Addison Wesley Publishing Company, 1992.

[15] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[16] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4:393–405, 1992.

[17] G. E. Hinton and T. E. Sejnowski. Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, volume 1, pages 282–317. MIT Press, 1986.

[18] G. E. Hinton and D. van Camp. Keeping neural networks simple. In *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam*, pages 11–18. Springer, 1993.

[19] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991. See www7.informatik.tu-muenchen.de/˜hochreit.

[20] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

[21] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[22] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[23] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[24] J. H. Holland. Properties of the bucket brigade. In *Proceedings of an International Conference on Genetic Algorithms*. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[25] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the National Academy of Sciences*, 79:2554–2558, 1982.

[26] M. I. Jordan and D. E. Rumelhart. Supervised learning with a distal teacher. Technical Report Occasional Paper #40, Center for Cog. Sci., Massachusetts Institute of Technology, 1990.

[27] T. Kohonen. *Self-Organization and Associative Memory*. Springer, second edition, 1988.

[28] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.

[29] S. Kullback. *Statistics and Information Theory*. J. Wiley and Sons, New York, 1959.

[30] Y. LeCun. Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva 85, Paris*, pages 599–604, 1985.

[31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[32] L. A. Levin. Laws of information (nongrowth) and aspects of the foundation of probability theory. *Problems of Information Transmission*, 10(3):206–210, 1974.

[33] L. Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In J. A. Meyer and S. W. Wilson, editors, *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 297–305. MIT Press/Bradford Books, 1991.

[34] S. Lindstädt. Comparison of two unsupervised neural network models for redundancy reduction. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend, editors, *Proc. of the 1993 Connectionist Models Summer School*, pages 308–315. Hillsdale, NJ: Erlbaum Associates, 1993.

[35] R. Linsker. Self-organization in a perceptual network. *IEEE Computer*, 21:105–117, 1988.

[36] R. Linsker. How to generate ordered maps by maximizing the mutual information between input and output. *Neural Computation*, 1(3):402–411, 1989.

[37] M. Minsky and S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[38] A. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[39] M. C. Mozer. A focused back-propagation algorithm for temporal sequence recognition. *Complex Systems*, 3:349–381, 1989.

[40] Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 357–363. IEEE Press, 1989.

[41] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4:173–193, 1992.

[42] D. B. Parker. Learning-logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT, 1985.

[43] D. B. Parker. Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order hebbian learning. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 593–600, 1987.

[44] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[45] B. A. Pearlmutter and G. E. Hinton. G-maximization: An unsupervised learning procedure for discovering regularities. In J. S. Denker, editor, *Neural Networks for Computing: American Institute of Physics Conference Proceedings 151*, volume 2, pages 333–338, 1986.

[46] F. J. Pineda. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1(2):161–172, 1989.

[47] S. Rasmussen, C. Knudsen, and R. Feldberg. Dynamics of programmable matter. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 211–254. Addison Wesley Publishing Company, 1992.

[48] T. S. Ray. An approach to the synthesis of life. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 371–408. Addison Wesley Publishing Company, 1992.

[49] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[50] J. Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14(3):1080–1100, 1986.

[51] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[52] M. Röscheisen, R. Hofmann, and V. Tresp. Neural control for rolling mills: Incorporating domain theories to overcome data deficiencies. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4*, pages 659–666. Morgan Kaufmann, 1992.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[54] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229, 1959.

[55] J. Schmidhuber. Accelerated learning in back-propagation nets. In R. Pfeifer, Z. Schreter, Z. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, pages 429 – 438. Amsterdam: Elsevier, North-Holland, 1989.

[56] J. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1989.

[57] J. Schmidhuber. A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992.

[58] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[59] J. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992.

[60] J. Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. *Neural Computation*, 4(1):131–139, 1992.

[61] J. Schmidhuber and R. Huber. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2(1 & 2):135–141, 1991.

[62] J. Schmidhuber and D. Prelinger. Discovering predictable classifications. *Neural Computation*, 5(4):625–635, 1993.

[63] B. Schürmann. Stability and adaptation in artificial neural systems. *Physical Review*, A 40(50):2681–2688, 1989.

[64] C. E. Shannon. A mathematical theory of communication (parts I and II). *Bell System Technical Journal*, XXVII:379–423, 1948.

[65] P.Y. Simard and Y. LeCun. Local computation of the second derivative information in a multi-layer network. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 5*. CA: Morgan Kaufmann, 1993. In preparation.

[66] R.J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7:1–22, 1964.

[67] P. Stolorz, A. Lapedes, and X. Xia. Predicting protein secondary structure using neural net and statistical methods. *Journal of Molecular Biology*, 225:363–377, 1992.

[68] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[69] R. S. Sutton. Integrated architectures for learning, planning and reacting based on dynamic programming. In *Machine Learning: Proceedings of the Seventh International Workshop*, 1990.

[70] G. Tesauro. Practical issues in temporal difference learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4*, pages 259–266. Morgan Kaufmann, 1992.

[71] V. Vapnik. Principles of risk minimization for learning theory. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4*, pages 831–838. Morgan Kaufmann, 1992.

[72] C.J.C.H Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[73] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1:193–209, 1990.

[74] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[75] R. J. Williams and J. Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 4:491–501, 1990.

[76] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1(2):270–280, 1989.

[77] D. Zipser. Recurrent network model of short-term active memory. *Neural Computation*, 3(2):179–193, 1991.

[78] A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the algorithmic concepts of information and randomness. *Russian Math. Surveys*, 25(6):83–124, 1970.