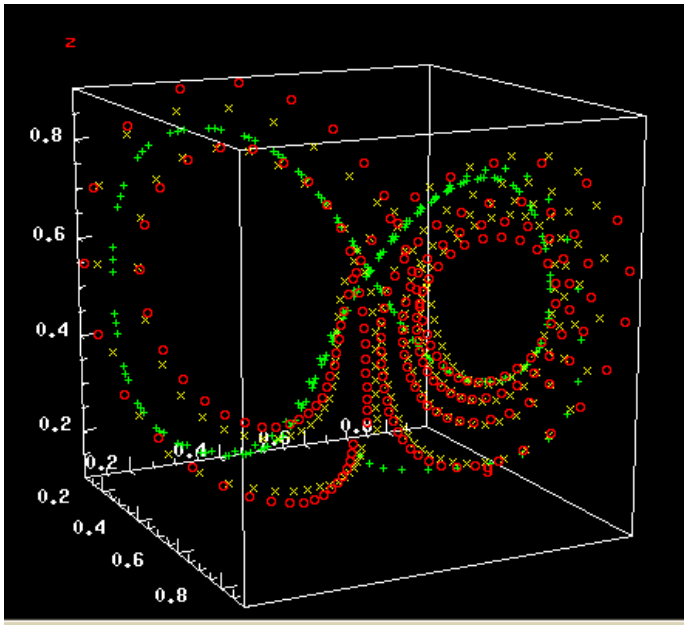
Universität Bielefeld
SS 2014

Script zur Vorlesung

Neuronale Netze II



Dr. Robert Haschke

(Stand: 7. Juli 2017)

Inhaltsverzeichnis

1 Optimierungsverfahren zum MLP-Lernen

1.1 Allgemeine Aspekte

a) Gute Performanz

- gute Approximation
- gute Generalisierung
- Vermeidung von Overfitting

b) Schnelligkeit

- wenige Datenbeispiele
- wenig Epochen
- wenig Rechenaufwand pro Schritt

c) Gute statistische Eigenschaften

- geringe Varianz der gelernten Parameter $\vec{w}$
- geringer Systematischer Fehler (Bias), d.h. geringe Abweichung vom Optimum $\vec{\omega}^*$

Dies sind widersprüchliche Forderungen:

Theorem 1.1.1. Bias Varianz Dilemma
Lernen entspricht Schätzung der Parameter $\vec{\omega}$ mit

- wahrer Wert: $\vec{\omega}^*$ (für gegebene Architektur)
- Schätzwert: $\vec{\omega}$ (durch Lernen ermittelt)
- Mittlerer Schätzwert: $\langle\vec{\omega}\rangle$

Idee. Zerlege den mittleren quadratischen Schätzfehler

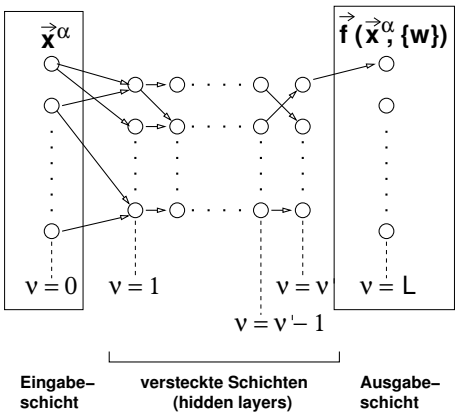
$$\begin{aligned}\left\langle(\vec{\omega}-\vec{\omega}^*)^2\right\rangle &= \left\langle[(\vec{\omega}-\langle\vec{\omega}\rangle)+(\langle\vec{\omega}\rangle-\vec{\omega}^*)]^2\right\rangle \\ &= \left\langle(\vec{\omega}-\langle\vec{\omega}\rangle)^2\right\rangle + (\langle\vec{\omega}\rangle-\vec{\omega}^*)^2 + \underbrace{2\underbrace{\langle\vec{\omega}-\langle\vec{\omega}\rangle\rangle}_{0}\cdot(\langle\vec{\omega}\rangle-\vec{\omega}^*)}_{0} \\ &= \text{Varianz der Schätzung} + \text{Bias}^2\end{aligned}$$

Bei festem Gesamtfehler: Veringerung der Varianz geht auf Kosten des Bias
$\rightarrow$ besseres Modell erforderlich.

d) Hohe Robustheit geringe Abhängigkeit von

- Anfangsbedingungen
- Parametern (Lernrate)
- Datenfehlern
- numerischer "Gutartigkeit"

1.2 Wiederholung MLP-Lernen



$$s_i^\nu = \sigma_i(a_i^\nu) \qquad a_i^\nu = \sum_j w_{i,j}^{\nu,\nu-1} s_j^{\nu-1}$$

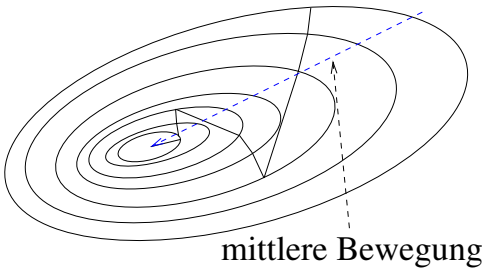
$$E = \sum_\alpha E^\alpha \qquad E^\alpha = \frac{1}{2}\sum_i (y_i^\alpha - s_i^L)^2$$

$$\delta_i^L = \frac{\partial E^\alpha}{\partial a_i^L} = -\sigma'(a_i^L)(y_i^\alpha - s_i^L)$$

$$\delta_i^\nu = \sigma'(a_i^\nu)\sum_k \omega_{k,i}^{\nu+1,\nu}\delta_k^{\nu+1}$$

$$\Delta\omega_{i,j}^{\nu,\nu-1} = -\eta\frac{\partial E^\alpha}{\partial \omega_{i,j}^{\nu,\nu-1}} = -\eta\,\delta_i^\nu\, s_j^{\nu-1}$$

1.3 Trägheitsterm oder Momentum



In langgestreckten Tälern von E führt der Gradientenabstieg zu Zick-Zack-Bewegungen. **Abhilfe**: Stärkere Beachtung des mittleren Schrittes durch Momentum-Term (Trägheitsterm):

$$\Delta w(t) = -\eta\,\frac{\partial E}{\partial w} + \underbrace{\alpha\,\Delta w(t-1)}_{\text{Trägheitsterm}}$$

Dabei muss gelten: $\alpha \ll 1$. Es ist sinvoll in der Nähe des Minimums $\alpha \to 1$ zu erhöhen.

Beispiel:

$$\nabla E(t) = \frac{\partial E}{\partial w} = \vec{a} + \vec{b} \qquad \vec{a} \text{ – effektive Richtung}$$

$$\nabla E(t-1) = \vec{a} - \vec{b} \qquad \pm\vec{b} \text{ – Oszillationsanteil}$$

$$\Delta w(t) = -\eta\,(\vec{a}+\vec{b}) \;-\; \alpha\,\eta\,(\vec{a}-\vec{b})$$

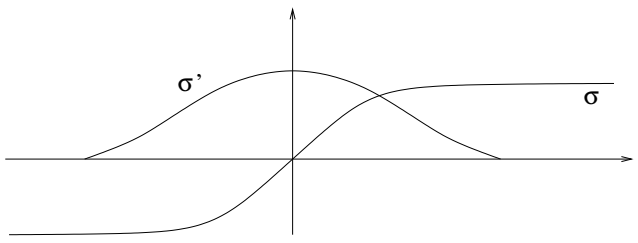
$$= -\eta\,(1+\alpha)\,\vec{a} \;-\; \eta\,(1-\alpha)\,\vec{b}$$

1.4 Flat-Spot Elimination

Standard-Lernregel liefert

$$\Delta\omega_{i,j} \sim \sigma_i'(\cdot)$$

mit σ (meist) sigmoid: $\sigma = \tanh$



Bemerkung. Gewichte zu Neuron i mit hoher Aktivierung nahe Sättigung können sich kaum noch ändern - Ursache von Plateaus im Fehlergebirge
Abhilfe: Ersetze $\sigma' \to \sigma' + C$ für eine kleine Konstante C überall in der Lernregel.
$\to$ Erfahrung: große Beschleunigung.

1.5 Weight-Decay

Plateaus in Fehlergebirgen können von großen Gewichten her stammen.
Abhilfe: Strafterm in Fehlerfunktion

$$E_{reg} = E + \frac{\lambda}{2} \sum_{i,j} \omega_{i,j}^2 \to \min$$
$$\Delta \vec{w} = -\eta \frac{\partial E}{\partial \vec{w}} - \lambda \vec{w}$$

Statistische Interpretation liefert den Wahrscheinlichsten Wert für $\vec{\omega}$ bei gegebenen Daten $\{x^\alpha, y^\alpha\}$ unter folgenden Annahmen:

1. Normalverteilte "a-priori" Wahrscheinlichkeit für $\vec{\omega}$
2. Normalverteilte Fehler $y^\alpha = f(x^\alpha, \omega)$ bzw. $y^\alpha - y^{net}$

Aus 2. folgt für die Fehler $\epsilon_\alpha = y^\alpha - y^{net}(x^\alpha, \omega)$:

$$P(\underbrace{x^\alpha, y^\alpha}_{z^\alpha}|\omega) = N \cdot exp(-\frac{\epsilon_\alpha^2}{2\sigma^2})$$
$$\Rightarrow P(z^1, \ldots, z^M|\omega) = \prod_{\alpha=1}^{M} P(z^\alpha|\omega)$$
$$= N' \cdot exp(-\frac{1}{2\sigma^2} \sum_{\alpha=1}^{M} \epsilon_\alpha^2)$$

Anwendung der Bayes Regel:

$$P(\omega|z^1, \ldots, z^M) = \frac{P(z^1, \ldots, z^M|\omega)P(\omega)}{P(z^1, \ldots, z^M)}$$
$$= N'' exp\left(-\frac{1}{2\sigma^2} \sum_{\alpha=1}^{M} \epsilon_\alpha^2\right) \cdot exp\left(-\frac{1}{2\sigma_\omega^2} \sum_{i,j} \omega_{i,j}^2\right)$$
$$= N'' exp\left(-\frac{1}{2\sigma^2}(E + \frac{\lambda}{2}\|\omega\|^2)\right) = N''' exp(-E_{reg}) \qquad \text{mit } \lambda = 2\frac{\sigma^2}{\sigma_\omega^2}$$

1.6 Entropie-Fehlermaß

Wir betrachten den Fall einer Klassifikationsaufgabe mit N Klassen. Die Neuronenantworten in der Ausgabeschicht sind idealerweise Eins für die richtige Klasse und Null sonst:

$$y^\alpha = (0, \ldots, 0, 1, 0, \ldots, 0)$$

Selbst bei so gewählten Sollantworten werden die Ausgaben des Netzes aufgrund von Ambiguitäten gegen die Wahrscheinlichkeitsverteilung für das Vorliegen der Klassen $C_1 \dots C_N$ konvergieren:

$$y_i(\vec{x}) \sim P_i(\vec{x}) = P(\vec{x} \in C_i), \quad i = 1 \dots N.$$

Um tatsächlich zu garantieren, dass die Netzwerkausgabe eine Wahrscheinlichkeitsverteilung ist, verwenden wir die **Softmax-Aktivierungsfunktion**:

$$y_i = \frac{exp(h_i(x))}{\sum_{j=1}^{N} exp(h_j(x))}, \text{ wobei } h_i(x) = \sum_j \vec{\omega}_{i,j}^{L,L-1} s_j^{L-1} (= a_i^L)$$

Damit sind die Ausgaben insbesondere normiert:

$$\sum_i y_i = 1$$

Ein natürliches Maß, um die beiden Wahrscheinlichkeitsverteilungen (Netzausgaben $y_i(x)$ und wahre Wahrscheinlichkeit $P_i(x)$) miteinander zu vergleichen, bildet die **Kullback-Leibler Distanz**:

$$D = -\sum_\alpha \sum_i P_i(x^\alpha) \log \left(\frac{y_i(\vec{x}^\alpha)}{P_i(x^\alpha)} \right)$$

auch Kreuzentropie genannt. D lässt sich interpretieren als Information, die auf Grundlage der Netzausgabe y_i (als Wahrscheinlichkeit interpretiert) noch fehlt, um die optimale, auf den tatsächlichen Wahrscheinlichkeiten beruhende Klassifikation zu erreichen.
Für den Lernschritt benötigen wir wieder den Gradienten $-\frac{\partial D}{\partial \omega_{i,j}}$:

$$\begin{aligned}
-\frac{\partial D}{\partial \omega_{i,j}^{L,L-1}} &= -\sum_k \frac{\partial D}{\partial y_k} \cdot \frac{\partial y_k}{\partial h_i} \cdot \frac{\partial h_i}{\partial \omega_{i,j}^{L,L-1}} \\
&= \sum_k \sum_\alpha \frac{P_k^\alpha}{y_k^\alpha} \cdot y_k^\alpha (\delta_{k,i} - y_i^\alpha) \cdot s_j^{L-1} \\
&= \sum_\alpha \sum_k P_k^\alpha (\delta_{k,i} - y_i^\alpha) s_j^{L-1} \\
&= \sum_\alpha \Big(P_i^\alpha - y_i \sum_k P_k^\alpha \Big) s_j^{L-1} \\
&= \sum_\alpha \underbrace{(P_i^\alpha - y_i^\alpha(x^\alpha, \omega))}_{\varepsilon^\alpha} s_j^{L-1}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial D}{\partial y_k} &= -\sum_\alpha P_k(x^\alpha) \cdot \frac{P_k^\alpha}{y_k^\alpha} \cdot \frac{1}{P_k^\alpha} \\
\frac{\partial y_k}{\partial h_i} &= \frac{\delta_{ki} e^{h_k} \cdot N - e^{h_i} \cdot e^{h_k}}{N^2} \\
&= \delta_{ki} \frac{e^{h_k}}{N} - \frac{e^{h_i}}{N} \cdot \frac{e^{h_k}}{N} \\
&= \delta_{ki} y_k - y_i \cdot y_k = y_k(\delta_{ki} - y_i) \\
\frac{\partial h_i}{\partial \omega_{ij}^{LL-1}} &= s_j^{L-1}
\end{aligned}$$

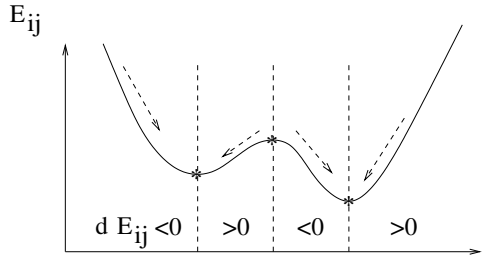
Der resultierende Gradient entspricht demnach dem Gradienten, den wir für ein MLP mit linearem Ausgabeneuron und quadratischer Fehlerfunktion mit Wahrscheinlichkeiten P_i^α als Sollwerten erhalten würden. Praktisch kann ein solches MLP sogar mit Zielwerten $y^{soll} \in \{0, 1\}$ anstelle der P_i^α trainiert werden.

Vorteile von Entropie-Maß + Softmax:

- statistische Fundierung
- "eingebaute" Plateau-Elimination (für die letzte Schicht) trotz nichtlinearer Ausgabe
- i.d.R. bessere Konvergenz
- speziell für 2D Softmax:
  $y_i = \frac{exp(h_i(x))}{\sum_{j=1}^{2} exp(h_j(x))} \to \frac{exp(h_1)}{exp(h_1)+exp(h_2)} = \frac{1}{1+exp(h_2-h_1)}$ = Fermifkt. für Differenz $h_2 - h_1$

1.7 Automatische Schrittweiten Steuerung

- globale Schrittweite (Lernrate) ist brauchbar, aber nicht optimal
- besser: Verwende $\eta_{i,j}$, d.h. eigene Schrittweite für jedes Gewicht
- Grundidee: beobachte $d\,E = \frac{\partial E}{\partial \omega_{i,j}}$



kritische Stellen: Vorzeichenwechsel $w_{i,j}$

Heuristik: falls $d\,E_{i,j}$ in zwei aufeinanderfolgenden Schritten gleiches Vorzeichen hat, vergrößere $\eta_{i,j}$, bei Vorzeichenwechsel verkleinere es z.B. durch **Super-SAB** mit multiplikativen änderungen:

$$\eta_{i,j(t+1)} = \begin{cases} \eta_{i,j}(t) \cdot \eta^+ & \text{falls } d\,E_{i,j}(t) \cdot d\,E_{i,j}(t+1) > 0 \\ \eta_{i,j}(t) \cdot \eta^- & \text{falls } d\,E_{i,j}(t) \cdot d\,E_{i,j}(t+1) < 0 \\ \eta_{i,j}(t) & \text{sonst} \end{cases} \quad \text{mit } 0 < \eta^- < 1 < \eta^+$$

Typische Wahl: $\eta^- = \frac{1}{2}, \eta^+ = 1,05$

1.8 Resilient Backpropagation (RPROP)

(nach Riedmüller, Braun ICANN 93') Eng verwandt mit Super-SAB, jedoch wird $\eta_{i,j}$ nicht als Lernrate verwendet, sondern direkt als Schrittweite für $\omega_{i,j}$. Das Vorzeichen wird aus der Gradientenrichtung $d\,E$ bestimmt:

$$\Delta\omega_{i,j} = -\eta_{i,j}\ \mathrm{sgn}(E_{i,j}) \quad \text{statt} \quad \Delta\omega_{i,j} = -\eta_{i,j}\,\frac{\partial E}{\partial \omega_{i,j}}$$

Aktualisierung von $\eta_{i,j}$ erfolgt wie bisher, eventuell mit anderen η^-, η^+.

Bemerkung:

- alle Beschleunigungsverfahren zur Schrittweitensteuerung erfordern **epochenweise** Berechnung der Gradienten $d\,E_{i,j}$
- typischerweise verschlechtert sich die Generalisierung (Abhilfe durch weight-decay)

1.9 Quadratische Verfahren

Idee. Verwende, dass E in der Nähe eines Minimums näherungsweise quadratisch ist – mit ω^* als Minimum.

Taylor-Entwicklung: $E(\omega^* + \Delta\omega) = E(\omega^*) + \nabla E \Delta\omega + \frac{1}{2}\Delta\omega^T \cdot H(\omega^*)\Delta\omega + O(\Delta\omega^3)$

wobei H die Hesse-Matrix ist:

$$H_{ij,kl} = \frac{\partial^2 E}{\partial\omega_{ij}\partial\omega_{kl}}$$

Jetzt minimiere bzgl. $\Delta\omega$, d.h.

$$0 = \frac{\partial E(\omega^* + \Delta\omega)}{\partial\Delta\omega} = \nabla E + H\Delta\omega \Leftrightarrow \Delta\omega = -H^{-1}\nabla E \text{ (Newton-Richtung)}$$

Dieses $\Delta\omega$ gibt die exakte (beste) Suchrichtung an, in der das Minimum zu finden ist. Praktisch ist die Newton Richtung kaum bestimmbar, denn H ist hochdimensional und schwer zu invertieren. Es ergeben sich verschiedene Verfahren, die H^{-1} auf unterschiedliche Weise approximieren, z.B.:

- verwende nur die Diagonalelemente $\frac{\partial^2 E}{\partial\omega_{ij}^2}$:

- bestimme $\frac{\partial^2 E}{\partial\omega_{ij}^2}$ numerisch mittels $\frac{d\,E_{ij}(t) - d\,E(t-1)}{\Delta\omega_{ij}(t-1)}$ $\rightarrow$ Grundlage für Quick-PROP (QPROP):

$$\Delta\omega_{ij}(t) = \frac{d\,E_{ij}(t)}{d\,E_{ij}(t-1) - d\,E_{ij}(t)}\Delta\omega_{ij}(t-1)$$

- Levenberg-Marquardt-Verfahren:
  Hesse-Matrix nicht notwendig invertierbar
  $\Rightarrow$ Regularisiere H: $\tilde{H} = H + \lambda\mathbf{1}$
  $\Rightarrow \quad \Delta w = -(H + \lambda\mathbf{1})^{-1}\nabla E$

  Levenberg-Marquardt verwendet statt Regularisierung mit $\lambda\mathbf{1}$ aber $\lambda \operatorname{diag} H$, um auch für große λ noch sinnvolle, von H abhängige Updateschritte zu machen:

  $\tilde{H} = H + \lambda \operatorname{diag} H$
  $\Delta w = -(H + \lambda \operatorname{diag} H)^{-1}\nabla E$

  $\lambda \gg 1 \quad \rightarrow \quad$ Gradientenabstieg (mit Schrittweite $\approx \lambda^{-1}$)
  $\lambda \ll 1 \quad \rightarrow \quad$ Newton-Verfahren

  verringere λ multiplikativ falls $E(t+1) < E(t)$
  vergrößere λ multiplikativ sonst

2 Support Vector Machine (SVM)

2.1 Optimaler Linearer Klassifikator

Das einfachste Modell einer SVM betrachtet ein linear separables 2-Klassen-Problem, d.h. es sind Datenbeispiele $\{\vec{x}^\alpha, y^\alpha\}$ mit $\vec{x}^\alpha \in \mathbb{R}^d$ und $y^\alpha \in \{-1, 1\}$ gegeben. Wie wir bereits beim Perzeptron gesehen haben, kann dieses Problem mittels eines einfachen linearen Klassifikators gelöst werden:

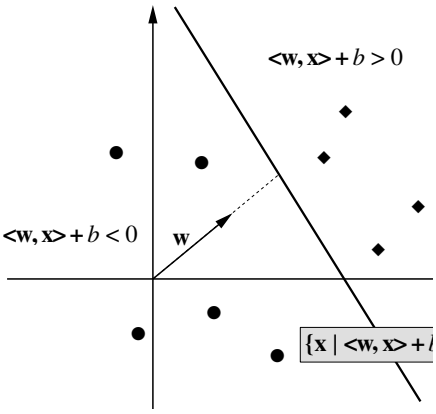
$$y(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x} + b) = \text{sgn}(\langle \vec{w}, \vec{x} \rangle + b)$$

wobei es $\vec{w} \in \mathbb{R}^d$ und $b \in \mathbb{R}$ gibt, so dass

$$\begin{aligned} y(\vec{x}^\alpha) &< 0 \quad \text{für alle } \alpha \text{ mit } y^\alpha < 0 \text{ und} \\ y(\vec{x}^\alpha) &> 0 \quad \text{für alle } \alpha \text{ mit } y^\alpha > 0 \end{aligned}$$

oder zusammengefaßt:

$$y^\alpha \cdot y(\vec{x}^\alpha) > 0 \quad \text{für alle } \alpha$$

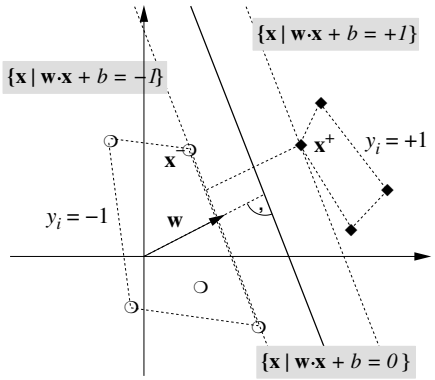


Ansatz der SVM: *Die Trennebene ist optimal (im Sinne höchster Generalisierungsfähigkeit), wenn der Abstand zu den Datenbeispielen maximal ist.*
Diese optimale Trennebene ist orthogonal zur kürzesten Verbindungslinie zwischen den konvexen Hüllen der beiden Klassen und liegt in der Mitte dieser Verbindungslinie.

Für die nächstliegenden Punkte $\vec{x}^+$ bzw. $\vec{x}^-$ gelte:

$$\begin{aligned} y(\vec{x}^+) &= \vec{w} \cdot \vec{x}^+ + b = +\varepsilon \qquad \text{und} \\ y(\vec{x}^-) &= \vec{w} \cdot \vec{x}^- + b = -\varepsilon \end{aligned}$$

Dann gilt: $y^\alpha \cdot y(\vec{x}^\alpha) = y^\alpha \cdot (\vec{w} \cdot \vec{x}^\alpha + b) \geq \varepsilon \quad \forall \alpha$



Der Abstand (Margin) zwischen den Klassen ist die Projektion von $\vec{x}^+ - \vec{x}^-$ auf $\hat{\vec{w}}$:

$$M = \frac{\vec{w}}{\|\vec{w}\|} \cdot (\vec{x}^+ - \vec{x}) = \frac{2\varepsilon}{\|\vec{w}\|}$$

Man normalisiert $\vec{w}$ und b typischerweise so, dass $\varepsilon = 1$ gilt (kanonische Form).
Der Abstand ist also umgekehrt proportional zur Norm $\vec{w}$. Eine Maximierung des Abstand führt daher zu folgendem Optimierungsproblem:

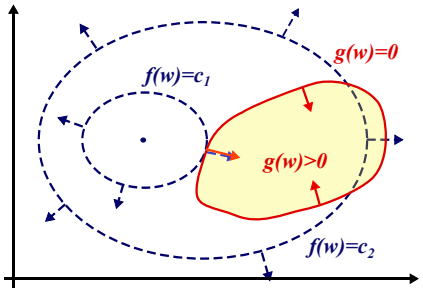
$$\begin{aligned} \text{Minimiere} \quad & ||w||^2 \\ \text{mit Nebenbedingungen} \quad & y^\alpha \cdot (\vec{w} \cdot \vec{x}^\alpha + b) \geq 1 \quad \forall \alpha \end{aligned}$$

2.1.1 Exkurs: Karush-Kuhn-Tucker-Methode zur Lösung konvexer Optimierungsprobleme mit Nebenbedingungen

$$\begin{aligned} \text{Minimiere} \quad & f(\vec{w}) \\ \text{mit Nebenbedingungen} \quad & g_\alpha(\vec{w}) = 0 \quad \alpha = 1, \ldots, M \qquad g_\beta(\vec{w}) \geq 0 \quad \beta = 1, \ldots, N \end{aligned}$$

1. Verwende Lagrange-Funktion

$$L(\vec{w}, \vec{\lambda}) = f(\vec{w}) - \sum_{\alpha=1}^{M} \lambda_\alpha \cdot g_\alpha(\vec{w}) - \sum_{\beta=1}^{N} \lambda_\beta \cdot g_\beta(\vec{w})$$



mit Lagrange-Multiplikatoren $\lambda_\alpha, \lambda_\beta$.

Minimiere L bzgl. $\vec{w}$ und λ, wobei $\lambda_\beta \geq 0$.

Was bewirken die zusätzlichen Terme?

Betrachten wir zunächst die Gleichheitsbedingung $g_\alpha = 0$. Diese Bedingung definiert eine $(d-1)$-dimensionale Mannigfaltigkeit im Parameterraum. Der Gradient $\nabla g(\vec{w})$ ist immer orthogonal zu dieser Mannigfaltigkeit. An einem Punkt $\vec{w}$ auf der Mannigfaltigkeit, der $f(\vec{w})$ minimiert, ist auch $\nabla f(\vec{w})$ orthogonal zur Mannigfaltigkeit, d.h.

$$\nabla f \parallel \nabla g \quad \Leftrightarrow \quad \nabla f + \lambda \nabla g = 0 \text{ für ein } \lambda \neq 0. \tag{2.1}$$

Falls dies nicht der Fall wäre, könnte man sich noch ein Stück tangential auf der Mannigfaltigkeit bewegen und die Funktion f weiter minimieren.

Betrachten wir nun die Nebenbedinung $g_\alpha \geq 0$. Entweder, das Minimum liegt im Inneren ($g_\alpha > 0$, dann spielt die Bedingung für die Optimierung keine Rolle) oder auf dem Rand (dann haben wir den vorigen Fall $g_\alpha = 0$).

Im ersten Fall kann durch Wahl von $\lambda = 0$ die zusätzliche Bedingung "abgeschaltet werden". Im letzteren Fall muss wie oben ein $\lambda \neq 0$ gewählt werden, genauer: $\nabla f = \lambda \nabla g$ mit $\lambda > 0$. Achtung: Hier kommt es jetzt auf das Vorzeichen von λ an, weil die Ungleichheitsbedingung eine Vorzugsrichtung vorgibt!

Betrachten wir das Zusammenspiel zwischen Minimierung bzgl. $\vec{w}$ und λ nochmal aus einem anderen Blickwinkel. Nehmen wir an, die Bedingung $g_\alpha \geq 0$ sei erfüllt. Die Minimierung von L bzgl. λ_α führt dazu, dass λ_α einen möglichst kleinen Wert annimmt ($\lambda_\alpha = 0$). Dann spielt der zusätzliche Term also keine Rolle mehr.

Ist die Nebenbedingung umgekehrt nicht erfüllt ($g_\alpha < 0$), versucht λ_α einen möglichst großen Wert anzunehmen, was zu einer Verkleinerung von L führt. Die Minimierung von L bzgl. $\vec{w}$ arbeitet dagegen an und verändert $\vec{w}$ so, dass – bei festem λ_α – die Nebenbedingung besser erfüllt wird.
Die Lagrange-Multiplikatoren können somit als Maß für die Abhängigkeit der Lösung von der jeweiligen Nebenbedingung interpretiert werden. Je größer ihr Wert, desto stärker beeinflußt die Nebenbedingung die optimale Lösung.

2. $\vec{w}^*$ ist Lösung des Optimierungsproblems falls

$$\begin{aligned} \nabla_{\vec{w}} L(\vec{w}^*, \lambda^*) &= 0 && \text{notwendige Bedingung für Sattelpunkt} \\ \lambda^*_\alpha \cdot g_\alpha(\vec{w}^*) &= 0 && \text{KKT-Komplementärbedingungen} \\ g_\alpha(\vec{w}^*) \geq 0 \quad \text{und} \quad \lambda^*_\alpha &\geq 0 \end{aligned}$$

An den KKT-Komplementärbedingungen erkennt man, dass Bedingungen die sicher erfüllt sind ($g_\alpha(\vec{w}^*) > 0$) einen Multiplikator $\lambda_\alpha = 0$ haben und damit nicht bei der Optimierung berücksichtigt werden. *Nur Bedingungen, die gerade noch erfüllt sind ($g_\alpha(\vec{w}^*) = 0$), haben positive Multiplikatoren und werden somit in der Optimierung berücksichtigt.*

1. Lagrange-Funktion für SVM-Optimierungsproblem:
$$L(\vec{w}, b, \vec{\lambda}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{\alpha=1}^{M} \lambda_\alpha \cdot (y^\alpha \cdot (\vec{w} \cdot \vec{x}^\alpha + b) - 1)$$

2. Löse Sattelpunktsbedingung
$$\nabla_{\vec{w}} L = \vec{w} - \sum_{\alpha=1}^{M} y^\alpha \lambda_\alpha \vec{x}^\alpha \stackrel{!}{=} 0 \qquad \Leftrightarrow \qquad \vec{w}^* = \sum_{\alpha=1}^{M} \lambda_\alpha y^\alpha \vec{x}^\alpha \tag{2.2}$$
$$\nabla_b L = \sum_{\alpha=1}^{M} y^\alpha \lambda_\alpha \stackrel{!}{=} 0$$
Der Gewichtsvektor ist eine Linearkombination der Datenpunkte $\vec{x}^\alpha$, wobei nur diejenigen berücksichtigt werden, für die $\lambda_\alpha \neq 0$ gilt, deren Nebenbedingungen also gerade noch erfüllt sind: $y^\alpha \cdot (\vec{w} \cdot \vec{x}^\alpha + b) = 1$. Dies sind offenbar diejenigen Vektoren, die am nächsten an der Trennebene dranliegen (Support-Vektoren). Das entspricht genau unserer Intuition: die optimale Trennebene wird natürlich nur durch diese Support-Vektoren bestimmt, die anderen Vektoren sind für die Trennebene irrelevant.

3. Setze in $L(\vec{w}, b, \vec{\lambda})$ ein und löse das duale Problem:
$$\begin{aligned} \text{maximiere} \quad W(\vec{\lambda}) &= L(\vec{w}^*, b^*, \vec{\lambda}) \\ &= \tfrac{1}{2}\vec{w}^{*T}\vec{w}^* - \textstyle\sum_\alpha \lambda_\alpha \cdot \big(y^\alpha \cdot (\vec{w}^*\vec{x}^\alpha + b^*) - 1\big) \\ &= \tfrac{1}{2}\vec{w}^{*T}\vec{w}^* - \vec{w}^* \underbrace{\sum_\alpha \lambda_\alpha y^\alpha \vec{x}^\alpha}_{\vec{w}^*} - b^* \underbrace{\sum_\alpha \lambda_\alpha y^\alpha}_{=0} + \textstyle\sum_\alpha \lambda_\alpha \\ &= \sum_{\alpha=1}^{M} \lambda_\alpha \; - \; \frac{1}{2} \sum_{\alpha,\beta=1}^{M} \lambda_\alpha \lambda_\beta y^\alpha y^\beta \cdot (\vec{x}^\alpha \cdot \vec{x}^\beta) \end{aligned}$$
$$\text{mit Nebenbedingungen} \quad \lambda_\alpha \geq 0 \qquad \text{und} \qquad \sum_{\alpha=1}^{M} y^\alpha \lambda_\alpha = 0$$
Die Optimierung erfolgt numerisch mittels Standard-Verfahren der quadratischen Programmierung.

Einsetzen von (2.2) in den Klassifikator ergibt:
$$y(\vec{x}) = \operatorname{sgn}\Big(\sum_{\substack{\alpha \\ \lambda_\alpha \neq 0}} \lambda_\alpha y^\alpha \cdot (\vec{x} \cdot \vec{x}^\alpha) + b^* \Big)$$
$$b^* = -\tfrac{1}{2}\left(\max_{y^\alpha=-1} \vec{w}^* \cdot \vec{x}^\alpha \; + \; \min_{y^\alpha=+1} \vec{w}^* \cdot \vec{x}^\alpha\right)$$

2.2 Numerische Optimierung des dualen Problems

- quadratisches, konvexes Problem

→ Lösung existiert und kann effizient berechnet werden
(quadratische Programmierung, Interior-Point-Methoden, etc.)

- häufig benötigt Matrix $(K_{\alpha\beta}) = \vec{x}^\alpha \cdot \vec{x}^\beta, \quad K \in \mathbb{R}^{M \times M}$

→ extrem speicheraufwendig bei vielen Daten

- Lösung: Beschränkung auf (noch unbekannte) Supportvektoren

Chunking: schrittweise Bestimmung der Menge der Supportvektoren

1. Initialisierung: $t = 0, \vec{w}^0 = 0, b^0 = 0, T^0 = \{\}$
   Zerlegung der Trainingsmenge D in disjunkte Teilmengen D_i
2. $t \leftarrow t + 1$
   Forme neue Trainingsmenge aus bisherigen Supportvektoren und potentiellen neuen Supportvektoren:
   $$T^t = \{\alpha \in T^{t-1} \mid \lambda_\alpha > 0\} \cup \{\alpha \in D_t \mid y^\alpha \cdot (\vec{w}^{t-1} \cdot \vec{x}^\alpha + b^{t-1}) < 1\}$$
3. Maximiere $W(\vec{\lambda})$ mit neuer Bestlösung $\vec{w}^t, b^t, \vec{\lambda}^t$

Sequential Minimal Optimisation (SMO)

- Optimiere immer nur bzgl. zweier Datenbeispiele

$\Rightarrow$ Lösung analytisch möglich

- viele Iterationen aber aufgrund guter Auswahl-Heuristik trotzdem sehr effizient
- linearer Speicheraufwand

2.3 Der Kerneltrick

Problem: Die SVM ist ein rein linearer Ansatz. Viele Probleme sind aber nichtlinear.
Idee: Benutze vorgeschaltete nichtlineare Feature-Extraktion $\vec{\phi} = \Phi(\vec{x})$, um die Daten in einen hochdimensionalen Feature-Raum $\mathbb{R}^d$ zu transformieren und verwende dort die lineare SVM. **Problem:** Berechnung im hochdimensionalen Feature-Space (speicher)aufwendig.
Aber: Benötigt wird immer nur das *Skalarprodukt* $\Phi(\vec{x}) \cdot \Phi(\vec{x}')$.
Idee: Ersetze Skalarprodukt durch Kernel $K(\vec{x}, \vec{x}') = \Phi(\vec{x}) \cdot \Phi(\vec{x}')$:

$$\begin{aligned} & & f(\vec{x}) &= \vec{w}^* \cdot \Phi(\vec{x}) + b^* \\ \text{mit} & & \vec{w}^* &= \sum_\alpha \lambda_\alpha y^\alpha \Phi(\vec{x}^\alpha) \\ \text{eingesetzt:} & & f(\vec{x}) &= \sum_\alpha \lambda_\alpha y^\alpha \underbrace{\Phi(\vec{x}^\alpha) \cdot \Phi(\vec{x})}_{K(\vec{x}^\alpha, \vec{x})} + b^* \end{aligned}$$

Berechne direkt $K(\cdot,\cdot)$, $\Phi(\cdot)$ wird nie explizit benötigt.

Beispiel 2.3.1. 2-dim Daten $(x_1, x_2) \in \mathbb{R}^2$

$$\begin{aligned} K(\vec{x}, \vec{z}) &= (\vec{x} \cdot \vec{z} + 1)^2 = \big((x_1, x_2)^t \cdot (z_1, z_2)^t + 1\big)^2 \\ &= (x_1 z_1 + x_2 z_2 + 1)^2 = x_1^2 z_1^2 + x_2^2 z_2^2 + 1^2 + 2x_1 x_2 z_1 z_2 + 2x_1 z_1 + 2x_2 z_2 \\ &= (x_1^2, x_2^2, 1, \sqrt{2}\, x_1 x_2, \sqrt{2}\, x_1, \sqrt{2}\, x_2)^t \cdot (z_1^2, z_2^2, 1, \sqrt{2}\, z_1 z_2, \sqrt{2}\, z_1, \sqrt{2}\, z_2)^t \\ (x_1, x_2)^t \mapsto \Phi(\vec{x}) &= (x_1^2, x_2^2, 1, \sqrt{2}\, x_1 x_2, \sqrt{2}\, x_1, \sqrt{2}\, x_2)^t \end{aligned}$$

Mercer's Bedingung
Frage: Welche $K(\cdot,\cdot)$ sind erlaubt, d.h. entsprechen Skalarprodukten für irgendein Φ?

- symmetrisch: $K(x, z) = K(z, x)$
- positiv semi-definit: $\int g(x) K(x, y) g(y)\, dxdy \geq 0$ für alle $g(x) \in L^2$

Häufig verwendete Kernel:

- $K(x, x') = e^{-\frac{||x-x'||^2}{2\sigma^2}}$ – RBF- oder Gauß-Kernel
- $K(x, x') = (\langle x, x' \rangle + 1)^p,\ p \in \mathbb{N}^+$ – Polynome bis p-ten Grades
- $K(x, x') = \tanh(\langle x, x' \rangle + \theta)$ – sigmoides Neuron
- häufig sind „verbotene Kernel“ in Praxis trotzdem einsetzbar

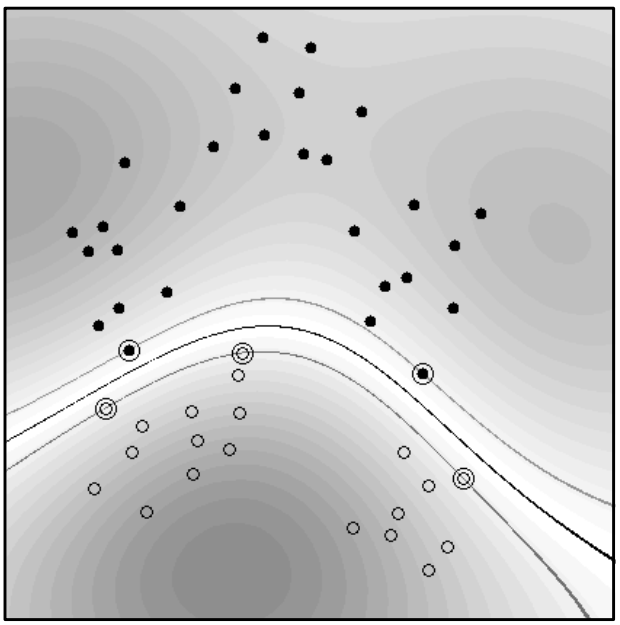


Abbildung 1: 2-Klassen-Problem mit RBF-Kernel: Supportvektoren sind eingekreist, dunklere Regionen entsprechen einer sichereren Entscheidung.

Bemerkungen

- $|\vec{w}^* \cdot \vec{x} + b^*|$ ist Maß für Abstand von Trennebene = Sicherheit der Klassifikation
- Kernel müssen/können datenspezifisch optimiert werden
- Kernel-Parameter, z.B. Radius σ beim RBF-Kernel müssen (außerhalb der SVM) optimiert werden (Meta-Parameter)
- für Multiklassenprobleme sind iterative eine-gegen-alle Verfahren nötig
- weiterführende Links: www.kernel-machines.org

2.4 Soft-Margin Klassifikator: Generalisierung auf nicht-separable Probleme

Bislang haben wir angenommen, dass Klassifikationsproblem sei linear separabel. Häufig ist dies jedoch nicht der Fall, z.B. aufgrund verrauschter Trainingsdaten.
Lösung: Einführung von Schlupf- oder Slackvariablen $\xi_\alpha \geq 0$ zur „Aufweichung“der Nebenbedingungen:

$$y^\alpha \cdot (\vec{w} \cdot \vec{x}^\alpha + b) \geq 1 - \xi_\alpha \quad \forall \alpha$$

Falls $\xi_\alpha = 0$ erhalten wir die ursprünglichen Bedingungen, falls $\xi_\alpha > 0$ sind Abweichungen davon erlaubt. Diese Abweichungen sollen natürlich klein bleiben. Wir minimieren also

$$\text{Minimiere} \quad f(\vec{w}) = \tfrac{1}{2}\|\vec{w}\|^2 + C\sum_\alpha \xi_\alpha$$

wobei $C > 0$ die Gewichtung zwischen Korrektheit und Softness bestimmt. Die Anwendung der KKT-Theorie führt zu ein ähnlichen dualen Optimierungsproblem wie zuvor:

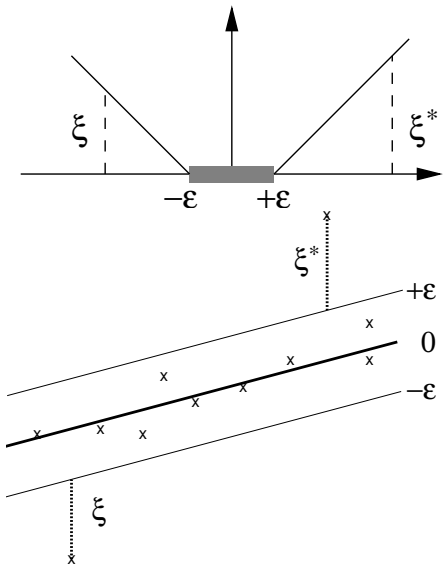
$$\begin{aligned} &\text{Maximiere} && W(\vec{\lambda}) \\ &\text{mit Nebenbedingungen} && 0 \le \lambda_\alpha \le C \qquad \text{und} \qquad \sum_{\alpha=1}^{M} y^\alpha \lambda_\alpha = 0 \end{aligned}$$

Der Einfluß der einzelnen Nebenbedingungen wird also nach oben hin durch C beschränkt. Support-Vektoren sind wieder diejenigen, für die die Nebenbedingung gerade noch erfüllt sind ($y^\alpha \cdot (\vec{w} \cdot \vec{x}^\alpha + b) = 1 - \xi_\alpha$). Hinzu kommen also noch alle Vektoren, die falsch klassifiziert werden ($\xi_\alpha > 0 \;\Leftrightarrow\; \lambda_\alpha = C$).

2.5 Support Vector Regression

Ziel: beliebige Funktionsapproximation $f : \mathbb{R}^d \to \mathbb{R}$.
Lösung: Lege Gerade bzw. Hyperebene durch die Daten und bestrafe Abweichungen wieder mit Slack-Variablen.

Approximationsansatz: $f(\vec{x}) = \vec{w} \cdot \vec{x} + b$
Kostenfunktion: $|y - f(\vec{x})|_\varepsilon = \max\{0, |y - f(\vec{x})| - \varepsilon\}$
(ε-insensitive loss – ignoriere Fehler kleiner als ε)



$$\begin{aligned} &\text{Minimiere} && \tfrac{1}{2}\|\vec{w}\|^2 \;+\; C\sum_{\alpha=1}^{M}(\xi_\alpha + \xi_\alpha^*) \\ &\text{mit Nebenbedingungen} && y^\alpha - (\vec{w} \cdot \vec{x}^\alpha + b) \le \varepsilon + \xi_\alpha^* \\ &&& (\vec{w} \cdot \vec{x}^\alpha + b) - y^\alpha \le \varepsilon + \xi_\alpha \\ &&& \xi_\alpha, \xi_\alpha^* \ge 0 \end{aligned}$$

Die ξ_α^* bestrafen positive Abweichungen,
die ξ_α negative Abweichungen.

Die Lösung erfolgt wieder analog zum Klassifikationsfall mittels der KKT-Theorie. Das duale Problem lautet nun:

$$\begin{aligned} \text{Maximiere} \quad W(\vec{\lambda}, \vec{\lambda}^*) = &-\varepsilon \sum_{\alpha=1}^{M}(\lambda_\alpha^* + \lambda_\alpha) + \sum_{\alpha=1}^{M} y^\alpha \cdot (\lambda_\alpha^* - \lambda_\alpha) \\ &- \frac{1}{2}\sum_{\alpha,\beta=1}^{M} (\lambda_\alpha^* - \lambda_\alpha)(\lambda_\beta^* - \lambda_\beta) \cdot (\vec{x}^\alpha \cdot \vec{x}^\beta) \end{aligned}$$

$$\text{mit Nebenbedingungen} \qquad 0 \le \lambda_\alpha, \lambda_\alpha^* \le C \qquad \text{und} \qquad \sum_{\alpha=1}^{M}(\lambda_\alpha^* - \lambda_\alpha) = 0$$

Die gesuchte Approximationsfunktion ergibt sich wieder als Linearkombination von Supportvektoren:

$$f(\vec{x}) = \sum_{\alpha=1}^{M} (\lambda_\alpha^* - \lambda_\alpha) \cdot (\vec{x}^\alpha \cdot \vec{x}) + b^*$$

wobei nur Datenbeispiele α eingehen, für die entweder $\lambda_\alpha > 0$ oder $\lambda_\alpha^* > 0$ ist. Dies sind die Datenbeispiele, die außerhalb des ε-Schlauches liegen, d.h. $|f(\vec{x}^\alpha) - y^\alpha| \geq \varepsilon$.

3 Topologische Merkmalskarten

3.1 Selbstorganisierende, Topologie-erhaltende Merkmalskarten (SOM)

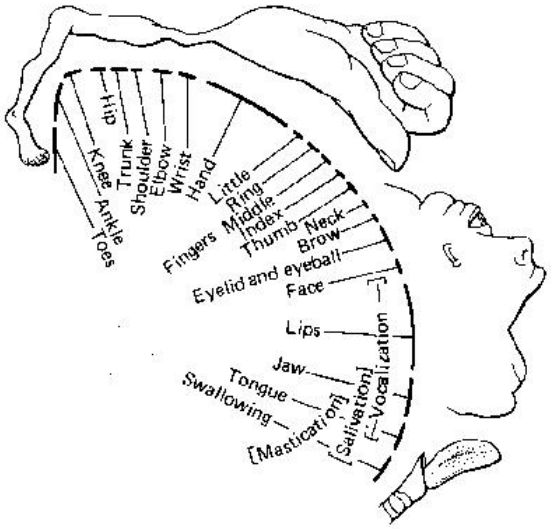
3.1.1 Fragestellung und Motivation

Wie gelangt man von „low-level“ sensorischen Daten zu symbolischen, „high-level“ Abstraktionen? (und zurück)?
Teilaspekte:

- Übergang von „physikalischen“ zu logischen Repräsentationen
- Ausblenden unwesentlicher Informationen
- Hervorhebung wesentlicher, struktureller Informationen („Topologie“)
- Bildung von Prototypen (Clusterung / symbolische Repräsentation)
- Lernen

Merkmalskarten im Gehirn (Cortex):

- visuelle, auditive und andere sensorische Karten – Reizwahrnehmung
- motorische Karten – Ansteuerung der Muskeln



- topographische Ordnung:
  ähnliche Reize führen zur Erregung benachbarter Neuronen
- Cortex kann schnell umlernen:
  - ◇ Bei Verlust eines Fingers erhöht sich die Empfindsamkeit und Beweglichkeit benachbarter Finger.
  - ◇ Blinde nutzen Gehirnregionen zum Hören und Fühlen, die üblicherweise zum Sehen genutzt werden.
- Minimierung von „Verdrahtungskosten“

Frage: Wie kann die Ausbildung solcher topologischer Merkmalskarten erklären?

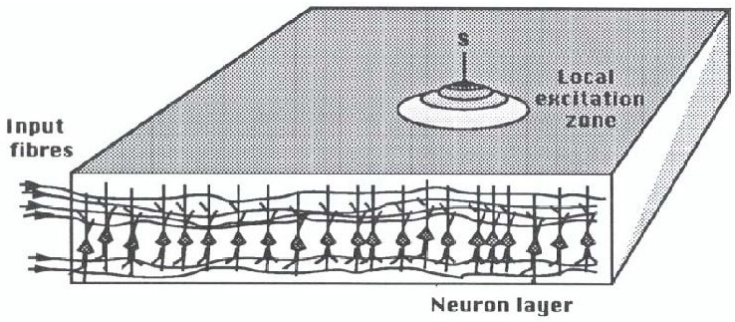


Abbildung 2: Kohonen-Modell: In einer zweidimensionalen Schicht bildet sich aufgrund des Wettbewerbs der Neuronen untereinander (laterale Inhibition) ein eng lokalisierter Peak heraus, der an der Stelle maximaler Input-Antwort $\vec{w} \cdot \vec{x}$ lokalisiert ist.

3.1.2 Kohonen-Modell

Ein erstes Modell dafür lieferte der finn. Wissenschaftler Teuvo Kohonen.

Ausgangssituation

- zweidimensionale Schicht von Neuronen
- Neuron c am Ort $\vec{c}$ mit Erregung bzw. Aktivierung y_c
- Eingabe $\vec{x} \in \mathbb{R}^L$ und zug. Gewichte $\vec{w}_c \in \mathbb{R}^L$
- laterale Inhibition der Neuronen untereinander über Gewichte $g_{cc'}$
- $y_c = \sigma\big(\underbrace{\vec{w}_c \vec{x}}_{\text{Reaktion auf Input}} + \underbrace{\sum_{c'} g_{cc'} y_{c'}}_{\text{laterale Inhibition}} - \underbrace{\theta}_{\text{threshold}} \big)$
- kurzreichweitige Erregung, langreichweitige Hemmung
- Wettbewerb ($g_{cc'}$) so austariert, dass Zelle n mit maximaler Reaktion auf die Eingabe zum Zentrum einer lokalen Erregungszone wird: Folie

$$n = \arg\max_c \; y_c \qquad \text{(WTA: winner takes all)}$$

- Lernregel: Verschiebung aller Prototypen in Richtung des Stimulus $\vec{x}$:

$$\Delta\vec{w}_c = \eta \cdot y_c \cdot (\vec{x} - \vec{w}_c) \qquad \text{(Hebb-Regel + Decay)}$$

η: Lernrate, y_c: Aktivität

3.1.3 SOM – mathematische Abstraktion von Kohonen's Modell

Vereinfachungen:

- festes Gitter von Knoten bzw. Neuronen mit zugehörigen Gewichten $w_c \in \mathbb{R}^L$
  Indizierung der Neuronen über Gitterposition $\vec{c}$

- Bestimme **direkt** das Neuron n mit maximaler Reaktion auf die Eingabe:

$$n = \arg\max_c \ \vec{w}_c \cdot \vec{x} \qquad \text{(WTA: winner takes all)}$$

- Ersetze Skalarprodukt $\vec{x} \cdot \vec{w}$ durch Abstand $\|\vec{x} - \vec{w}\|$:

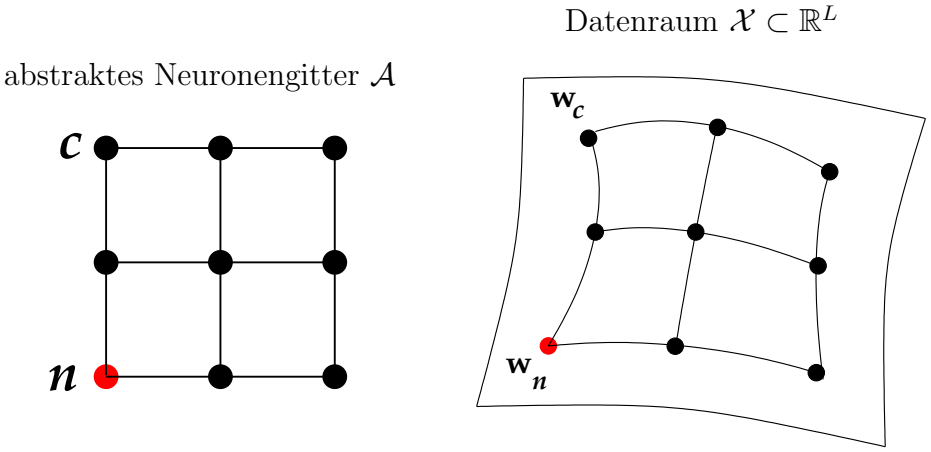
$$2\,\vec{x} \cdot \vec{w} = \|\vec{x}\|^2 + \|\vec{w}\|^2 - \|\vec{x} - \vec{w}\|^2$$
$$\Rightarrow \quad n = \arg\min_c \|\vec{x} - \vec{w}_c\| \qquad \text{(WTA: winner takes all)}$$

Damit können die Gewichte $\vec{w}_c$ als **Prototypen** für die Eingabe interpretiert werden.

- Ersetze y_c in der Lernregel durch lokalisierte Gaussverteilung:

$$y_c \quad \longrightarrow \quad h_{cn} = \mathrm{e}^{-\frac{\|\vec{c}-\vec{n}\|^2}{2\sigma^2}}$$

Visualisierung



Die den Neuronen zugeordneten Gewichtsvektoren werden als Prototypen im Eingabe- bzw. Datenraum eingezeichnet. Ausserdem wird die Gitterstruktur, die ja zunächst nur im abstrakten Gitter existiert auf den Eingaberaum übertragen, so dass dort die durch die SOM repräsentierte Mannigfaltigkeit (d.h. die gekrümmte Fläche) sichtbar wird.

Algorithmus

1. Wähle Stimulus $\vec{x}^\alpha$ zufällig
2. Bestimme Gewinnerneuron n mit

$$n = \arg\min_c \|\vec{x}^\alpha - \vec{w}_c\|$$
$$\text{d.h.} \quad \|\vec{x}^\alpha - \vec{w}_n\| < \|\vec{x}^\alpha - \vec{w}_c\| \quad \forall c \neq n$$

3. Lernschritt:

$$\Delta w_c = \eta \cdot h_{cn} \cdot (\vec{x}^\alpha - \vec{w}_c) \quad \text{für alle Neuronen } c$$

wobei η Lernrate und h_{cn} Nachbarschaftsfunktion ist: $h_{cn} = \mathrm{e}^{-\frac{\|\vec{c}-\vec{n}\|^2}{2\sigma^2}}$

4. (a) Verringere die Lernrate η sowie die Reichweite σ der Nachbarschaft:
   $\Rightarrow$ das Netz wird weniger sensitiv auf Ausreißer
   (b) Verwende adaptive Schrittweitensteuerung für Lernrate und Nachbarschaft
5. GOTO 1 bis Qualität zufriedenstellend ist
   oder „keine Veränderungen“ mehr eintreten

Bemerkungen

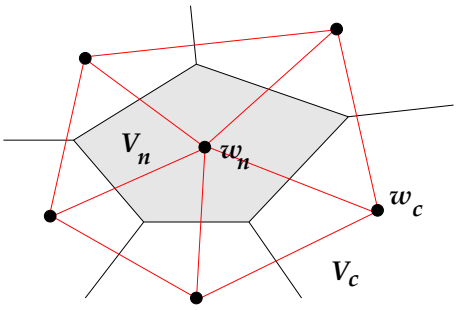
- Prototypen $\vec{w}_c$ sind entsprechend der Gitter-Nachbarschaft durch „Gummibänder“ verbunden
- Lernregel zieht Winner-Prototyp in Richtung des Stimulus (VQ-Lernregel)
- aufgrund der Gummibänder (h_{cn}) werden benachbarte Prototypen (im Sinne der Gitterstruktur) ebenfalls adaptiert (garantiert glatte Mannigfaltigkeit)
- das σ der Exponentialfunktion steuert die Reichweite der Änderungen
- Antwort des Gewinnerneuron n auf wiederkehrende Muster wird erhöht
- topologisch benachbarte Neuronen sind auf ähnliche Muster spezialisiert
- *unüberwachtes Lernverfahren*

Anwendungen Je nachdem, ob der Koordinatenindex $\vec{n}$ des Gewinner-Neurons oder dessen zug. Gewichtsvektor $\vec{w}_n$ als Ausgabe der SOM gewählt wird, erhält man zwei unterschiedliche Anwendungen:

- lokales Koordinatensystem: Abbildung $\Phi : \mathcal{X} \to \mathcal{A} = \{\vec{c}\}$
  $\Phi(\vec{x}) = \vec{n}$ liefert lokale Koordinaten der Datenmannigfaltigkeit. Da die Beschreibungsdimension L der Daten häufig deutlich größer als die Dimension m des SOM-Gitters ist, erhält man somit auch eine *nichtlineare Dimensionsreduktion*.
- Vektorquantisierung: Abbildung $\Phi : \mathcal{X} \to \mathcal{X}_d = \{\vec{w}_c\}$
  Die Daten $\vec{x}$ werden auf den nächsten Prototyp $\Phi(\vec{x}) = \vec{w}_n$ abgebildet.

Voronoizelle Die Menge aller Daten, für die n Gewinner-Neuron ist heißt *Voronoizelle* zu n

$$V_n = \{\vec{x} | \quad ||\vec{x} - \vec{w}_n|| \leq ||\vec{x} - \vec{w}_c|| \quad \forall c\}$$



- Die Grenzen der Voronoizellen sind die Mittelsenkrechten der Verbindungslinien zwischen zwei Knoten.
- Voronoizellen sind konvex, d.h. die Verbindungsstrecke zweier Punkte ist auch in der Voronoizelle enthalten.
- Die **Delaunay-Triangulierung** entsteht durch die Verbindung aller Knoten i und j, deren Voronoizellen V_i und V_j eine gemeinsame Kante haben. Sie repräsentiert die Topologie der Neuronen.

Vergleich zur Standard-Vektorquantisierung (VQ) VQ $\approx$ SOM ohne Nachbarschaft. Der Standard-VQ-Algorithmus adaptiert *nur* den Winner-Knoten. Die VQ-Lernregel

$$\Delta w_n = \eta \cdot (\vec{x}^{\alpha} - \vec{w}_n)$$

ist interpretierbar als stochastischer Gradient von

$$E(w) = \frac{1}{2} \int_x (\vec{w}_s - \vec{x})^2 \; p(x) \; d^d\vec{x}$$

wobei $x \in \mathbb{R}^L$ und $p(\vec{x})$ die Datendichte ist. Der VQ-Algorithmus versucht also den *Quantisierungsfehler* E zu minimieren.
Man kann zeigen, dass

$$\langle \Delta\vec{w}_n \rangle = 0 = \eta \left(\int_{V_n} \vec{x} \; p(\vec{x}) \; d^L\vec{x} - \vec{w}_n \int_{V_n} p(\vec{x}) \; d^L\vec{x} \right)$$

ein stationärer Punkt des VQ-Verfahrens ist. Auflösen nach $\vec{w}_n$ führt zu

$$\vec{w}_n = \frac{\int_{V_n} \vec{x} \cdot p(\vec{x}) \; d^L\vec{x}}{\int_{V_n} p(\vec{x}) \; d^L\vec{x}}$$

Für VQ liegen die Prototypen $\vec{w}_n$ also im Erwartungswert der Daten ihrer Voronoizelle V_n. VQ berechnet diese Prototypen iterativ, anhand der Beispieldaten $\{\vec{x}^{\alpha}\}$.

Der SOM-Algorithmus ist eine Erweiterung des VQ-Verfahrens unter Einbeziehung von Nachbarschaft. Bei variabler Reichweite σ der Nachbarschaft existiert allerdings keine allgemeine Fehlerfunktion mehr. Für festes σ minimiert SOM den Fehler:

$$E = \left\langle \frac{1}{2} \sum_c h_{cn} \cdot ||\vec{x} - \vec{w}_c||^2 \right\rangle_{\vec{x}}$$

Wie vorher gilt

$$\vec{w}_n = \frac{\sum_c h_{cn} \cdot \int_{V_c} \vec{x} \cdot p(\vec{x}) \; d^L\vec{x}}{\sum_c h_{cn} \cdot \int_{V_c} p(\vec{x}) \; d^L\vec{x}}$$

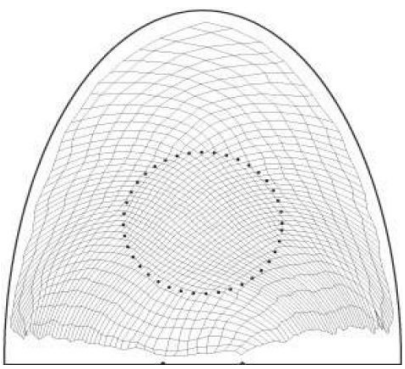
weitere Eigenschaften der SOM

- **Dichteverteilung**: SOM zeigt Fovealeffekt, d.h. es gibt höhere Auflösung in Regionen hoher Datendichte. Formal definiere Vergrößerungsfaktor $m(\vec{x})$ = mittlere Anzahl von Neuronen pro Volumenelement $d^L\vec{x}$, d.h.

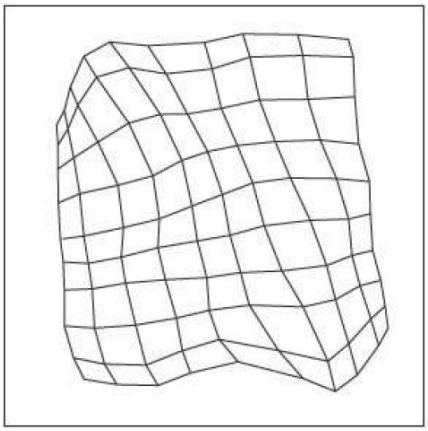
  $$\int m(x) \; d^L x = l = \text{Anzahl der Neuronen}$$

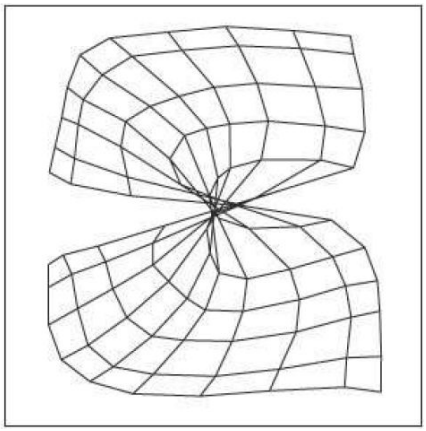
  Dann gilt $m(x) \propto p^{\frac{2}{3}}$ (Ritter, 91)

  In Regionen hoher Datendichte häufen sich die Knoten.



- **Topologische Ordnung / Topologische Defekte**
  Topologische Defekte liegen vor, wenn die durch die SOM abgebildete Nachbarschaftsrelation nicht der Nachbarschaftsrelation der Daten entspricht. Sie entstehen, wenn die Nachbarschaftsreichweite σ zu schnell verringert wird und können i.A. nur schwer detektiert werden.





3.2 Growing Neural Gas

(Fritzke 1994/95 [?, ?])

3.2.1 Motivation

- SOM setzt feste Netzwerkgröße voraus, d.h. bekannte Topologie
- suche Verfahren zum unüberwachten Lernen hochdimensionaler Daten *unbekannter* Topologie
- Intrinsische Dimension (ID) = Anzahl unabhängiger Parameter in einem generativen Modell der Daten:
  $P(x) \propto f(\vec{v}) + \eta$, wobei $\dim(\vec{v}) = \text{ID}$
  - ◇ sehr oft die ID viel kleiner als $\dim(\vec{x})$
  - ◇ ID kann variieren
  - ◇ ID ist auch ein interessantes Lernziel

Problem: Finde für gegebene Verteilung $P(\vec{x})$, eine topologische (Netz-)Struktur, die die Topologie der Daten optimal approximiert.

3.2.2 Ansatz

- Menge von Knoten: $\mathcal{A} = \{c_1, \ldots, c_N\}$ mit
  - ◇ zugehörigem Referenzvektor $w_i = \vec{w}_{c_i} \in \mathbb{R}^L$
  - ◇ und akkumuliertem lokalen Fehler $e_i = e_{c_i}$
- Menge von Kanten: $\mathcal{C} = \mathcal{A} \times \mathcal{A}$
  - ◇ ungerichtet / symmetrisch: $(c_i, c_j) \in \mathcal{C} \Leftrightarrow (c_j, c_i) \in \mathcal{C}$
  - ◇ Kantenalter $\text{age}(c_i) \in \mathbb{R}$
- Mechanismen zum Einfügen und Löschen von Knoten und Kanten

3.2.3 Algorithmus

1. Start: zwei zufällige Knoten $\mathcal{A} = \{c_1, c_2\}$ aus $p(\vec{x})$, keine Verbindung: $\mathcal{C} = \emptyset$
2. **Matching:** Wähle zufälligen Input $\vec{x}$ entsprechend $p(\vec{x})$ und suche nächsten sowie zweitnächsten Nachbarn:

$$\begin{aligned} \text{nearest:} \quad & n = \arg\min_{c \in \mathcal{A}} \|\vec{x} - \vec{w}_c\| \\ \text{second nearest:} \quad & s = \arg\min_{c \in \mathcal{A} \setminus \{n\}} \|\vec{x} - \vec{w}_c\| \end{aligned}$$

3. **Referenzvektor-Adaptation (VQ)**: Adaptiere den Gewinner n und seine nächsten topologischen Nachbarn $\mathcal{N}(n)$ mit unterschiedlichen Lernraten $\eta_1 > \eta_2$:

$$\begin{aligned} \Delta w_n &= \eta_1 \, (\vec{x} - \vec{w}_n) && \text{– Gewinner} \\ \Delta w_c &= \eta_2 \, (\vec{x} - \vec{w}_c) \quad \forall\, c \in \mathcal{N}(n) && \text{– Nachbarn des Gewinners} \end{aligned}$$

4. **Kanten-Update**

   - Erzeuge neue Kante (n, s) zwischen Gewinner n und zweitem Gewinner s
   - Setze Kantenalter zurück: $\text{age}(n, s) = 0$
   - Inkrementiere Alter aller Kanten zu Nachbarn: $\text{age}(n, c) \leftarrow \text{age}(n, c) + 1 \quad \forall\, c \in \mathcal{N}(n)$
   - Lösche zu alte Kanten: $\text{age}(n, c) > a_{\max}$
   - Lösche kantenlose Knoten

5. **Knoten-Update**

   - Fehler-Update: $\Delta e_n = \|\vec{x} - \vec{w}_n\|^2$
   - Alle λ Iteration füge *einen* neuen Knoten ein:
     - ◇ Suche Knoten q mit größtem Fehler: $q = \arg\max_{c \in \mathcal{A}} e_c$
     - ◇ Finde topologischen Nachbarn p mit höchstem Fehler: $p = \underset{c \in \mathcal{N}(q)}{\arg\max} e_c$
     - ◇ Erzeuge neuen Knoten r zwischen p und q: $\vec{w}_r = \frac{1}{2}(\vec{w}_p + \vec{w}_q)$
     - ◇ Ersetze Kante (q, p) durch neue Kanten (q, r) und (p, r)
     - ◇ Reduzierung der Fehlervariablen

$$\begin{aligned} \Delta e_q &= -\alpha e_q \\ \Delta e_p &= -\alpha e_p \\ e_r &= \tfrac{1}{2}(e_p + e_q) \end{aligned}$$

   - Fehlerabklingterm: $\Delta e_c = -\beta\, e_c \quad \forall\, c \in \mathcal{A}$

6. GOTO 1 bis Abbruchbedingung erfüllt

Bemerkungen

- sehr viele (aber recht robuste) Parameter, d.h. gleiche Parameter für viele Situationen geeignet
  - ◇ Lernraten η_1 und η_2
  - ◇ Knotenerzeugungsintervall λ und maximales Kantenalter $a_{\max}$
  - ◇ Fehlerabklingraten α und β
- Verfahren kann mit nicht-stationären Signalen umgehen: neue Knoten entstehen, wenn sie gebraucht werden, ungenutzte Knoten verschwinden (meistens)
- Knotenerzeugung hängt an festem Zeitintervall λ und lässt die absolute Größe des Quantisierungsfehlers unberücksichtigt.

3.2.4 GNG mit Utility-Kriterium

Problem: Bei zu schneller Änderung der zugrundeliegenden Datenverteilung, bleiben häufig unnötige Knoten zurück. Diese erhalten aufgrund der geänderten Verteilung keine Stimuli mehr und inkrementieren also auch das Kantenalter zu benachbarten Neuronen nicht. Dies ist aber der einzige Mechanismus, um Kanten und daraufhin auch Knoten zu entfernen.

Lösung:

- zusätzliche Utility-Variable u_c pro Knoten
  (als Maß für Vergrößerung des Quantisierungsfehlers bei Wegfall des Knotens)
- $\Delta u_n = \|\vec{x} - \vec{w}_s\| - \|\vec{x} - \vec{w}_n\|$
  (Differenz des Quantisierungsfehlers von Winner n zu Second-Winner s)
- Decay aller Utility-Variablen: $\forall\, c \in \mathcal{A} \quad \Delta u_c = -\beta u_c$

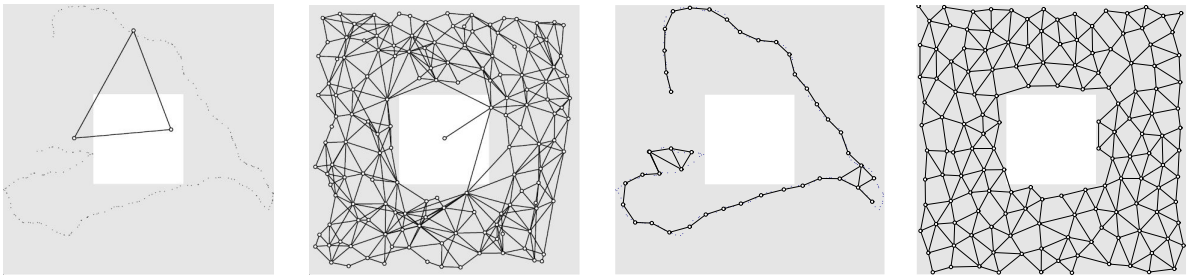
$\Rightarrow$ geringe Utility, falls (lokale) Knotendichte hoch oder Knoten selten aktiviert wird

- Lösche Knoten $i = \arg\min_c u_c$ mit minimaler Utility, falls die (erwartete) Verringerung des Quantisierungsfehlers durch Einfügen eines neuen Knotens an anderer Stelle (am Knoten q mit höchstem lokalen Fehler) größer ist: $u_i < k^{-1} e_q$

3.3 Instantaneous Topological Map – ITM

(J.Jokusch, 2000, AGNI [**?**])

SOM und GNG sind ungeeignet für korrelierte Stimuli, wie z.B. Explorationsbewegungen in der Robotik. Bei der Präsentation solch nahe beieinander liegender Stimuli "gewinnt" häufig dasselbe Neuron mehrfach hintereinander und folgt aufgrund der VQ-Lernregel dem Stimulus entlang dessen Pfad (siehe Abb. 3).



GNG: 200 Stimuli | GNG: 15000 Stimuli | ITM: 200 Stimuli | ITM: 15000 Stimuli

Abbildung 3: Vergleich zwischen GNG und ITM Performance bei korrelierten Stimuli. Nach wenigen Stimuli approximiert das GNG die präsentierte Trajektorie aufgrund von destruktiver Interferenz nur schlecht, das ITM hingegen zeichnet die Trajektorie korrekt auf. Im asymptotischen Limit unterscheiden sich beide Varianten nur unwesentlich. Courtesy of Ján Jockusch.

Idee: Kanten- und Knotenerzeugung sowie -löschung nur aufgrund *lokaler* Eigenschaften

3.3.1 Algorithmus

1. Start: zwei zufällige Knoten $\mathcal{A} = \{c_1, c_2\}$ aus $p(\vec{x})$, keine Verbindung: $\mathcal{C} = \emptyset$

2. **Matching:** Wähle zufälligen Input $\vec{x}$ entsprechend $p(\vec{x})$ und suche nächsten sowie zweitnächsten Nachbarn:

$$\begin{aligned} \text{nearest:} \quad & n = \arg\min_{c \in \mathcal{A}} \|(\vec{x}, \vec{w}_c)\| \\ \text{second nearest:} \quad & s = \arg\min_{c \in \mathcal{A} \setminus \{n\}} \|(\vec{x}, \vec{w}_c)\| \end{aligned}$$

3. **Referenzvektor-Adaptation (VQ)**: Adaptiere *nur* den Gewinner n:

$$\Delta w_n = \eta \, (\vec{x} - \vec{w}_n)$$

4. **Kanten-Update**

   - Erzeuge neue Kante (n, s) zwischen Gewinner n und zweitem Gewinner s
   - Für jeden Nachbarn c von n:
     Falls $\vec{w}_s$ im Thaleskreis von $\vec{w}_n$ und $\vec{w}_c$ liegt, lösche die Kante (n, c). (Die neue Kante (n, s) ersetzt die überflüssige alte Kante (n, c).)

$$\forall\, c \in \mathcal{N}(n) : \text{Falls } (\vec{w}_n - \vec{w}_s) \cdot (\vec{w}_c - \vec{w}_s) < 0 \text{ entferne Kante } (n, c).$$

- Lösche kantenlose Knoten

5. **Knoten-Update**

   - Falls Stimulus außerhalb des Thaleskreises um $\vec{w}_n$ und $\vec{w}_s$ liegt *und* außerhalb des Kreises mit Radius $e_{\max}$ um $\vec{w}_n$
     $((\vec{w}_n - \vec{x}) \cdot (\vec{w}_s - \vec{x}) > 0 \quad \text{und} \quad \|\vec{x} - \vec{w}_n\| > e_{\max})$:
     - ◇ Erzeuge am Ort des Stimulus einen neuen Knoten r: $\vec{w}_r = \vec{x}$
     - ◇ Verbinde r mit n
   - Falls $\vec{w}_n$ und $\vec{w}_s$ näher als $\frac{1}{2}\ e_{max}$, lösche Knoten s

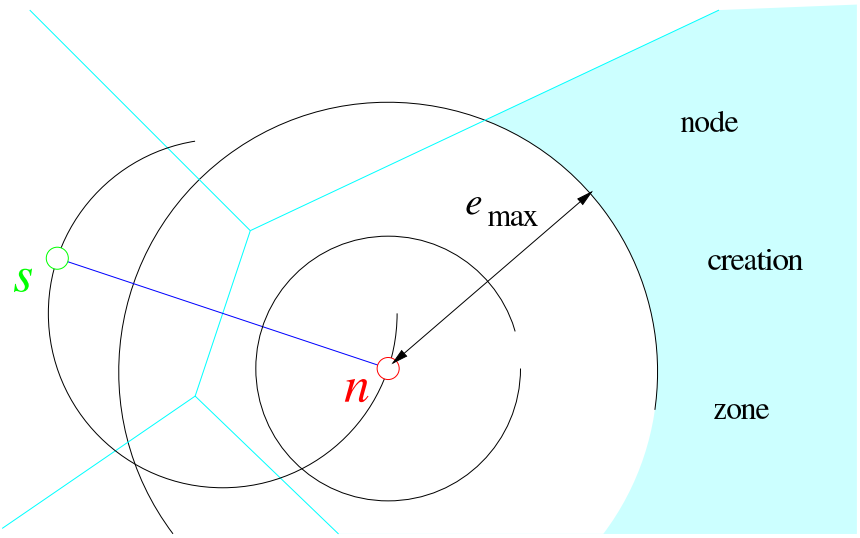


Abbildung 4: Knoten-Update-Mechanismus im ITM: neue Knoten werden erzeugt, sobald der neue Stimulus $\vec{x}$ zu großen Abstand zum Winner-Knoten n hat und der Second-Winner nicht bereits verantworlich zeichnet (Thales-Bedingung). Courtesy of Ján Jockusch.

Bemerkungen

- Nur 2 Parameter: maximaler Quantisierungsfehler $e_{\max}$ und Lernrate η
- Referenzvektor-Adaptation (VQ-Lernregel) nicht notwendig. $\eta > 0$ sorgt lediglich für gleichmäßige Verteilung der Knoten.
- $e_{\max}$ hängt stark von den Daten ab. Wähle z.B. mittleren Stimulus-Abstand aufeinanderfolgender korrelierter Stimuli.
- Kein Fovea-Effekt wie bei SOM oder GNG.
- Nicht mehr benötigte Knoten in nicht-stationären Verteilungen werden *nicht* gelöscht.
- SOM: Strenge Topologie, adaptiert Knotenposition
  ITM: Generiert Topologie, feste Knotenposition

3.4 Parametrisierte SOM – PSOM

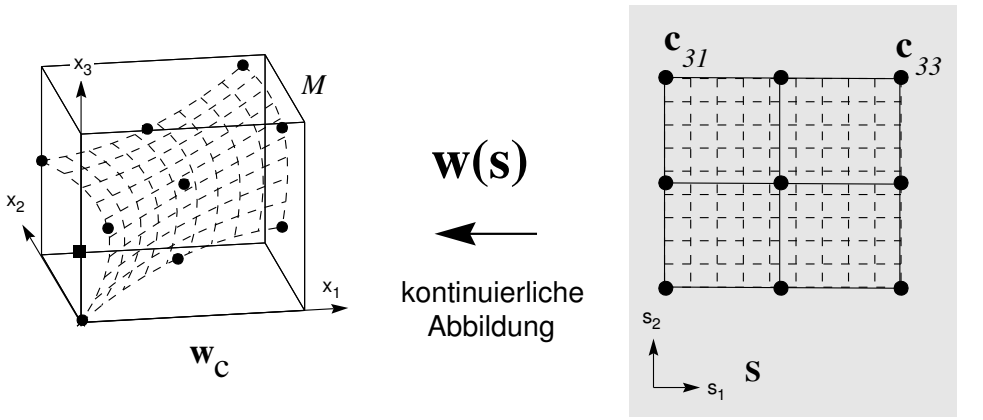
(Ritter '93, Walter '96, AGAI [?, ?, ?, ?])

Idee: Generalisiere die diskrete Abbildung der SOM zu einer kontinuierlichen Abbildung auf einer Mannigfaltigkeit.

- adaptives Modell einer glatten Mannigfaltigkeit im Eingaberaum
- braucht nur wenig Trainingsdaten
- erbt eine modifizierte Lernregel von der SOM
- typische Anwendung in der Robotik, wo Trainingsdaten teuer sind, aber aktiv gesammelt werden können

3.4.1 Die kontinuierliche Abbildung

- Erweitere diskretes Neuronengitter der SOM ($\vec{c} \in \mathcal{A}$)
  auf kontinuierlichen Parameterraum / Koordinatenraum $\vec{s} \in S \supset \mathcal{A}$
- $\vec{w}_{\vec{c}} \in \mathcal{X} \subseteq \mathbb{R}^L$ ist Referenzvektor zu Knoten $\vec{c}$
- erweitere die diskrete Abbildung $\Psi_d : \mathcal{A} \to \mathcal{X}_d$ $(\vec{c} \mapsto \vec{w}_{\vec{c}})$ der SOM
  auf eine kontinuierliche Abbildung $\Psi : S \to \mathcal{M} \subseteq \mathcal{X}$ $(s \mapsto \vec{w}(s))$
- $\mathcal{M}$ ist glatte Mannigfaltigkeit im Eingaberaum $\mathcal{X}$



Mannigfaltigkeit M
im Eingaberaum X

Koordinatensystem S
aufgespannt durch $\boldsymbol{c} \in \mathcal{A}$

Ansatz: $\vec{w}(s) = \sum_{c \in \mathcal{A}} H(c,s)\vec{w}_c$

wobei die Basisfunktionen $H(c,s)$ folgende Eigenschaften besitzen sollen:

1. $\mathcal{M}$ soll durch die Prototypen (Referenzvektoren) verlaufen,
   d.h. $\vec{w}(c) = \vec{w}_c$ bzw. $H(c,c) = 1$ und $H(c,c') = 0$ für alle $c' \neq c$

$\Rightarrow$ $H(c,s)$ muß orthonormales Funktionensystem sein: $H(c,c') = \delta_{cc'}$

2. Konstante Funktionen sollen darstellbar sein, d.h. $\forall s \in S \quad \sum_{c \in \mathcal{A}} H(c,s) = 1$

$\Rightarrow$ $H(c,s)$ produziert „Zerlegung der 1“

Lösung: Multi-dimensionale Lagrange-Polynome auf regelmäßigem Gitter

eindimensionales Gitter: $\mathcal{A} = \{c_1, c_2, \cdots, c_n\} \subset \mathbb{R}$

$$w(s) = \sum_{c \in \mathcal{A}} l_c(s, \mathcal{A}) \cdot \vec{w}_c$$

$$\text{wobei} \quad l_c(s, \mathcal{A}) = \prod_{c' \in \mathcal{A},\, c' \neq c} \frac{s - c'}{c - c'}$$

multidimensionales Gitter: Faktorisierung entlang der einzelnen Dimensionen

$$\mathcal{A}_\nu = \{c_1^\nu, \ldots, c_{n_\nu}^\nu\}, \quad \nu = 1 \ldots m$$
$$\vec{c} \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k \quad (\text{d.h. } \vec{c} = (c_{i_1}^1, c_{i_2}^2, \ldots, c_{i_m}^m)^T)$$
$$H(\vec{c}, \vec{s}) = \prod_{\nu=1}^{m} l_{i_\nu}^\nu(s_\nu, \mathcal{A}_\nu)$$
$$\vec{w}(\vec{s}) = \sum_{\vec{c} \in \mathcal{A}} H(\vec{c}, \vec{s}) \cdot \vec{w}_{\vec{c}}$$

Dabei ist jedem Knoten $c_{i_\nu}^\nu$ entlang einer Dimension ν genau ein Lagrange-Polynom $l_{i_\nu}^\nu$ zugeordnet.

3.4.2 Kontinuierliches Matching

Aufgabe der PSOM ist es, die Umkehrabbildung zu $\Psi : S \to \mathcal{M}$ zu realisieren. D.h. zu gegebenem Stimulus $\vec{x}$ sind die – jetzt kontinuierlichen – Koordinaten $\vec{s}$ gesucht, so dass $\vec{w}(\vec{s})$ möglichst nahe am Stimulus $\vec{x}$ liegt.

Suche kontinuierliche Koordinaten $\vec{s}^*$ des zum Stimulus $\vec{x}$ nächst-gelegenen Punktes $\vec{w}(\vec{s}^*) \in \mathcal{M}$:

$$\vec{s}^* = \arg\min_{\vec{s} \in S} \|\vec{x} - \vec{w}(\vec{s})\|$$

Der nun kontinuierliche Suchraum $s \in S$ erfordert anstatt einer einfachen diskreten Suche nun eine nichtlineare Optimierung von

$$E(\vec{s}) = \frac{1}{2} \sum_{k=1}^{L} (x_k - w_k(\vec{s}))^2 \to \min$$

mittels Gradientenabstieg:

- Starte bei Gewinnerknoten: $\vec{s}_0 = \arg\min_{a \in \mathcal{A}} \|\vec{x} - \vec{w}(\vec{s})\|$
- iterativ Gradientenabstieg:

$$\vec{s}_{t+1} = \vec{s}_t - \eta \, \nabla_{\vec{s}} E(\vec{s}_t) = \vec{s}_t - \eta \sum_{k=1}^{L} \nabla_{\vec{s}} w_k(\vec{s}_t) \, (x_k - w_k(\vec{s}_t))$$

- schneller konvergiert der Levenberg-Marquardt-Algorithmus (quadratisches Verfahren):

$$\text{Löse} \quad (H(\vec{s}_t) + \lambda \cdot \mathbf{1}) \, \vec{\delta_s} = -\nabla_{\vec{s}} E(\vec{s}_t)$$

- ◇ Ursprung: lokale quadratische Approximation:
  $E(\vec{s}+\vec{\delta_s}) \approx q(\vec{\delta_s}) = E(\vec{s}) + \nabla_{\vec{s}} E(\vec{s}) \cdot \vec{\delta_s} + \frac{1}{2}\vec{\delta_s}^t \cdot H(\vec{s}) \cdot \vec{\delta_s}$
- ◇ Minimum von $q(\vec{\delta_s})$ falls $\nabla_{\vec{\delta_s}} q = 0$

Beides erfordert die Berechnung des Gradienten $\nabla_{\vec{s}} E(\vec{s}_t)$ (bzw. der Hessematrix $H = \frac{\partial^2}{\partial \vec{s}^2} E(\vec{s}_t)$), die aber aufgrund der gewählten Lagrange-Polynome sehr gut analytisch bestimmt werden können (Walter '96).

3.4.3 Lernen

1. **ad-hoc Lernen:** Wähle geeignete Gitterknoten $\vec{c} \in \mathcal{A}$ und bestimme zugehörige Referenzvektoren $\vec{w}_{\vec{c}}$.

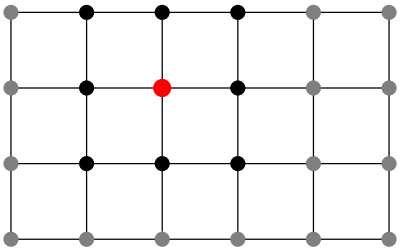
2. **modifizierte SOM-Lernregel:** Verschiebe Referenzvektoren gewichtet relativ zu ihren Beitrag in der aktuellen Ausgabe:

   $$\Delta\vec{w}_{\vec{c}} = \eta \cdot H(\vec{c}, \vec{s}^*) \cdot (\vec{x} - \vec{w}(\vec{s}^*))$$

   Beachte: $\Delta\vec{w}_{\vec{c}} = \eta \cdot H(\vec{c}, \vec{s}^*) \cdot (\vec{x} - \vec{w}_{\vec{c}})$ konvergiert nicht bzw. zerstört sogar die gelernte Mannigfaltigkeit.

3.4.4 Bemerkungen

- PSOM ist globales Lernverfahren (im Gegensatz zu SOM, GNG und ITM):
  Lokale Änderungen z.B. der Referenzvektoren führen zu globalen Änderungen:
  - ◇ aufgrund dem großen Support der Polynome innerhalb einer Dimension und
  - ◇ aufgrund der Faktorisierung auch in jeder anderen Dimension
- Viele Stützstellen bzw. Gitterpunkte entlang einer Dimension führen zu einem hohen Polynomgrad und damit sehr schnell zu unerwünschten Oszillationen. Daher beschränkt man sich i.d.R. auf 3 bis 4 Gitterpunkte je Gitterdimension.
- Um trotzdem größere Raumbereiche abdecken zu können, verwendet man zur (lokalen) Approximation nicht alle Gitterpunkte sondern nur ein lokales $3 \times \cdots \times 3$ Subgitter.



  Das damit auftretende Unstetigkeitsproblem ist lösbar (Walter '96).
- Verwendung von Tschebycheff-Polynomen ist auch sinnvoll. → braucht ein unregelmäßiges Gitter
- Topologische Defekte können auftreten
- Wahl der Topologie legt immer ein Modell zur Interpretation der Daten fest.

3.4.5 Assoziative Vervollständigung

Idee: Verwende SOM, GNG, ITM, PSOM und andere Verwandte als Eingabe-Ausgabe Funktion und lerne diese Abbildung überwacht. Dazu wird der bisherige Referenzvektor $\vec{w}_c \equiv \vec{w}_c^{in}$ im Eingaberaum $\mathcal{X}^{in}$ durch einen zusätzlichen Ausgabevektor $\vec{w}_c^{out}$ im Ausgaberaum $\mathcal{X}^{out}$ ergänzt. Beide Vektoren fassen wir wieder in einem Vektor $\vec{w}_c$ zusammen:

$$\vec{w}_c = \begin{pmatrix} \vec{w}_c^{in} \\ \vec{w}_c^{out} \end{pmatrix} \in \mathcal{X}^{in} \oplus \mathcal{X}^{out} = \mathcal{X} \subset \mathbb{R}^L$$

Während der VQ-Lernschritt alle Komponenten von $\vec{w}_c$ berücksichtigt, darf das Matching nur die Eingabekomponenten $\vec{w}^{in}$ einbeziehen. Dies geschieht mittels einer angepaßten Norm:

$$\begin{aligned} P &= diag(p_1, \ldots, p_L) \qquad p_k \in \{0, 1\} \\ \|\vec{x} - \vec{x}'\|_P &= (\vec{x} - \vec{x}')^T \cdot P \cdot (\vec{x} - \vec{x}') \\ &= \sum_{k=1}^{L} p_k (x_k - x'_k)^2 \end{aligned}$$

Wähle $p_k = 1$ für die Eingabe- und $p_k = 0$ für die Ausgabekomponenten. Alle topologischen Lernverfahren können dann die folgende Ein-Ausgabe-Funktion realisieren:

$$\begin{aligned} &\Phi : \mathcal{X}^{in} \to \mathcal{X}^{out} \\ &\qquad \vec{x} \mapsto \vec{w}_n^{out} \qquad \text{mit} \quad n = \arg\min_{c \in \mathcal{A}} \|\vec{x} - \vec{w}_c\|_P \end{aligned}$$

Während SOM, GNG und ITM damit jeder Voronoi-Zelle einen konstanten Output $\vec{w}_c^{out}$ zuordnen – also eine stückweise konstante Abbildung realisieren – interpoliert die PSOM natürlich auch im Ausgaberaum.

Eine beliebige Auswahl von Ein- und Ausgabekomponenten anhand der $p_k \in \{0, 1\}$ erlaubt auch eine assoziative Vervollständigung:

- Selektiere mit P vorhandene Komponenten und vervollständige den Rest

$$x = \begin{pmatrix} x_1 \\ * \\ x_3 \\ x_4 \\ * \end{pmatrix}$$

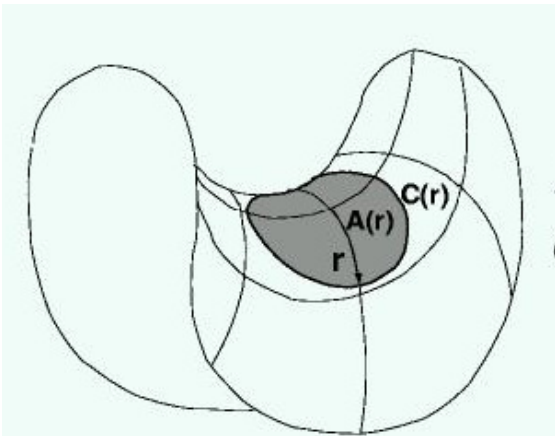
- funktioniert gut bei funktionaler Abhängigkeit der Komponenten voneinander
- kann zur Invertierung von Funktionen verwendet werden

3.5 Hyperbolische SOM (HSOM)

(Ritter '99, AGAI [**?**])

Motivation SOMs eignen sich hervorragend, um hochdimensionaler Daten zu visualisieren, da sie die inhärente Struktur der Daten (ihre Topologie) auf eine niedrig-dimensionale Nachbarschaft abbilden. Diese Fähigkeit hängt offenbar auch davon ab, wie die *Anzahl der möglichen Nachbarn* mit der Entfernung von einem Knoten wächst. Die bisherigen SOM-Ansätze verwenden als zugrundeliegenden Raum den euklidischen Raum, dessen Volumen polynomiell mit dem Nachbarschaftsradius wächst: $V(r) \propto r^d$.
Die Abbildung hierarchischer, baumartiger Datenstrukturen (wie z.B. das WWW) erfordert jedoch ein exponentielles Wachstum der Anzahl der Nachbarn. Der hyperbolische Raum bietet ein solches exponentielles Wachstum des Volumens und stellt daher den idealen Raum zur Abbildung hierarchischer Datenstruktur dar.

Der hyperbolische Raum Der zweidimensionale hyperbolische Raum $\mathbb{H}^2$ ist ein Raum gleichmäßiger negativer Krümmung, so dass man ihn sich lokal als *Sattel* im euklidischen Raum vorstellen kann (Abb. 5). Eine isometrische, d.h. längenerhaltende Einbettung in den euklidischen Raum $\mathbb{R}^2$ ist – ähnliche wie bei der Kugel – offenbar nicht möglich.



$$\text{Umfang:} \quad C(r) = 2\pi \sinh(r)$$
$$\text{Fläche:} \quad A(r) = 4\pi \sinh^2(r/2)$$

wobei

$$\sinh(x) = \tfrac{1}{2}(\mathrm{e}^x - \mathrm{e}^{-x})$$
$$\cosh(x) = \tfrac{1}{2}(\mathrm{e}^x + \mathrm{e}^{-x})$$

Umfang und Fläche wachsen exponentiell mit r!

Abbildung 5: Der hyperbolische Raum – lokal als Sattel vorstellbar – bietet ein exponentielles Wachstum der Nachbarschaft.

3.5.1 Modelle und Visualisierungen

Alle Modelle des zweidimensionalen hyperbolischen Raumes $\mathbb{H}^2$ gehen von Polarkoordinaten (r, θ) aus.

Hyperboloid / Minkowski-Modell

- Abbildung auf 3-dimensionalen Hyperboloid: $z^2 = x^2 + y^2 + 1$

$$x = \sinh(r)\cos(\theta)$$
$$y = \sinh(r)\sin(\theta)$$
$$z = \cosh(r)$$

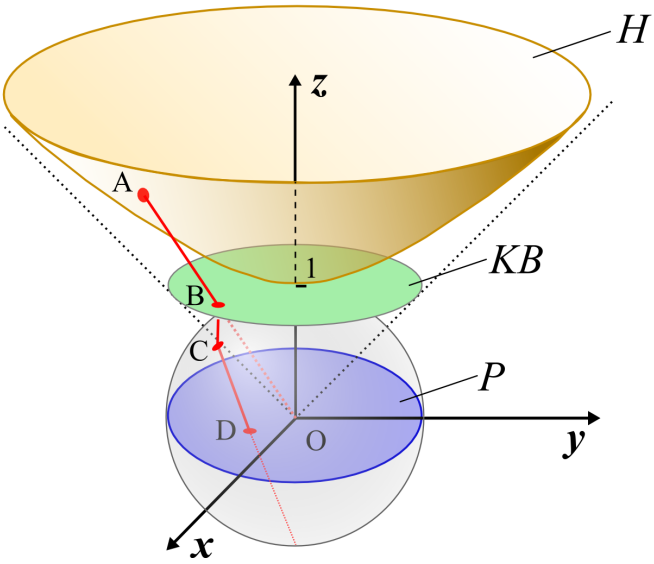


Abbildung 6: Unterschiedliche Modelle / Einbettungen des $\mathbb{H}^2$ im euklidischen Raum: Hyperboloid H im $\mathbb{R}^3$, Poincaré Disk P im $\mathbb{R}^2$. Courtesy of Jörg Ontrup.

- Mit der Minkowski-Metrik stellt dies eine isometrische Einbettung dar.

$$\|\vec{p} - \vec{p}'\|_M = (x - x')^2 + (y - y')^2 - (z - z')^2$$

  Die Minkowski-Metrik ist nicht positiv definit, also keine echte Metrik.

Poincaré Disk Mittels mehrerer Projektionsschritte gelangt man vom Hyperboloid auf die Poincaré Disk (Abb. 6):

- Klein-Beltrami-Modell: Projektion auf die Ebene $z = 1$ mittels Strahlen durch den Ursprung O: $A \to B$
- senkrechte Projektion auf die Einheitskugel um den Ursprung: $B \to C$
- stereographische Projektion auf den Einheitskreis um den Ursprung (mittels eines Strahls durch den Südpol der Kugel): $C \to D$

→ liefert Poincaré-Disk mit folgenden Eigenschaften:

- ◇ Transformation:

$$x = \tanh(\tfrac{1}{2}r)\cos(\theta)$$
$$y = \tanh(\tfrac{1}{2}r)\sin(\theta)$$

- ◇ Abbildung auf die endliche Scheibe: $x^2 + y^2 < 1$
- ◇ Winkel bleiben erhalten (und damit geometrische Formen)
- ◇ extremer Fischaugen-Effekt:
  - ∗ Ursprung des $\mathbb{H}^2$ fast 1:1 abgebildet
  - ∗ entfernte Regionen exponentiell gestaucht
- ◇ Linien werden auf Kreise abgebildet

◇ Punkte $z \in P \equiv \mathbb{C}$ – aufgefasst als komplexe Zahlen – können zur Bestimmung des hyperbolischen Abstands genutzt werden:

$$\mathrm{d}(z_1, z_2) = 2 \operatorname{arctanh} \left(\left| \frac{z_1 - z_2}{1 - z_1 \bar{z}_2} \right| \right) \tag{3.1}$$

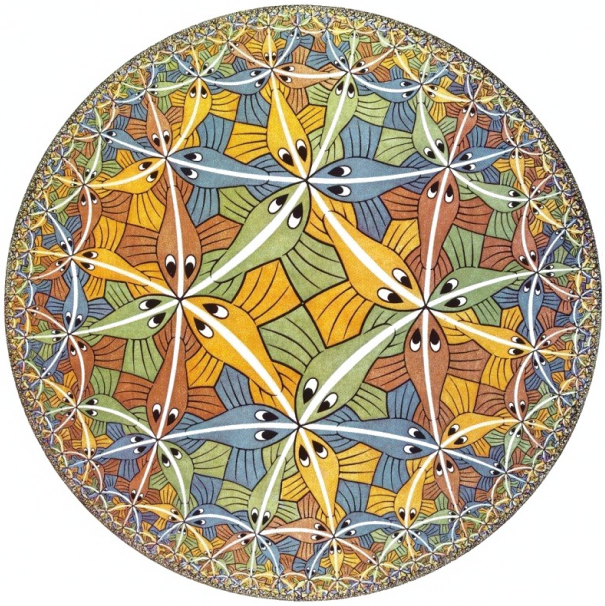


Abbildung 7: Den Fischaugen-Effekt der Poincaré-Disk kann man sehr gut an den Kunstwerken der Reihe „Circle Limit“ von M.C. Escher erkennen.

3.5.2 Tesselierung des $\mathbb{H}^2$

Wie erzeugt man eine sinnvolle Gitterstruktur im $\mathbb{H}^2$?

- umgebe jeden Knoten mit $n \geq 7$ gleichseitigen Dreiecken
- je Knoten n Kanten: 1 Elternknoten, 2 Geschwister, $n - 3$ Kinder

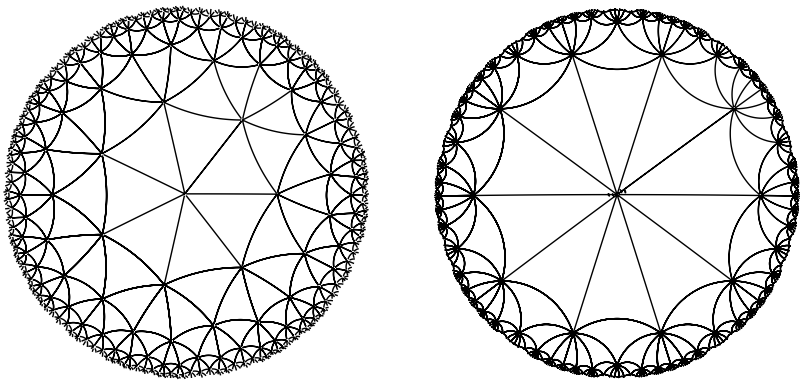


Abbildung 8: Tesselierung des $\mathbb{H}^2$ mit $n = 7$ bzw. $n = 10$ Kanten.

3.5.3 HSOM Lernregel

Bis auf die andere zugrundeliegende Netzwerk-Topologie und die neue Metrik (3.1) ändert sich am Standard-SOM-Algorithmus nichts!

- speichere zu jedem Knoten c dessen Position auf der Poincaré-Disk als komplexe Zahl z_c
- nutze neue Metrik (3.1) im Lernschritt: $\Delta\vec{w}_c = \eta \cdot h_{cn} \cdot (\vec{x} - \vec{w}_c)$

$$h_{cn} = \mathrm{e}^{-\frac{\mathrm{d}^2(z_c, z_n)}{2\sigma^2}} \qquad \mathrm{d}(z_c, z_n) = 2\,\mathrm{arctanh}\left(\left|\frac{z_c - z_n}{1 - z_c\bar{z}_n}\right|\right)$$

3.5.4 Hierarchically Growing Hyperbolic SOM (H^{2}SOM)

Jörg Ontrup, AG AI [**?**]

Idee: Nutze hierachische Struktur auch zur Suche ($\rightarrow$ Tree-Search)
und erweitere die HSOM nur nach Bedarf (ähnlich GNG)

- Start: Initialisiere Root-Knoten mit Mittelwert der Daten: $\vec{w}_c = \langle\vec{x}\rangle$
- nach λ Trainingsschritten erweitere die HSOM an den Knoten c, deren mittlerer Quantisierungsfehler $\bar{e}_c$ einen Schwellwert e_{max} überschreitet
- beschränke Adaption immer auf die *äußersten* Knoten,
  Elternknoten bleiben unverändert

$\rightarrow$ Performance-Steigerung (bei N Knoten): $\mathcal{O}(N) \rightarrow \mathcal{O}(\log_n N)$, typischerweise Faktor 60

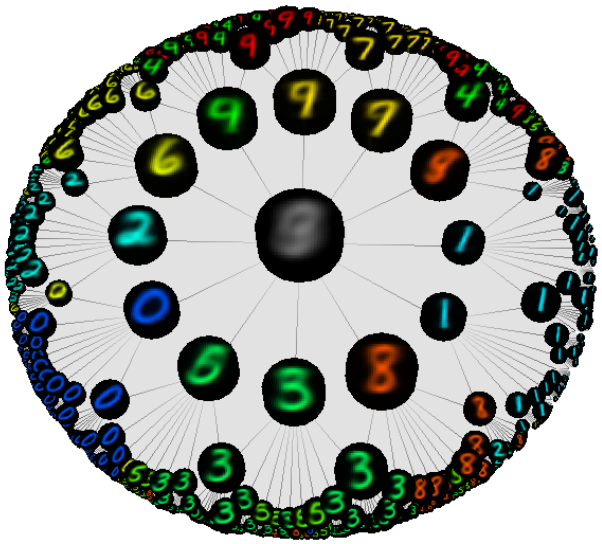


Abbildung 9: HSOM der MNIST-Datenbank von handschriftlichen Zahlen. Courtesy of Jörg Ontrup.

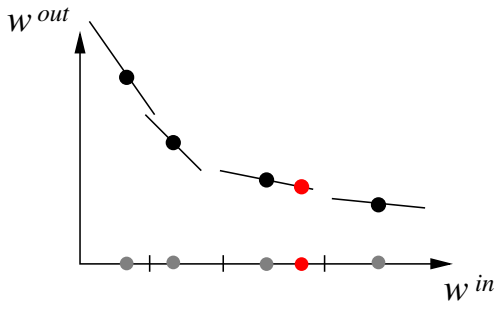
3.6 Local Linear Maps

Die Local Linear Map (LLM) erweitert – ähnlich wie die in Abschnitt 3.4.5 vorgestellte Eingabe-Ausgabe-Abbildung – die topologischen Lernverfahren SOM, GNG, ITM um eine Ausgabe-Komponente, die überwacht gelernt wird. Zusätzlich zu dem in Abschnitt 3.4.5 eingeführten Ausgabevektor $\vec{w}_c^{out}$, der ein stückweise konstante Abbildung ermöglichte, wird nun aber noch eine lokal-lineare Abbildung A_c gelernt, die zu einer stückweise linearen Abbildung (über jeder Voronoizelle) und damit zu einer besseren Approximation der Daten führt.

Idee: Bessere Approximation der Eingabe-Ausgabe-Abbildung mittels lokaler linearer Abbildungen A_c:

$$\Phi : \mathcal{X}^{in} \to \mathcal{X}^{out} \qquad \vec{x} \mapsto \vec{y}_n(\vec{x})$$
$$\vec{y}_c(\vec{x}) = \vec{w}_c^{out} + A_c \cdot (\vec{x} - \vec{w}_c^{in})$$

Beispiel: $\Phi : \mathbb{R} \to \mathbb{R}$



A_c – Steigung der Tangenten

3.6.1 LLM Lernregel

1. Lerne unüberwacht Referenzvektoren $\vec{w}_c^{in}$ – Clusterung im Eingaberaum
   Nutze Standard SOM-, GNG- oder ITM-Lernregel:

   $$\text{z.B. SOM:} \qquad \Delta\vec{w}_c^{in} = \eta_{in} \cdot h_{cn} \cdot (\vec{x}^{\alpha} - \vec{w}_c^{in})$$

2. Lerne überwacht die zugehörigen Ausgabevektoren $\vec{w}_c^{out}$
   und die lokale lineare Abbildung A_c:

   $$\Delta\vec{w}_n^{out} = \eta_{out} \cdot \underbrace{(\vec{y}^{\alpha} - \vec{y}_n(\vec{x}^{\alpha}))}_{(1)} + \underbrace{A_n \Delta\vec{w}_n^{in}}_{(2)}$$

   (1) Fehlerkorrekturregel benötigt Sollausgabe $\vec{y}^{\alpha}$
   (2) Kompensiert Drift von w_c^{in} durch Verschiebung von $\vec{w}_c^{out}$ entlang der Ebene

   $$\Delta A_n = \eta_A \cdot (\vec{y}^{\alpha} - \vec{y}_n(\vec{x}^{\alpha})) \frac{(\vec{x}^{\alpha} - \vec{w}_n^{in})^t}{\|\vec{x}^{\alpha} - \vec{w}_n^{in}\|^2}$$

Die Matrix-Adaptation implementiert eine komponentenweise lineare Fehlerkorrektur ähnlich der Perzeptron-Lernregel. Falls die Netzwerkausgabe bereits korrekt ist ($\vec{y}^{\alpha} = \vec{y}_n(\vec{x}^{\alpha})$)), werden nur noch die Eingabe-Referenzvektoren $\vec{w}_c^{in}$ verschoben und mit ihnen die Ausgabe-Vektoren $\vec{w}_c^{out}$ aufgrund der Drift-Korrektur.

Konvergenz der Lernregel Um die Konvergenz zu zeigen, bestimmen wir zunächst die Änderung der Netzwerkausgabe $\Delta\vec{y}$ unter einem Lernschritt und zeigen anschließend, dass der Approximationsfehler $\vec{y}^{\alpha} - \vec{y}_c$ abnimmt.

$$\begin{aligned}
\Delta\vec{y}_c &= \vec{y}'_c - \vec{y}_c \\
&= \vec{w}'^{out}_c + A'_c \cdot (\vec{x} - \vec{w}'^{in}_c) \; - \; \big(\vec{w}^{out}_c + A_c \cdot (\vec{x} - \vec{w}^{in}_c)\big) \\
&= \vec{w}'^{out} + A'\vec{x} - A'\vec{w}'^{in} \; - \; \vec{w}^{out} - A\vec{x} + A\vec{w}^{in} \\
&= \Delta\vec{w}^{out} + \Delta A\vec{x} - A'\vec{w}'^{in} + A\vec{w}^{in} + (A'\vec{w}^{in} - A'\vec{w}^{in}) \\
&= \Delta\vec{w}^{out} + \Delta A\vec{x} - (A' - A)\vec{w}^{in} - A'(\vec{w}'^{in} - \vec{w}^{in}) \\
&= \Delta\vec{w}^{out} + \Delta A(\vec{x} - \vec{w}^{in}) - (A + \Delta A)\Delta\vec{w}^{in} \\
&= \Delta\vec{w}^{out} + \Delta A(\vec{x} - \vec{w}^{in}) - (A\Delta\vec{w}^{in} + \underbrace{\Delta A\Delta\vec{w}^{in}}_{\Delta^2 \approx 0}) \\
&\approx \Delta\vec{w}^{out} + \Delta A(\vec{x} - \vec{w}^{in}) - A\Delta\vec{w}^{in}
\end{aligned}$$

Setze die Lernregeln für Δ-Ausdrücke ein:

$$= \eta_{out} \cdot h_{cn} \cdot (\vec{y}^{\alpha} - \vec{y}_c) + A_c\Delta\vec{w}^{in}_c + \eta_A \cdot h_{cn}(\vec{y}^{\alpha} - \vec{y}_c) \cdot \frac{\cancel{\|\vec{x} - \vec{w}^{in}_c\|^2}}{\cancel{\|\vec{x} - \vec{w}^{in}_c\|^2}} - A_c\Delta\vec{w}^{in}_c$$

Nun können wir die Änderung des Fehlers bestimmen:

$$\begin{aligned}
\Delta(\vec{y}^{\alpha} - \vec{y}_c) &= (\vec{y}^{\alpha} - \vec{y}'_c) - (\vec{y}^{\alpha} - \vec{y}_c) = \cancel{\Delta\vec{y}^{\alpha}} - \Delta\vec{y}_c = -\Delta\vec{y}_c \\
&= -(\eta_{out} + \eta_A)h_{cn}(\vec{y}^{\alpha} - \vec{y}_c)
\end{aligned}$$

Dies ist die bereits bekannte zeit-diskrete Version der Differentialgleichung, die einen exponentiellen Abfall beschreibt:

$$\begin{aligned}
\Delta z &= -\tau \cdot z \qquad \text{mit } z = \vec{y}^{\alpha} - \vec{y}_c \text{ und } \tau = (\eta_{out} + \eta_A) \cdot h_{cn} > 0 \\
\text{DGL:} \quad \dot{z} &= -\tau \cdot z \\
\text{mit Lösung:} \quad z(t) &= z_0 \cdot \mathrm{e}^{-\tau t} \to 0
\end{aligned}$$

Damit konvergiert auch der Ausgabefehler $\vec{y}^{\alpha} - \vec{y}_c$ exponentiell schnell gegen Null.

3.6.2 Verallgemeinerung auf Ausgabe-Mischungen

Die Winner-Takes-All-Charakteristik der zugrundeliegenden SOM-Ansätze führt für die Local Linear Map zu Unstetigkeiten der Ein-Ausgabe-Abbildung an den Grenzen der Voronoi-Zellen.
Ziel: Vermeide Unstetigkeiten an Grenzen der Voronoi-Zellen **Ansatz:** Mische die Ausgabe durch *gewichtete Überlagerung aller* Knotenausgaben:

$$\vec{y}^{net}(\vec{x}) = \sum_c \vec{y}_c \cdot g_c(\vec{x}) = \sum_c \big(\vec{w}^{out}_c + A_c(\vec{x} - \vec{w}^{in}_c)\big) \cdot g_c(\vec{x})$$

wobei $g_c(\vec{x})$ eine Mischungsfunktion zur Gewichtung der einzelnen Knotenbeiträge ist. Typischerweise verwendet man eine Soft-Max-Gewichtung mit Gauss'schen Mischungsfunktionen:

$$g_i(x) = \frac{R\left(\frac{||x - w^{in}_i||}{\sigma_i}\right)}{\sum_j R\left(\frac{||x - w^{in}_j||}{\sigma_j}\right)} \qquad \text{(soft-max)}$$

- $R(s) = \mathrm{e}^{-\frac{1}{2}s^2}$
- Radien σ_i geben die Einfluss-Breite an
  - ◇ für $\sigma_i^{-1} \to \infty$ bzw. $\sigma_i \to 0$ erhält man wieder Winner-Takes-All
  - ◇ für $\sigma_i \to \infty$ erhält man Gleichverteilung: $g_i(x) = (\#\text{ Knoten})^{-1}$

Lernregeln Die modifizierten Lernregeln beinhalten nun jeweils auch den Gewichtungsfaktor $g_c(\vec{x})$, um die Knoten entsprechend ihrem Anteil an der Gesamtausgabe upzudaten:

$$\begin{aligned}
\Delta\vec{w}_c^{in} &= \eta_{in} h_{cn}\, g_c(\vec{x}^\alpha) \cdot (\vec{x}^\alpha - \vec{w}_c^{in}) \\
\Delta\vec{w}_c^{out} &= \eta_{out} \quad g_c(\vec{x}^\alpha) \cdot (\vec{y}^\alpha - \vec{y}_c(\vec{x}^\alpha)) + A_c \Delta\vec{w}_c^{in} \\
\Delta A_c &= \eta_A \quad g_c(\vec{x}^\alpha) \cdot (\vec{y}^\alpha - \vec{y}_c(\vec{x}^\alpha))\, \frac{(\vec{x}^\alpha - \vec{w}_c^{in})^t}{\|\vec{x}^\alpha - \vec{w}_c^{in}\|^2}
\end{aligned}$$

Die Radien σ_c sollen den mittleren Abstand der Datenpunkte innerhalb der Voronoizelle V_c approximieren:

$$\begin{aligned}
\Delta\sigma_n &= \eta_\sigma(||\vec{x} - \vec{w}_n^{in}|| - \sigma_n) \qquad \text{(nur für den Winner-Knoten } n\text{)} \\
\Rightarrow \qquad \sigma_c &= \langle ||\vec{x} - \vec{w}_c^{in}|| \rangle_{V_c}
\end{aligned}$$

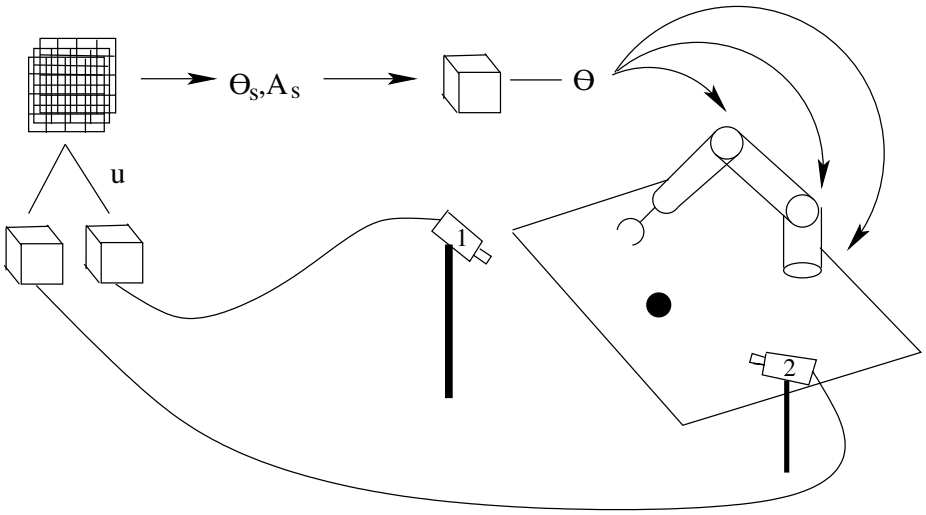
Die Referenzvektoren $\vec{w}_c^{in}$ liegen nun im gewichteten Mittel der Daten, die sie repräsentieren:

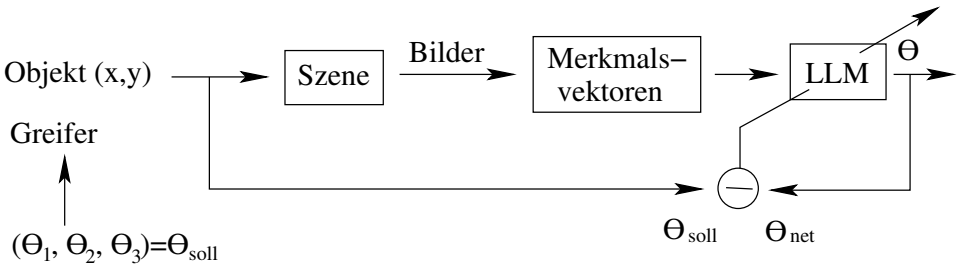
$$\begin{aligned}
\left\langle \Delta\vec{w}_c^{in} \right\rangle = 0 &= \eta_{in} \langle (\vec{x} - \vec{w}_c^{in}) \cdot g_c(\vec{x}) \rangle_{\vec{x} \in V_c} \\
\Leftrightarrow \qquad \vec{w}_c^{in} &= \frac{\langle \vec{x} \cdot g_c(\vec{x}) \rangle_{V_c}}{\langle g_c(\vec{x}) \rangle_{V_c}}
\end{aligned}$$

Beispiel 3.6.1. Visuelles Lernen in der Robotik

Aufgabe: Lerne inverse Kinematik eines Roboterarms aufgrund der Bildkoordinaten zweier Kameras.

- *Input*: Kamerabilder 256 × 256 Pixel
  Vorverarbeitung: Bestimmung der Bildkoordinaten (x,y) des Zielpunktes
- *Output*: Gelenkwinkel $\vec{\theta} = (\theta_1, \theta_2, \theta_3)^t$
- *Topologie*: 3D-Gitter (3-dimensionale Raumkoordinaten)




Objekt (x,y)
Szene
Bilder
Merkmals–
vektoren
LLM
Θ
Greifer
(Θ1, Θ2, Θ3)=Θsoll
Θsoll
Θnet

3.7 Einschub: Regression

Ziel: Finde optimale Gewichte für die lineare Funktionsapproximation $\hat{y}_\alpha = \sum w_i x_i = \mathbf{x}_\alpha^t \cdot \mathbf{w}$ bei gegebenen Trainingsdaten $(\mathbf{x}_\alpha, y_\alpha)$ – bei Minimierung des quadratischen Fehlers.

Modell:	$\hat{\mathbf{y}}_\alpha^t = \sum_{i=1}^{N} x_i^\alpha \cdot \mathbf{w}_i = \mathbf{x}_\alpha^t \cdot \mathbf{W}$	$\hat{\mathbf{y}} = \mathbf{X} \cdot \mathbf{w}$	$\hat{\mathbf{Y}} = \mathbf{X} \cdot \mathbf{W}$
		$\hat{\mathbf{y}} \in \mathbb{R}^{M \times 1}$	$\hat{\mathbf{Y}} \in \mathbb{R}^{M \times L}$
Eingabe:	$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^t \\ \vdots \\ \mathbf{x}_M^t \end{bmatrix} = \left[\mathbf{x}_1, \dots \mathbf{x}_M\right]^t$	$\mathbf{X} \in \mathbb{R}^{M \times N}$	
	$\mathbf{Y} = \left[\mathbf{y}_1, \dots \mathbf{y}_M\right]^t$	$\mathbf{y} = \left[y_1, \dots y_M\right]^t$	
gesucht:	$\mathbf{W} = \left[\mathbf{w}_1 \dots \mathbf{w}_N\right]^t$	$\mathbf{w} \in \mathbb{R}^{N \times 1}$	$\mathbf{W} \in \mathbb{R}^{N \times L}$

N – Dimension der Eingaben $\mathbf{x}_\alpha$
L – Dimension der Ausgaben $\mathbf{y}_\alpha$
M – # Trainingsbeispiele $(\mathbf{x}_\alpha, \mathbf{y}_\alpha)$

Minimiere	$E(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^t \cdot (\mathbf{y} - \mathbf{X}\mathbf{w})$	$+$	$\frac{1}{2}\lambda\|\mathbf{w}\|^2$
Minimum bei:	$\nabla_\mathbf{w} E \stackrel{!}{=} 0 = \mathbf{X}^t \cdot (\mathbf{y} - \mathbf{X}\mathbf{w})$	$+$	$\lambda\mathbf{w}$
Lösung:	$\mathbf{w} = (\mathbf{X}^t\,\mathbf{X})^{-1}\mathbf{X}^t \cdot \mathbf{y}$		$\mathbf{w} = (\mathbf{X}^t\,\mathbf{X} + \lambda\mathbf{1})^{-1}\mathbf{X}^t \cdot \mathbf{y}$
mehr-dimensional:	$\mathbf{W} = \underbrace{(\mathbf{X}^t\,\mathbf{X})^{-1}\mathbf{X}^t}_{\text{Pseudo-Inverse}} \cdot \mathbf{Y}$		$\mathbf{W} = \underbrace{(\mathbf{X}^t\,\mathbf{X} + \lambda\mathbf{1})^{-1}\mathbf{X}^t}_{\text{Ridge Regression}} \cdot \mathbf{Y}$

3.7.1 Principal Component Regression (PCR)

Idee: Regression nicht direkt auf Daten X, sondern nur auf den wichtigsten Hauptkomponenten:

Eigenwert-Zerlegung: $\underbrace{\mathbf{X}^t\mathbf{X}}_{\text{Kovarianz der Daten}} = \mathbf{V}\,\mathbf{\Lambda}^2\,\mathbf{V}^t$

$\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_N]$
$\mathbf{U} = [\mathbf{v}_1, \dots, \mathbf{v}_k]$

$$\mathbf{W} = (\mathbf{U}^t\,\mathbf{X}^t \cdot \underbrace{\mathbf{X}\,\mathbf{U}}\,)^{-1}\,\mathbf{U}^t\,\mathbf{X}^t \cdot \mathbf{Y} = (\mathbf{S}^t\mathbf{S})^{-1}\mathbf{S}^t\mathbf{Y}$$

$\mathbf{S} = \mathbf{X}'$ – Projektion von $\mathbf{X}$ auf $\mathbf{U}$

3.7.2 Partial Least Squares (PLS)

PCR betrachtet die Richtungen maximaler Varianz der Eingabedaten $\mathbf{X}$. Diese müssen aber nicht unbedingt stark mit der Ausgabe $\mathbf{Y}$ korrelieren, d.h. die PCA-Analyse ist u.U. kontraproduktiv.

Idee: Regression nicht entlang der Richtungen maximaler Autokorrelation der Daten, sondern entlang maximaler Korrelation zwischen $\mathbf{X}$ und $\mathbf{Y}$. Im Falle eindimensionaler Ausgaben, können die Projektionsrichtung $\mathbf{u}$ und das Regressionsgewicht sehr einfach (ohne iterativen Prozess) bestimmt werden:

		Init: $\mathbf{X}_{\text{res}} \equiv \mathbf{X}$, $\mathbf{y}_{\text{res}} \equiv \mathbf{y}$
		for $i = 1 \dots r$
		$\mathbf{u}_i = {\mathbf{X}_{\text{res}}}^t \cdot \mathbf{y}_{\text{res}}$
$\mathbf{u} = \mathbf{X}^t \cdot \mathbf{y}$	projection direction	$\mathbf{s}_i = \mathbf{X}_{\text{res}} \cdot \mathbf{u}_i$
$\mathbf{s} = \mathbf{X} \cdot \mathbf{u} = \mathbf{X}\mathbf{X}^t \cdot \mathbf{y}$	projection	$\beta_i \equiv w_i = \dfrac{\mathbf{s}_i^t \cdot \mathbf{y}_{\text{res}}}{\mathbf{s}_i^t \mathbf{s}_i}$
$\beta \equiv w = \dfrac{\mathbf{s}^t \cdot \mathbf{y}}{\mathbf{s}^t \mathbf{s}}$	1-dim regression	$\mathbf{p}_i = \dfrac{\mathbf{s}_i^t \cdot \mathbf{X}_{\text{res}}}{\mathbf{s}_i^t \mathbf{s}_i}$
		$\mathbf{y}_{\text{res}} = \mathbf{y}_{\text{res}} - w_i \mathbf{s}_i$
		$\mathbf{X}_{\text{res}} = \mathbf{X}_{\text{res}} - \mathbf{s}_i \cdot \mathbf{p}_i$
		oder $= \mathbf{X}_{\text{res}} - \mathbf{s}_i \cdot \mathbf{u}_i$

Der Algorithmus auf der rechten Seite bestimmt iterativ mehrere Projektionsrichtungen $\mathbf{u}_i$ auf Basis des restlichen Prädiktionsfehlers $\mathbf{y}_{\text{res}}(\mathbf{X}_{\text{res}})$. Solange der Regressionsparameter w_i im Verhältnis zum Vorgänger w_{i-1} noch groß genug ist, wird versucht eine weitere Projektionsrichtung hinzuzufügen.

3.8 Locally Weighted Projection Regression (LWPR)

LWPR ist ein sehr populäres und effizientes Lernverfahren, dass den LLMs in der Struktur sehr ähnelt. Die lineare Funktionsapproximation innerhalb der rezeptiven Felder wird aber mittels inkrementeller PLS gemacht. Ein rezeptives Feld / ein Prototyp c speichert folgende Informationen:

- w_c^{in} – Zentrum im Eingaberaum
- $\beta_0^c, \dots, \beta_N^c$ – PLS Regressionsparameter, $\beta_0^c \equiv w_c^{out}$
- $\bar{x}_c, \bar{y}_c \equiv \beta_0^c$ – gewichteter Mittelwert von Ein- und Ausgaben x, y

$$w_c(x) = \exp\left(-\tfrac{1}{2}(x - w_c^{\text{in}})^t \cdot D_c \cdot (x - w_c^{\text{in}})\right) \qquad \text{(mit Mahalanobis-Distanz)}$$

$$g_c(x) = \frac{w_c(x)}{\sum_c w_c(x)} \qquad \text{(soft-max)}$$

$$\bar{x}_c = \sum_\alpha w_c(x_\alpha) \cdot x_\alpha \;/\; \sum_\alpha w_c(x_\alpha)$$

$$\bar{y}_c = \sum_\alpha w_c(x_\alpha) \cdot y_\alpha \;/\; \sum_\alpha w_c(x_\alpha)$$

- N_c – Anzahl an Datenbeispielen, die das rezeptive Feld “gesehen” hat

Die Gesamtausgabe ist wieder eine mit soft-max gewichtete Summe aller Teilausgaben:

$$\hat{y} = \sum_c g_c(x) \cdot \hat{y}_c(x)$$

$$\hat{y}_c = \beta_0^c + \beta_1^c s_1 + \dots + \beta_r^c s_r \qquad \text{(PLS-Regression)}$$

$$x_0 = x - \bar{x}_c \qquad s_i = x_{i-1} \cdot u_i \qquad x_i = x_{i-1} - s_i \cdot p_i$$

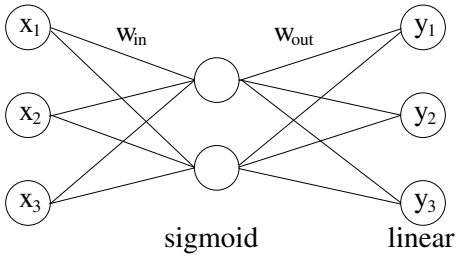
Bemerkungen:

- LWPR ist **inkrementell**. Neue rezeptive Felder werden eingefügt, wenn $\max_c w_c(x) < w_{\text{gen}}$, d.h. kein rezeptives Feld das neue Sample gut genug repräsentiert.
- Prototypen werden nicht verschoben: w_c^{in} bleibt konstant, $\bar{x}_c$ wird aktualisiert.
- Die Ausgabe eines rezeptiven Feldes geht nur in die Gesamtausgabe ein, wenn dessen Statistik zuverlässig ist: $N_c > 2d$ (d.h. mit genügend Datenbeispielen gelernt wurde).
- Für jede Ausgabedimension wird ein eigenes, *eindimensionales* Modell gelernt. Im Gegensatz zu LLM sind die Anzahl und Position der Prototypen in den unterschiedlichen Dimensionen also unabhängig wählbar.

4 Radiale Basisfunktionen (RBF)

4.1 Motivation für lokale Verfahren

Beispiel 4.1.1. Zweischichtiges MLP mit linearer Ausgabeschicht



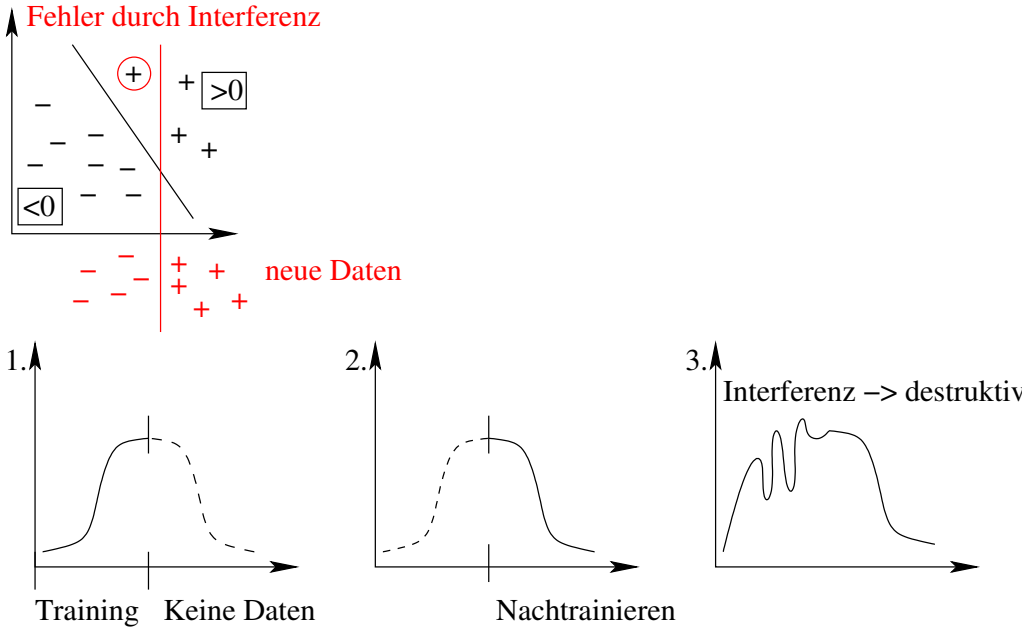
$$y_i(\mathbf{x}) = \sum_{j=1}^{N} w_{ij}^{out} \sigma(\mathbf{w}_j^{in} \cdot \mathbf{x})$$

Bei MLPs liegt eine verteilte Repräsentation vor:

- jedes Neuron liefert einen Funktionsbeitrag im Halbraum,
- jedes Neuron liefert eine Änderung in der Nachbarschaft einer Hyperebene,
- ermöglicht gute
  - ◇ Interpolationsfähigkeit
  - ◇ Generalisierungsfähigkeit

Aber: Änderung an einem Neuron kann Netzantwort für weite Bereiche des Inputs verändern. Dieses Phänomen macht inkrementelles Lernen (Nachlernen von neuen Beispielen) infolge „destruktiver Interferenz“ unmöglich.

Beispiel 4.1.2. Destruktive Interferenz bei MLPs



Halbebenen sind nicht-lokal und verhindern damit inkrementelles Lernen. Lokale Verfahren verhindern solche Interferenz, z.B. SOM, RBF, LLM, GNG, SVM (aber nicht PSOM).

Definition 4.1.1. Lokales Verfahren: Jedes Neuron liefert nur innerhalb eines begrenzten Bereiches des Eingaberaums einen Beitrag.

4.2 Radiale Basisfunktionen

Gegeben sei eine Trainingsmenge $\{x^\alpha, y^\alpha\}$ mit M Elementen (überwachtes Lernen).

Statt der sigmoiden Aktivierungsfunktion $\sigma(\mathbf{w}_j^{in} \cdot \mathbf{x})$ der MLPs verwenden wir lokalisierte „Basisfunktionen“:

$$G_j(\mathbf{x}, \mathbf{w}^{in}) = R\left(\frac{||\mathbf{x} - \mathbf{w}_j^{in}||}{\sigma_j}\right), \qquad \text{mit} \quad j \in 1, \ldots, N \quad \text{und z.B.} \quad R(s) = e^{-\frac{1}{2}s^2}$$

Die „Rampenantwort“ eines Neurons wird dabei ersetzt durch eine lokalisierte Antwort, das Skalarprodukt $\mathbf{w}_j^{in} \cdot \mathbf{x}$ durch den Abstand $\|\mathbf{w}_j^{in} - \mathbf{x}\|$[1].
Die Netzantwort für eine Eingabe $\mathbf{x}$ ist – wie bei der Ausgabeschicht des MLP – die gewichtete Summe der N Basisfunktionen:

$$y_i(\mathbf{x}) = \sum_{j=1}^{N} w_{ij}^{out} \cdot G_j(\mathbf{x}, \mathbf{w}_j^{in}) \quad \text{mit} \quad i \in 1, \ldots, L,\ j \in 1, \ldots, N. \tag{4.1}$$

mit L als Anzahl der Ausgabedimensionen. Die w_{ij}^{out} gewichten die Basisfunktionen dabei für jede Ausgabekomponente unabhängig.

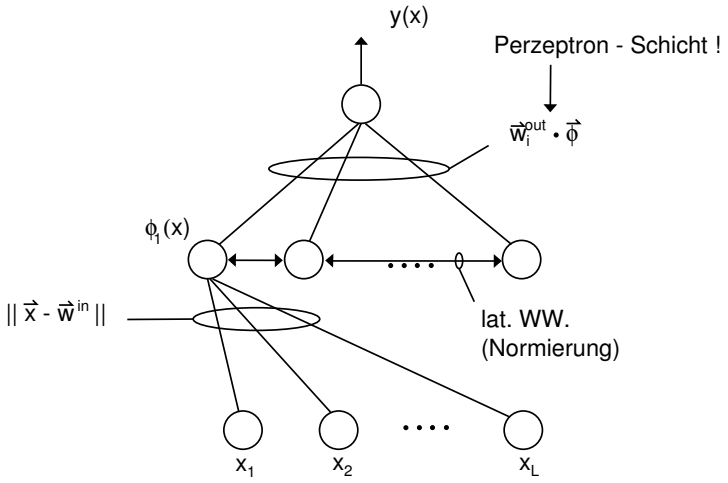


Abbildung 10: Netzwerkarchitektur des RBF-Ansatzes: In der mittleren Schicht bilden die Basisfunktionen ein Feature-Set, das über die Ausgabeschicht linear kombiniert wird. Der Unterschied zum MLP aus Beispiel 4.1.1 liegt in der Bestimmung der Aktivitäten bzw. Feature in der mittleren Schicht.

Das **Training des RBF-Netzes** zerfällt in 2 Teile, die getrennt voneinander optimiert werden können.

1. Bestimmung geeigneter **Zentren** $\mathbf{w}_j^{in}$ und **Radien** σ_j, $j = 1 \ldots N$. Die Gewichte $\mathbf{w}_j^{in}$ können aufgrund des Übergangs zur Differenz $\mathbf{x} - \mathbf{w}_j^{in}$ als Prototypen im Eingaberaum aufgefasst und *unüberwacht* gelernt werden!

[1] Man beachte, dass im Falle $\|\mathbf{x}\|, \|\mathbf{w}\| = \text{const}$ beide Ausdrücke semantisch äquivalent sind:
$\|\mathbf{w} - \mathbf{x}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{w}\|^2 - 2\mathbf{w} \cdot \mathbf{x}$

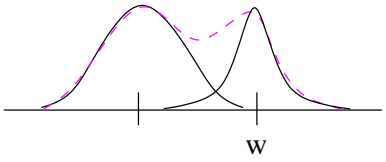
2. Bestimmung geeigneter Ausgabegewichte $\mathbf{w}_{ij}^{out}$ $i = 1 \dots L$, $j = 1 \dots N$. Die Ausgabegewichte können aufgrund der einfachen Linearkombination von Basisfunktionen mittels linearer Regression direkt bestimmt werden!

Bestimmung geeigneter Zentren
Hier sind viele Varianten denkbar.

- Am Einfachsten ist eine zufällige Wahl der Zentren aus dem Datenraum oder aber eine Zufallsauswahl auf den Trainingsbeispielen. Dies ermöglicht eine einfache aber grobe Abdeckung des Eingaberaums.

- Die bereits vorgestellten topologischen Lernverfahren (SOM, GNG, VQ) bieten eine wesentlich bessere Abdeckung des Eingaberaums.

Bestimmung geeigneter Radien σ_j, wobei meist $G_j(\mathbf{x}, \mathbf{w}^{in}) = R\left(\frac{||x - w_k^{in}||}{\sigma_k}\right)$, d.h., dass jede Basisfunktion einen eigenen Radius σ_k hat. Die Radien σ_k müssen so gewählt sein, dass die einzelnen RBF's zu einer „glatten" Funktion verschmelzen!

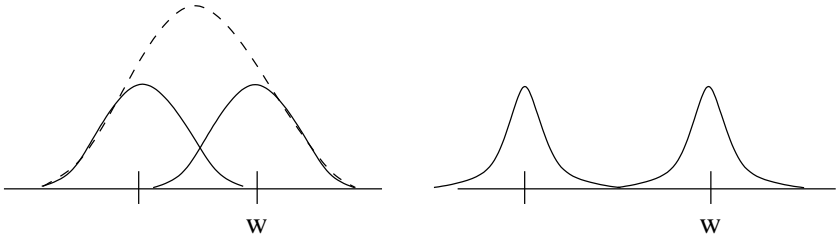


Ein einfaches Verfahren ist die „Nächste Nachbar-Heuristik":

$$\sigma_k = \lambda \cdot \min_l ||\mathbf{w}_k^{in} - \mathbf{w}_l^{in}||$$

und Optimierung der Approximation bzgl. des einzigen verbleibenden Parameters $\lambda \approx 1$.

Problem: zu dicht beieinanderliegende Prototypen produzieren starke Peaks, zu weit entfernt liegende Prototypen verschmelzen nicht.



Varianten der RBF

- Verwende Soft-Max-Normierung:

$$G'_j = \frac{G_j(\mathbf{x}, \mathbf{w}^{in})}{\sum_j G_j(\mathbf{x}, \mathbf{w}^{in})} \qquad y_i(\mathbf{x}) = \sum_{j=1}^{N} w_{ij}^{out} \cdot G'_j(\mathbf{x}, \mathbf{w}_j^{in})$$

  $\rightarrow$ kein Abfallen der Funktion, Ausgaben liegen in der konvexen Hülle der w_{ij}^{out}
  $\rightarrow$ stärkere Interpolation zwischen den Knoten

- Verwende alternative Basisfunktionen $G(x, w) = R(||x - w||)$, z.B. Inverse-Hardy-Multiquadriken (HIMQ) $R(s) = \frac{1}{\sqrt{s^2 + c^2}}$ $\rightarrow$ schneller Abfall mit s.

Erinnerung: Für Ausgabe-Mischungen bei der LLM wurde im letzten Kapitel folgende Formel entwickelt (gewichtete Überlagerung aller Knotenausgaben):

$$\mathbf{y}^{net}(\mathbf{x}) = \sum_c \mathbf{y}_c \cdot g_c(\mathbf{x}) = \sum_c \left(\mathbf{w}_c^{out} + A_c(\mathbf{x} - \mathbf{w}_c^{in})\right) \cdot g_c(\mathbf{x})$$

wobei $g_c(\mathbf{x})$ eine Mischungsfunktion zur Gewichtung der einzelnen Knotenbeiträge ist. Typischerweise verwendet man eine Soft-Max-Gewichtung mit Gauss'schen Mischungsfunktionen:

$$g_i(x) = \frac{R\left(\frac{||x-w_i^{in}||}{\sigma_i}\right)}{\sum_j R\left(\frac{||x-w_j^{in}||}{\sigma_j}\right)} \qquad \text{(soft-max)}$$

Der einzige Unterschied zur Berechnung der RBF besteht in der zusätzlich vorhandenen linearen Abbildung bei der LLM.

Bestimmung der Ausgabegewichte w_{ij}^{out}: Sind die Zentren und Radien bestimmt, ergeben sich die Ausgabegewichte als Lösung eines linearen Regressionsproblems. Dazu schreiben wir die Gleichung (4.1) zunächst in Vektornotation:

$$\begin{array}{lllll} & \mathbf{y}(\mathbf{x}) = W^{out} \cdot \mathbf{G}(\mathbf{x}) & & & \\ \text{mit} & W^{out} = (w_{ij}^{out}) & W^{out} \in \mathbb{R}^{L\times N} & L \text{ – Ausgabedimension: } \mathbf{y} \in \mathbb{R}^L \\ & \mathbf{G}(\mathbf{x}) = (G_j(\mathbf{x}, \mathbf{w}_j^{in})) & \mathbf{G} \in \mathbb{R}^N & N \text{ – Anzahl der Basisfunktionen} \end{array}$$

Ziel: Minimierung des Interpolationsfehlers

$$E(W^{out}) = \frac{1}{2}\sum_{\alpha=1}^{M} ||\mathbf{y}(\mathbf{x}^\alpha) - \mathbf{y}^\alpha||^2 \equiv \frac{1}{2}\left\langle (W \cdot \mathbf{G}(\mathbf{x}^\alpha) - \mathbf{y}^\alpha)^T \cdot (W \cdot \mathbf{G}(\mathbf{x}^\alpha) - \mathbf{y}^\alpha)\right\rangle_\alpha \qquad \text{bzgl. } \mathbf{W}$$

Lösung: $W^T = \left\langle \mathbf{G}_\alpha \cdot \mathbf{G}_\alpha^T \right\rangle_\alpha^{-1} \cdot \left\langle \mathbf{G}_\alpha \cdot \mathbf{y}_\alpha^T \right\rangle_\alpha$

- es existieren numerische Verfahren zur Bestimmung von W^T
- alternativ: Bestimmung der w_{ij}^{out} mit LLM-Verfahren
- alternativ: Bestimmung der w_{ij}^{out} z.B. mittels Gradientenabstieg:

$$\Delta w_{ij}^{out} = -\eta \frac{\partial E}{\partial w_{ij}^{out}} = \eta\,(y_i^\alpha - y_i(\mathbf{x}^\alpha)) \cdot G_j(\mathbf{x}^\alpha, \mathbf{w}_j^{in})$$

4.3 Generalisierte RBF's

Ansatz. Verwende nicht nur rundliche, sondern ellipsoide (anisotrope) rezeptive Felder bzw. Basisfunktionen.

→ Ersetze den (isotropen) euklidischen Abstand $||\mathbf{x} - \mathbf{w}_j^{in}||$ durch die Mahalanobis-Distanz

$$(\mathbf{x} - \mathbf{w}_j^{in})^T \cdot C_j^{-1} \cdot (\mathbf{x} - \mathbf{w}_j^{in})$$

- Die symmetrischen Matrizen C_j stellen die Kovarianzmatrizen der jetzt ellipsoiden Gaußglocken dar. Falls wir unterschiedliche Matrizen C_j für jede Basisfunktion zulassen ($C_i \neq C_j$) erhalten wir eine sehr große Flexibilität und in der Regel eine viel bessere Approximation.

- Bestimmung der C_j ist nichtlineares Optimierungsproblem mit (zu) vielen Parametern (Overfitting-Gefahr)
- Heuristik zur Bestimmung der C_j: Schätzung der Kovarianz aus den Daten innerhalb jeder Voronoizelle V_j (erfordert ausreichend viele Datenpunkte pro Voronoizelle)
- Abwägung zwischen Einschränkung der Variabilität und Erhöhung der Anzahl von Basisfunktionen versus wenige Basisfunktionen mit voller Flexibilität

Vereinfachte GRBF's:

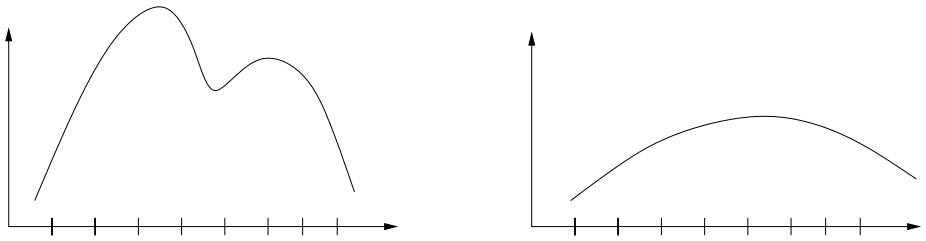
$$C_j = diag\{\sigma_{j1}^2, \sigma_{j2}^2, \ldots, \sigma_{jd}^2\} \qquad \text{für alle Basisfunktionen } j$$

Die Bestimmung der Parameter σ_{ji} stellt ein nichtlineares Optimierungsproblem dar, dass z.B. mittels des Levenberg-Marquardt-Verfahrens gelöst werden kann. Diese Verfahren sind aber aufwendig und man läuft Gefahr, in lokalen Minima stecken zu bleiben.

Näherungsansatz (Botros & Atkeson [**?**]): Wähle σ_{ji} entsprechend dem *mittlerem Funktionsgradienten* und der Varianz der Daten innerhalb der Voronoizelle V_j eines Prototypen $\mathbf{w}_j^{in}$:

$$q_{ji} = \frac{1}{\sigma_{ji}} = \frac{\langle |\nabla_i f(\mathbf{x}^\alpha)| \rangle_{V_j}}{\langle ||\nabla f(\mathbf{x}^\alpha)|| \rangle_{V_j}} \cdot \frac{1}{\lambda \cdot \sqrt{\left\langle (x_i^\alpha - w_{ji}^{in})^2 \right\rangle_{V_j}}}$$

- zweiter Faktor entspricht Nächster-Nachbar-Heuristik: Varianz der Daten entlang der i-Achse innerhalb der Voronoizelle des Prototypen $\mathbf{w}_j^{in}$.
- Parameter $\lambda \approx 1$ regelt wieder den Einfluß der Varianz
- erster Faktor: Schätze (normalisierte) Steilheit der Funktion entlang der i-Achse
  - ◇ hohe Steilheit = starke Änderungen → kleine σ
  - ◇ geringe Steilheit = wenig Änderungen → große σ



- aufgrund der Normierung führen nur stark unterschiedliche Ableitungen entlang der Ausgabedimensionen i zu einer Verformung der sonst kugelförmigen Gaußglocken

⇒ Gesamteffekt = Kompromiss zwischen Daten-Treffen und „Gradienten bereitstellen können".

Anwendung geschieht iterativ, da die Funktion dessen Gradienten bestimmt werden soll ja gerade approximiert werden soll und daher anfang unbekannt ist.

Startpunkt: Wähle $C^{(0)}$ aufgrund der Varianz im Eingaberaum

$$C^{(0)} \to f^{(0)} \to C^{(1)} \to f^{(1)} \to C^{(2)} \cdots$$

4.4 Mathematischer Hintergrund

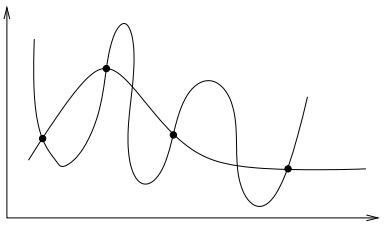
4.4.1 Der Regularisierungsansatz

(Girosi et al, NC 1995 [**?**])

Gegeben: M Datenpunkte $(\mathbf{x}^\alpha, \mathbf{y}^\alpha)$ mit $\alpha = 1 \dots M$
Gesucht: „möglichst gute“ Interpolationsfunktion $f(\mathbf{x})$

Probleme:

- Datenpunkte allein legen $f(\mathbf{x})$ nicht eindeutig fest
- Datenpunkte selbst häufig verrauscht



→ zusätzliche einschränkende Bedingungen notwendig

- ◇ Einschränkung auf Funktionenfamilie $\mathcal{F}$, z.B. durch Netzarchitektur bei MLP oder Festlegung von Basisfunktionen bei RBF
- ◇ Einführung von Regularisierungstermen in der Fehlerfunktion zur Beschränkung z.B. auf möglichst glatte Lösungen

$$E(f) = \underbrace{\frac{1}{2}\sum_{\alpha=1}^{M} ||f(\mathbf{x}^\alpha) - \mathbf{y}^\alpha||^2}_{\text{quad. Approximationsfehler}} \quad + \quad \underbrace{\lambda \cdot \Phi(f)}_{\text{Regularisierungsterm}}$$

- Der Regularisierungsansatz wird zeigen, dass beide Ansätze eng miteinander verknüpft sind!

Glattheit einer Funktion ist gut durch ihre Fourierkoeffizienten $\tilde{f}(\mathbf{k})$ ausdrückbar:

$$\tilde{f}(\mathbf{k}) = \int e^{-i\mathbf{kx}} \cdot f(\mathbf{x})\, d^d\mathbf{x}$$

$$\text{Rücktransformation} \qquad f(\mathbf{x}) = \frac{1}{(2\pi)^d} \int e^{+i\mathbf{kx}} \cdot \tilde{f}(\mathbf{k})\, d^d\mathbf{k}$$

$|\mathbf{k}|$ – Wellenzahl $\quad \propto \quad$ Frequenz

Ansatz für ein Glattheitsmaß $\Phi(f)$:

$$\Phi(f) = \frac{1}{2} \cdot \left(\frac{1}{2\pi}\right)^d \cdot \int \frac{|\tilde{f}(\mathbf{k})|^2}{\tilde{G}(\mathbf{k})}\, d^d\mathbf{k} \tag{4.2}$$

mit folgenden Eigenschaften der Gewichtungsfunktion

$\tilde{G}(\mathbf{k}) > 0$ und
$\tilde{G}(\mathbf{k}) \to 0$ für $|\mathbf{k}| \to \infty$ und
$\tilde{G}(\mathbf{k}) = \tilde{G}(-\mathbf{k})$ (Symmetrie – nötig für die Rücktransformation)

Interpretation: hochfrequente Fourierkomponenten ($|\mathbf{k}| \gg 0$) werden stark bestraft, $\frac{1}{\tilde{G}}$ ist Hochpassfilter im Fourierraum.

Bemerkung. Je nach Wahl von $\tilde{G}$ ist $\sqrt{\Phi(f)}$ Norm oder Halbnorm

$$\text{Norm} \begin{cases} \text{Halbnorm} \begin{cases} 1.\ ||f+g|| \leq ||f|| + ||g|| \quad \text{Dreiecksungleichung} \\ 2.\ ||f|| \geq 0 \\ 3.\ ||\lambda \cdot f|| = |\lambda| \cdot ||f|| \end{cases} \\ 4.\ ||f|| = 0 \quad \Leftrightarrow \quad f \equiv 0 \end{cases}$$

Halbnorm: es existiert ein nichttrivialer Nullraum $N = \{f \in \mathcal{F} | \Phi(f) = 0\}$. Funktionen, die sich um ein $f \in N$ unterscheiden sind für Φ gleich glatt.

Theorem 4.4.1. Jede Lösung $f^*()$ der Minimierungsaufgabe

$$f^* = \arg\min_f E(f) \tag{4.3}$$

ist von der Gestalt:

$$f(\mathbf{x}) = \sum_{\alpha=1}^{M} w_\alpha \cdot G(\mathbf{x} - \mathbf{x}^\alpha) + \sum_{\beta=1}^{M'} q_\beta P_\beta(\mathbf{x}) \tag{4.4}$$

wobei $\quad G(\mathbf{x}) = \left(\frac{1}{2\pi}\right)^d \int e^{i\mathbf{kx}} \cdot \tilde{G}(k) d^d k \quad$ (Fourier-Rücktransformation von $\tilde{G}$)

und $\{P_\beta(\cdot)\}$ stellt eine Basis des Nullraums von $\Phi(\cdot)$ dar.

Aufgrund des quadratischen Fehlermaßes erhält man eine lineare Überlagerung der Basisfunktionen, wobei die Gewichtungsparameter w_α und q_β folgenden linearen Gleichungen genügen:

$$\begin{aligned} (G + \lambda \mathbf{1}) \cdot \mathbf{w} + P^t \mathbf{q} &= \mathbf{y} \\ P\mathbf{w} &= 0 \end{aligned}$$

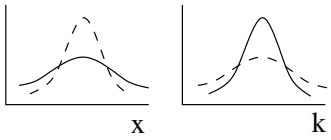
wobei $(\mathbf{y})_\alpha = y^\alpha$, $(\mathbf{w})_\alpha = w_\alpha$, $(\mathbf{q})_\beta = (q_\beta)$, $(G)_{ij} = G(\mathbf{x}^i - \mathbf{x}^j)$ und $(P)_{\beta\alpha} = P_\beta(\mathbf{x}^\alpha)$.

Bemerkungen

1. Die Wahl eines Regularisierungsansatzes (in Form der $\tilde{G}$) legt die Funktionenklasse $\mathcal{F}$ fest (in Form der Basisfunktionen G und P_β) und umgekehrt!
2. Der gewählte Regularisierungsansatz führt automatisch zur *radialen* Basisfunktionen, d.h. RBF's sind Lösungen eines durch Glattheitsforderungen regularisierten Interpolationsproblems.
3. Minimierung des „Datenfehlers“ alleine würde Glattheitsterm $\Phi(f)$ unterschlagen

4.4.2 Beispiele

Gaußfunktion für $\tilde{G}$
Die Fouriertransformierte $G = \mathcal{F}(\tilde{G})$ ist wieder eine Gaußfunktion mit Standardabweichung $\sigma = \tilde{\sigma}^{-1}$:



$$f(x) = e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}$$
$$\leftrightarrow \tilde{f}(k) = const \cdot e^{-\frac{x^2 \cdot \sigma^2}{2}}$$

Nullraum N = $\{0\}$ $\Rightarrow$ keine zusätzlichen Funktionen $P_\beta(\cdot)$ notwendig. D.h. unser ursprünglicher RBF-Ansatz mit Gaußschen Basisfunktionen entspricht einem Gauß-förmigen Hochpassfilter im Frequenzraum.

Polynome für $\tilde{G}$

$$\tilde{G} = ||\mathbf{k}||^{-2m}, \quad m \in \mathbb{N}$$
$$\rightarrow G(\mathbf{x}) = \begin{cases} ||\mathbf{x}||^{2m-d} \cdot \ln ||\mathbf{x}|| & \text{falls } 2m > d \text{ und } d \text{ gerade} \\ ||\mathbf{x}||^{2m} & \text{sonst} \end{cases}$$
$$\Rightarrow \text{Nullraum}N = \{\text{Polynome mit Grad } \leq m\}$$

Beispiel $m = 1$

$$\tilde{G} = \frac{1}{k^2}$$
$$\Phi(f) = \frac{1}{2} \int k^2 |\tilde{f}(k)|^2 \, d^d\mathbf{k}$$

Erinnere: Die Fouriertransformierte einer Ableitung ∇f entspricht:

$$\mathcal{F}(\nabla f)(\mathbf{k}) = -i \cdot \mathbf{k} \cdot \tilde{f}(\mathbf{k})$$
$$\text{also} \qquad \Phi(f) = \frac{1}{2} \int |\nabla f(\mathbf{x})|^2 \, d^d\mathbf{x}$$

$\Rightarrow m = 1$ versucht den mittleren Gradienten klein zu halten.

5 Mixture Models

Mischungsmodelle (oder sog. Komitee-Maschinen) basieren auf dem Prinzip von „divide-and-conquer“: Sie trainieren mehrere Expertennetze, die sich auf Teilaufgaben spezialisieren, und kombinieren deren Ausgaben zu einer Gesamtantwort des Netzes.[2] Wir werden drei unterschiedliche Architekturen zur Ausgabemischung behandeln:

- Ensemble Averaging (datenunabhängige lineare Funktion)
- Boosting (adaptive Gewichtung der Trainingsbeispiele)
- Mixture of experts (dataabhängige Mischungsfunktion)

Mischungsmodelle können aufgrund ihrer Modularität die Trainingszeit verkürzen und Overfitting vorbeugen.

5.1 Mixture of Experts.

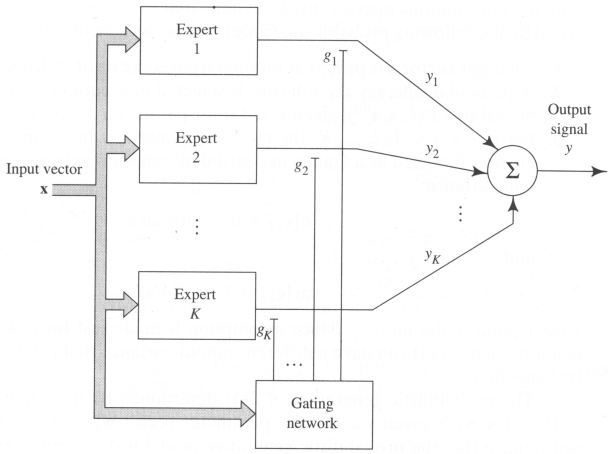


Abbildung 11: Mixture-of-Experts-Modell: Die Expertenausgabe wird über ein Gating-Netzwerk gewichtet aufsummiert.

Dieser Ansatz nutzt ein adaptives *Gating-Netzwerk* g, welches sich *datenabhängig* für eine Expertenausgabe entscheidet:

Experten:	$f_i(\mathbf{x}) = f_i(\mathbf{x}, \mathbf{w}_i)$ $\quad i = 1, \ldots, N$	N – # Experten
Gating-Netz:	$g_i(\mathbf{x}) = g_i(\mathbf{x}, \mathbf{v}_i)$	Gewichtung der Experten
Gesamtausgabe:	$y = \sum_{i=1}^{N} g_i(\mathbf{x}) \cdot f_i(\mathbf{x})$	

g ist als Wahrscheinlichkeitsverteilung interpretierbar, falls $\sum_i g_i(\mathbf{x}) = 1$.
Verwende typischerweise Soft-Max-Aktivierung:

$$g_i(\mathbf{x}, \mathbf{v}_i) = \frac{\exp(s_i(\mathbf{x}, \mathbf{v}_i))}{\sum_j \exp(s_j(\mathbf{x}, \mathbf{v}_j))} \quad \Rightarrow \quad \begin{array}{l} g_i(\mathbf{x}) > 0 \\ \sum_i \; g_i(\mathbf{x}) = 1 \end{array}$$

mit $s_i(\mathbf{x}, \mathbf{v}_i) = \mathbf{x} \cdot \mathbf{v}_i$ oder $s_i(\mathbf{x}, \mathbf{v}_i) = -\|\mathbf{x} - \mathbf{v}_i\|^2$.

[2] Beachte die Ähnlichkeit zum LLM. Dort ist jedes einzelne Neuron ein Experte für seine Voronoi-Zelle und lernt innerhalb dieser einen Teil der Gesamtabbildung. Entweder wird dann mittels WTA der zuständige Experte (=Neuron) ausgewählt oder mittels Soft-Max-Mischung eine Ausgabemischung erzeugt.

Statistische Interpretation Dem Mixture-of-Experts-Ansatz liegt ein bestimmtes Datenmodell zugrunde: Die Ausgabe y zu einem Eingabevektor $\mathbf{x}$ wird von Teilprozessen $i = 1, \ldots, N$ mit a-priori W.keit $P(i|\mathbf{x})$ und prozeßspezifischer W.keit $P(y|i, \mathbf{x})$ erzeugt.

1. wähle Eingabevektor $\mathbf{x}$ entsprechend einer geg. Verteilung (meist Gleichverteilung)
2. Wähle Regel i entsprechend der W.keit $P(i|\mathbf{x}) = g_i(\mathbf{x}, \mathbf{v}_i^*)$
3. Regel i erzeugt Ausgabe $y = f_i(\mathbf{x}, \mathbf{w}_i^*) + \eta_i$ wobei η_i Gaußsches Rauschen modelliert.

$$P(y|i, \mathbf{x}) = \mathcal{N}_i \cdot \exp\left[-\frac{(y - f_i(\mathbf{x}, \mathbf{w}_i^*))^2}{2\sigma_i^2}\right] =: h_i(y, \mathbf{x}, \mathbf{w}_i^*)$$

Die Wahrscheinlichkeit für ein Datenpaar $(\mathbf{x}, y)$ ist damit gegeben durch:

$$P(y|\mathbf{x}) = \sum_{i=1}^{N} P(i|\mathbf{x}) \cdot P(y|i, \mathbf{x}) = \sum_{i=1}^{N} g_i(\mathbf{x}, \mathbf{v}_i^*) \cdot h_i(y, \mathbf{x}, \mathbf{w}_i^*)$$

Für die Wahrscheinlichkeiten $P(i|\mathbf{x})$ und $P(y|i, \mathbf{x})$ haben wir dabei funktionale Modelle unterstellt, die spezifische Parameter $\mathbf{v}_i^*$ und $\mathbf{w}_i^*$ verwenden. Das Lernproblem besteht nun darin, diese Parameter aufgrund von Trainingsdaten zu schätzen.
Zur besseren Lesbarkeit führen wir folgende Bezeichnungen ein:

$\Theta = (\mathbf{v}_1, \ldots, \mathbf{v}_N, \mathbf{w}_1, \ldots, \mathbf{w}_N)$	Parametersatz
$\mathbf{z}^\alpha = (\mathbf{x}^\alpha, y^\alpha)$	Datenbeispiel
$\mathbf{z} = (\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^M)$	alle Trainingsdaten

Das Lernziel besteht damit in der Aufgabe: Maximiere die W.keit, dass die gegebenen Daten $\mathbf{z}$ von Parametern Θ erzeugt wurden:

$$\text{Maximiere} \qquad p(\Theta|\mathbf{z})$$

Da alle Datenbeispiele unabhängig voneinander sind, können wir die Likelihood, d.h. die Wahrscheinlichkeit, dass die Trainingsdaten $\mathbf{z}$ von einem Parametersatz Θ erzeugt wurden, leicht bestimmen:

$$\begin{aligned} P(\mathbf{z}|\Theta) &= \prod_{\alpha} P(y^\alpha|\mathbf{x}^\alpha, \Theta) \\ &= \prod_{\alpha} \left[\sum_{i=1}^{N} P(y^\alpha|i, \mathbf{x}^\alpha, \Theta) \cdot P(i|\mathbf{x}^\alpha, \Theta)\right] \end{aligned}$$

Mittels der Bayes-Regel können wir daraus die uns interessierende Wahrscheinlichkeit $p(\Theta|\mathbf{z})$ bestimmen:

$$p(\Theta|\mathbf{z}) \overset{Bayes}{=} \frac{P(\mathbf{z}|\Theta) \cdot p(\Theta)}{\int P(\mathbf{z}|\Theta') \cdot p(\Theta') \cdot d\Theta'}$$

Unter Annahme einer Gleichverteilung für Θ, können wir den bzgl. Θ konstanten Nenner sowie $p(\Theta)$ bei der Maximierung außer Acht lassen. Eine Maximierung von $p(\Theta|\mathbf{z})$ ist daher äquivalent zum Maximum-Likelihood-Schätzer:

$$\text{Maximiere} \qquad P(\mathbf{z}|\Theta)$$

Einsetzen unseres generativen Datenmodells und Minimierung der negativen Log-Likelihood ergibt die Energiefunktion:

$$\begin{aligned} E(\Theta) : &= -\log P(\mathbf{z}|\Theta) \\ &= -\sum_{\alpha} \log \left[\sum_{i=1}^{N} P(i|\mathbf{x}^{\alpha}, \mathbf{v}_i) \cdot P(y|\mathbf{x}^{\alpha}, i, \mathbf{w}_i) \right] \\ &= -\sum_{\alpha} \log \left[\sum_{i=1}^{N} g_i(\mathbf{x}^{\alpha}, \mathbf{v}_i) \cdot h_i(\mathbf{x}^{\alpha}, y^{\alpha}, \mathbf{w}_i) \right] \end{aligned}$$

Diese Energiefunktion wird dem Datenmodell demnach besser gerecht als die übliche quadratische Fehlerfunktion.

Gradienten-Lernregel Die Optimierung der Energiefunktion bzgl. der Parameter $\mathbf{v}_i$ und $\mathbf{w}_i$ erfolgt wieder per Gradientenabstieg. Wir müssen also zunächst die Gradienten bestimmen. Dazu macht es Sinn zunächst die Ableitung $\partial g_i / \partial s_j$ zu bestimmen:

$$\begin{aligned} g_i(\mathbf{x}^{\alpha}, \mathbf{v}_i) &= \frac{e^{s_i}}{\sum_j e^{s_j}} \\ \frac{\partial g_i}{\partial s_j} &= \frac{\delta_{ij} e^{s_i} \cdot \sum - e^{s_j} \cdot e^{s_i}}{\sum \cdot \sum} = \delta_{ij} \cdot \frac{e^{s_i}}{\sum} - \frac{e^{s_i}}{\sum} \cdot \frac{e^{s_j}}{\sum} = \delta_{ij} \cdot g_i - g_i \cdot g_j) \end{aligned}$$

Damit können wir den Gradienten $\partial E / \partial s_j$ mittels der Kettenregel bestimmen:

$$\begin{aligned} -\frac{\partial E}{\partial s_j} &= \sum_{\alpha} \left[\frac{1}{R_{\alpha}} \sum_i h_i^{\alpha} \cdot \overbrace{(\delta_{ij} \cdot g_i^{\alpha} - g_i^{\alpha} \cdot g_j^{\alpha})}^{\partial g_i / \partial s_j} \right] \qquad \text{mit} \qquad R_{\alpha} = \sum_i g_i^{\alpha} \cdot h_i^{\alpha} \\ &= \sum_{\alpha} \left[\frac{g_j^{\alpha} h_j^{\alpha}}{R_{\alpha}} - g_j^{\alpha} \frac{\sum_i h_i^{\alpha} g_i^{\alpha}}{R_{\alpha}} \right] = \sum_{\alpha} \left[\frac{g_j^{\alpha} h_j^{\alpha}}{R_{\alpha}} - g_j^{\alpha} \right] \end{aligned}$$

Dabei rührt der Faktor $1/R_{\alpha}$ von der Ableitung des Terms $\log R_{\alpha}$ her.
Mit Hilfe der statistischen Interpretation können wir diesen Gradienten interpretieren:

$$\frac{g_j^{\alpha} h_j^{\alpha}}{R_{\alpha}} = \frac{g_j^{\alpha} h_j^{\alpha}}{\sum_i g_i^{\alpha} h_i^{\alpha}} = \underbrace{\frac{P(j|\mathbf{x}^{\alpha}) \cdot P(y^{\alpha}|j, \mathbf{x}^{\alpha})}{\sum_i P(i|\mathbf{x}^{\alpha}) \cdot P(y^{\alpha}|i, \mathbf{x}^{\alpha})}}_{P(y^{\alpha}|\mathbf{x}^{\alpha})} \overset{Bayes}{=} P(j|\mathbf{x}^{\alpha}, y^{\alpha})$$

Der erste Term ist also die a-posteriori Wahrscheinlichkeit $P(j|\mathbf{x}^{\alpha}, y^{\alpha})$, dass Prozess j das Datenbeispiel $(\mathbf{x}^{\alpha}, y^{\alpha})$ erzeugt hat – im Gegensatz zur a-priori Wahrscheinlichkeit $g_j^{\alpha} = P(j|\mathbf{x}^{\alpha})$ wird neben $\mathbf{x}^{\alpha}$ also auch die erzeugte Ausgabe y^{α} berücksichtigt. Der Gradient ist damit die Differenz von a-posteriori und a-priori Wahrscheinlichkeit:

$$-\frac{\partial E}{\partial s_j} = \sum_{\alpha} \left[\underbrace{P(j|\mathbf{x}^{\alpha}, y^{\alpha})}_{\text{a-posteriori}} - \underbrace{P(j|\mathbf{x}^{\alpha})}_{\text{a-priori}} \right].$$

Der Gradient verschwindet also erst (und das Lernen im Gating-Netz endet) falls die a-posteriori und die a-priori Verteilungen übereinstimmen, das Gating-Netz also gelernt hat, die a-priori Wahrscheinlichkeit richtig vorherzusagen.

Der Gradient bzgl. der Gewichte $\mathbf{w}_i$ der Experten ergibt sich direkt nach der Kettenregel:

$$-\frac{\partial E}{\partial \mathbf{w}_j} = \sum_\alpha \frac{1}{R_\alpha} \cdot g_j^\alpha \cdot \underbrace{h_j^\alpha \cdot \frac{y^\alpha - f_j(\mathbf{x}^\alpha, \mathbf{w}_j)}{\sigma_j^2}}_{\partial h_j / \partial f_j} \cdot \frac{\partial f_j}{\mathbf{w}_j}$$

$$= \sum_\alpha \sigma_i^{-2} \cdot \underbrace{\frac{g_j^\alpha \cdot h_j^\alpha}{R_\alpha}}_{P(j|\mathbf{x}^\alpha, y^\alpha)} \cdot \underbrace{\big(y^\alpha - f_j(\mathbf{x}^\alpha, \mathbf{w}_j)\big) \cdot \frac{\partial f_j}{\mathbf{w}_j}}_{\partial E^j_{\text{square}} / \partial \mathbf{w}_j}$$

Wir erhalten also gerade den Gradienten bzgl. der quadratischen Energiefunktion E^j_{square} des zugeordneten Expertennetzes j, gewichtet mit der a-posteriori Wahrscheinlichkeit, dass dieses Netz überhaupt zuständig für das Datenbeispiel war und damit den Fehler produziert hat. Die Varianz des weißen Rauschens wird typischerweise als konstant über alle Teilprozesse hinweg angenommen ($\sigma_i = \sigma = \text{const}$), so dass der Faktor σ_i^{-2} konstant ist und in die Lernrate η integriert werden kann.
Mit dem Übergang vom Batch- zum Online-Lernen (Weglassen der Summe über die Datenbeispiele α) erhalten wir insgesamt folgende Gradientenschritte:

$$\Delta \mathbf{v}_i = -\eta \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial \mathbf{v}_i} = \eta \cdot \big(P(i|\mathbf{x}^\alpha, y^\alpha) - P(i|\mathbf{x}^\alpha)\big) \cdot \frac{\partial s_i}{\partial \mathbf{v}_i}$$

$$\Delta \mathbf{w}_i = -\eta \frac{\partial E}{\partial \mathbf{w}_i} \qquad = \eta \cdot \sigma^{-2} \cdot P(i|\mathbf{x}^\alpha, y^\alpha) \cdot \big(y^\alpha - f_j(\mathbf{x}^\alpha, \mathbf{w}_j)\big) \cdot \frac{\partial f_i}{\partial \mathbf{w}_i}$$

Dank der Gewichtung des quadratischen Fehlers der Expertennetze mit der a-posteriori Wahrscheinlichkeit wird das Expertennetz am stärksten trainiert, von dem das Gating-Netz glaubt, es sei für die aktuelle Eingabe $\mathbf{x}$ zuständig. Auch wenn dies anfangs falsch sein mag, entsteht dadurch tatsächlich eine Aufteilung der Experten auf disjunkte Teilgebiete des Eingaberaums.

Expectation-Maximisation-Algorithmus (EM) Die Berechnung der Gradienten wäre sehr viel einfacher, wenn die Likelihood-Funktion $P(\mathbf{z}|\Theta) = \prod_\alpha \sum_i g_i h_i$ nur Produkte enthalten würde: $P'(\mathbf{z}|\Theta) = \prod_\alpha \prod_i g_i h_i$. Dieses Ziel kann erreicht werden, wenn bekannt wäre, welcher Teilprozess j eine Ausgabe y^α erzeugt hat.
Der EM-Algorithmus ist ein allgemeines Verfahren zur Optimierung einer Kostenfunktion bei Vorliegen unvollständig beobachtbarer Daten $\mathbf{z}^\alpha$.

- **Ziel:** Maximierung einer Likelihood-Funktion bei unvollständig beobachtbaren Daten $\mathbf{z}^\alpha$:

$$\text{Maximiere} \qquad L(\Theta) = logP(\mathbf{z}|\Theta)$$

- **Idee:** Wir nehmen an, die nicht-beobachtbaren Daten γ^α stünden zur Verfügung. Wir maximieren nun bzgl. der Likelihood-Funktion unter Annahme vollständiger Daten $\tilde{z}^\alpha = (\mathbf{z}^\alpha, \gamma^\alpha)$:

$$\text{Maximiere} \qquad L_{\text{complete}}(\Theta) = logP(\tilde{z}|\Theta)$$

- $L_{\text{complete}}(\Theta)$ ist eine Zufallsvariable aufgrund der unbekannten Daten γ^α.

EM-Algorithmus:

1. Initialisiere $t = 0$, Θ_t

2. Expectation: $Q_t(\Theta) = E_\gamma \left[L_{\text{complete}}(\Theta_t) \right]$
3. Maximization: $\Theta_{t+1} = \arg\max_\Theta Q_t(\Theta)$
4. $t \to t+1$ Wiederhole Schritte 2-4

Man kann zeigen: $L(\Theta_{t+1}) \geq L(\Theta_t)$ für alle t, d.h. es handelt sich um einen echten Aufstieg und der Prozess konvergiert.

Wir wenden diese allgemeine Idee nun auf den Mixture-of-Experts-Ansatz an.
Einführung von Indikatorvariablen γ_i^α zur Anzeige, welcher Teilprozess eine Ausgabe erzeugt hat:

$$\gamma_i^\alpha = \begin{cases} 1 & \text{falls Prozess } i \text{ das Datenbeispiel } (\mathbf{x}^\alpha, y^\alpha) \text{ erzeugt hat} \\ 0 & \text{sonst} \end{cases}$$

Die γ_i^α fassen wir als statistisch unabhängige, diskrete Zufallsvariablen auf, deren Erwartungswert gerade der a-posteriori Wahrscheinlichkeit für Prozess i entspricht:

$$E\left[\gamma_i^\alpha\right] = P(i|\mathbf{x}^\alpha, y^\alpha) = \frac{g_i h_i}{\sum_j g_j h_j} =: \hat{\gamma}_i^\alpha$$

Für die Wahrscheinlichkeiten $P(\tilde{z}|\Theta)$ gilt nun:

$$\begin{aligned} P(y^\alpha, \gamma^\alpha|\mathbf{x}^\alpha, \Theta) &= P(y^\alpha, j|\mathbf{x}^\alpha, \Theta) && \text{unter der Annahme } \gamma_i^\alpha = \delta_{ij} \\ &= \prod_{i=1}^N P(y^\alpha, i|\mathbf{x}^\alpha, \Theta)^{\gamma_i^\alpha} && \text{nur der Faktor } j \text{ des Produkts ist ungleich Eins} \\ &= \prod_{i=1}^N \left[P(i|\mathbf{x}^\alpha, \Theta) \cdot P(y^\alpha|i, \mathbf{x}^\alpha, \Theta)\right]^{\gamma_i^\alpha} = \prod_{i=1}^N \left[g_i(\mathbf{x}^\alpha) \cdot h_i(\mathbf{x}^\alpha)\right]^{\gamma_i^\alpha} \\ P(\tilde{z}|\Theta) &= \prod_\alpha P(y^\alpha, \gamma^\alpha|\mathbf{x}^\alpha, \Theta) = \prod_\alpha \prod_{i=1}^N \left[g_i(\mathbf{x}^\alpha) \cdot h_i(\mathbf{x}^\alpha)\right]^{\gamma_i^\alpha} \end{aligned}$$

Wir haben unser Ziel, die Summe $\sum_i$ durch ein Produkt $\prod_i$ zu ersetzen, also erreicht. Damit ergibt sich die vereinfachte negative Log-Likelihood-Funktion:

$$\begin{aligned} E_c(\Theta) &= -\log L_c(\Theta) = -\log P(\tilde{z}|\Theta) \\ &= -\sum_\alpha \sum_{i=1}^N \gamma_i^\alpha \cdot \left(\log g_i(\mathbf{x}^\alpha) + \log h_i(\mathbf{x}^\alpha)\right) \end{aligned}$$

Die so erzielte Entkopplung der Terme g_i und h_i ermöglicht nun die unabhängige Minimierung der Expertennetze (bzgl. $\mathbf{w}_i$) und des Gating-Netzes (bzgl. $\mathbf{v}_i$).

- Expectation-Schritt: Ersetze γ_i^α in $E_c(\Theta)$ durch den Erwartungswert

$$\begin{aligned} \hat{\gamma}_i^\alpha &:= E\left[\gamma_i^\alpha\right] = \frac{g_i h_i}{\sum_j g_j h_j} \\ \Rightarrow \qquad Q_t(\Theta) &= -\sum_\alpha \sum_i \hat{\gamma}_i^\alpha \cdot \left(\log g_i(\mathbf{x}^\alpha) + \log h_i(\mathbf{x}^\alpha)\right) \end{aligned}$$

- Maximization-Schritt: Minimiere bzgl. $\mathbf{v}_i$ und $\mathbf{w}_i$:

$$\begin{aligned}
\mathbf{v}_i^{t+1} &= \arg\min_{\mathbf{v}_i} -\sum_\alpha \sum_i \hat{\gamma}_i^\alpha \log g_i(\mathbf{x}^\alpha, \mathbf{v}_i) \\
\mathbf{w}_i^{t+1} &= \arg\min_{\mathbf{w}_i} -\sum_\alpha \sum_i \hat{\gamma}_i^\alpha \log h_i(\mathbf{x}^\alpha, \mathbf{w}_i) \\
&= \arg\min_{\mathbf{w}_i} -\sum_\alpha \sum_i \hat{\gamma}_i^\alpha \cdot \frac{1}{2\sigma^2}\big(y^\alpha - f_i(\mathbf{x}^\alpha, \mathbf{w}_i)\big)^2
\end{aligned}$$

Ein Gradientenabstieg auf diesen neuen Fehlerfunktionen liefert diesselben Lernregeln wie zuvor, jedoch war die Herleitung unter Ausnutzung des EM-Verfahrens viel einfacher. Weitere Details in Jordan & Jacobs, 1994 [**?**].

5.2 Probability density estimation

The following draws on the material presented in Bishop, Chapter 2.6. The mixture of experts framework including the EM algorithm can also be used for the unsupervised task to estimate a probability density from data as sum of Gaussian distributions.
Given data $\{\mathbf{x}^\alpha\}, \alpha = 1 \dots M$ we assume that the density $p(\mathbf{x})$ can be approximated by a sum of N component distributions

$$p(\mathbf{x}) = \sum_{i=1}^{N} p(\mathbf{x}|i)P(i),$$

where again $P(i)$ is the prior probability of the data point having been generated from component i of the mixture. Assume the prior satify

$$\sum_{i=1}^{N} P(i) = 1, \qquad 0 \le P(i) \le 1$$

and let the component density functions $p(x|i)$ are normalized Gaussians with mean μ_i and width σ^2 so that

$$p(\mathbf{x}|i) = \frac{1}{(2\pi\sigma_i)^{d/2}} \exp\left(-\frac{\|\mathbf{x} - \mu_i\|^2}{2\sigma_i^2}\right)$$
$$\int p(\mathbf{x}|i)d\mathbf{x} = 1.$$

Using the Bayes rule we get the posterior probabilities for the mixture component i given the data as

$$P(i|\mathbf{x}) = \frac{p(\mathbf{x}|i)P(i)}{p(\mathbf{x})}$$

and the negative log-likelihood

$$E = -\sum_{\alpha=1}^{M} \log P(\mathbf{x}^\alpha) = -\sum_{\alpha=1}^{M} \log\left[\sum_{i=1}^{N} p(\mathbf{x}^\alpha|i)P(i)\right].$$

Minimization of E now gives the desired optimal mixture parameters. Note the similarity to the mixture of experts approach: here the $\mathbf{x}^\alpha$ play the role of the outputs y^α, and are replaced by the mean μ_i, which is now one of the parameters. However, there rest of the formulas are identical.

Gradient descent for Gaussian mixture. Using the Gaussian density in the log-likelihood the gradient can easily be derived analytically as

$$\frac{\partial E}{\partial \mu_i} = \sum_{\alpha=1}^{M} P(i|\mathbf{x}^\alpha) \frac{(\mu_i - \mathbf{x}^\alpha)}{\sigma_i^2}.$$

Similarily for the width parameter we obtain

$$\frac{\partial E}{\partial \sigma_i} = \sum_{\alpha=1}^{M} P(i|\mathbf{x}^\alpha) \left[\frac{d}{\sigma_i} - \frac{\|\mathbf{x}^\alpha - \mu_i\|^2}{\sigma_i^3} \right].$$

Minimization with respect to the mixing probabilities $P(i)$ is more tedious and must be carried out with respect to the constraints $\sum P(i) = 1$. Using the softmax activation

$$P(i) = \frac{\gamma_i}{\sum_{k=1}^{N} \gamma_k}$$

again turns the problem into an unconstrained one in the auxiliary variables γ_i and some computation (similar to the previous section) yields

$$\frac{\partial E}{\partial \gamma_i} = -\sum_{\alpha=1}^{M} \Big[P(i|\mathbf{x}^\alpha) - P(j) \Big]),$$

i.e. the difference between prior and posterior probabilities. Setting the gradients to zero we get at the minimum

$$\begin{aligned} \mu_i^\star &= \frac{\sum_M P(i|\mathbf{x}^\alpha)\mathbf{x}^\alpha}{\sum_M P(i|\mathbf{x}^\alpha)} \\ \sigma_i^\star &= \frac{1}{d} \frac{\sum P(i|\mathbf{x}^\alpha)\|\mathbf{x}^\alpha - \mu_i^\star\|^2}{\sum P(i|\mathbf{x}^\alpha)} \\ P^\star(i) &= \frac{1}{N} \sum_M P(i|\mathbf{x}^\alpha), \end{aligned}$$

which is the intuitive result that at the minimum the mean of the mixture component is the mean of the data weighted be the posterior probabilities that the corresponding data were generated from the respective component. Similarily, the variance is the data variance weighted by the posterior probabilities and the prior for component i is given by the average over the posteriors for that very component.

Remarks.

- A direct minimisation approach not using the gradients can be derived starting from the last (recursive!) formulas
- it can be shown that this approach leads to the same log-likelihood energy to be minimized as the EM approach

6 Verarbeitung von Zeitreihen

bisher: statische Eingabe → statische Ausgabe
jetzt: Verarbeitung zeitlich-variabler Eingaben und Ausgaben

Anwendungen:

- Filterung
- Kompression
- Steuerung und Regelung
- Klassifikation von Zeitreihen, z.B. Sprache
- Zeitreihenvorhersage, z.B. Börsenkurse, Wetter, etc.

Grundannahme:
Ausgabe hängt nicht nur von aktueller Eingabe, sondern von Eingabe-Historie, dem Kontext ab.

Taxonomie von Zeitreihen

- deterministisch (auch det. Chaos) vs. stochastisch
- zeit-diskret vs. zeit-kontinuierlich
- stochastische:
  - ◇ Markov: endlich viele Vorgänger-Eingaben berücksichtigt
  - ◇ Nicht-Markov: beliebige Abhängigkeiten
- Meist Zeitreihen N-ter Ordnung: $N < \infty$

$$u(t) = F(u_{t-1}, u_{t-2}, \ldots, u_{t-N}) \qquad \text{diskret}$$

$$\dot{u}(t) = \int_0^N F(u - \tau)d\tau \qquad \text{kontinuierlich}$$

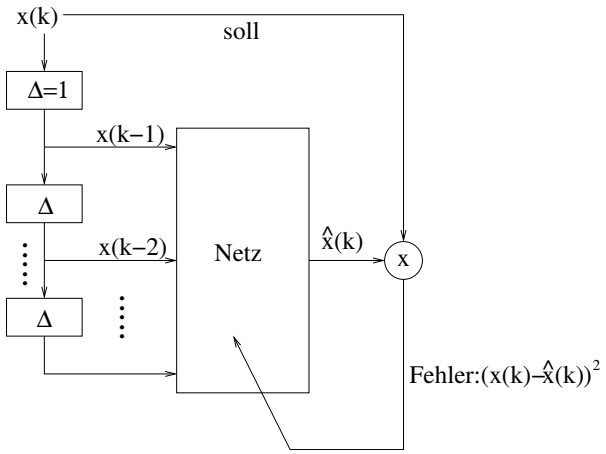
- einfachstes Modell: lineare Zeitreihen

$$u(t) = \sum_{\tau=1}^{N} a_\tau u_{t-\tau}$$

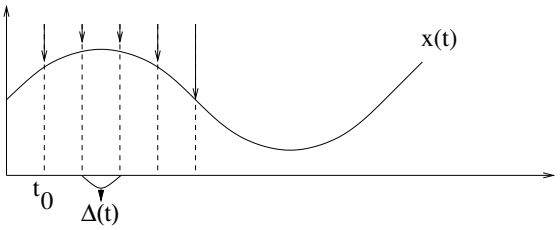
6.1 Architekturen zur Repräsentation von Kontext

6.1.1 Time-Delay-Neural-Network (TDNN)

Zeitverzögerungspuffer für die letzten N Eingaben:

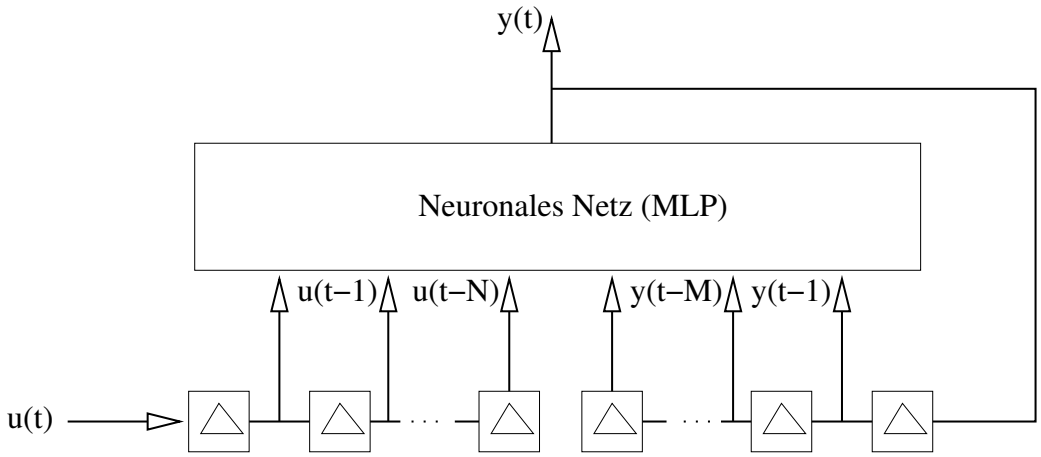


- Standard-Netz (z.B. MLP) mit Back-Prop einsetzbar
- Time-Delay-Line stellt ein Zeitfenster der Länge N dar
- optimale Breite N des Zeitfensters unbekannt
  (bestimmt durch Ordnung der zugrundeliegenden Funktion)
- mit N wächst Anzahl der Gewichte $O(H * N)$
- kontinuierliche Signale müssen diskretisiert werden, d.h. modelliere $x(t)$ durch $x(t_0), x(t_0+\Delta t), x(t_0+2\Delta t), \ldots, x(t_0+k\Delta t)$



- Vorverarbeitung und Analyse der geg. Zeitreihen notwendig

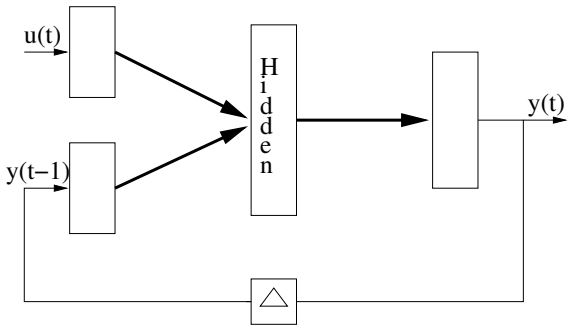
6.1.2 NARX recurrent neural networks (Nonlinear AutoRegressive with eXogenous inputs)



- Time-Delay-Line der Eingabe und zusätzlich
- rekurrente Rückkopplung der Ausgabe über eine zweite Time-Delay-Line
- Turing-universell
- echtes rekurrentes Netz: Loop in Netzwerkstruktur

6.1.3 Jordan-Netze (1988)

- Rückkopplung der Ausgabe:



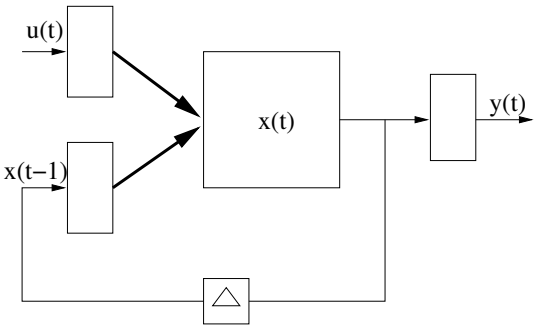
$$\begin{aligned} y(t) &= f(u(t), y(t-1)) \\ &= f(u(t), f(u(t-1), y(t-2))) \\ &= \dots \end{aligned}$$

- Information über alle vorherigen Zustände vorhanden (im Prinzip)
- **Nachteil:** gesamte Kontext-Information muss in Ausgabe kodiert werden
- **Training:** Standard-Back-Prop mit Teacher-Forcing:
  Für $y(t-1)$ setze korrekte Sollausgabe $y^{\text{soll}}(t-1)$ ein.

$$\Delta w_{ij}^{\text{rec}} = -\eta \delta_i y_j^{soll}(t-1)$$

6.1.4 Elman-Netze (1988-1991)

- Rückkopplung der Hidden-States



$$x_i(t) = \sigma(\sum w_{ij}^{\text{in}} u_j(t) + \sum w_{ij}^{\text{rek}} x_j(t-1))$$

$$y_i(t) = \sigma(\sum w_{ij}^{\text{out}} x_j(t))$$

$$\begin{aligned} \mathbf{x}(t) &= \sigma(W^{\text{in}} u(t) + W^{\text{rek}} x(t-1)) \\ \mathbf{y}(t) &= \sigma(W^{\text{out}} x(t)) \end{aligned}$$

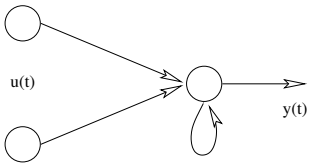
- Netz kann flexibel eine Repräsentation des Kontextes in den hidden units lernen
- vollständiges rekurrentes Netzwerk
- Vereinfachung des Lernprozesses durch Beschränkung (Truncation) auf MLP-Lernen, d.h. ignoriere rekurrente Verbindung beim Lernen. Fehler werden nicht rekursiv rückpropagiert.

6.1.5 Cascade-Corellation Network (CCN)

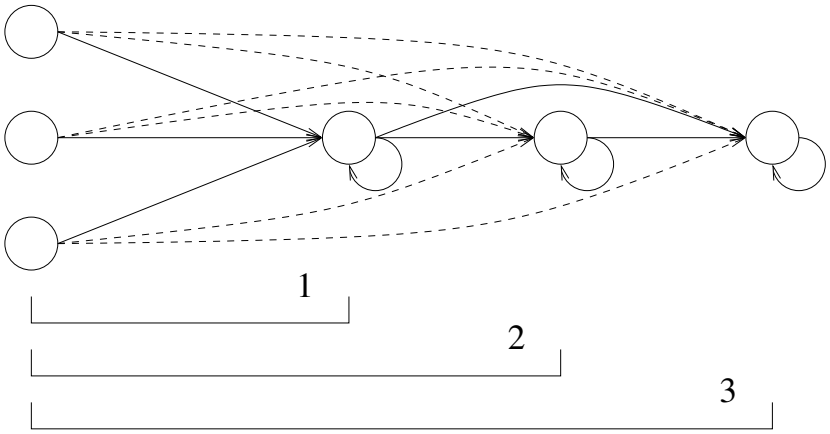
- die Wahl der Netzarchitektur bestimmt die Kapazität des Netzes
- "richtige" Architektur ist noch schwieriger als bei MLP zu finden, da rekurrente Gewichte W^{rec} hinzukommen

Idee: Baue die Netzwerkstruktur inkrementell auf.

1. Beginne mit Eingabe und einem Knoten (pro Ausgabe), der selbstrekurrent ist.



2. Trainiere bis der Fehler minimal wird.
3. Füge (pro Ausgabe) einen weiteren Knoten hinzu, der selbstrekurrent ist und nur Feed-Forward-Verbindungen von den bisherigen Knoten besitzt.



4. Trainiere *nur* die neuen Verbindungen.
5. Wiederhole 3-4 solange bis "zufrieden".

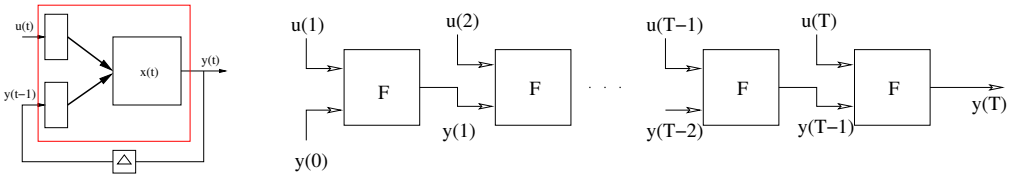
Bemerkungen:

- sehr effizient, da wenige Verbindungen zu trainieren
- rekurrente Verbindung kann trainiert oder festgelegt werden $w_{ii} \leq 1$
- auch auf MLP anwendbar (bei Weglassen der Selbstrekurrenz)

6.2 Backpropagation Through Time - BPTT

- Ausgabe eines RNNs kann für endlichen Zeithorizont T auch durch entsprechendes Feedforward-Netzwerk erzeugt werden
- **Idee:** Entfaltung in der Zeit

$$\begin{aligned} y(t) &= \sigma(W^{\text{in}}u(t) + W^{\text{rek}}y(t-1)) = F(y(t-1)) \\ &= F(F(F(\dots F(y(t-T-1))\dots))) \end{aligned}$$



Alle Feedforward-Schichten verwenden dieselben Gewichte W^{in} und W^{rek}.

- Trainiere mit Standard-Backprop und mittele die Gewichtsupdates Δw über alle Schichten $\hat{=}$ Zeitschritte.
  In jedem Schritt muss der Fehler $e(t)$ dieses Schrittes injeziert werden:

$$e_i(t) = \begin{cases} y_i(t) - y_i^{\text{soll}}(t), \text{falls Neuron i Ausgabeneuron} \\ 0, \text{sonst} \end{cases}$$

Back-Prop

- Beginne in letzter Schicht (T):

$$\delta_i(T) = +\sigma'(a_i(T)) \cdot e_i(T)$$

- Back-Prop-Schritt von Schicht zu Schicht:

$$\delta_i(t) = \sigma'(a_i(t)) \cdot \big(\underbrace{e_i(t)}_{\text{injection error}} + \textstyle\sum_k w_{ki}\delta_k(t+1) \big)$$

- Mittele die Gewichtsupdates aller Schichten:

$$\Delta w_{ij} = -\eta \textstyle\sum_{t=1}^{T} \delta_i(t) y_j(t-1)$$

Bemerkungen:

- Problem: Gradienten verschwinden schnell bei zu vielen Schichten
- Truncated BPTT: Back-Prop häppchenweise in Blöcken zu $\tau \approx 3 \ll T$
- Aufwand: $O(T \cdot n^2)$

7 Neural Dynamics

RNN zeigen komplexes dynamisches Verhalten:

- stabile konstante Ausgabe
- Oszillationen bel. komplexer Natur
- deterministisches Chaos:
  beliebig nahe beinander startende Trajektorien divergieren mit der Zeit

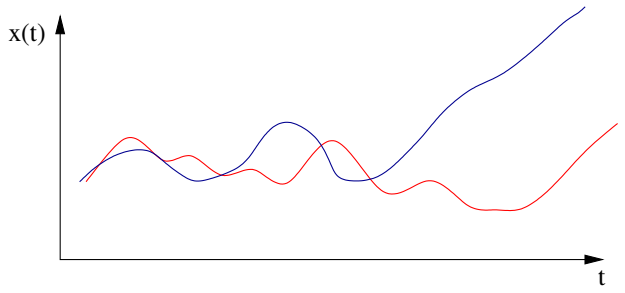


Abbildung 12: Stabilität eines RNN ist nicht explizit gegeben

Anwendungen von dynamischem Verhalten für Berechnungen:

- stabile Gleichgewichte als gespeicherte Muster
- stabile Gleichgewichte als (lokale) Minima von Optimierungsproblemen
- stabile Gleichgewichte als Bindungszustand bei Segmentierungsaufgaben
- stabile Oszillationen als Bindungszustand in Bildsegmentierung
- spezifische Oszillationen als *Central Pattern Generators* (CPG) zur Erzeugung von Bewegungsmustern
- Ausgabe-Trajektorie zur Zeitserien-Vorhersage
- diskrete Netzwerkzustände als Implementation von endlichen Automaten
- ...

7.1 Stabilität Dynamischer Systeme

Wir betrachten zeit-kontinuierliche und zeit-diskrete rekurrente neuronale Netze

$$\text{zeit-kontinuierlich:} \qquad \dot{\boldsymbol{x}} = -D\boldsymbol{x} + W\sigma(\boldsymbol{x}) + W_u \cdot u(t) \qquad t \in \mathbb{R}$$

mit den Variablen:

- $\boldsymbol{x}$ - Zustandsvektor $\cong$ Ausgabe aller Neuronen $\boldsymbol{x} = (x_1, \cdots, x_n)^t$
- D - Diagonalmatrix mit Elementen $d_i > 0$, häufig $D = \mathbf{1}$
- W - Gewichtsmatrix $W \in \mathbb{R}^{n \times n}$
- $W_u, u(t)$ - Externe Eingabe mit Gewichtsmatrix W_u: $\quad I(t) = W_u \cdot u(t)$
- σ - sigmoide Aktivierungsfunktion

Im diskreten Fall:

$$x(t+1) = W \cdot \sigma(x(t)) + I(t) \qquad t \in Z \tag{1}$$
$$y(t+1) = \sigma\left(W \cdot y(t) + I(t)\right) \qquad t \in Z \tag{2}$$

Beide diskreten Systeme (1) und (2) haben identisches Verhalten, da sie lediglich unterschiedliche Variablen desselben Prozesses modellieren:

(1) $\mathbf{x} \cong$ Zellpotential (synaptische Summation): $x(t+1) = W \cdot y(t) + I$

(2) $\mathbf{y} \cong$ Feuerrate: $y(t+1) = \sigma(x(t+1))$

Die Simulation der kontinuierlichen Differentialgleichung erfordet immer eine Zeit-Diskretisierung, z.B. mittels Euler-Approximation:

$$x(t+\Delta t) \approx x(t) + \Delta t \cdot \dot{x}(t) = x(t) + \Delta t(-D \cdot x + W \cdot \sigma(x) + I)$$

Für $\Delta t = 1$ und $D = \mathbf{1}$ führt die Euler-Diskretisierung auf das zeit-diskrete RNN (1).

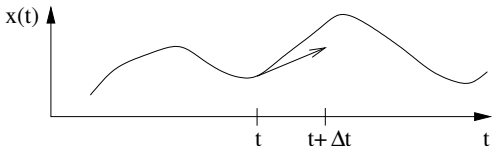
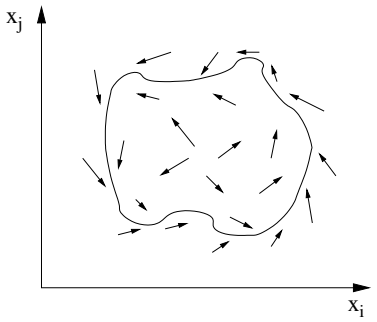


Abbildung 13: Zeit-Diskretisierung mittels Euler-Approximation

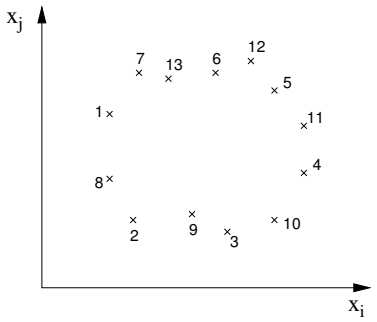
Aber: Die Dynamik beider Modelle (des zeit-kontinuierlichen und des zeit-disrekten) unterscheidet sich – auch qualitativ – auf Grund des unvermeidlichen Diskretisierungsfehlers. Nur im Limit $\Delta t \to 0$ erhält man dieselbe Dynamik.

Visualisierung: Phasenplots und **Vektorfelder**

Phasenplot: Trage zwei Zustandskomponenten x_i und x_j gegeneinander auf.
Vektorfeld: Zusätzlich werden im Phasenplot die Geschwindigkeitsvektoren $\dot{\boldsymbol{x}}$ der DGL (auf einem Gitter) eingezeichnet.

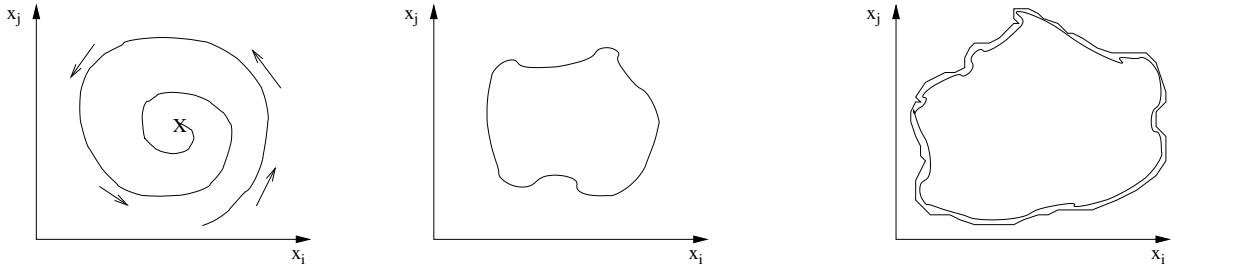


Vektorfeld eines
zeit-kontinuierlichen Systems



zeit-diskretes System:
zeitliche Informationen gehen verloren

Anhand der Phasendiagramme kann man sehr gut das qualitative Verhalten eines dynamischen Systems einschätzen. Jede Art von Verhalten zeigt nämlich ein charakteristisches Phasendiagramm:



Fixpunkt Oszillation: geschlossene Kurve Chaos: verschmierte Trajektorie

Stabilität von Fixpunkten
Fixpunkte bzw. Gleichgewichte $\bar{\boldsymbol{x}}$ zeigen die einfachste Art dynamischen Verhaltens: Ihre Trajektorie ändert sich nicht im Verlauf der Zeit, d.h. $\boldsymbol{x}(t) \equiv \bar{\boldsymbol{x}}$. Wir betrachten dazu die folgenden (vereinfachten) Systeme von oben:

$$\dot{\boldsymbol{x}} = -\boldsymbol{x} + \overbrace{W \cdot \sigma(\boldsymbol{x}) + I}^{F} = -\boldsymbol{x} + F(\boldsymbol{x}, I)$$
$$\boldsymbol{x}(t+1) = W \cdot \sigma(\boldsymbol{x}) + I = F(\boldsymbol{x}, I)$$

Die Fixpunktbedingungen lauten demnach:

$$\dot{\bar{\boldsymbol{x}}} = 0 \qquad \text{bzw.} \qquad \boldsymbol{x}(t+1) = \boldsymbol{x}(t) \equiv \bar{\boldsymbol{x}} \qquad \Rightarrow \qquad F(\bar{\boldsymbol{x}}, I) = \bar{\boldsymbol{x}}$$

Fixpunkte werden nach ihrer lokalen und globalen Stabilität klassifiziert.
Lokal: Lokale Linearisierung
Global: Lyapunov-Funktion

Definition Stabilität:

(i) Ein Fixpunkt $\bar{\boldsymbol{x}}$ heißt (Lyapunov) stabil, falls es für jede (noch so kleine) Umgebung $U_\varepsilon(\bar{\boldsymbol{x}})$ eine (eventuell kleinere) Umgebung $U_\delta(\bar{\boldsymbol{x}})$ gibt, so dass Trajektorien, die in $U_\delta(\bar{\boldsymbol{x}})$ starten, $U_\varepsilon(\bar{\boldsymbol{x}})$ nicht mehr verlassen. [Dies erlaubt ausdrücklich zunächst ein Verlassen der Umgebung $U_\delta(\bar{\boldsymbol{x}})$.]

$$\forall\, \varepsilon > 0\ \exists\, \delta > 0\ \forall\, t \|\boldsymbol{x}(0) - \bar{\boldsymbol{x}}\| < \delta \quad \Rightarrow \quad \|\boldsymbol{x}(t) - \bar{\boldsymbol{x}}\| < \varepsilon$$

„Eine kleine Störung (um δ) bleibt klein (um ε).“

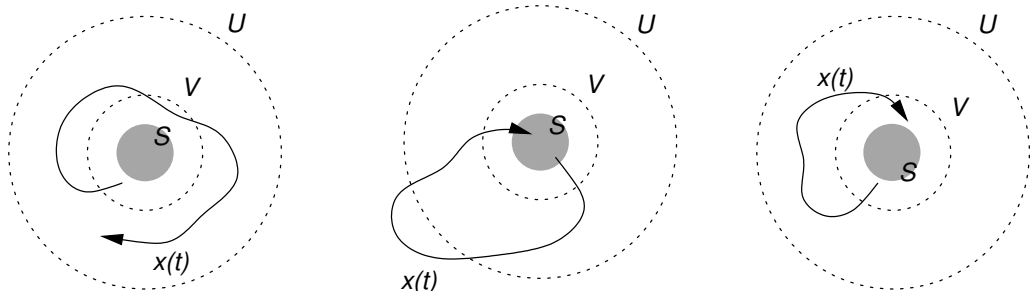
(ii) $\bar{\boldsymbol{x}}$ heißt anziehend, falls alle Trajektorien zu $\bar{\boldsymbol{x}}$ konvergieren, falls sie nur nahe genug am Fixpunkt starten:

$$\exists\, \delta > 0\ \|\boldsymbol{x}(0) - \bar{\boldsymbol{x}}\| < \delta \quad \Rightarrow \quad \lim_{t\to\infty} \boldsymbol{x}(t) = \bar{\boldsymbol{x}}$$

(iii) Der Fixpunkt ist asymptotisch stabil, falls (i) und (ii) gleichzeitig gelten.

(iv) Der Fixpunkt heißt global asymptotisch stabil (GAS), falls (i) und (ii) global für alle Trajektorien gelten, d.h. unabhängig von δ sind.

(v) Ein Dynamik heißt global konvergent, falls jede Trajektorie gegen einen (nicht notwendigerweise identischen) Fixpunkt konvergiert. Die Menge der Fixpunkte kann dabei eine diskrete Menge von isolierten Punkten oder eine ganze Mannigfaltigkeit sein.



(a) stabil & nicht anziehend (b) instabil & anziehend (c) asymptotisch stabil

Abbildung 14: Stabilität eines Fixpunktes S.

7.1.1 Lokale Stabilität

Die lokale Stabilität kann mit Hilfe einer Linearisierung der Systemdynamik in der Nähe des Fixpunktes bestimmt werden. Dazu machen wir eine Taylor-Entwicklung von $F(x)$ in der Nähe des Fixpunktes $\bar{x}$.

$$F(\boldsymbol{x}) = F(\bar{\boldsymbol{x}} + \Delta\boldsymbol{x}) = F(\bar{\boldsymbol{x}}) + \frac{\partial F}{\partial \boldsymbol{x}}|_{\bar{\boldsymbol{x}}=\boldsymbol{x}}\Delta\boldsymbol{x} + O(\Delta\boldsymbol{x}^2) \approx \bar{\boldsymbol{x}} + J(\bar{\boldsymbol{x}}) \cdot \Delta\boldsymbol{x}$$

mit $J(\bar{\boldsymbol{x}}) = \left(\frac{\partial F_i}{\partial x_j}\right)_{ij}$ – Jacobi-Matrix von F an der Stelle $\bar{\boldsymbol{x}}$.
Mittels Variablentransformation $z = \Delta\boldsymbol{x} = \boldsymbol{x} - \bar{\boldsymbol{x}}$ gilt:

$$\begin{aligned} \dot{z} = \dot{\boldsymbol{x}} = -\boldsymbol{x} + F(\boldsymbol{x}) &\overset{\text{Taylor}}{\approx} -(z + \bar{\boldsymbol{x}}) + \bar{\boldsymbol{x}} + J(\bar{\boldsymbol{x}}) \cdot z \\ &\approx -z + J(\bar{\boldsymbol{x}}) \cdot z = (-\mathbf{1} + J(\bar{\boldsymbol{x}})) \cdot z \end{aligned}$$

$$\text{bzw.} \qquad z(t+1) = \boldsymbol{x}(t+1) - \bar{\boldsymbol{x}} = F(\boldsymbol{x}(t)) - \bar{\boldsymbol{x}} \overset{\text{Taylor}}{\approx} \bar{\boldsymbol{x}} - \bar{\boldsymbol{x}} + J(\bar{\boldsymbol{x}}) \cdot z$$

Wir erhalten im zeit-kontinuierlichen und zeit-diskreten Fall jeweils ein lineares System:

$$\begin{array}{lll} \dot{z} = A \cdot z & z(t) = e^{A \cdot t} z_0 & A = -\mathbf{1} + J(\bar{\boldsymbol{x}}) \\ z(t+1) = A \cdot z(t) & z(t) = A^t \cdot z_0 & A = J(\bar{\boldsymbol{x}}) \end{array}$$

Um diese Lösungen besser interpretieren zu können, diagonalisieren wir die Matrix A:

$$\begin{aligned} &A = U\Lambda U^{-1} \\ &U - \text{Matrix von Eigenvektoren} \quad U = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n] \\ &\Lambda - \text{Diagonalmatrix der zug. Eigenwerte} \quad \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n) \end{aligned}$$

Damit folgt aufgrund der Reihenentwicklung $\mathrm{e}^{At} = \sum \frac{1}{n!} A^n t^n$ und wegen $A^n = U\Lambda^n U^{-1}$:

$$\begin{array}{ll} \mathrm{e}^{At} = U\mathrm{e}^{\Lambda t}U^{-1} & \mathrm{e}^{\Lambda t} = \text{diag}(\mathrm{e}^{\lambda_1 t}, \ldots, \mathrm{e}^{\lambda_n t}) \\ A^t = U\Lambda^t U^{-1} & \Lambda^t = \text{diag}(\lambda_1^t, \ldots, \lambda_n^t) \end{array}$$

Um uns die zeitliche Entwicklung einer Trajektorie anzuschauen, die bei $\boldsymbol{z}_0$ startet, entwickeln wir $\boldsymbol{z}_0$ nach der Eigenvektor-Basis U:

$$\boldsymbol{z}^0 = \sum_{i=0}^{n} a_i^0 \cdot \boldsymbol{v}_i \qquad \text{bzw. in Vektornotation:} \qquad \boldsymbol{z}_0 = U\boldsymbol{a}_0$$

und erhalten für die kontinuierliche bzw. diskrete Dynamik:

$$\boldsymbol{z}(t) = U\,\mathrm{e}^{\Lambda t}\,U^{-1}\,U\boldsymbol{a}_0 = U\mathrm{e}^{\Lambda t}\boldsymbol{a}_0 = U\boldsymbol{a}(t)$$
$$\boldsymbol{z}(t) = U\,\Lambda^t\,U^{-1}\,U\boldsymbol{a}_0 = U\Lambda^t\boldsymbol{a}_0 = U\boldsymbol{a}(t)$$

Die zeitliche Entwicklung der Anteile $a_i(t)$ von $\boldsymbol{z}(t)$ am Eigenvektor $\boldsymbol{v}_i$ hängt nur vom zugehörigen Eigenwert λ_i ab. Wir verdeutlichen uns dies nochmal anhand der diskreten Dynamik:

$$\boldsymbol{z}(t) = \sum_{\substack{i=0\\ \mathrm{Im}\,\lambda_i=0}}^{n} \lambda_i^t a_i^0 \boldsymbol{v}_i \quad + \sum_{\substack{i=0\\ \mathrm{Im}\,\lambda_i>0}}^{n} \lambda_i^t a_i^0 \boldsymbol{v}_i + \bar{\lambda}_i^t \bar{a}_i^0 \bar{\boldsymbol{v}}_i$$

Dabei haben wir berücksichtigt, dass komplexe Eigenwerte als konjugiert-komplexe *Paare* auftreten. Wir führen für alle beteiligten komplexen Zahlen die exponentielle Repräsentation ein:

$$\lambda_i = |\lambda_i|\mathrm{e}^{\mathrm{i}\omega_i} \qquad a_i = |a_i|\mathrm{e}^{\mathrm{i}\phi_i} \qquad v_{ji} = |v_{ji}|\mathrm{e}^{\mathrm{i}\theta_{ji}}$$

Dabei ist v_{ji} die j-te Komponente des i-ten Eigenvektors. Wir erhalten:

$$z_j(t) = \sum_{\substack{i=0\\ \mathrm{Im}\,\lambda_i=0}}^{n} |\lambda_i|^t a_i^0 \boldsymbol{v}_i \quad + \sum_{\substack{i=0\\ \mathrm{Im}\,\lambda_i>0}}^{n} 2|\lambda_i|^t |a_i^0||v_{ji}|\,\cos(\omega_i \cdot t + \phi_i + \theta_{ji})$$

Imaginäre Eigenwerte der Jacobi-Matrix führen demnach zu Oszillationen. Der Fixpunkt $\bar{\boldsymbol{x}}$ ist anziehend, falls der Betrag aller Eigenwerte kleiner als Eins ist, da dann gilt $|\lambda_i|^t \to 0$. Für den zeit-kontinuierlichen Fall zeigt eine ähnliche Rechnung, dass der Fixpunkt anziehend ist, falls $\mathrm{Re}(\lambda_i) < 0$. Genauer gilt:

$$\mathrm{e}^{\lambda_i t} + \mathrm{e}^{-\lambda_i t} = \mathrm{e}^{\mathrm{Re}\,\lambda_i t} \cdot \mathrm{e}^{\mathrm{i}\,\mathrm{Im}\,\lambda_i t} + \mathrm{e}^{\mathrm{Re}\,\lambda_i t} \cdot \mathrm{e}^{-\mathrm{i}\,\mathrm{Im}\,\lambda_i t} = \mathrm{e}^{\mathrm{Re}\,\lambda_i t} \cdot 2\,\cos(\mathrm{Im}\,\lambda_i\, t)$$

Die *lokale* Dynamik in der Nähe eines Fixpunktes kann qualitativ durch reine Betrachtung der Eigenwerte bestimmt werden. Die folgende Übersicht fasst die unterschiedlichen Dynamiken und ihre zugehörigen Eigenwerte zusammen.

Grobman-Hartman-Theorem: Falls alle Eigenwerte der Jacobi-Matrix eines zeit-diskreten Systems vom Betrage entweder echt kleiner oder echt größer Eins sind, d.h. kein Eigenwert auf dem Einheitskreis liegt, dann genügt bereits die Linearisierung zur Bestimmung der Stabilität des Fixpunkts. Analog müssen die Eigenwerte eines zeit-kontinuierlichen Systems in der negativen oder positiven Halbebene liegen, nicht jedoch auf der imaginären Achse.

Auf dieser Grenze zwischen Stabilität und Instabilität kann man nicht von der Stabilität des linearisierten Systems auf die des nicht-linearen Systems schließen. Außerdem ändert sich beim Überschreiten der Grenze offenbar das qualitative Verhalten der Dynamik. Dies nennt man *Bifurkation*. Abhängig vom Ort, wo ein Eigenwert λ_i der Jacobi-Matrix $J(\bar{\boldsymbol{x}})$ diese Grenze überschreitet, unterscheidet man folgende Fixpunkt-Bifurkationen:

1. **saddle-node:** $\lambda_i^{diskret} = 1 \quad$ bzw. $\quad \lambda_i^{kont} = 0$
   Entstehung zweier neuer Fixpunkte „aus dem Nichts“
2. **Periodenverdoppelung:** $\lambda_i^{diskret} = -1$
3. **Neimark-Sacker:** $\lambda_i^{diskret} = 1 \cdot \mathrm{e}^{\mathrm{i}\omega_i} \quad$ bzw. **Hopf:** $\lambda_i^{kont} = 0 + \mathrm{i}\omega_i$
   Entstehung einer (zunächst harmonischen) Oszillation um den Fixpunkt

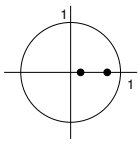
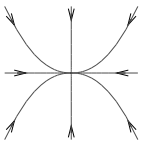
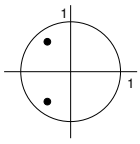
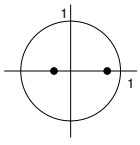
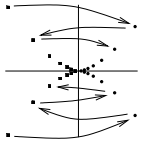
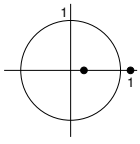
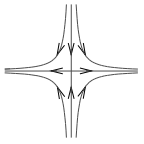
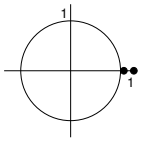
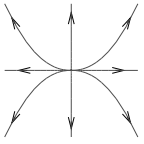
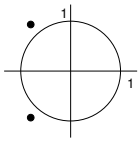
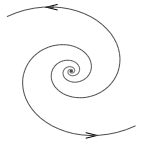
stability	eigenvalues	phase portrait		J	$\ln J$
stable			node	$\begin{pmatrix}\lambda_1 & \\ & \lambda_2\end{pmatrix}$ $0<\lambda_{1,2}<1$	$\begin{pmatrix}\ln\lambda_1 & \\ & \ln\lambda_2\end{pmatrix}$ $\ln\lambda_{1,2}<0$
			focus	$\lambda\begin{pmatrix}\cos\omega & \sin\omega \\ -\sin\omega & \cos\omega\end{pmatrix}$ $0<\lambda<1, \omega\in[0,\pi]$	$\begin{pmatrix}\ln\lambda & \omega \\ -\omega & \ln\lambda\end{pmatrix}$ $\ln\lambda\pm\mathrm{i}\omega,\ \ln\lambda<0$
			flip node	$\begin{pmatrix}-\lambda_1 & \\ & \lambda_2\end{pmatrix}$ $0<\lambda_{1,2}<1$	
unstable			saddle	$\begin{pmatrix}\lambda_1 & \\ & \lambda_2\end{pmatrix}$ $0<\lambda_1<1<\lambda_2$	$\begin{pmatrix}\ln\lambda_1 & \\ & \ln\lambda_2\end{pmatrix}$ $\ln\lambda_1<0<\ln\lambda_2$
unstable			node	$\begin{pmatrix}\lambda_1 & \\ & \lambda_2\end{pmatrix}$ $\lambda_{1,2}>1$	$\begin{pmatrix}\ln\lambda_1 & \\ & \ln\lambda_2\end{pmatrix}$ $\ln\lambda_{1,2}>0$
			focus	$\lambda\begin{pmatrix}\cos\omega & \sin\omega \\ -\sin\omega & \cos\omega\end{pmatrix}$ $\lambda>1, \omega\in[0,\pi]$	$\begin{pmatrix}\ln\lambda & \omega \\ -\omega & \ln\lambda\end{pmatrix}$ $\ln\lambda\pm\mathrm{i}\omega,\ \ln\lambda>0$

Tabelle 1: Unterschiedliche Dynamiken zeit-diskreter und zeit-kontinuierlicher Systeme in Abhängigkeit von den Eigenwerten $|\lambda_i|e^{i\omega_i}$ (zeit-diskret) bzw. $\ln\lambda_i+i\omega_i$ (zeit-kontinuierlich).

7.1.2 Globale Stabilität

Ein allgemeineres Hilfsmittel, die Stabilität von Fixpunkten nachzuweisen, ist die Lyapunov-Funktion (auch anwendbar auf der Stabilitätsgrenze des linearisierten Systems).

Lyapunov-Funktion V: Eine skalare Funktion $V(\boldsymbol{x}(t))$ heißt Lyapunov-Funktion innerhalb einer Umgebung $U(\bar{\boldsymbol{x}}) \in X$ falls

(i) $\dot{V}(\boldsymbol{x}(t)) = \frac{d}{dt}V(\boldsymbol{x}(t))$ existiert und ist stetig entlang der Trajektorien $\boldsymbol{x}(t) \in U$,

(ii) $V(\boldsymbol{x}(t))$ ist positiv definit auf U: $V(\boldsymbol{x}(t)) > 0$ für alle $\boldsymbol{x}(t) \neq \bar{\boldsymbol{x}}$ und $V(\bar{\boldsymbol{x}}) = 0$

(iii) $\dot{V}(\boldsymbol{x}(t)) \leq 0$ entlang jeder Trajektorie $\boldsymbol{x}(t) \in U$.

Eine analoge Definition gilt für zeit-diskrete Systeme. Dabei wird die Zeitableitung $\dot{V}$ durch eine diskrete Differenz $V(\boldsymbol{x}(t+1) - \boldsymbol{x}(t))$ ersetzt.

Satz: Ein Fixpunkt $\bar{\boldsymbol{x}}$ ist stabil, falls eine Lyapunov-Funktion in einer Umgebung $U(\bar{\boldsymbol{x}})$ existiert. Für $\dot{V} < 0$ folgt sogar asymptotische Stabilität.

Die Lyapunov-Funktion fungiert als verallgemeinerte Energie-Funktion, die entlang von Trajektorien stetig abnimmt und schliesslich in ein (lokales) Minimum führt.

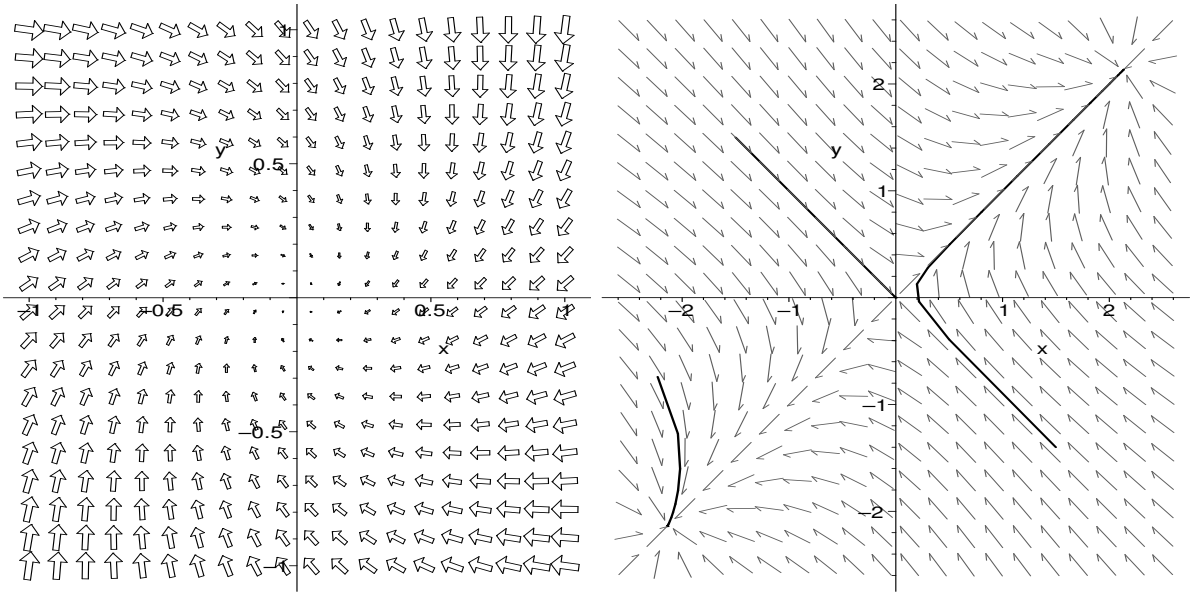


Abbildung 15: Vektorfeld einer zwei-dimensionalen Hopfield-Dynamik mit einem anziehenden Fixpunkt (links) bzw. zwei anziehenden Fixpunkten (rechts). Im rechten Phasendiagramm sind drei examplarische Trajektorien gezeigt.

Spezialfall Gradientensysteme: Die bislang betrachteten Optimierungsverfahren mittels Gradientenabstieg verwenden den Gradienten einer Energie-Funktion $V = E$ als Systemdynamik und haben damit automatisch eine Lyapunov-Funktion:

$$\dot{\boldsymbol{x}}(t) = -\eta \nabla_{\boldsymbol{x}} E(\boldsymbol{x}(t))$$
$$\dot{E}(\boldsymbol{x}(t)) = \nabla_{\boldsymbol{x}} E \cdot \dot{\boldsymbol{x}}(t) = -\eta \big(\nabla_{\boldsymbol{x}} E\big)^2 < 0$$

Beispiel: Cohen-Grossberg Energie-Funktion für Hopfield-Netzwerke Das klassische Beispiel eines Netzwerkes mit global konvergenter Dynamik ist das Hopfield-Modell. Aufgrund seiner symmetrischen Gewichtsmatrix ($w_{ij} = w_{ji}$) besitzt es eine Lyapunov-Funktion. Die Hopfield-Dynamik

$$\dot{x}_i(t) = -x_i + \sum w_{ij}\sigma(x_j) + I_i, \quad w_{ij} = w_{ji}, w_{ii} = 0 \; \forall \, i, j \tag{7.1}$$

hat bei Nutzung der Notation $\boldsymbol{y} = \sigma(\boldsymbol{x})$ folgende Lyapunov-Funktion:

$$E(\boldsymbol{x}) = -\frac{1}{2} \sum_{i,j=1}^{N} w_{ij} y_i y_j + \sum_{i=1}^{N} \int_0^{v_i} \sigma^{-1}(s) ds - \sum_{i=1}^{N} y_i I_i \tag{7.2}$$

Beweis: Wir bestimmen zunächst den Gradienten $\nabla_{\boldsymbol{y}} E$:

$$\frac{\partial}{\partial y_k} \Big(\sum_i y_i \sum_j w_{ij} y_j \Big) = \sum_j w_{kj} y_j + \sum_i y_i w_{ik} = \sum_j y_j \overbrace{(w_{kj} + w_{jk})}^{2w_{kj}}$$
$$\Rightarrow \quad \frac{\partial E}{\partial y_k} = -\sum_j w_{kj} \cdot y_j + \sigma^{-1}(y_k) - I_k$$

Damit können wir nun $\dot{E}$ bestimmen:

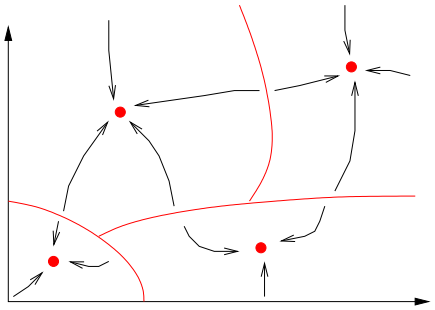
$$\begin{aligned}\dot{E}(\boldsymbol{y}(t)) = \nabla_{\boldsymbol{y}} E \cdot \dot{\boldsymbol{y}} = \sum_i \frac{\partial E}{\partial y_i} \cdot \dot{y}_i &= \sum_i \Big(-\sum_j w_{ij}\, y_j + \sigma^{-1}(y_i) - I_i\Big) \cdot \dot{y}_i \\ &= -\sum_i \underbrace{\Big(-x_i + \sum_j w_{ij}\, y_j + I_i\Big)}_{\dot{x}_i} \cdot \sigma'(x_i) \cdot \dot{x}_i \\ &= -\sum_i \underbrace{\sigma'(x_i)}_{\geq 0} \cdot \underbrace{\dot{x}_i^2}_{\geq 0} \leq 0\end{aligned}$$

Für eine monoton wachsende Aktivierung σ, fällt E(t) damit monoton entlang einer Trajektorie. Damit konvergiert die Hopfield-Dynamik immer zu einem Fixpunkt.
Die Energiefunktion kann im Allgemeinen sehr komplex sein und die Hauptaufgabe für viele Anwendungen besteht darin, die Gewichtsmatrix so zu wählen, dass die lokalen Minima gewünschten Endzuständen entsprechen, die dann als abgerufenes Muster interpretiert werden können.

7.2 Hopfield-Modell als Assoziativ-Speicher

Offenbar ist das Hopfield-Netz aufgrund seiner symmetrischen Gewichte global konvergent, d.h. jede Trajektorie konvergiert zu einem Fixpunkt. Wenn es gelänge, dem Netz gezielt gewünschte Fixpunkte aufzuprägen, fungiert die rekurrente Systemdynamik als „Abruf“ dieser eingespeicherten Muster.

- Interpretiere asymptotisch stabile Fixpunkte des Hopfield-Netzes als gespeicherte Muster.
- Diese sind auch bei unvollständiger oder verrauschter Eingabe korrekt abrufbar, denn die Attraktorbecken der Fixpunkte sind typischerweise zusammenhängend.



Diskretes Hopfield-Modell:

$$\boldsymbol{x}(t+1) = \operatorname{sgn}(W(\boldsymbol{x}(t))) \qquad \text{mit } \boldsymbol{x} \in \{-1, 1\}^n \qquad \text{(Binärmuster)}$$

Update:

- synchron (alle Neuronen gleichzeitig, erforder globalen Zeittakt)
- asynchron (Neuronen zufällig nacheinander, biologisch plausibler)

Wie wählt man die Gewichtsmatrix, um binäre Muster $p^1, \cdots, p^m \in \mathbb{R}^n$ zu speichern?
Einfachster Ansatz: **Hebb-Regel** $\Delta w_{ij} = {p_i}^\alpha \cdot {p_j}^\alpha$:

$$\Rightarrow W = \frac{1}{n} \cdot \sum_{\alpha=1}^{m} p_\alpha^T \cdot p_\alpha$$

$$w_{ij} = \frac{1}{n} \cdot \sum_{\alpha=1}^{m} p_i^\alpha \cdot p_j^\alpha$$

n - Anzahl der Neuronen bzw. Dimension der Muster
m - Anzahl der Muster

Um zu klären, ob die Muster p^α mit dieser Regel tatsächlich gespeichert werden, untersuchen wir, wie stabil der Abruf eines Musters p^β ist, d.h. wieviele Bits sich bei Präsentation dieses Musters ändern würden.

Ziel: $\text{sgn}(W \cdot p^\beta) \stackrel{!}{=} p^\beta$

$$
\begin{aligned}
(W \cdot p^\beta)_i = \overbrace{h_i{}^\beta}^{syn.Sum.} &= \sum_j w_{ij} \cdot p_j{}^\beta = \frac{1}{n} \sum_j \sum_\alpha p_i^\alpha \cdot p_j^\alpha \cdot p_j^\beta \\
&= \frac{1}{n} \sum_{\alpha=\beta} \sum_j p_i^\alpha \cdot \underbrace{p_j^\alpha \cdot p_j^\beta}_{=1} \quad + \quad \frac{1}{n} \sum_{\alpha \neq \beta} \sum_j p_i^\alpha \cdot p_j^\alpha \cdot p_j^\beta \quad \binom{\text{Ausklammern}}{\text{des Terms } \alpha = \beta} \\
&= \underbrace{p_i{}^\beta}_{\text{gewünschtes Musterbit}} + \underbrace{\frac{1}{n} \sum_{\alpha \neq \beta} \sum_j p_i^\alpha \cdot p_j^\alpha \cdot p_j^\beta}_{\text{Cross-Talk-Term } C_i^\beta}
\end{aligned}
$$

Der Cross-Talk-Term beschreibt den Einfluß der anderen gespeicherten Muster ($\alpha \neq \beta$) auf den Abruf des Musters β. Falls der Cross-Talk-Term klein ist, genauer:

$$-p_i{}^\beta \cdot C_i{}^\beta < 1$$

bleibt das Bit $p_i{}^\beta$ aufgrund der sgn-Funktion erhalten. Ansonsten flippt es offenbar. Für orthogonale Muster verschwindet der Cross-Talk-Term, denn:

$$\sum_j p_j{}^\alpha \cdot p_j{}^\beta = 0 \qquad \forall\, \alpha \neq \beta$$

Die Hebb-Regel führt zu vielen weiteren gespeicherten Mustern (spurious states)

- Inverse Muster: $W(-p^\beta) = -W \cdot p^\beta = -p^\beta - C^\beta$
- Überlagerung bestimmter Muster (ungerader Anzahl):
  $W(p^\beta + p^\gamma + p^\delta) = p^\beta + p^\gamma + p^\delta + \underbrace{C^\beta + C^\gamma + C^\delta}_{C^{\beta\gamma\delta}}$

Bermerkungen:

- Attraktionsbecken der gemischten Muster sind kleiner als die der Originale, d.h. deren Abruf ist unwahrscheinlicher.
- Mittels statistischer Physik kann man zeigen, dass bei $\alpha = \frac{m}{n} = 0,138$ die Muster instabil werden. $P_{error} = 0,0036$
- α heißt Ladefaktor (load parameter)
- Fast exakter Abruf nur möglich, falls $m \leq \frac{n}{2 \log n}$
- Weitere Details in [**?**]

Bestimmung der Wahrscheinlichkeit, dass ein Bit flipt

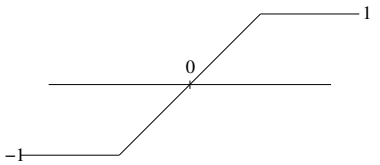
- Nehme an, $p_j^\alpha \in \{-1, 1\}$ seien statistisch unabhängig.
- Cross-talk-Term wird zu Binomialverteilung
  mit Mittelwert $\mu = 0$ und Varianz $\sigma^2 = M/N = \alpha$.
- Im Limes großer Netzwerke ($n \to \infty$) wirkt der Zentrale Grenzwertsatz.
- Die Binomialverteilung kann durch eine Gaussverteilung $\mathcal{N}(\mu, \sigma)$ ersetzt werden.
- Bestimme Wahrscheinlichkeit

$$P_{error} = P(-p_i^\beta \cdot C_i^\beta > 1) = \int_1^\infty \mathcal{N}(\mu, \sigma)(x)dx$$

7.3 Brain-State-in-a-Box (BSB)

Das bekannte BSB-Modell ist ein andere Variante des diskreten Hopfield-Netzes. Dabei wird die sgn-Funktion durch eine stückweise lineare Aktivierung ersetzt:

$$\sigma(s) = \begin{cases} 1 & s > 1 \\ s & -1 \leq s \leq 1 \\ -1 & s < -1 \end{cases}$$



Die Netzwerk-Aktivitäten $x_i(t)$ können sich damit auch frei im Inneren des Hyperwürfels $\{-1, 1\}^n$ bewegen und sind nicht mehr auf die Ecken beschränkt.

7.4 Competitive Layer Model (CLM)

Das Competetive Layer Model (CLM) ist ein komplexes rekurrentes Netzwerk, um Binding-Probleme aller Art zu lösen. Unter Binding versteht man dabei allgemein die Zuordnung von ähnlichen Feature-Vektoren zueinander oder zu bedeutungstragenden Klassenlabeln. Die Gruppierung ähnlicher Feature-Vektoren wie sie bei Segmentierungsaufgaben anfällt ist ein konkretes Beispiel.

- Clustering
- Bild-, Textursegmentierung
- perzeptuelle Gruppierung
- Bewegungsklassifikation

Die Idee des CLM basiert darauf, dass sich kompatible, also zueinander gehörige Merkmale anziehen und umgekehrt nicht-kompatible Merkmale sich abstoßen. Durch die Wechselwirkung zwischen Anziehung und Abstoßung bilden sich Cluster zusammenghöriger Merkmale aus, die jeweils verschiedenen Neuronenschichten zugeordnet werden. Anhand dieser Zuordnung zu verschiedenen Schichten kann man die Gruppierung dann ablesen.
Das CLM besteht aus L Schichten von topographisch angeordneten Neuronen. Jedem Feature-Vektor $\mathbf{m}_r$ ist eine Spalte von L Neuronen $x_{r\alpha}$ quer durch alle Schichten zugeordnet, wobei r einfach den Ort oder die Nummer des Merkmals indiziert.
Die Merkmale $\mathbf{m}_r$ gehen nicht direkt als Eingabe in das CLM ein, sondern bestimmen die Gewichtsmatrix des CLM in Form der symmetrischen Wechselwirkung $f_{rr'}$ zwischen zwei Neuronen r und r'. Diese Wechselwirkung definiert letztlich die Kompatibilität zwischen zwei Merkmalen und bestimmt damit hauptsächlich das Gruppierungsergebnis. In jeder Neuronenschicht α kommt dabei dieselbe Wechselwirkungsmatrix $f_{rr'}$ zur Anwendung.

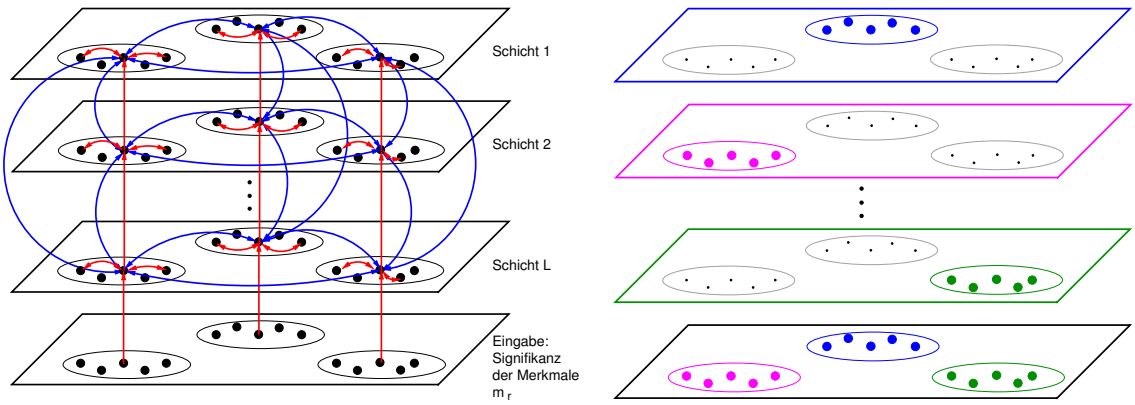


Abbildung 16: Schichtweiser Aufbau des CLM: Die Eingabeschicht versorgt jedes Neuron in einer Spalte mit der konstanten Eingabe h_r, die als Signifikanz des Merkmals $\mathbf{m}_r$ interpretiert werden kann. Innerhalb jeder Schicht wird über die Kompatibilität der Merkmale eine laterale Wechselwirkung definiert, die entweder anziehend (rot) oder abstoßend (blau) sein kann. Zwischen den Schichten besteht eine Konkurrenz, die mittels negativer Gewichte (blau) realisiert wird und WTA-Verhalten produziert. Jeder Gruppe wird durch die Dynamik genau eine Schicht zugeordnet (rechts).

Zusätzlich zu dieser lateralen Wechselwirkung gibt es interlaterale (vertikale) Konkurrenz zwischen den Schichten, die dafür sorgt, dass (im Gleichgewicht) pro Spalte genau nur eine Schicht aktiv ist. Anhand dieser Zuordnung von Merkmalen (einer Spalte r) zu einer Schicht α kann man schließlich das Gruppierungsergebnis ablesen. Insgesamt ergibt sich folgende Dynamik:

$$\dot{x}_{r\alpha} = -x_{r\alpha} + \sigma\Big(\underbrace{Jh_r}_{\text{Eingabe}} - \underbrace{J\sum_{\beta} x_{r\beta}}_{\text{WTA-Konkurrenz}} + \underbrace{\sum_{r'} f_{rr'} x_{r'\alpha}}_{\text{laterale WW}} \Big) \tag{7.3}$$

Als Aktivierungsfunktion σ kommt dabei die lineare Threshold-Funktion $\sigma(x) = \max(0, x)$ zum Einsatz, die besonders einfach zu berechnen ist (ein einfacher Vergleich). Die CLM-Dynamik hat folgende Lyapunov-Funktion:

$$E = -\sum_{r\alpha} Jh_r x_{r\alpha} + \frac{1}{2}\sum_{r\alpha\beta} Jx_{r\alpha}x_{r\beta} - \frac{1}{2}\sum_{\alpha rr'} f_{rr'}x_{r\alpha}x_{r'\alpha} \tag{7.4}$$

Damit konvergiert das CLM immer zu einem stabilen Gleichgewicht.

- Gruppierung: $\alpha(r) = \arg\max_\beta \sum_{r'} f_{rr'}x_{r'\beta} \approx \arg\max_\beta x_{r\beta}$
- Aktivität des Winner-Neurons = Eingabe: $x_{r\alpha(r)} \approx h_r$

Wie lernt man die laterale Interaktion $f_{rr'}$?

- Definiere symmetrisches Ähnlichkeitsmaß $d_{rr'}(m_r, m'_r)$
- Vektor-Quantisierung im Ähnlichkeitsraum $d_{rr'}$:
  Berechne für jedes Merkmalspaar m_r – m'_r die Ähnlichkeit und bilde $d_{rr'}$ auf einen von 20 – 30 Prototypen ab. Diese Prototypen repräsentieren typisch auftretende Merkmalspaare.

- Zähle für jeden Prototyp die Anzahl positiver c^+ (zusammengehöriger) und negativer c^- Merkmalspaare in den Trainingsdaten.

- Interaktionsfunktion entsteht aus linearer Gewichtung beider Verteilungen:

$$f_{rr'} = c_p^+ - \Lambda c_p^- \qquad \text{mit } p = \arg\min ||d_{rr'} - d_p|| \text{ – Index des Sieger-Prototypen}$$

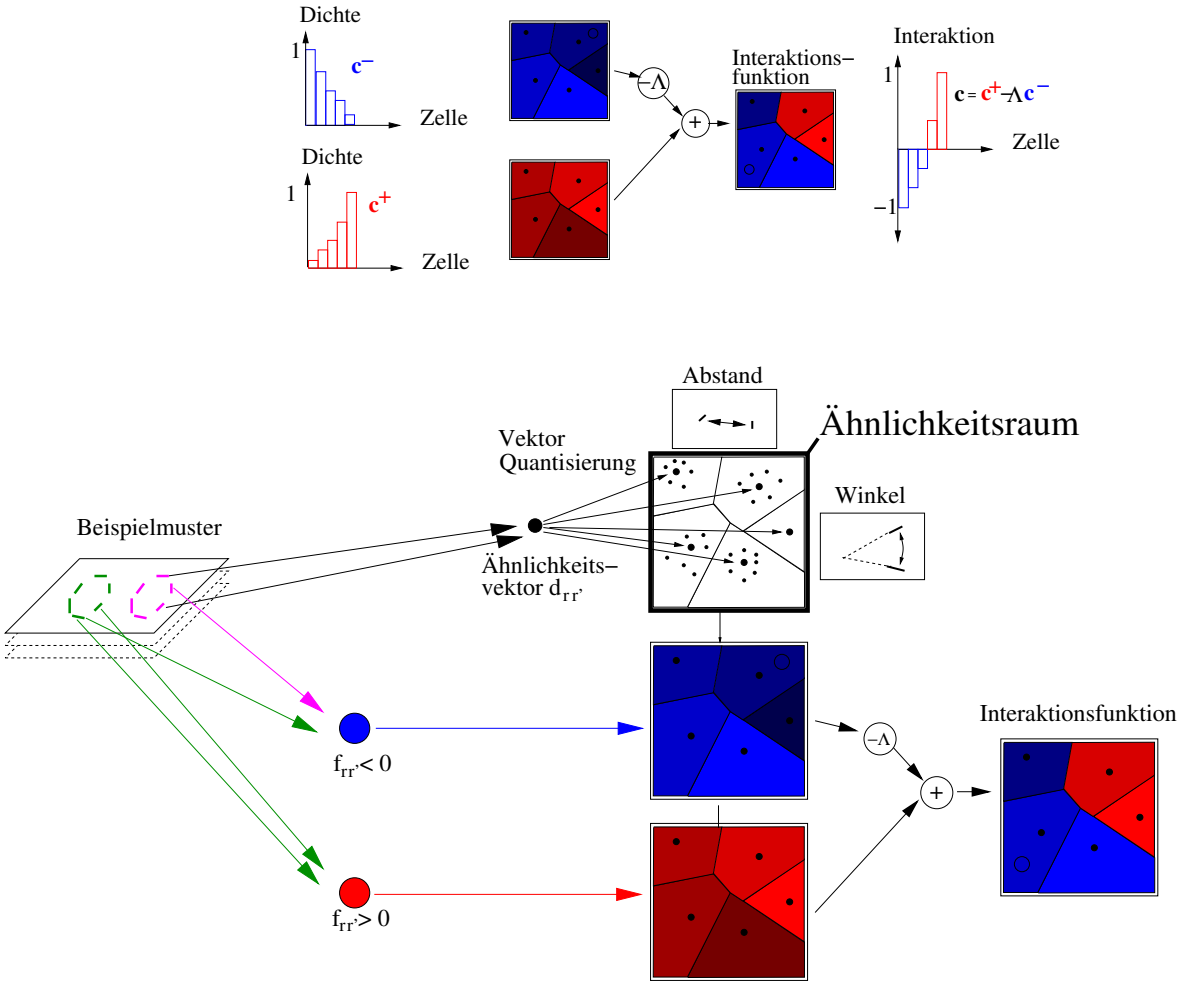


Abbildung 17: Zusammenfassung CLM-Lernen

8 Lernverfahren für Rekurrente Netze

8.1 Backpropagation Through Time

- Entfaltung des RNN in ein äquivalentes MLP + Standard-Backprop
- Batch-Algorithmus
- Komplexität $\mathcal{O}(T \cdot n^2)$ (n – Anzahl der Neuronen)

8.2 Real-Time Recurrent Learning

- Online-fähiges Verfahren
- Batch-Algorithmus
- Komplexität $\mathcal{O}(T \cdot n^4)$

Aufgabe: Lernen von Ein-Ausgabe-Trajektorien, d.h. die Beispieldaten stammen aus Trajektorienpaaren $\{\mathbf{u}(t)^\alpha, d(t)^\alpha\}_{t=0\ldots T-1}$
diskretes Netzwerkmodell:

$$\boldsymbol{y}(t+1) = \sigma\big(W_{rec}\,\boldsymbol{y}(t) + W_u \boldsymbol{u}(t)\big)$$

Fasse Netzwerkzustände $\boldsymbol{y}$ und Eingabe $\boldsymbol{u}$ in einem Vektor zusammen:

$$\boldsymbol{x} = [\boldsymbol{y}, \boldsymbol{u}] \qquad W = [W_{rec}, W_u]$$
$$\Rightarrow \boldsymbol{x}(t+1) = \sigma\big(W\,\boldsymbol{x}(t)\big)$$

Fehlerfunktion:

$$E(W) = \frac{1}{2}\sum_\alpha \sum_{t=1}^{T} \sum_{l\in O} \big(x_l(t) - d_l(t)\big)^2 \qquad E(W) = \frac{1}{2}\sum_\alpha \sum_{t=1}^{T} E^\alpha(W,t)$$

Training durch Gradientenabstieg:

Batch: $$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \sum_\alpha \sum_{t=0}^{T} \frac{\partial E^\alpha(W,t)}{\partial w_{ij}}$$

Online: $$\Delta w_{ij}(t) = -\eta \frac{\partial E^\alpha(W,t)}{\partial w_{ij}} = -\eta \sum_{l\in O} \underbrace{\big(x_l(t) - d_l^\alpha(t)\big)}_{\varepsilon_l^\alpha(t)} \cdot \frac{\partial x_l(t)}{\partial w_{ij}} = -\eta \sum_{l=0}^{n} \varepsilon_l^\alpha(t) \cdot \frac{\partial x_l(t)}{\partial w_{ij}}$$

Der Fehler $\varepsilon_l^\alpha(t)$ ist nur für die Ausgabe-Neuronen verschieden von Null:

$$\varepsilon_l^\alpha(t) = \begin{cases} x_l(t) - d_l^\alpha(t) & l \in O \qquad \text{Ausgabe-Neuron} \\ 0 & \text{sonst} \end{cases}$$

$p_{ij}^l(t+1) \equiv \dfrac{\partial x_l(t+1)}{\partial w_{ij}}$ – Empfindlichkeit des Neurons l auf Änderung des Gewichts w_{ij}

$$= \sigma'\big(h_l(t)\big) \cdot \overbrace{\Big[\delta_{il} \cdot x_j(t) + \sum_{k=0}^{n} w_{lk} \frac{\partial x_k(t)}{\partial w_{ij}}\Big]}^{\sum_k u'\,v + u\,v'}$$

Ausgehend von $p_{ij}^l(0)$ können wir alle $p_{ij}^l(t)$ für eine Trajektorie iterativ berechnen: (RTRL, Williams & Zipser, 1989)

$$\begin{aligned} p_{ij}^l(0) &= 0 \qquad x_l(0) \\ x_l(t+1) &= \begin{cases} u_l(t+1) & \text{falls } l \in \text{Eingabe} \\ \sigma\big(\sum_k w_{lk} x_k(t)\big) \end{cases} \\ p_{ij}^l(t+1) &= \sigma'\big(h_l(t)\big) \cdot \Big[\delta_{il} \cdot x_j(t) + \sum_{k=0}^{n} w_{lk} \cdot p_{ij}^k(t)\Big] \\ \Delta w_{ij}(t+1) &= -\eta \sum_{l=0}^{n} \varepsilon_l^\alpha(t+1) \cdot p_{ij}^l(t+1) \end{aligned}$$

8.3 Virtual Teacher Forcing

Atiya & Parlos, 1990:
Vereinigender Ansatz zur Herleitung verschiedener rekurrenter Lernverfahren.
Lernen als allg. Optimierungsproblem: Minimiere $E(W)$ unter Nebenbedingungen $g \equiv 0$.
Die Nebenbedingungen $\boldsymbol{g}$ achten dabei auf die Einhaltung der Dynamikgleichungen:

$$\boldsymbol{g}(t+1) \equiv -\boldsymbol{x}(t+1) + W\sigma(\boldsymbol{x}(t)) = 0 \qquad t = 0, \ldots, T-1$$

Fasse die Größen *aller Zeitschritte* in großen Vektoren zusammen:

$$\begin{aligned} \boldsymbol{x} &= \big(\boldsymbol{x}^t(1), \ldots, \boldsymbol{x}^t(T)\big)^t && \in \mathbb{R}^{nT} \\ \boldsymbol{g} &= \big(\boldsymbol{g}^t(1), \ldots, \boldsymbol{g}^t(T)\big)^t && \in \mathbb{R}^{nT} \\ \boldsymbol{w} &= \big(\mathbf{w}_1, \ldots, \mathbf{w}_n\big)^t && \in \mathbb{R}^{n^2} \qquad \mathbf{w}_i \text{ – Zeilenvektoren von } W \end{aligned}$$

Idee: Leite die Fehlerfunktion $E(\boldsymbol{w})$ nicht nach $\boldsymbol{w}$ sondern nach Zuständen $\boldsymbol{x}$ ab:

$$\Delta x(t) = -\eta \frac{\partial E}{\partial \boldsymbol{x}} = -\eta\varepsilon(t)$$

Dieser „Gradientenabstieg“ auf den Zuständen $\boldsymbol{x}$ bewirkt quasi „direkt“ eine Korrektur des Fehlers $\varepsilon(t)$. Die zugehörigen Gewichtsänderungen bestimmen wir mittels der Nebenbedinungen $\boldsymbol{g}$:

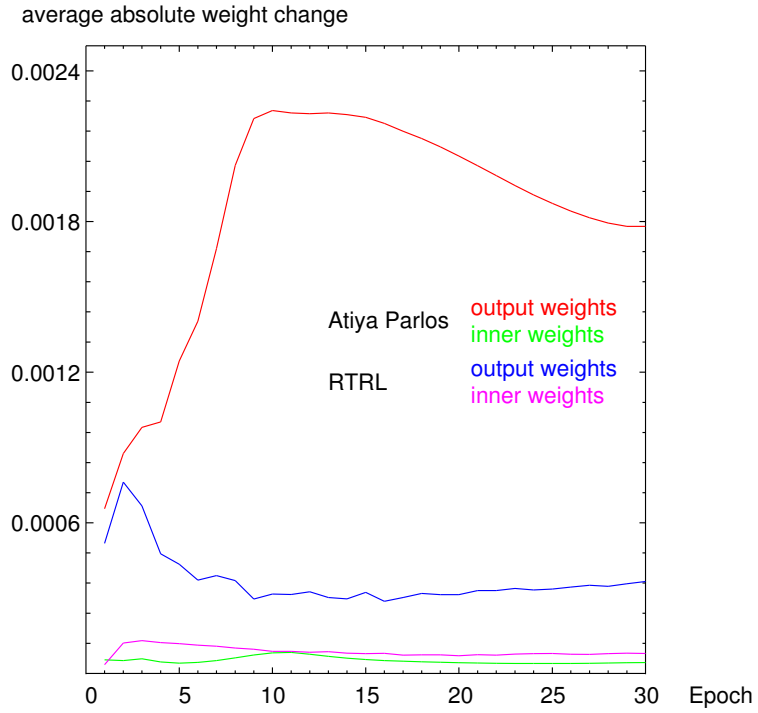
$$\boldsymbol{g} \equiv 0 \qquad \Leftrightarrow \qquad \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{w}}\Delta\boldsymbol{w} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} = 0 \qquad \Leftrightarrow \qquad \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{w}}\Delta\boldsymbol{w} = -\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\Delta\boldsymbol{x}$$

Diese Technik heißt *virtual teacher forcing*, weil die (vom Lehrer) angestrebten Zielzustände $\boldsymbol{x} + \Delta\boldsymbol{x}$ die eigentlichen Gewichtsänderungen bewirken aber niemals wirklich in das Netz eingespeist werden.
Dieses unterbestimmte LGS ($\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{w}} \in \mathbb{R}^{nT \times n^2}$) wird mittels der Pseudo-Inversen gelöst:

$$\Delta\boldsymbol{w} = -\left[\left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{w}}\right)^t \left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{w}}\right)\right]^{-1} \left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{w}}\right) \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \cdot \Delta\boldsymbol{x} \tag{8.1}$$

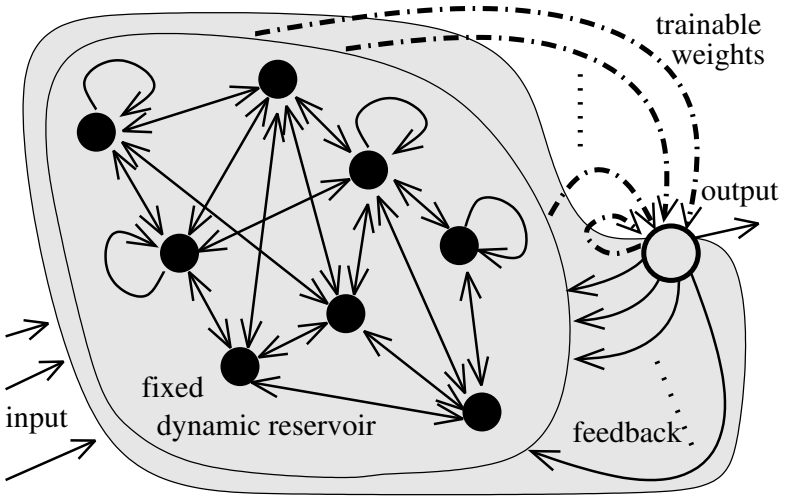
Bemerkungen zum Atiya-Parlos-Lernen

- entspricht nicht exaktem Gradienten $\frac{\partial E}{\partial \boldsymbol{w}}$
- anderer Pfad im Gewichtsraum als RTRL
- Matrizen faktorisieren aufgrund ihrer Block-Diagonalform
- Komplexität: $\mathcal{O}(T \cdot n^2)$
- Ausgabegewichte werden überproportional stark variiert



8.4 Reservoir Computing

Der überproportional hohe Einfluß der Ausgabegewichte, motiviert die Betrachtung eines sog. dynamischen Reservoirs. Die Hidden Neuronen des Reservoirs werden nicht adaptiert, sondern entwickeln – getrieben von der externen Eingabe $\boldsymbol{u}(t)$ – „lediglich“ eine komplexe Dynamik in einem neuen hoch-dimensionalen Merkmalsraum. Ähnlich wie auch schon bei den SVMs kombiniert das Ausgabeneuron diesen hoch-dimensionalen Zustandsvektor *linear* um seine Ausgabe zu produzieren.



- feste, zufällige Initialisierung der Gewichte des Reservoirs
- „gutes“ Reservoir erzeugt möglichst reichhaltige Dynamik
- Verteilung der Eigenwerte von $W_{reservoir}$ entlang der Bifurkationsgrenze zwischen stabilem und instabilem Verhalten
- sehr erfolgreich in Zeitserien-Vorhersage
- drei Ansätze
  - ◇ Echo State Networks [**?**, **?**]
    - ∗ keine Rückkopplung von Ausgabeneuron in Reservoir
    - ∗ kein Lernen im Reservoir
  - ◇ Liquid State Machines [**?**]
    ähnlich zu Echo State Networks, aber spikende Neuronen
  - ◇ Backpropagation-Decorrelation Learning (BPDC) [**?**, **?**]
    - ∗ Rückkopplung vom Ausgabeneuron ins Reservoir
    - ∗ effizientes Training (Anpassung) des Reservoirs
    - ∗ Komplexität: $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$

8.5 Backpropagation-Decorrelation Learning

We consider fully connected recurrent networks

$$\boldsymbol{x}(k+\Delta t) = (1-\Delta t)\boldsymbol{x}(k) + \Delta t \mathsf{W}\boldsymbol{f}(\boldsymbol{x}(k)) + \Delta t \mathsf{W}_u \boldsymbol{u}(k), \tag{8.2}$$

where $x_i, i=1,\ldots,N$ are the states, $\mathsf{W}\in\mathbb{R}^{N\times N}$ is the weight matrix, W_u the input weight matrix and $k=\hat{k}\Delta t, \hat{k}\in N_+$ is a discretized time variable[3]. For small Δt we obtain an approximation of the continuous time dynamics $d\boldsymbol{x}/dt = -\boldsymbol{x} + \mathsf{W}\boldsymbol{f}(\boldsymbol{x})$ and for $\Delta t = 1$ the standard discrete dynamics. We assume that $\boldsymbol{f}$ is a standard sigmoidal differentiable activation function with $\boldsymbol{f}' \leq 1$ and is applied component wise to the vector $\boldsymbol{x}$. We further assume that W is initialized with small random values in a certain weight initialization interval $[-a,a]$ (which can be adaptively rescaled as shown in Section 3.1). Denote by $O\subset\{1,..,N\}$ the set of indices s of N_O output neurons (i.e. x_s output $\Rightarrow s\in O$) and let for a single output neuron w.r. $O=\{1\}$ such that x_1 is the respective output of the structured network shown in Fig. 1.
If all but the output weights are fixed, we can regard the inner neurons as dynamical reservoir triggered by the input signal and providing a dynamical memory. The output layer linearly combines these states to read out the desired output. In [**?**], the *Backpropagation-Decorrelation* rule

$$\Delta w_{ij}^{BPDC}(k+1) = \frac{\eta}{\Delta t}\frac{f(x_j(k))}{\sum_s f(x_s(k))^2+\varepsilon}\gamma_i(k+1), \tag{8.3}$$

$$\text{where}\quad \gamma_i(k+1) = \sum_{s\in 0}\Big((1-\Delta t)\delta_{is} + \Delta t w_{is} f'(x_s(k))\Big)e_s(k) - e_i(k+1), \tag{8.4}$$

has been introduced, where η is the learning rate, ε a regularization constant ($\varepsilon = 0.002$ throughout), and $e_s(k)$ are the non-zero error components for $s\in O$ at time $k: e_s(k) = x_s(k) - y_s(k)$ with respect to the teaching signal $y_s(k)$.

[3]With a small abuse of notation we also interpret time arguments and indices $(k+1)$ as $(k+1)\Delta t$.

The BPDC rule in (8.5) is obtained from (8.1) by modification: (i) by restriction of learning in BPDC to the output weights motivated by the observation that in AP recurrent learning the hidden neurons update slowly and in a highly coupled way [**?**]; (ii) by omitting the momentum term introducing the old update $\Delta w_{ij}^{AP_{batch}}$ (second term in (8.1)); (iii) by replacing the full autocorrelation matrix C_k by the instantaneous autocorrelation matrix $C(k) = \varepsilon I + \boldsymbol{f}_k \boldsymbol{f}_k^T$ in the first term and use the small rank adjustment matrix inversion lemma

$$C(k)^{-1}\boldsymbol{f}_k = \left[\frac{1}{\epsilon}\mathbf{1} - \frac{[\epsilon^{-1}\boldsymbol{f}_k][\epsilon^{-1}\boldsymbol{f}_k]^T}{1+\epsilon^{-1}\boldsymbol{f}_k^T\boldsymbol{f}_k}\right]\boldsymbol{f}_k = \boldsymbol{f}_k\left[\frac{1}{\epsilon} - \frac{1}{\epsilon^2}\frac{\boldsymbol{f}_k^T\boldsymbol{f}_k}{1+\frac{1}{\epsilon}\|\boldsymbol{f}_k\|^2}\right] = \frac{\boldsymbol{f}_k}{\sum \boldsymbol{f}(x_s(k))^2 + \epsilon}$$

$$\Rightarrow \Delta w_{ij}^{BPDC}(k{+}1) = \frac{\eta}{\Delta t}[C(k)^{-1}\boldsymbol{f}_k]_j \gamma_i(k+1) \tag{8.5}$$

From this derivation the BPDC rule is interpreted as an effective mixture of reservoir computing, error minimization, and decorrelation mechanisms.

8.6 Stability and BPDC

The operator framework Using the standard notation for nonlinear feedback systems [**?**, **?**] [4] the network (8.2) is composed of a linear feedforward and a nonlinear feedback operator Φ:

$$\dot{\boldsymbol{x}} = -\boldsymbol{x} + \boldsymbol{e}, \qquad \boldsymbol{e} = \mathsf{W}_u\boldsymbol{u} + \mathsf{W}\boldsymbol{f}(\boldsymbol{y}), \quad \boldsymbol{y} = \boldsymbol{x}.$$

The Laplace transformation of the linear part yields the forward operator $\mathsf{G}_\mathrm{I}(s) = (\mathsf{C}{+}s\mathbf{1})^{-1}$ while the activation function $\boldsymbol{f}$ defines the feedback operator Φ, see Fig. 8.4. Φ does not explicitely have to be stated in the frequency domain because it will be approximated by its gain which is defined by the maximum slope of $\boldsymbol{f}$. Denote this network interpretation as $((\mathsf{G}_\mathrm{I}, \mathsf{W}), \Phi)$ for the input-output equation

$$\boldsymbol{y} = \mathsf{G}_\mathrm{I}(\mathsf{W}_u\boldsymbol{u} + \mathsf{W}\Phi(\boldsymbol{y})) = \mathsf{G}_\mathrm{I}\mathsf{W}_u\boldsymbol{u} + \mathsf{G}\Phi(\boldsymbol{y}), \quad (\mathsf{G} = \mathsf{G}_\mathrm{I}\mathsf{W}).$$

The network acts as nonlinear feedback system implementing a loop operator $\mathsf{H} \doteq ((\mathsf{G}_\mathrm{I}, \mathsf{W}), \Phi)$ which transforms L_2^n signals[5] $\mathsf{W}_u\boldsymbol{u}$ into L_2 output signals $\boldsymbol{y}$. Using the small gain theorem, this system is input-output stable (and the origin is globally exponentially stable for the respective unforced dynamics (8.2) with $\boldsymbol{u}\equiv 0$), if the operator gains $\gamma(\mathsf{G}_\mathrm{I}), \gamma(\mathsf{G}) = \gamma(\mathsf{G}_\mathrm{I}\mathsf{W})$ and $\gamma(\Phi)$ are finite and the

$$\text{small gain condition:} \qquad \gamma(\mathsf{G})\gamma(\Phi) < 1 \tag{8.6}$$

holds. The small gain condition yields the loop gain estimate for $\boldsymbol{u}' = \mathsf{W}_u\boldsymbol{u}$

$$\gamma(\mathsf{H}) \le \frac{\gamma(\mathsf{G}_\mathrm{I})}{1-\gamma(\mathsf{G})\gamma(\Phi)} \quad \text{where} \quad \gamma(\mathsf{H}) = \sup_{\boldsymbol{u}'}\frac{\|\mathsf{H}(\boldsymbol{u}')\|_2}{\|\boldsymbol{u}'\|_2} = \|\mathsf{H}\|_2 \tag{8.7}$$

is the gain induced by the $L_2^p, p{=}n, 1$ norms for the operator $\mathsf{H}\colon L_2^n \to L_2$. Note that $\gamma(\mathsf{G}_\mathrm{I}){=}\gamma(\Phi){=}1$ by definition. This implies also bounds on the output signal: $\|\mathbf{y}\|_2 \le \gamma(H)\|\boldsymbol{u}_1\|_2$. To derive the stability condition now decompose the network into reservoir and output neuron subsystems: let $\boldsymbol{x}^r = (x_2, ..., x_n)^T$ and W_r^r the sub matrix connecting only these

[4] We give the framework only for continuous time, an analog derivation is possible for discrete time using the z-transform and sequence spaces, see ([**?**],chap. 6).

[5] Strictly speaking this is true only after assuring stability of the system, see [**?**, **?**].

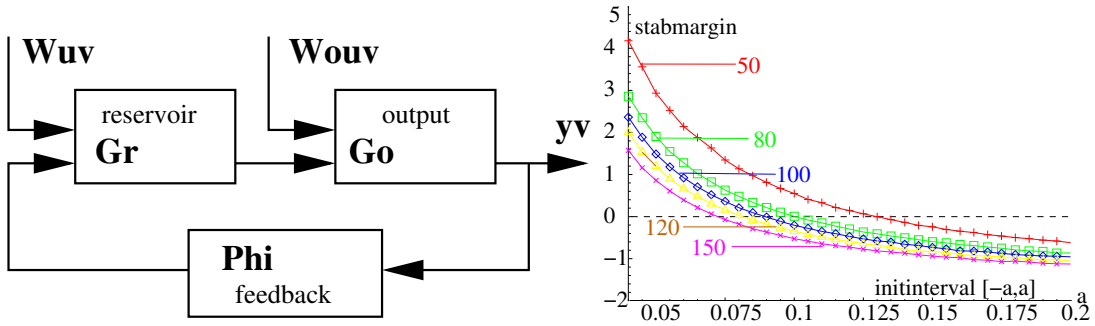


Abbildung 18: Left: System composed of inner reservoir subsystem, output subsystem and feedback.Right: Stability margin vs. weight initialization intervals different network sizes (averaged over 50 network initializations.)

inner neurons. Then $((\mathsf{G}_\mathrm{I}, \mathsf{W}_r^r), \Phi)$ denotes the reservoir subsystem while $((\mathsf{G}_\mathrm{I}, w_{11}), \Phi)$ yields the 1-dimensional output subsystem. The dimensions of G_I and Φ have to be adjusted, respectively, but their gains remain equal to one. In Fig. 18 the network is shown as composition of the subsystems connected by the original feedback weights. The composite system is stable, if the subsystems are stable and because $\gamma(\mathsf{G}_\mathrm{I}) = \gamma(\Phi) = 1$ and thus $\gamma(\mathsf{G}) = \gamma(\mathsf{G}_\mathrm{I}\mathsf{W}) = \|\mathsf{W}\|$ we obtain for the subsystems the inequalities $\|\mathsf{W}_r^r\| < 1$ and $w_{11} < 1$ and, as proved in the Appendix,

$$\|\boldsymbol{w}_{\boldsymbol{x}_r}^o\| < (1 - \|\mathsf{W}_r^r\|)(1 - |w_{11}|)\|\boldsymbol{w}_o^{\boldsymbol{x}_r}\|^{-1} \tag{8.8}$$

for the overall network. Here $\boldsymbol{w}_{\boldsymbol{x}_r}^o$ is the vector of trainable weights and $\boldsymbol{w}_o^{\boldsymbol{x}_r}$ the vector of feedback weights from the output to the reservoir.
The condition (8.8) can be easily monitored online, because the matrix and vector norms on the right hand side can be precomputed at initialization time and while learning only the norm of the output weight vector $\|\boldsymbol{w}_{\boldsymbol{x}_r}^o\|$ has to be updated. Fig. 18 right shows the averaged right hand side (stability margin) of (8.8) for different network sizes and initialization intervals.

Online Stability and rescaling Condition (8.8) can also be used to enforce network stability under learning. The idea is to rescale the full weight matrix by a discount factor λ such that the resulting $\mathsf{W}^+ := \lambda\mathsf{W}$ obeys the stability constraint. Given the current $\|\boldsymbol{w}_{\boldsymbol{x}_r}^o\|$ not fulfilling (8.8), solve

$$\lambda\|\boldsymbol{w}_{\boldsymbol{x}_r}^o\| = \frac{(1 - \lambda\|\mathsf{W}_r^r\|)(1 - \lambda|w_{11}|)}{\lambda\|\boldsymbol{w}_o^{\boldsymbol{x}_r}\|^{-1}} \tag{8.9}$$

for λ to exactly obey the stability condition with the scaled matrix $\lambda\mathsf{W}$. This is a quadratic form in λ and allows for the closed form solution

$$\lambda = -1/2\frac{\|\mathsf{W}_r^r\| + |w_{11}| \pm \sqrt{\|\mathsf{W}_r^r\|^2 - 2\,|w_{11}|\|\mathsf{W}_r^r\| + |w_{11}|^2 + 4\|\boldsymbol{w}_{\boldsymbol{x}_r}^o\|\|\boldsymbol{w}_o^{\boldsymbol{x}_r}\|}}{\|\boldsymbol{w}_{\boldsymbol{x}_r}^o\|\|\boldsymbol{w}_o^{\boldsymbol{x}_r}\| - |w_{11}|\|\mathsf{W}_r^r\|}, \tag{8.10}$$

such that it also can be evaluated online at no significant extra computational burden. To avoid the need for too fast rescaling in practice it is useful to use a slightly smaller λ, e.g. $\lambda - 0.02$ (used in the experiments below). We finally get the stabilized online learning algorithm:

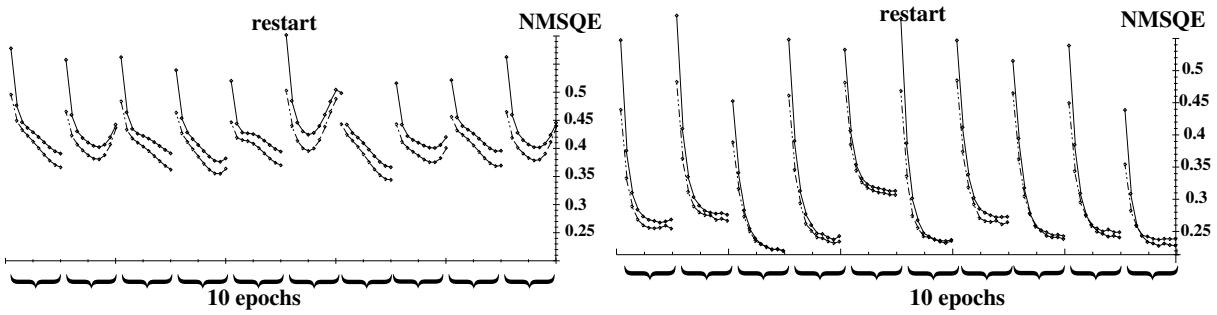


Abbildung 19: Left: Train and test errors for fixed stable initialization. Right Train and test errors for identical parameters, but stable online rescaling. (Ten runs of ten epochs learning of laser data with 50 nodes, 20 inputs, learning rate 0.03, 1000 points training, 1000 points test data).

0) initialize network with weights in $[-a, a]$, compute $\|\mathsf{W}_r^r\|$, $\|\boldsymbol{w}_o^{\boldsymbol{x}_r}\|$, $|w_{11}|$;

1) check stability condition (8.8) for $\|\boldsymbol{w}_{\boldsymbol{x}_r}^o\|$;

2) if (8.8) does not hold

   ◇ compute scaling factor λ according to (8.10),

   ◇ rescale weight matrix $\mathsf{W}^+ := \lambda \mathsf{W}$,

   ◇ rescale $\|\mathsf{W}_r^r\|^+ := \lambda \|\mathsf{W}_r^r\|$, $\|\boldsymbol{w}_o^{\boldsymbol{x}_r}\|^+ := \lambda \|\boldsymbol{w}_o^{\boldsymbol{x}_r}\|$, $\|w_{11}\|^+ := \lambda \|w_{11}\|$,

   ◇ replace the norms by the updated $\|\cdot\|^+$ values in (8.8);

3) iterate network;

4) apply learn step $\Delta \boldsymbol{w}^{BPDC}$;

5) goto 1) (unless stopping criterion is met).