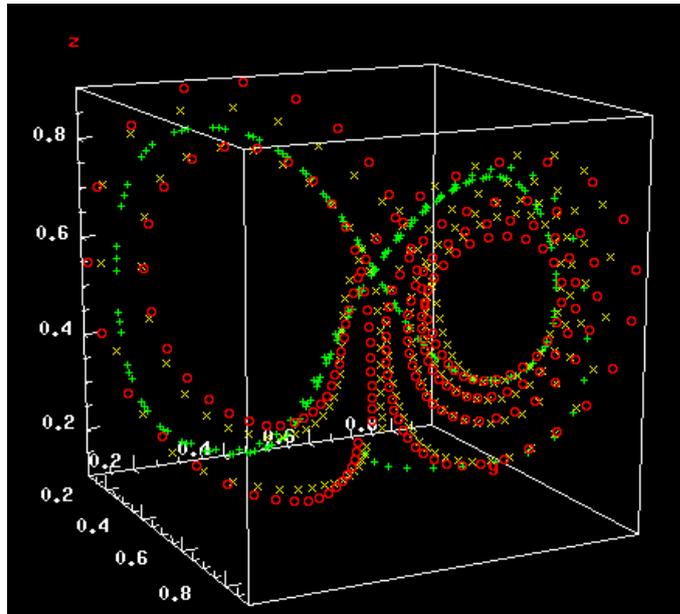


Script to the lecture
Advanced Neural Networks



Dr. Robert Haschke

(Stand: September 7, 2018)

Contents

1	Optimierungsverfahren zum MLP-Lernen	3
1.1	Allgemeine Aspekte	3
1.2	Wiederholung MLP-Lernen	4
1.3	Trägheitsterm oder Momentum	4
1.4	Flat-Spot Elimination	4
1.5	Weight-Decay	5
1.6	Entropie-Fehlermaß	5
1.7	Automatische Schrittweiten Steuerung	7
1.8	Resilient Backpropagation (RPROP)	7
1.9	Quadratische Verfahren	7
2	Support Vector Machine (SVM)	9
2.1	Optimaler Linearer Klassifikator	9
2.1.1	Exkurs: Karush-Kuhn-Tucker-Methode zur Lösung konvexer Opti- mierungsprobleme mit Nebenbedingungen	9
2.2	Numerische Optimierung des dualen Problems	11
2.3	Der Kerneltrick	12
2.4	Soft-Margin Klassifikator: Generalisierung auf nicht-separable Probleme	13
2.5	Support Vector Regression	14
3	Topological Maps	16
3.1	SOM	16
3.1.1	Problem and motivation	16
3.1.2	Kohonen model	17
3.1.3	SOM algorithm	17
3.2	Growing Neural Gas	22
3.2.1	Motivation	22
3.2.2	Approach	22
3.2.3	Algorithm	22
3.2.4	GNG with utility	24
3.3	Instantaneous Topological Map – ITM	25
3.3.1	Algorithmus	25
3.4	Parameterized SOM – PSOM	27
3.4.1	The continuous mapping	27
3.4.2	Continuous Matching	28
3.4.3	Learning	29
3.4.4	Comments	29
3.4.5	Associative Map	30
3.5	Hyperbolische SOM (HSOM)	31
3.5.1	Modelle und Visualisierungen	31
3.5.2	Tesselation of \mathbb{H}^2	33
3.5.3	HSOM learning rule	34
3.5.4	Hierarchically Growing Hyperbolic SOM (H^2 SOM)	34
3.6	Local Linear Maps	35
3.6.1	LLM learning rule	35
3.6.2	Generalisation on output mixtures	36

3.7	Prerequisite: Regression	39
3.7.1	Principal Component Regression (PCR)	39
3.7.2	Partial Least Squares (PLS)	39
3.8	Locally Weighted Projection Regression (LWPR)	40
4	Radiale Basisfunktionen (RBF)	42
4.1	Motivation für lokale Verfahren	42
4.2	Radiale Basisfunktionen	43
4.3	Generalisierte RBF's	45
4.4	Mathematischer Hintergrund	47
4.4.1	Der Regularisierungsansatz	47
4.4.2	Beispiele	48
5	Mixture Models	50
5.1	Mixture of Experts.	50
5.2	Probability density estimation	55
6	Verarbeitung von Zeitreihen	57
6.1	Architekturen zur Repräsentation von Kontext	57
6.1.1	Time-Delay-Neural-Network (TDNN)	57
6.1.2	NARX recurrent neural networks (Nonlinear AutoRegressive with eXogenous inputs)	58
6.1.3	Jordan-Netze (1988)	58
6.1.4	Elman-Netze (1988-1991)	59
6.1.5	Cascade-Corellation Network (CCN)	60
6.2	Backpropagation Through Time - BPTT	60
7	Neural Dynamics	62
7.1	Stabilität Dynamischer Systeme	62
7.1.1	Lokale Stabilität	65
7.1.2	Globale Stabilität	67
7.2	Hopfield-Modell als Assoziativ-Speicher	69
7.3	Brain-State-in-a-Box (BSB)	71
7.4	Competitive Layer Model (CLM)	71
8	Lernverfahren für Rekurrente Netze	74
8.1	Backpropagation Through Time	74
8.2	Real-Time Recurrent Learning	74
8.3	Virtual Teacher Forcing	75
8.4	Reservoir Computing	76
8.5	Backpropagation-Decorrelation Learning	77
8.6	Stability and BPDC	78

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{w}^*

Dies sind widersprüchliche Forderungen:

Theorem 1.1.1. Bias Varianz Dilemma

Lernen entspricht Schätzung der Parameter \vec{w} mit

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

Idee. Zerlege den mittleren quadratischen Schätzfehler

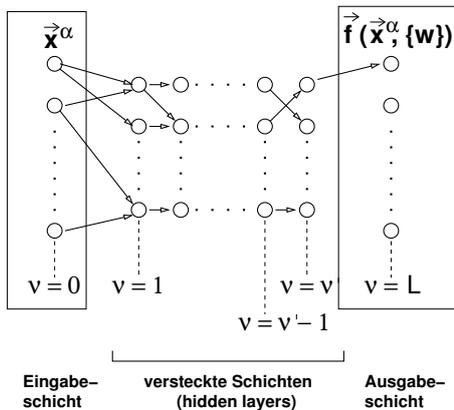
$$\begin{aligned}\langle (\vec{w} - \vec{w}^*)^2 \rangle &= \langle [(\vec{w} - \langle \vec{w} \rangle) + (\langle \vec{w} \rangle - \vec{w}^*)]^2 \rangle \\ &= \langle (\vec{w} - \langle \vec{w} \rangle)^2 \rangle + \langle (\langle \vec{w} \rangle - \vec{w}^*)^2 \rangle + \underbrace{2 \langle \vec{w}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

Bei festem Gesamtfehler: Verringerung der Varianz geht auf Kosten des Bias
→ besseres Modell erforderlich.

d) Hohe Robustheit geringe Abhängigkeit von

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

1.2 Wiederholung MLP-Lernen



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

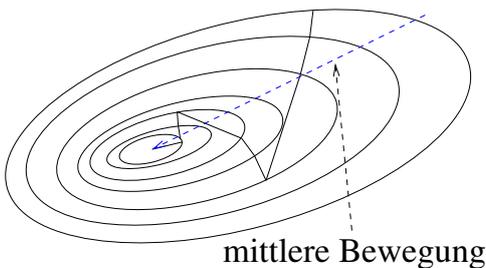
$$E = \sum_\alpha E^\alpha \quad E^\alpha = \frac{1}{2} \sum_i (y_i^\alpha - s_i^L)^2$$

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

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

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

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 sinnvoll in der Nähe des Minimums $\alpha \rightarrow 1$ zu erhöhen.

Beispiel:

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

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

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

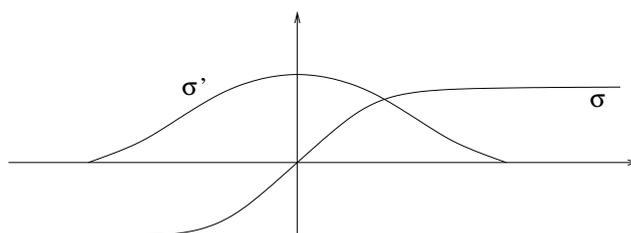
$$= -\eta (1 + \alpha) \vec{a} - \eta (1 - \alpha) \vec{b}$$

1.4 Flat-Spot Elimination

Standard-Lernregel liefert

$$\Delta w_{i,j} \sim \sigma'_i(\cdot)$$

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 $\sigma' \rightarrow \sigma' + C$ für eine kleine Konstante C überall in der Lernregel.

→ Erfahrung: gro"se Beschleunigung.

1.5 Weight-Decay

Plateaus in Fehlergebirgen können von gro"sen Gewichten her stammen.

Abhilfe: Strafterm in Fehlerfunktion

$$E_{reg} = E + \frac{\lambda}{2} \sum_{i,j} \omega_{i,j}^2 \rightarrow \min$$

$$\Delta \vec{w} = -\eta \frac{\partial E}{\partial \vec{w}} - \lambda \vec{w}$$

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

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

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

$$P(\underbrace{x^\alpha, y^\alpha}_{z^\alpha} | \omega) = N \cdot \exp\left(-\frac{\epsilon_\alpha^2}{2\sigma^2}\right)$$

$$\Rightarrow P(z^1, \dots, z^M | \omega) = \prod_{\alpha=1}^M P(z^\alpha | \omega)$$

$$= N' \cdot \exp\left(-\frac{1}{2\sigma^2} \sum_{\alpha=1}^M \epsilon_\alpha^2\right)$$

Anwendung der Bayes Regel:

$$P(\omega | z^1, \dots, z^M) = \frac{P(z^1, \dots, z^M | \omega) P(\omega)}{P(z^1, \dots, z^M)}$$

$$= N'' \exp\left(-\frac{1}{2\sigma^2} \sum_{\alpha=1}^M \epsilon_\alpha^2\right) \cdot \exp\left(-\frac{1}{2\sigma_\omega^2} \sum_{i,j} \omega_{i,j}^2\right)$$

$$= N''' \exp\left(-\frac{1}{2\sigma^2} \left(E + \frac{\lambda}{2} \|\omega\|^2\right)\right) = N''' \exp(-E_{reg}) \quad \text{mit } \lambda = 2 \frac{\sigma^2}{\sigma_\omega^2}$$

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:

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

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 $-\frac{\partial D}{\partial \omega_{i,j}}$:

$$\begin{aligned} -\frac{\partial D}{\partial \omega_{i,j}^{L,L-1}} &= - \sum_k \frac{\partial D}{\partial y_k} \cdot \frac{\partial y_k}{\partial h_i} \cdot \frac{\partial h_i}{\partial \omega_{i,j}^{L,L-1}} & \frac{\partial D}{\partial y_k} &= - \sum_{\alpha} P_k(x^\alpha) \cdot \frac{P_k^\alpha}{y_k^\alpha} \cdot \frac{1}{P_k^\alpha} \\ &= \sum_k \sum_{\alpha} \frac{P_k^\alpha}{y_k^\alpha} \cdot y_k^\alpha (\delta_{k,i} - y_i^\alpha) \cdot s_j^{L-1} & \frac{\partial y_k}{\partial h_i} &= \frac{\delta_{ki} e^{h_k} \cdot N - e^{h_i} \cdot e^{h_k}}{N^2} \\ &= \sum_{\alpha} \sum_k P_k^\alpha (\delta_{k,i} - y_i^\alpha) s_j^{L-1} & &= \delta_{ki} \frac{e^{h_k}}{N} - \frac{e^{h_i}}{N} \cdot \frac{e^{h_k}}{N} \\ &= \sum_{\alpha} \left(P_i^\alpha - y_i \sum_k P_k^\alpha \right) s_j^{L-1} & &= \delta_{ki} y_k - y_i \cdot y_k = y_k (\delta_{ki} - y_i) \\ &= \sum_{\alpha} \underbrace{(P_i^\alpha - y_i^\alpha(x^\alpha, \omega))}_{\varepsilon^\alpha} s_j^{L-1} & \frac{\partial h_i}{\partial \omega_{ij}^{LL-1}} &= s_j^{L-1} \end{aligned}$$

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.

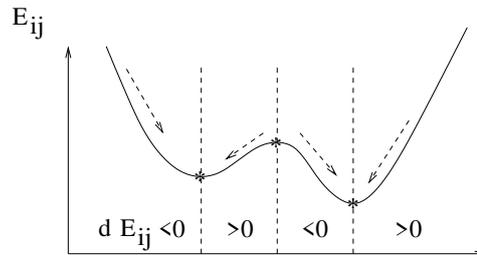
Vorteile von Entropie-Maß + Softmax:

- statistische Fundierung
- "eingebaute" Plateau-Elimination (für die letzte Schicht) trotz nichtlinearer Ausgabe
- i.d.R. bessere Konvergenz
- speziell für 2D Softmax:

$$y_i = \frac{\exp(h_i(x))}{\sum_{j=1}^2 \exp(h_j(x))} \rightarrow \frac{\exp(h_1)}{\exp(h_1) + \exp(h_2)} = \frac{1}{1 + \exp(h_2 - h_1)} = \text{Fermifkt. für Differenz } h_2 - h_1$$

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 $dE = \frac{\partial E}{\partial \omega_{i,j}}$



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

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

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

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 dE bestimmt:

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

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

Bemerkung:

- alle Beschleunigungsverfahren zur Schrittweitensteuerung erfordern **epochenweise** Berechnung der Gradienten $dE_{i,j}$
- 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 $\frac{d E_{ij}(t) - d E_{ij}(t-1)}{\Delta \omega_{ij}(t-1)} \rightarrow$ 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: $\tilde{H} = H + \lambda \mathbf{1}$
 $\Rightarrow \Delta w = -(H + \lambda \mathbf{1})^{-1} \nabla E$

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

$$\tilde{H} = H + \lambda \text{diag } H$$

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

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

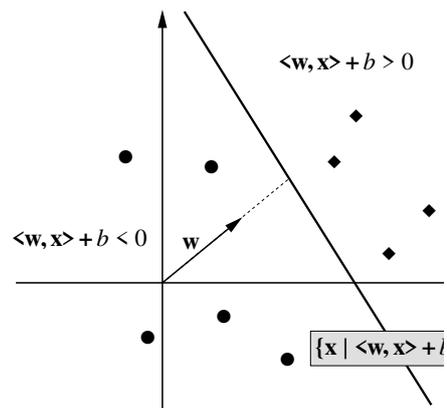
$$y(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x} + b) = \text{sgn}(\langle \vec{w}, \vec{x} \rangle + b)$$

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.

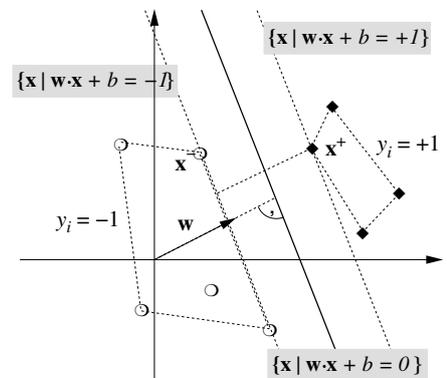
Für die nächstliegenden Punkte \vec{x}^+ bzw. \vec{x}^- gelte:

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

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

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

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

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

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

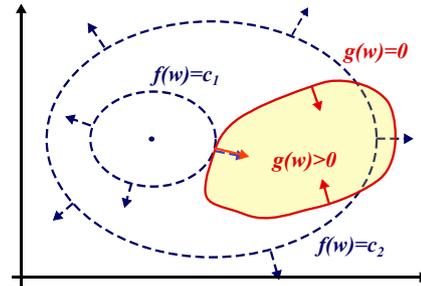
1. Verwende Lagrange-Funktion

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

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.

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

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 Nebenbedingung $g_{\alpha} \geq 0$. 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

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

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

$$\begin{aligned} \nabla_{\vec{w}} L &= \vec{w} - \sum_{\alpha=1}^M y^{\alpha} \lambda_{\alpha} \vec{x}^{\alpha} \stackrel{!}{=} 0 & \Leftrightarrow & \vec{w}^* = \sum_{\alpha=1}^M \lambda_{\alpha} y^{\alpha} \vec{x}^{\alpha} & (2.2) \\ \nabla_b L &= \sum_{\alpha=1}^M y^{\alpha} \lambda_{\alpha} \stackrel{!}{=} 0 \end{aligned}$$

Der Gewichtsvektor ist eine Linearkombination der Datenpunkte \vec{x}^{α} , 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:

$$\begin{aligned} \text{maximiere } W(\vec{\lambda}) &= L(\vec{w}^*, b^*, \vec{\lambda}) \\ &= \frac{1}{2} \vec{w}^{*T} \vec{w}^* - \sum_{\alpha} \lambda_{\alpha} \cdot (y^{\alpha} \cdot (\vec{w}^* \cdot \vec{x}^{\alpha} + b^*) - 1) \\ &= \frac{1}{2} \vec{w}^{*T} \vec{w}^* - \underbrace{\vec{w}^* \sum_{\alpha} \lambda_{\alpha} y^{\alpha} \vec{x}^{\alpha}}_{\vec{w}^*} - \underbrace{b^* \sum_{\alpha} \lambda_{\alpha} y^{\alpha}}_{=0} + \sum_{\alpha} \lambda_{\alpha} \\ &= \sum_{\alpha=1}^M \lambda_{\alpha} - \frac{1}{2} \sum_{\alpha, \beta=1}^M \lambda_{\alpha} \lambda_{\beta} y^{\alpha} y^{\beta} \cdot (\vec{x}^{\alpha} \cdot \vec{x}^{\beta}) \end{aligned}$$

$$\text{mit Nebenbedingungen } \lambda_{\alpha} \geq 0 \quad \text{und} \quad \sum_{\alpha=1}^M y^{\alpha} \lambda_{\alpha} = 0$$

Die Optimierung erfolgt numerisch mittels Standard-Verfahren der quadratischen Programmierung.

Einsetzen von (2.2) in den Klassifikator ergibt:

$$\begin{aligned} y(\vec{x}) &= \text{sgn} \left(\sum_{\substack{\alpha \\ \lambda_{\alpha} \neq 0}} \lambda_{\alpha} y^{\alpha} \cdot (\vec{x} \cdot \vec{x}^{\alpha}) + b^* \right) \\ b^* &= -\frac{1}{2} (\max_{y^{\alpha}=-1} \vec{w}^* \cdot \vec{x}^{\alpha} + \min_{y^{\alpha}=+1} \vec{w}^* \cdot \vec{x}^{\alpha}) \end{aligned}$$

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

- Optimierte immer nur bzgl. zweier Datenbeispiele
- ⇒ 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 $K(\vec{x}, \vec{x}') = \Phi(\vec{x}) \cdot \Phi(\vec{x}')$:

$$f(\vec{x}) = \vec{w}^* \cdot \Phi(\vec{x}) + b^*$$

mit $\vec{w}^* = \sum_{\alpha} \lambda_{\alpha} y^{\alpha} \Phi(\vec{x}^{\alpha})$

eingesetzt: $f(\vec{x}) = \sum_{\alpha} \lambda_{\alpha} y^{\alpha} \underbrace{\Phi(\vec{x}^{\alpha}) \cdot \Phi(\vec{x})}_{K(\vec{x}^{\alpha}, \vec{x})} + b^*$

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

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

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

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: $\int g(x)K(x, y)g(y) dx dy \geq 0$ für alle $g(x) \in L^2$

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

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

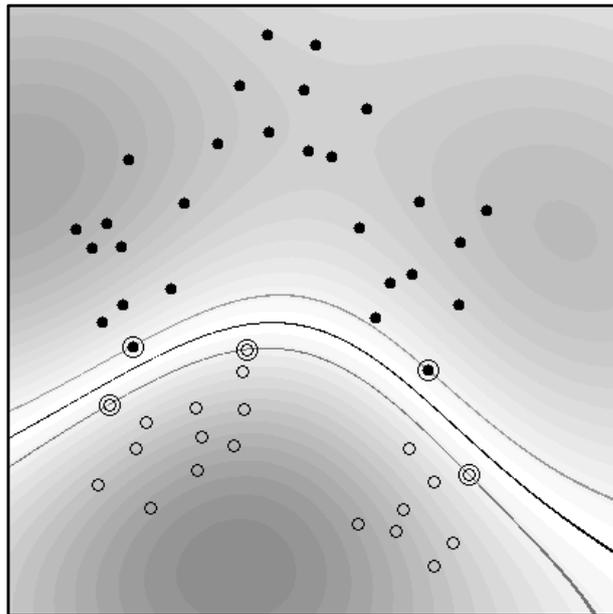


Figure 1: 2-Klassen-Problem mit RBF-Kernel: Supportvektoren sind eingekreist, dunklere Regionen entsprechen einer sichereren Entscheidung.

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

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

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

$$\text{Maximiere } W(\vec{\lambda})$$

$$\text{mit Nebenbedingungen } 0 \leq \lambda_{\alpha} \leq C \quad \text{und} \quad \sum_{\alpha=1}^M y^{\alpha} \lambda_{\alpha} = 0$$

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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$.

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)

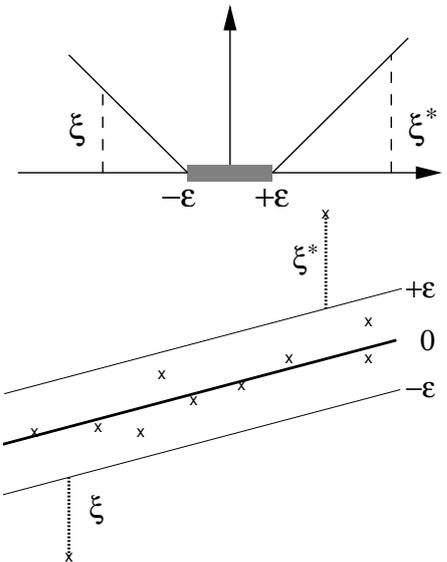
$$\text{Minimiere } \frac{1}{2} \|\vec{w}\|^2 + C \sum_{\alpha=1}^M (\xi_{\alpha} + \xi_{\alpha}^*)$$

$$\text{mit Nebenbedingungen } y^{\alpha} - (\vec{w} \cdot \vec{x}^{\alpha} + b) \leq \varepsilon + \xi_{\alpha}^*$$

$$(\vec{w} \cdot \vec{x}^{\alpha} + b) - y^{\alpha} \leq \varepsilon + \xi_{\alpha}$$

$$\xi_{\alpha}, \xi_{\alpha}^* \geq 0$$

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:

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

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

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 Topological Maps

3.1 Self-organizing, topology-preserving map (SOM)

3.1.1 Problem and motivation

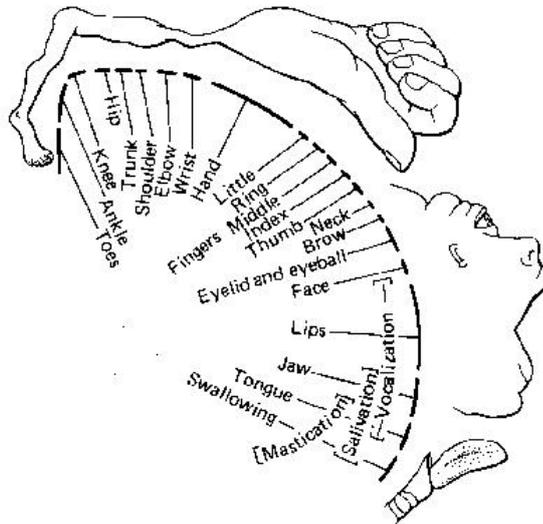
How to get from "low-level" sensory data to symbolic, "high-level" abstractions? (and back)?

Aspects:

- Transition from "physical" to logical representations
- Ignore insignificant information
- Focus on significant, structural information ("Topology")
- Finding prototypes (Clustering / symbolic representation)
- Learning

Topological maps in the brain (cortex):

- visual, auditory and other sensory maps – stimulus perception
- motor maps – controlling of muscles



- topological structure:
similar stimuli trigger excitation of neighboring neurons
- Cortex can retrain fast:
 - ◇ If losing fingers, the sensation and moveability of neighboring fingers amplifies
 - ◇ Blind people use brain areas normally reserved to seeing for hearing and feeling
- Minimization of "wiring costs"

Question: How can we explain the formation of such topological maps?

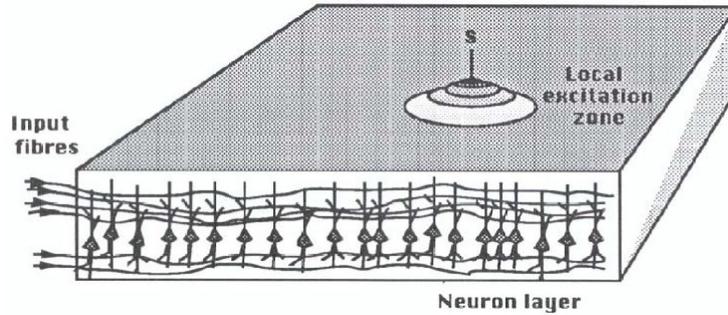


Figure 2: Kohonen model: In a two-dimensional layer of neurons, a local peak forms at the location of the maximum reaction to the input ($\vec{w} \cdot \vec{x}$) due to the competition of neurons among each other (lateral inhibitions).

3.1.2 Kohonen model

A first model for this was introduced by the Finish scientist Teuvo Kohonen.

Approach

- two-dimensional layer of neurons
- Neuron c at place \vec{c} with excitation / activation y_c
- Input $\vec{x} \in \mathbb{R}^L$ and corresponding weights $\vec{w}_c \in \mathbb{R}^L$
- lateral inhibition of the neurons via weights $g_{cc'}$
- $y_c = \sigma \left(\underbrace{\vec{w}_c \vec{x}}_{\text{reaction to input}} + \underbrace{\sum_{c'} g_{cc'} y_{c'}}_{\text{lateral inhibition}} - \underbrace{\theta}_{\text{threshold}} \right)$
- short distance excitation, long distance inhibition
- Competition ($g_{cc'}$) is balanced such that neuron n with maximum reaction to the input becomes the center of a local excitation zone:

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

- Learning rule: Shift all weights towards the stimulus \vec{x} :

$$\Delta \vec{w}_c = \eta \cdot y_c \cdot (\vec{x} - \vec{w}_c) \quad (\text{Hebb-Rule + weight decay})$$

\nearrow \nwarrow
learning rate activity

3.1.3 SOM – mathematical abstraction of the Kohonen model

Simplifications:

- Fixed grid of nodes with associated weights $w_c \in \mathbb{R}^L$
- Indexing of neurons via grid positions \vec{c}

- **Directly** determine the neuron n with maximum reaction to the input:

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

- Replace scalarproduct $\vec{x} \cdot \vec{w}$ with distance $\|\vec{x} - \vec{w}\|$:

$$2 \vec{x} \cdot \vec{w} = \|\vec{x}\|^2 + \|\vec{w}\|^2 - \|\vec{x} - \vec{w}\|^2$$

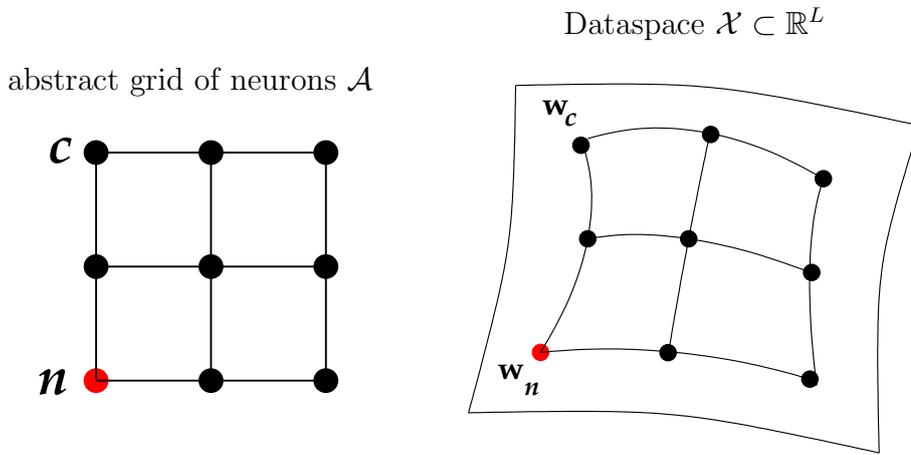
$$\Rightarrow n = \arg \min_c \|\vec{x} - \vec{w}_c\| \quad (\text{WTA: winner takes all})$$

Therefore the weights \vec{w}_c can be interpreted as **prototypes** for the input.

- Replace y_c in the learning rule by a local Gaussian distribution:

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

Visualization



The weights \vec{w}_c of the neurons are drawn as prototypes in the input/data space. Moreover the grid structure (which so far is only an abstract representation of the neighbourhood structure) is transferred to the input space such that the manifold of the SOM (i.e. the curved surface) becomes visible.

Algorithm

1. Choose a stimulus \vec{x}^α randomly
2. Determine winner neuron n according to:

$$n = \arg \min_c \|\vec{x}^\alpha - \vec{w}_c\|$$

i.e. $\|\vec{x}^\alpha - \vec{w}_n\| < \|\vec{x}^\alpha - \vec{w}_c\| \quad \forall c \neq n$

3. Learning step:

$$\Delta w_c = \eta \cdot h_{cn} \cdot (\vec{x}^\alpha - \vec{w}_c) \quad \text{for all neurons } c$$

Where η is the learning rate and h_{cn} the neighborhood function: $h_{cn} = e^{-\frac{\|\vec{c}-\vec{n}\|^2}{2\sigma^2}}$

4. (a) Reduce learning rate η as well as the range σ of the neighborhood:
 \Rightarrow the net will become less sensitive to outliers
 (b) Use adaptive step size control for learning rate and neighborhood
5. GOTO 1 until quality is satisfactory
 or "no changes" occur anymore

Comments

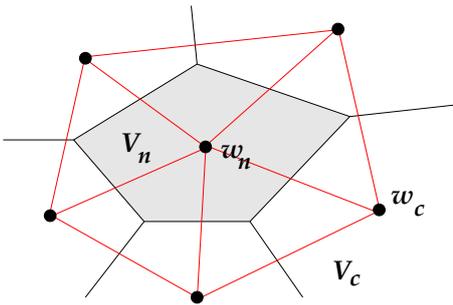
- Prototypes \vec{w}_c are connected according to the grid structure through "rubber bands"
- Learning rule pulls winner prototype towards the stimulus (VQ learning rule)
- Because of the rubber bands (h_{cn}) adjacent prototypes (according to the grid structure) will also be adapted (guarantees smooth manifold)
- σ in the exponential function controls the range of adaptations
- Response of the winner neuron n gets stronger for repeated input stimuli
- Topologically neighboring neurons specialize on similar input stimuli
- *unsupervised learning method*

Applications If either the coordinate index \vec{n} of the winner neuron or its associated weight vector \vec{w}_n is chosen as the output of the SOM, we get two different types of applications:

- local coordinate system: mapping $\Phi : \mathcal{X} \rightarrow \mathcal{A} = \{\vec{c}\}$
 $\Phi(\vec{x}) = \vec{n}$ yields local coordinates \vec{n} of stimulus \vec{x} in the data manifold. Because the descriptive dimension L of the data is usually much larger than the dimension m of the SOM grid, we also achieve a *non-linear dimension reduction*.
- Vector quantization: mapping $\Phi : \mathcal{X} \rightarrow \mathcal{X}_d = \{\vec{w}_c\}$
 The data \vec{x} will be projected onto the nearest prototype $\Phi(\vec{x}) = \vec{w}_n$

Voronoi cell The set of input stimuli, for which n becomes the winner neuron, is denoted as *Voronoi cell* of n

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



- The boundaries of a Voronoi cell are the perpendicular bisectors of the lines connecting two nodes.
- Voronoi cells are convex, i.e. the connecting line of any two points is also inside the Voronoi cell.
- The **Delaunay triangulation** is constituted through the connection of all nodes i and j , whose Voronoi cells V_i and V_j share a common edge. It captures the topology of the neuron grid.

Comparison to standard vector quantization (VQ) $VQ \approx$ SOM without neighborhood. The standard-VQ-algorithm adapts *only* the winner node. The VQ learning rule

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

can be interpreted as the stochastic gradient of

$$E(w) = \frac{1}{2} \int_x \|\vec{w}_s - \vec{x}\|^2 p(x) d^d \vec{x} = \left\langle \frac{1}{2} \|\vec{w}_s - \vec{x}\|^2 \right\rangle_{p(x)}$$

where $x \in \mathbb{R}^L$ and $p(\vec{x})$ denotes the data's probability density. The VQ algorithm therefore tries to minimize the *average quantization error* E .

It can be shown that

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

is a stationary point of the VQ algorithm. Solving equation for \vec{w}_n yields

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

Thus, for VQ the prototypes \vec{w}_n are distributed according to the expectation values of their corresponding Voronoi cells V_n . VQ calculates these prototypes iteratively, according to the training data $\{\vec{x}^\alpha\}$.

The SOM algorithm extends the VQ algorithm by also considering the neighborhood. If the range σ of the neighborhood is variable there does not exist a general error function anymore. However, for a fixed σ SOM minimizes the error:

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

Similarly as before we get:

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

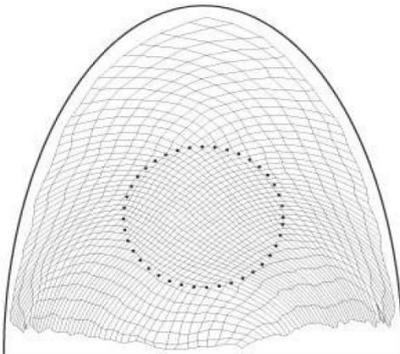
More properties of the SOM

- **Density distribution:** SOM exhibits a fovea effect, i.e. there is a higher resolution in regions of higher data density. Defining the zoom factor $m(\vec{x})$ as the expected amount

$$\int m(x) d^L x = l = \text{number of neurons}$$

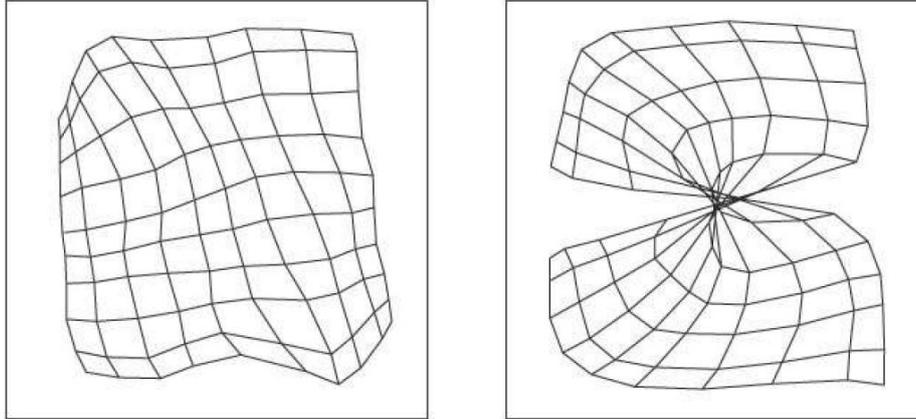
of neurons per volume element $d^L \vec{x}$, i.e. We can prove: $m(x) \propto p^{\frac{2}{3}}$ (Ritter, 91)

In regions of higher data density the neurons accumulate.



- **Topological order / Topological defects**

Topological defects are present if the neighborhood relation of the SOM is not matching the neighborhood relation of the data. Defects occur when the neighborhood range σ is reduced too quickly. Generally, it is hard to detect them.



3.2 Growing Neural Gas

(Fritzke 1994/95 [?, ?])

3.2.1 Motivation

- SOM requires fixed network size, i.e known topology
- search unsupervised learning method of high dimensional data for *unknown* topology
- Intrinsic dimension (ID) = Amount of independent parameters in a generative model of the data:

$$P(x) \propto f(\vec{v}) + \eta, \text{ where } \dim(\vec{v}) = \text{ID}$$

- ◊ very often ID much smaller than $\dim(\vec{x})$
- ◊ ID is variable
- ◊ ID is also an interesting learning target

Problem: Find for given distribution $P(\vec{x})$, a topological (map-)structure that approximates optimally the topology of the data

3.2.2 Approach

- Set of nodes: $\mathcal{A} = \{c_1, \dots, c_N\}$ with
 - ◊ assigned reference vector $w_i = \vec{w}_{c_i} \in \mathbb{R}^L$
 - ◊ and accumulated local error $e_i = e_{c_i}$
- Set of nodes: $\mathcal{C} = \mathcal{A} \times \mathcal{A}$
 - ◊ undirected / symmetrical: $(c_i, c_j) \in \mathcal{C} \Leftrightarrow (c_j, c_i) \in \mathcal{C}$
 - ◊ Edge age $\text{age}(c_i) \in \mathbb{R}$
- Mechanism to add and delete nodes and edges

3.2.3 Algorithm

1. Start: two random nodes $\mathcal{A} = \{c_1, c_2\}$ from $p(\vec{x})$, no connection: $\mathcal{C} = \emptyset$
2. **Matching:** Choose random input \vec{x} according to $p(\vec{x})$ and search for nearest and second nearest neighbour:

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

3. **Reference Vector Adaptation(VQ):** Adapt winner n and its nearest topological neighbours $\mathcal{N}(n)$ with different learning rates $\eta_1 > \eta_2$:

$$\begin{aligned} \Delta w_n &= \eta_1 (\vec{x} - \vec{w}_n) && - \text{Winner} \\ \Delta w_c &= \eta_2 (\vec{x} - \vec{w}_c) \quad \forall c \in \mathcal{N}(n) && - \text{Neighbours of the Winner} \end{aligned}$$

4. Edge-Update

- Create new edge (n, s) between winner n and second winner s
- Reset edge age: $\text{age}(n, s) = 0$
- Increment edge age of all edges in the neighborhood of the winner: $\text{age}(n, c) \leftarrow \text{age}(n, c) + 1 \quad \forall c \in \mathcal{N}(n)$
- Delete too old edges: $\text{age}(n, c) > a_{\max}$
- Delete edgeless nodes

5. Node-Update

- Error-Update: $\Delta e_n = \|\vec{x} - \vec{w}_n\|^2$
- Insert *one* new node every λ iterations:
 - ◇ Search for node q with highest error: $q = \arg \max_{c \in \mathcal{A}} e_c$
 - ◇ Find topological neighbour p with highest error: $p = \arg \max_{c \in \mathcal{N}(q)} e_c$
 - ◇ Insert new node r between p and q : $\vec{w}_r = \frac{1}{2}(\vec{w}_p + \vec{w}_q)$
 - ◇ Replace edge (q, p) by new edges (q, r) and (p, r)
 - ◇ Reduce error variables

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

- Error decay term: $\Delta e_c = -\beta e_c \quad \forall c \in \mathcal{A}$

6. GOTO 1 until termination criteria met

Comments

- a lot of (but pretty robust) parameters, i.e same parameters suited for a lot of situations
 - ◇ learning rate η_1 and η_2
 - ◇ Node creation interval λ and maximum edge age a_{\max}
 - ◇ Error decay variables α and β
- Approach can deal with non stationary data: new nodes appear if needed, unused nodes vanish (usually)
- Node creation is bound to time interval λ and does not consider the absolute quantity of the quantization error.

3.2.4 GNG with utility

Problem: If we change the data distribution too fast often a lot of unnessecary nodes get left behind. These nodes do not receive any stimuli anymore because of the changed distribution and therefore also do not increment the edge age of adjacent neurons. This is a problem because it's the only mechanism to delete edges and therefore also delete to nodes.

Solution:

- Additional utility variable u_c per node
(as a measurement of the increased quantization error when node gets deleted)
- $\Delta u_n = \|\vec{x} - \vec{w}_s\| - \|\vec{x} - \vec{w}_n\|$
(Difference of the quantization error of the winner n and second winner s)
- Decay for all utility variables: $\forall c \in \mathcal{A} \quad \Delta u_c = -\beta u_c$

\Rightarrow small utility if (local) node density high or node is activated seldomly

- Delete node $i = \arg \min_c u_c$ with minimal utility if the (expected) reduction of the quantization error is higher than through the insertion of a new node at a different place (at the node q with highest local error) : $u_i < k^{-1} e_q$

3.3 Instantaneous Topological Map – ITM

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

SOM and GNG are not suited for correlated stimuli. An example for correlated stimuli would be for example explorational movements in the robotics department. If we generate stimuli along a trajectory the same neuron "wins" a lot of times in a row and follows the stimuli along it's path due to the VQ-Learning-Rule. (siehe Abb. 3).

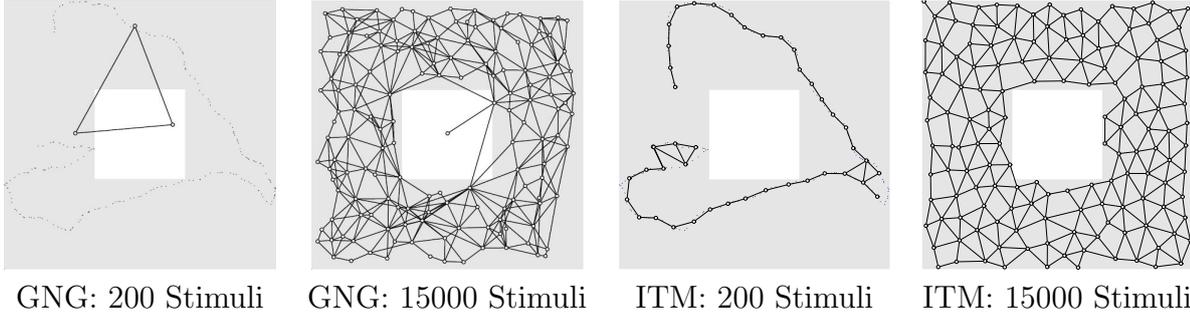


Figure 3: Comparison between GNG and ITM performance on correlated stimuli. After a few stimuli GNG approximates the presented trajectory badly because of destructive interference. For ITM on the other hand, the trajectory is correctly drawn. Although in the asymptotic limit these two approaches do not differ too much. Courtesy of Ján Jokusch.

Idea: Edge- and node-creation aswell as -deletion only due to *local* properties

3.3.1 Algorithmus

1. Start: Two random nodes $\mathcal{A} = \{c_1, c_2\}$ from $p(\vec{x})$, no connection: $\mathcal{C} = \emptyset$
2. **Matching:** Choose random input \vec{x} according to $p(\vec{x})$ and search for nearest and second nearest node:

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

3. **Reference vector-adaptation (VQ):** Adapt *only* the winner n :

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

4. Edge-Update

- Create new edge (n, s) between winner n and second winner s
- For every neighbour c of n :
If \vec{w}_s is in the thales circle of \vec{w}_n and \vec{w}_c , delete edge (n, c) . (The new edge (n, s) replaces the unnessecary old edge (n, c) .)

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

- Delete edgeless nodes

5. Node-Update

- If stimuli is outside of the thales circle around \vec{w}_n and \vec{w}_s and outside of the circle with radius e_{\max} around \vec{w}_n
 $((\vec{w}_n - \vec{x}) \cdot (\vec{w}_s - \vec{x}) > 0 \quad \text{and} \quad \|\vec{x} - \vec{w}_n\| > e_{\max})$:
 - ◇ Insert a new node at the place of the stimuli r : $\vec{w}_r = \vec{x}$
 - ◇ Connect r with n
- If \vec{w}_n and \vec{w}_s closer than $\frac{1}{2} e_{\max}$, delete node s

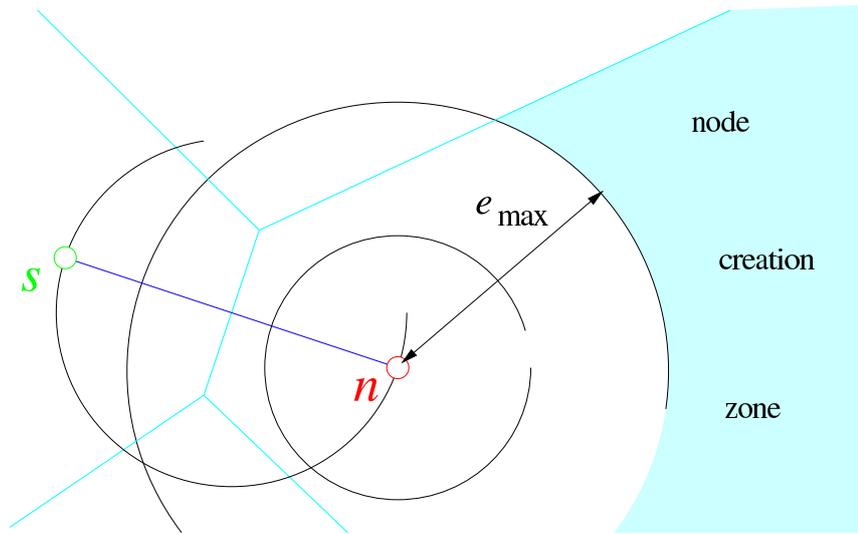


Figure 4: Node-Update-Mechanism of ITM: new nodes get inserted as soon as the new stimuli \vec{x} has a too large distance to the winner n and it's not inside the circle between the nearest and second nearest node. (Thales-Condition). Courtesy of Ján Jockusch.

Comments

- Only 2 parameters: maximum quantization error e_{\max} and learning rate η
- Reference vector adaptation (VQ-Learning Rule) not necessary. $\eta > 0$ is only responsible for the evenly distributed nodes.
- e_{\max} depends strongly on the data. Choose e.g. expected stimuli distance of successive correlated stimuli.
- No Fovea-Effect as in SOM or GNG.
- Nodes in non stationary distribution which are not needed anymore do *not* get deleted.
- SOM: Strict topology, adapts node positions
 ITM: Generates topology, fixed node positions

3.4 Parameterized SOM – PSOM

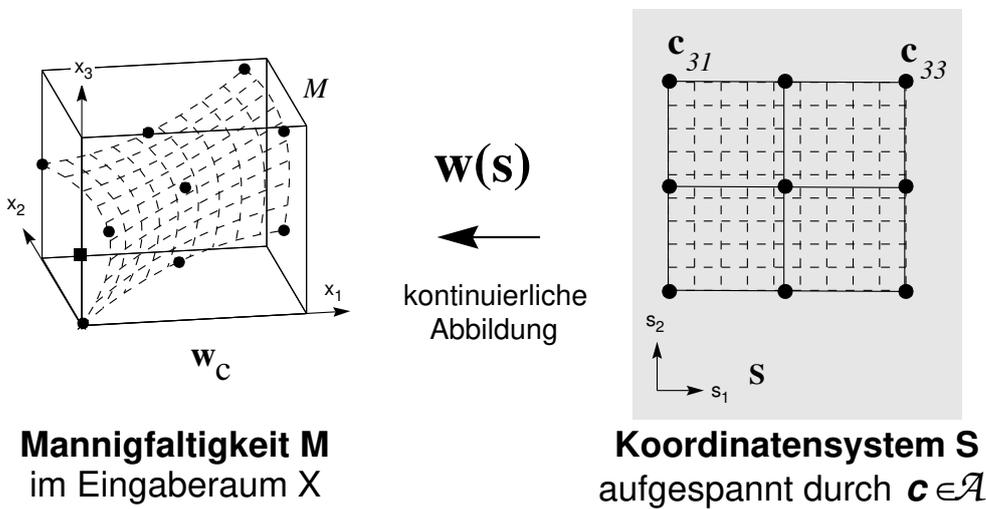
(Ritter '93, Walter '96, AGAI [?, ?, ?, ?])

Idea: Generalize the discrete mapping of the SOM to a continuous mapping on a manifold.

- adaptive model of a flat manifold in the input space
- only few training data necessary
- inherits a modified learning rule of the SOM
- typical application in robotics where training data is expensive but able to be actively collected

3.4.1 The continuous mapping

- Extend discrete neuron grid of the SOM ($\vec{c} \in \mathcal{A}$) to a continuous parameter space / coordinate system $\vec{s} \in S \supset \mathcal{A}$
- $\vec{w}_{\vec{c}} \in \mathcal{X} \subseteq \mathbb{R}^L$ is reference vector to node \vec{c}
- extend discrete mapping $\Psi_d : \mathcal{A} \rightarrow \mathcal{X}_d$ ($\vec{c} \mapsto \vec{w}_{\vec{c}}$) of SOM to a continuous mapping $\Psi : S \rightarrow \mathcal{M} \subseteq \mathcal{X}$ ($s \mapsto \vec{w}(s)$)
- \mathcal{M} is flat manifold in input space \mathcal{X}



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

where the basis function $H(c, s)$ should have the following properties:

1. \mathcal{M} should cross the (reference vectors) ,
i.e $\vec{w}(c) = \vec{w}_c$ or $H(c, c) = 1$ and $H(c, c') = 0$ for all $c' \neq c$

$\Rightarrow H(c, s)$ has to be orthonormal function system: $H(c, c') = \delta_{cc'}$

2. Constant functions should be representable, i.e $\forall s \in S \sum_{c \in \mathcal{A}} H(c, s) = 1$

$\Rightarrow H(c, s)$ produces "Decomposition of 1"

Solution: Multi dimensional lagrange polynoms on regular grid

one dimensional grid: $\mathcal{A} = \{c_1, c_2, \dots, c_n\} \subset \mathbb{R}$

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

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

multidimensional grid: Factorization along the particular dimensions

$$\mathcal{A}_\nu = \{c_{1\nu}^\nu, \dots, c_{n_\nu}^\nu\}, \quad \nu = 1 \dots m$$

$$\vec{c} \in \mathcal{A}_1 \times \dots \times \mathcal{A}_m \quad (\text{d.h. } \vec{c} = (c_{i_1}^1, c_{i_2}^2, \dots, c_{i_m}^m)^T)$$

$$H(\vec{c}, \vec{s}) = \prod_{\nu=1}^m l_{i_\nu}^\nu(s_\nu, \mathcal{A}_\nu)$$

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

For every node $c_{i_\nu}^\nu$ along a dimension ν exactly one lagrange polynom $l_{i_\nu}^\nu$ is assigned.

3.4.2 Continuous Matching

Task of the PSOM is to realize the inverse mapping $\Psi : S \rightarrow \mathcal{M}$. I.e to a given stimulus \vec{x} the – now continuous – coordinates \vec{s} are searched for, such that $\vec{w}(\vec{s})$ is close to \vec{x} .

Search for continuous coordinates \vec{s}^* for the stimulus \vec{x} which is closest to the nearest point $\vec{w}(\vec{s}^*) \in \mathcal{M}$:

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

The now transformed search space (continuous) $s \in S$ requires instead of a simple discrete search now a non linear optimization of

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

through gradient descent:

- Start at winner node: $\vec{s}_0 = \arg \min_{a \in \mathcal{A}} \|\vec{x} - \vec{w}(\vec{s})\|$
- iterative gradient descent:

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

- the Levenberg-Marquardt-Algorithm converges faster (quadratic approach):

$$\text{Solve } (H(\vec{s}_t) + \lambda \cdot \mathbf{1}) \vec{\delta}_s = -\nabla_{\vec{s}} E(\vec{s}_t)$$

◇ Origin: local quadratic 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 of $q(\vec{\delta}_s)$ if $\nabla_{\vec{\delta}_s} q = 0$

Both approaches require the calculation of the gradient $\nabla_{\vec{s}} E(\vec{s}_t)$ (or the Hessian Matrix $H = \frac{\partial^2}{\partial \vec{s}^2} E(\vec{s}_t)$), which is although easily analytically defined through the chosen lagrange polynoms (Walter '96).

3.4.3 Learning

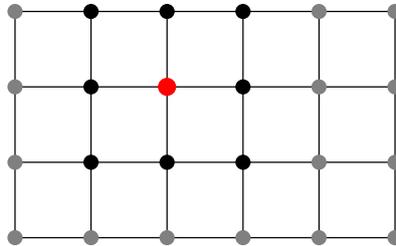
1. **ad-hoc Learning:** Choose fitting grid nodes $\vec{c} \in \mathcal{A}$ and define assigned reference vectors $\vec{w}_{\vec{c}}$.
2. **modified SOM-Learning Rule:** Shift reference vectors weighted relatively to its contribution in the current state:

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

Note: $\Delta \vec{w}_{\vec{c}} = \eta \cdot H(\vec{c}, \vec{s}^*) \cdot (\vec{x} - \vec{w}_{\vec{c}})$ does not converge i.e even destroys the learned manifold.

3.4.4 Comments

- PSOM is global learning rule (in contrast to SOM, GNG and ITM):
Local changes e.g of the reference vectors lead to global changes:
 - ◇ Due to the big support of a polynom in a specific dimension and
 - ◇ because of the factorization in every other dimension
- Alot of anchor points i.e grid points along a dimension lead to a high polynomial degree and therefore to unnessecary oscillations. Thaths why only 3 to 4 grid points should be chosen per grid dimension.
- To cover a broader space, for a (local) approximation not all grid points but only a local $3 \times \dots \times 3$ sub grid is used.



The emerging problem of discontinuity is solvable (Walter '96).

- Usage of Tschebycheff-Polynoms is also meaningful. → needs a non regular grid
- Topological defects can occur
- The chosing of the topology also lays ground for a model to interpret the data.

3.4.5 Associative Map

Idea: Use SOM, GNG, ITM, PSOM and other related approaches as Input-Output functions and learn this mapping supervised. For this the bisherige reference vector $\vec{w}_c \equiv \vec{w}_c^{in}$ in input space \mathcal{X}^{in} is extended by an additional output vector \vec{w}_c^{out} in output space \mathcal{X}^{out} . Both vectors are concatenated to a single one \vec{w}_c :

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

While the VQ-Learning step considers all components \vec{w}_c the matching should only consider the input components \vec{w}_c^{in} . This is done by modifying the norm:

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

Chose $p_k = 1$ for the input- and $p_k = 0$ for the output components. All topological learning methods can realize the following input-output function:

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

While SOM, GNG and ITM only assigns a constant output to every voronoi cell \vec{w}_c^{out} – meaning a piecewise constant mapping – PSOM is also interpolating in output space.

An arbitrary choosing of input- and output components due to $p_k \in \{0, 1\}$ also allows for a associative mapping:

- Select P with given components and complete the rest

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

- will function well if components are functionally dependent on each other
- can be used to invert functions

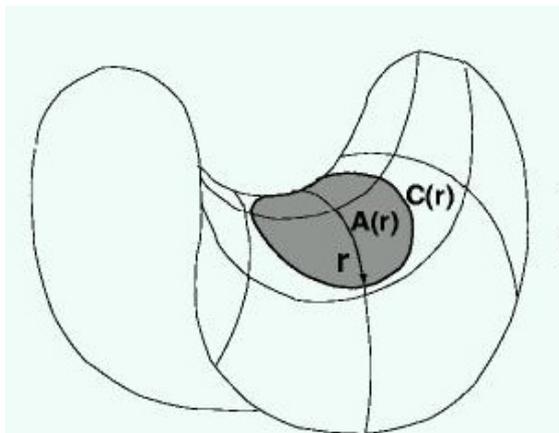
3.5 Hyperbolische SOM (HSOM)

(Ritter '99, AGAI [?])

Motivation SOMs are very useful when it comes to visualize high dimensional data because they project the inherent structure to a lower dimensional neighborhood. This ability also depends upon how the *Number of possible neighbours* rises with the distance from a node. The previous SOM-Approaches uses an euclidean space whose volume rises in a polynomial fashion with the neighborhood radius: $V(r) \propto r^d$.

The mapping of hierarchical tree-like data structures (like e.G das WWW) requires an exponential growth of the neighborhood. The hyperbolic space provides such an exponential growth of the volume and therefore serves as the ideal space for the mapping of hierarchical data structures.

The hyperbolic space The two dimensional hyperbolic space \mathbb{H}^2 is a space with evenly negative curvature such that it can be imagined as a *saddle* in euclidian space (Figure. 5). An isometrical, meaning distance maintaining embedding in euclidian space \mathbb{R}^2 is – similar to the sphere – not possible



$$\text{Umfang: } C(r) = 2\pi \sinh(r)$$

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

wobei

$$\sinh(x) = \frac{1}{2}(e^x - e^{-x})$$

$$\cosh(x) = \frac{1}{2}(e^x + e^{-x})$$

Perimeter und surface are growing exponentially with r !

Figure 5: The hyperbolic space – imaginable as a saddle – provides an exponential growth of the neighborhood.

3.5.1 Modelle und Visualisierungen

All models of the two dimensional hyperbolical space \mathbb{H}^2 are represented through polar coordinates (r, θ) .

Hyperboloid / Minkowski-Model

- Projection on 3-dimensionalen Hyperboloid: $z^2 = x^2 + y^2 + 1$

$$x = \sinh(r) \cos(\theta)$$

$$y = \sinh(r) \sin(\theta)$$

$$z = \cosh(r)$$

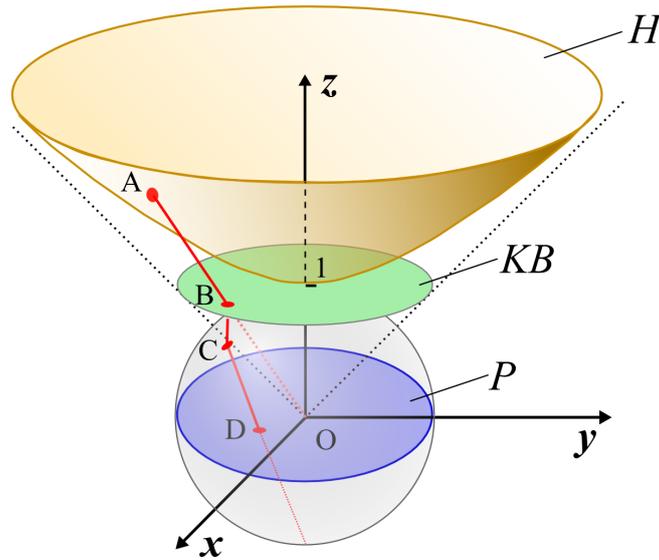


Figure 6: Different models / Embedding of \mathbb{H}^2 in euclidian space: Hyperboloid H in \mathbb{R}^3 , Poincaré Disk P in \mathbb{R}^2 . Courtesy of Jörg Ontrup.

- With the minkowski this is an isometrical embedding.

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

The minkowski metric is although not positive definite, so it's not a valid metric.

Poincaré Disk With multiple projection steps we arrive at the Poincaré Disk (Abb. 6 from hyperbolic space:

- Small-Beltrami-Modell: Projection on surface $z = 1$ with rays through the origin O : $A \rightarrow B$
- orthogonal projection on uni sphere around origin: $B \rightarrow C$
- stereographical projection on unit circle around origin (due to a ray preceeding through the south pole of the sphere): $C \rightarrow D$

→ provides Poincaré-Disk with the following properties:

- ◇ Transformation:

$$\begin{aligned} x &= \tanh\left(\frac{1}{2}r\right) \cos(\theta) \\ y &= \tanh\left(\frac{1}{2}r\right) \sin(\theta) \end{aligned}$$

- ◇ Mapping on finite disc: $x^2 + y^2 < 1$
- ◇ Angles are kept (and therefore geometrical forms)
- ◇ extremely strong fish eye effect:
 - * Origin of \mathbb{H}^2 mapped almost 1:1
 - * distant regions exponentially compressed
- ◇ Lines are mapped on circle

◇ Points $z \in P \equiv \mathbb{C}$ – designated as complex numbers – can be used to determine hyperbolic distance:

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

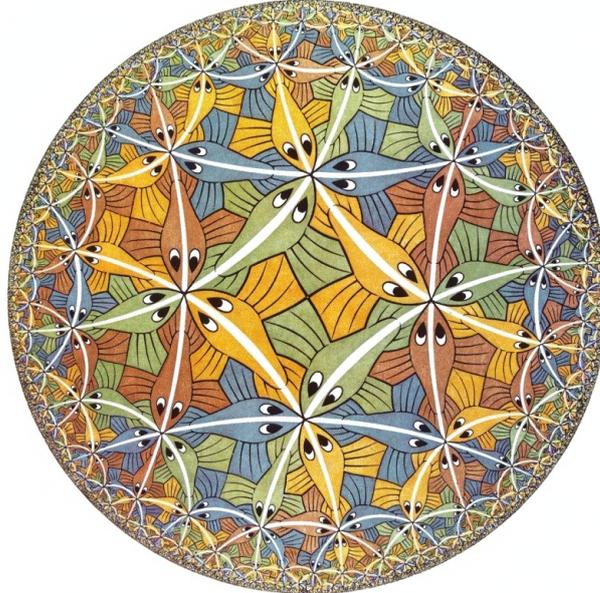


Figure 7: The fish eye effect of the poincaré-disc can be shown on the art series "Circle Limit" from M.C. Escher.

3.5.2 Tessellation of \mathbb{H}^2

How to generate a useful grid structure in \mathbb{H}^2 ?

- surround every node with $n \geq 7$ same sided triangles
- per node n edges: 1 parent, 2 siblings, $n - 3$ children

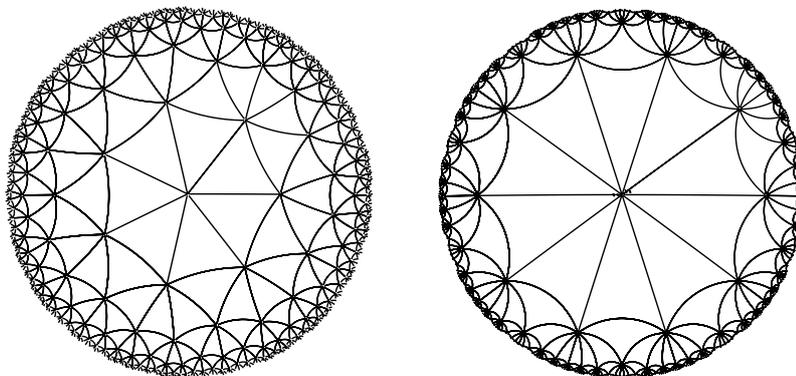


Figure 8: Tessellation of \mathbb{H}^2 with $n = 7$ or $n = 10$ edges.

3.5.3 HSOM learning rule

There is no difference from standard SOM algorithms except the change of topology (3.1) and the distance metric!

- save for every node c that position on the poincaré disc as complex number z_c
- use new metric (3.1) in learning step: $\Delta \vec{w}_c = \eta \cdot h_{cn} \cdot (\vec{x} - \vec{w}_c)$

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

3.5.4 Hierarchically Growing Hyperbolic SOM (H²SOM)

Jörg Ontrup, AG AI [?]

Idea: Use hierarchical structure also for search (\rightarrow Tree-Search) and extend the HSOM only when necessary (similar to GNG)

- Start: initialize root nodes with mean of the data: $\vec{w}_c = \langle \vec{x} \rangle$
 - After λ training steps the HSOM gets extended at the nodesc, where the mean of the quantization error \bar{e}_c exceeds a threshold e_{max}
 - govern adaption allways on the *outermost* nodes, parental nodes remain unchanged
- \rightarrow Performance increase (at N nodes): $\mathcal{O}(N) \rightarrow \mathcal{O}(\log_n N)$, typically by factor 60

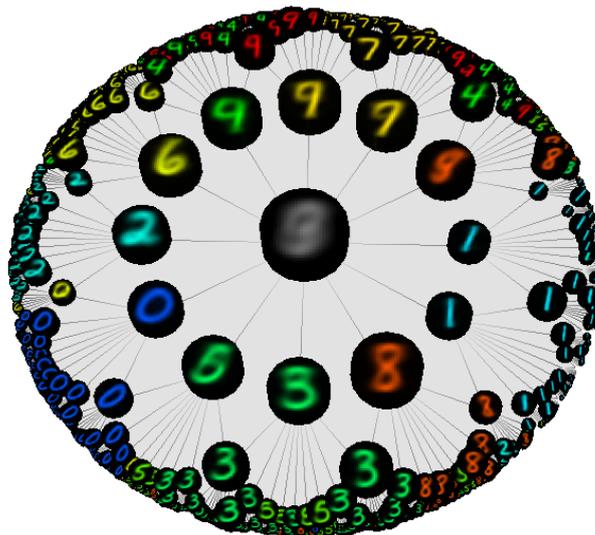


Figure 9: HSOM of MNIST-database from hand written digits. Courtesy of Jörg Ontrup.

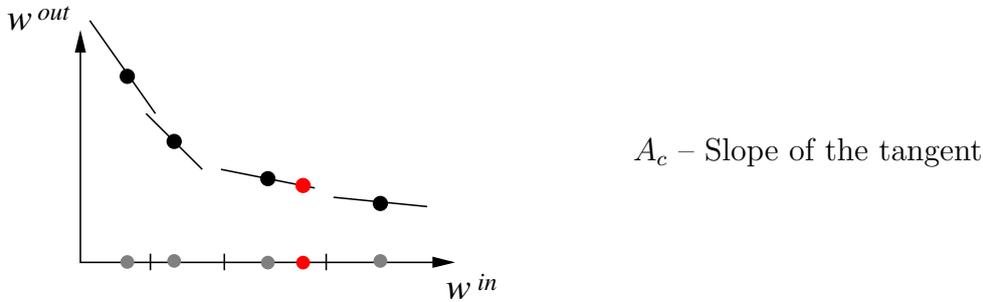
3.6 Local Linear Maps

The Local Linear Map (LLM) extends – introduced similar as in section 3.4.5 an Input-Output-Mapping – the topological learning methods SOM, GNG, ITM with an output component which is trained supervised. Additionally to the output vector \vec{w}_c^{out} in section 3.4.5, which enabled a piecewise constant mapping, we want to also learn a lokal linear mapping A_c , which leads to a piecewise linear projection (over every voronoi cell) and therefore to a better approximation of the data.

Idea: Improve approximation of the input output mapping through lokale linear mappings A_c :

$$\begin{aligned}\Phi : \mathcal{X}^{in} &\rightarrow \mathcal{X}^{out} & \vec{x} &\mapsto \vec{y}_n(\vec{x}) \\ \vec{y}_c(\vec{x}) &= \vec{w}_c^{out} + A_c \cdot (\vec{x} - \vec{w}_c^{in})\end{aligned}$$

Example: $\Phi : \mathbb{R} \rightarrow \mathbb{R}$



3.6.1 LLM learning rule

1. Learn unsupervised reference vectors \vec{w}_c^{in} – clustering in input space
Use standard SOM-, GNG- or ITM-learning rule:

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

2. Learn the output vectors supervised \vec{w}_c^{out}
and the local linear mapping 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) Error term needs target \vec{y}^α

(2) Compensate drift of w_c^{in} through moving of \vec{w}_c^{out} along the plane

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

The matrix adaptaton implements a componentwise linear error correction similar to the perceptron learning rule. If output is already correct ($\vec{y}^\alpha = \vec{y}_n(\vec{x}^\alpha)$) only the input reference vectors \vec{w}_c^{in} are moved and with them the output vectors \vec{w}_c^{out} due to the drift correction.

Convergence of learning rule To show convergence we firstly determine the change of the output $\Delta\vec{y}$ after a learning step and show afterwards that the approximation error $\vec{y}^\alpha - \vec{y}_c$ reduces.

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

Set the learning rule for Δ -expressions:

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

Now we can determine the change of the error:

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

This is the already known time discrete version of the differential equation that describes an exponential decay:

$$\Delta z = -\tau \cdot z \quad \text{mit } z = \vec{y}^\alpha - \vec{y}_c \text{ und } \tau = (\eta_{out} + \eta_A) \cdot h_{cn} > 0$$

$$\text{DGL: } \dot{z} = -\tau \cdot z$$

$$\text{with solution: } z(t) = z_0 \cdot e^{-\tau t} \rightarrow 0$$

Therefore the output error $\vec{y}^\alpha - \vec{y}_c$ converges exponentially fast towards zero.

3.6.2 Generalisation on output mixtures

The winner-takes-all-characteristic of the SOM approaches leads to discontinuities of the the input output function at the borders of the voronoi cells.

Target: Avoid discontinuities at the borders of the voronoi cells **Approach:** Mix output through *weighted superposition of all* node outputs:

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

where $g_c(\vec{x})$ is a mixture function for the weighting of the singular node contributions. Typically one chooses a soft-max weighting with gaussian mixture functions:

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

- $R(s) = e^{-\frac{1}{2}s^2}$
- Radii σ_i gives the area of influence
 - ◇ for $\sigma_i^{-1} \rightarrow \infty$ or $\sigma_i \rightarrow 0$ we would again receive winner takes all
 - ◇ for $\sigma_i \rightarrow \infty$ we receive an equal distribution: $g_i(x) = (\# \text{ Knoten})^{-1}$

Learning rule The modified learning rules now also contain the weighted factors $g_c(\vec{x})$, to update the weights according to their overall contribution:

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

The radii σ_c should approximate the expected distance of the data points within the voronoi cell V_c :

$$\begin{aligned} \Delta \sigma_n &= \eta_\sigma (\|\vec{x} - \vec{w}_n^{in}\| - \sigma_n) \quad (\text{only for winner node } n) \\ \Rightarrow \sigma_c &= \langle \|\vec{x} - \vec{w}_c^{in}\| \rangle_{V_c} \end{aligned}$$

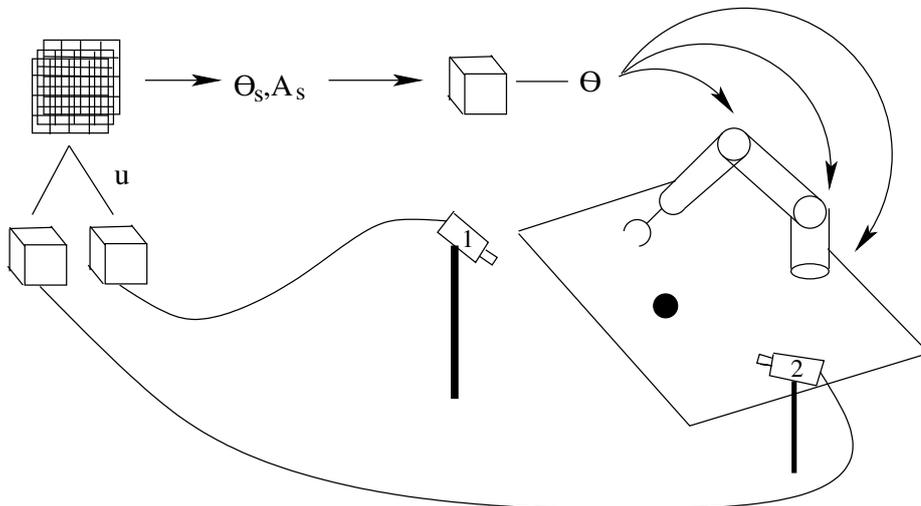
The reference vectors \vec{w}_c^{in} now also lie in the mean of the data that they come to represent:

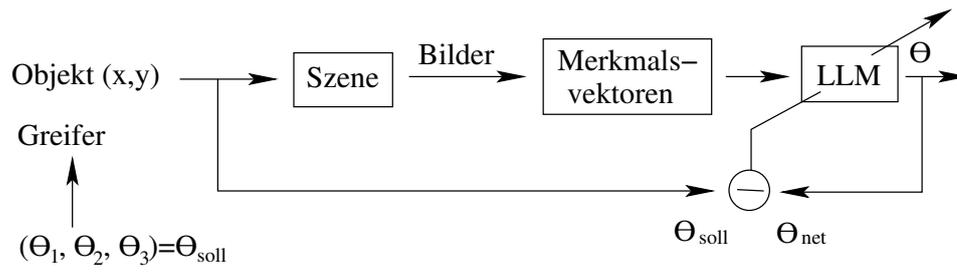
$$\begin{aligned} \langle \Delta \vec{w}_c^{in} \rangle &= 0 = \eta_{in} \langle (\vec{x} - \vec{w}_c^{in}) \cdot g_c(\vec{x}) \rangle_{\vec{x} \in V_c} \\ \Leftrightarrow \vec{w}_c^{in} &= \frac{\langle \vec{x} \cdot g_c(\vec{x}) \rangle_{V_c}}{\langle g_c(\vec{x}) \rangle_{V_c}} \end{aligned}$$

Example 3.6.1. Visual learning in robotics

Task: Learn inverse kinematics of a robot arms based on the coordinates of the images from two cameras.

- *Input:* Camera images 256×256 pixel
Preprocessing: Compute image coordinates (x,y) of the target
- *Output:* Joint angle $\vec{\theta} = (\theta_1, \theta_2, \theta_3)^t$
- *Topology:* 3D-Grid (3-dimensional space coordinates)





3.7 Prerequisite: Regression

Target: Find optimal weights for linear function approximation $\hat{y}_\alpha = \sum w_i x_i = \mathbf{x}_\alpha^t \cdot \mathbf{w}$ on given training data $(\mathbf{x}_\alpha, y_\alpha)$ – through minimization of quadratic error.

$$\text{model: } \hat{y}_\alpha^t = \sum_{i=1}^N x_i^\alpha \cdot \mathbf{w}_i = \mathbf{x}_\alpha^t \cdot \mathbf{W} \quad \hat{\mathbf{y}} = \mathbf{X} \cdot \mathbf{w} \quad \hat{\mathbf{Y}} = \mathbf{X} \cdot \mathbf{W}$$

$$\hat{\mathbf{y}} \in \mathbb{R}^{M \times 1} \quad \hat{\mathbf{Y}} \in \mathbb{R}^{M \times L}$$

$$\text{Input: } \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^t \\ \vdots \\ \mathbf{x}_M^t \end{bmatrix} = [\mathbf{x}_1, \dots, \mathbf{x}_M]^t \quad \mathbf{X} \in \mathbb{R}^{M \times N}$$

$$\text{needed: } \mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_M]^t \quad \mathbf{y} = [y_1, \dots, y_M]^t$$

$$\mathbf{W} = [\mathbf{w}_1 \dots \mathbf{w}_N]^t \quad \mathbf{w} \in \mathbb{R}^{N \times 1} \quad \mathbf{W} \in \mathbb{R}^{N \times L}$$

N – dimension of inputs \mathbf{x}_α

L – dimension of outputs \mathbf{y}_α

M – # Training examples $(\mathbf{x}_\alpha, \mathbf{y}_\alpha)$

$$\begin{array}{ll} \text{Minimize} & E(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^t \cdot (\mathbf{y} - \mathbf{X}\mathbf{w}) \quad + \quad \frac{1}{2}\lambda\|\mathbf{w}\|^2 \\ \text{Minimum at:} & \nabla_{\mathbf{w}} E \stackrel{!}{=} 0 = \mathbf{X}^t \cdot (\mathbf{y} - \mathbf{X}\mathbf{w}) \quad + \quad \lambda\mathbf{w} \\ \text{Solution:} & \mathbf{w} = (\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \cdot \mathbf{y} \quad \mathbf{w} = (\mathbf{X}^t \mathbf{X} + \lambda \mathbf{1})^{-1} \mathbf{X}^t \cdot \mathbf{y} \\ \text{multi-dimensional:} & \mathbf{W} = \underbrace{(\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \cdot \mathbf{Y}}_{\text{Pseudo-Inverse}} \quad \mathbf{W} = \underbrace{(\mathbf{X}^t \mathbf{X} + \lambda \mathbf{1})^{-1} \mathbf{X}^t \cdot \mathbf{Y}}_{\text{Ridge Regression}} \end{array}$$

3.7.1 Principal Component Regression (PCR)

Idea: Regression not directly on data X but only on most important components:

$$\text{Eigenvalue-decomposition } \underbrace{\mathbf{X}^t \mathbf{X}}_{\substack{\text{covariance} \\ \text{of the} \\ \text{data}}} = \mathbf{V} \mathbf{\Lambda}^2 \mathbf{V}^t \quad \mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_N]$$

$$\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_k]$$

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

$\mathbf{S} = \mathbf{X}'$ – Projection of \mathbf{X} on \mathbf{U}

3.7.2 Partial Least Squares (PLS)

PCR observes the direction of maximum variance of the input data \mathbf{X} . These do not have to correlate with the output \mathbf{Y} , i.e the pca analysis is contra productive.

Idea: Regression not along maximum autocorrelation of the data, but along maximal correlation between \mathbf{X} and \mathbf{Y} . In the case of one dimensional outputs the projection directions \mathbf{u} and the weights of regression (without an iterative process) can be defined easily through:

Init: $\mathbf{X}_{\text{res}} \equiv \mathbf{X}$, $\mathbf{y}_{\text{res}} \equiv \mathbf{y}$
for $i = 1 \dots r$

$$\begin{aligned} \mathbf{u} &= \mathbf{X}^t \cdot \mathbf{y} && \text{projection direction} \\ \mathbf{s} &= \mathbf{X} \cdot \mathbf{u} = \mathbf{X}\mathbf{X}^t \cdot \mathbf{y} && \text{projection} \\ \beta \equiv w &= \frac{\mathbf{s}^t \cdot \mathbf{y}}{\mathbf{s}^t \mathbf{s}} && \text{1-dim regression} \end{aligned}$$

$$\begin{aligned} \mathbf{u}_i &= \mathbf{X}_{\text{res}}^t \cdot \mathbf{y}_{\text{res}} \\ \mathbf{s}_i &= \mathbf{X}_{\text{res}} \cdot \mathbf{u}_i \\ \beta_i \equiv w_i &= \frac{\mathbf{s}_i^t \cdot \mathbf{y}_{\text{res}}}{\mathbf{s}_i^t \mathbf{s}_i} \\ \mathbf{p}_i &= \frac{\mathbf{s}_i^t \cdot \mathbf{X}_{\text{res}}}{\mathbf{s}_i^t \mathbf{s}_i} \\ \mathbf{y}_{\text{res}} &= \mathbf{y}_{\text{res}} - w_i \mathbf{s}_i \\ \mathbf{X}_{\text{res}} &= \mathbf{X}_{\text{res}} - \mathbf{s}_i \cdot \mathbf{p}_i \\ \text{oder} &= \mathbf{X}_{\text{res}} - \mathbf{s}_i \cdot \mathbf{u}_i \end{aligned}$$

The algorithm on the right side iteratively constitutes several projection directions \mathbf{u}_i on the basis of the remaining prediction error $\mathbf{y}_{\text{res}}(\mathbf{X}_{\text{res}})$. As long as der Regressionsparameter w_i is large enough compared to the preceding one w_{i-1} a new projection direction will be added.

3.8 Locally Weighted Projection Regression (LWPR)

LWPR ist a very popular and efficient learning method that strongly resembles LLMs in structure. The linear function approximations within the receptive fields will be done through incrementing PLS. A receptive field / a prototype c saves the following information:

- w_c^{in} – Center of input space
- $\beta_0^c, \dots, \beta_N^c$ – PLS regression parameter, $\beta_0^c \equiv w_c^{\text{out}}$
- $\bar{x}_c, \bar{y}_c \equiv \beta_0^c$ – weighted mean of inputs and outputs x, y

$$\begin{aligned} w_c(x) &= \exp\left(-\frac{1}{2}(x - w_c^{\text{in}})^t \cdot D_c \cdot (x - w_c^{\text{in}})\right) && \text{(mit Mahalanobis-Distanz)} \\ g_c(x) &= \frac{w_c(x)}{\sum_c w_c(x)} && \text{(soft-max)} \\ \bar{x}_c &= \sum_{\alpha} w_c(x_{\alpha}) \cdot x_{\alpha} / \sum_{\alpha} w_c(x_{\alpha}) \\ \bar{y}_c &= \sum_{\alpha} w_c(x_{\alpha}) \cdot y_{\alpha} / \sum_{\alpha} w_c(x_{\alpha}) \end{aligned}$$

- N_c – Number of data examples that have "seen" the receptive field

Die total output is again a weighted sum soft-max of partial outputs:

$$\begin{aligned} \hat{y} &= \sum_c g_c(x) \cdot \hat{y}_c(x) \\ \hat{y}_c &= \beta_0^c + \beta_1^c s_1 + \dots + \beta_r^c s_r && \text{(PLS-Regression)} \\ x_0 &= x - \bar{x}_c && \mathbf{s}_i = x_{i-1} \cdot \mathbf{u}_i && \mathbf{x}_i = x_{i-1} - \mathbf{s}_i \cdot \mathbf{p}_i \end{aligned}$$

Comment:

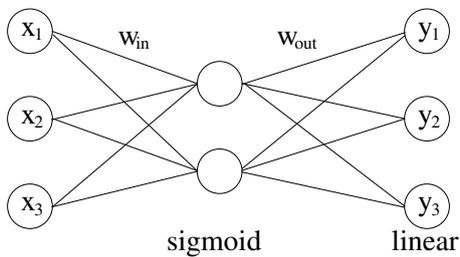
- LWPR is **incremental**. New receptive fields will be added if $\max_c w_c(x) < w_{\text{gen}}$, i.e. no receptive field which sufficiently represents the samples.

- Prototypes wont be moved: w_c^{in} keep being constant, \bar{x}_c will be updated.
- The output of a receptive field only contributes to the total output if it's statistic is reliable: $N_c > 2d$ (i.e trained with sufficient data examples).
- For every output dimension an own, *one dimensional* model is learned. In contrast to LLM the amount and position of the prototpyes is independently choosable in the different dimension.

4 Radiale Basisfunktionen (RBF)

4.1 Motivation für lokale Verfahren

Example 4.1.1. Zweischichtiges MLP mit linearer Ausgabeschicht



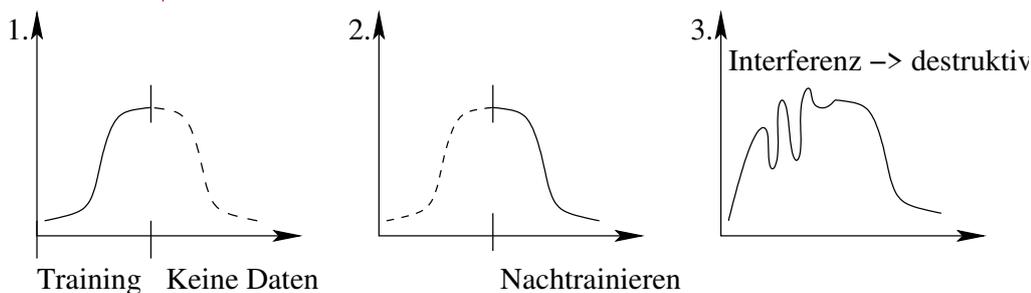
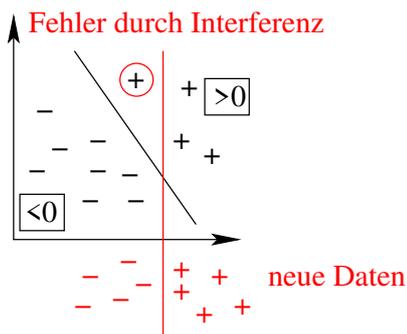
$$y_i(\mathbf{x}) = \sum_{j=1}^N w_{ij}^{out} \sigma(\mathbf{w}_j^{in} \cdot \mathbf{x})$$

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.

Example 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":

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

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 Ausgangsschicht des MLP – die gewichtete Summe der N Basisfunktionen:

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

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

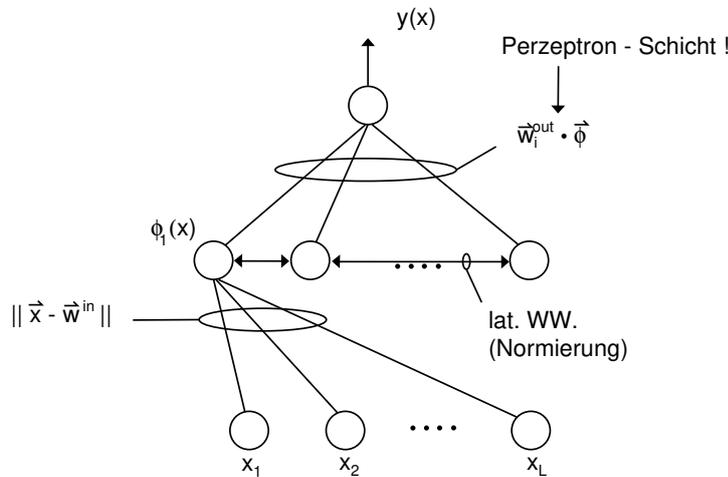


Figure 10: Netzwerkarchitektur des RBF-Ansatzes: In der mittleren Schicht bilden die Basisfunktionen ein Feature-Set, das über die Ausgangsschicht linear kombiniert wird. Der Unterschied zum MLP aus Beispiel 4.1.1 liegt in der Bestimmung der Aktivitäten bzw. Feature in der mittleren Schicht.

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 , $j = 1 \dots N$. 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!

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

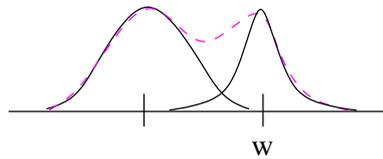
2. Bestimmung geeigneter Ausgabegewichte \mathbf{w}_{ij}^{out} $i = 1 \dots L, j = 1 \dots N$. Die Ausgabegewichte können aufgrund der einfachen Linearkombination von Basisfunktionen mittels linearer Regression direkt bestimmt werden!

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 $G_j(\mathbf{x}, \mathbf{w}^{in}) = R\left(\frac{\|\mathbf{x} - \mathbf{w}_k^{in}\|}{\sigma_k}\right)$, 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!

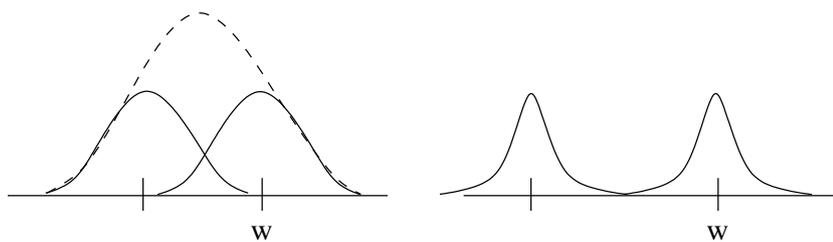


Ein einfaches Verfahren ist die "Nächste Nachbar-Heuristik":

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

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:

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

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

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

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

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

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

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

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:

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

Ziel: Minimierung des Interpolationsfehlers

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

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

- 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:

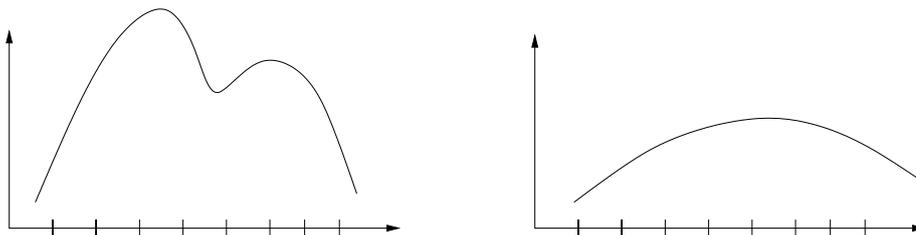
$$C_j = \text{diag}\{\sigma_{j1}^2, \sigma_{j2}^2, \dots, \sigma_{jd}^2\} \quad \text{für alle Basisfunktionen } j$$

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} :

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

- 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 Gaußlocken

⇒ 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

$$C^{(0)} \rightarrow f^{(0)} \rightarrow C^{(1)} \rightarrow f^{(1)} \rightarrow C^{(2)} \dots$$

4.4 Mathematischer Hintergrund

4.4.1 Der Regularisierungsansatz

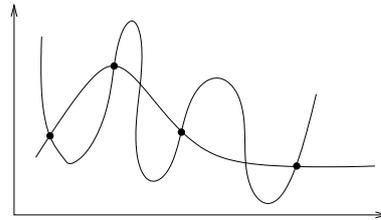
(Girosi et al, NC 1995 [?])

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

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

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

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

$$\begin{aligned} \tilde{f}(\mathbf{k}) &= \int e^{-i\mathbf{k}\mathbf{x}} \cdot f(\mathbf{x}) d^d \mathbf{x} \\ \text{Rücktransformation} \quad f(\mathbf{x}) &= \frac{1}{(2\pi)^d} \int e^{+i\mathbf{k}\mathbf{x}} \cdot \tilde{f}(\mathbf{k}) d^d \mathbf{k} \\ |\mathbf{k}| - \text{Wellenzahl} &\propto \text{Frequenz} \end{aligned}$$

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

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

mit folgenden Eigenschaften der Gewichtungsfunktion

$$\begin{aligned} \tilde{G}(\mathbf{k}) &> 0 \quad \text{und} \\ \tilde{G}(\mathbf{k}) &\rightarrow 0 \quad \text{für } |\mathbf{k}| \rightarrow \infty \text{ und} \\ \tilde{G}(\mathbf{k}) &= \tilde{G}(-\mathbf{k}) \quad (\text{Symmetrie - nötig für die Rücktransformation}) \end{aligned}$$

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

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

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

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 $f^*(\cdot)$ 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 $G(\mathbf{x}) = \left(\frac{1}{2\pi}\right)^d \int e^{i\mathbf{k}\mathbf{x}} \cdot \tilde{G}(k) d^d k$ (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 Gewichtungparameter 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 $(P)_{\beta\alpha} = P_{\beta}(\mathbf{x}^{\alpha})$.

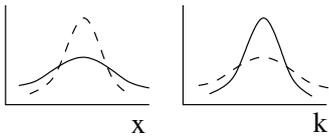
Bemerkungen

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 $\sigma = \tilde{\sigma}^{-1}$:



$$f(x) = e^{-\frac{1}{2} \frac{x^2}{\sigma^2}}$$

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

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:

$$\mathcal{F}(\nabla f)(\mathbf{k}) = -i \cdot \mathbf{k} \cdot \tilde{f}(\mathbf{k})$$

also

$$\Phi(f) = \frac{1}{2} \int |\nabla f(\mathbf{x})|^2 d^d \mathbf{x}$$

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

5 Mixture Models

Mixture models are based on the divide-and-conquer principle. They train expert network which specialize in different subtasks and combine their outputs to a combined output of the network ² We consider three different types of output mixtures:

- Ensemble Averaging (data independent linear function)
- Boosting (adaptive weighting of the training samples)
- Mixture of experts (data dependent mixture function)

Mixture models are able to reduce the training time due to its modularity and avoid overfitting.

5.1 Mixture of Experts.

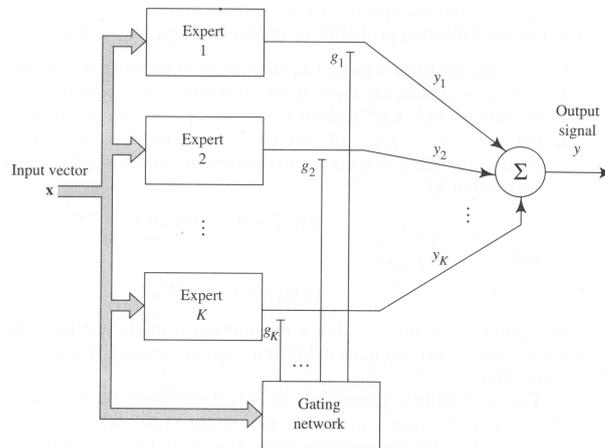


Figure 11: Mixture-of-Experts-Modell: Die Expertenausgabe wird über ein Gating-Netzwerk gewichtet aufsummiert.

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

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

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

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

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

²Note the similarity to LLM. There every neuron is an expert for it's Voronoi Cell and learns in this a part of the combined projection. Either it will be done through a WTA where the corresponding expert(=Neuron) is chosen or via Soft-Max Mixture which generates a output Mixture

Statistische Interpretation Dem Mixture-of-Experts-Ansatz liegt ein bestimmtes Datenmodell zugrunde: Die Ausgabe y zu einem Eingabevektor \mathbf{x} wird von Teilprozessen $i = 1, \dots, N$ 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:

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

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

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

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:

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

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:

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

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:

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

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:

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

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}^{α} 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:

$$\begin{aligned} -\frac{\partial E}{\partial \mathbf{w}_j} &= \sum_{\alpha} \frac{1}{R_{\alpha}} \cdot g_j^{\alpha} \cdot h_j^{\alpha} \cdot \underbrace{\frac{y^{\alpha} - f_j(\mathbf{x}^{\alpha}, \mathbf{w}_j)}{\sigma_j^2}}_{\partial h_j / \partial f_j} \cdot \frac{\partial f_j}{\partial \mathbf{w}_j} \\ &= \sum_{\alpha} \sigma_i^{-2} \cdot \underbrace{\frac{g_j^{\alpha} \cdot h_j^{\alpha}}{R_{\alpha}}}_{P(j|\mathbf{x}^{\alpha}, y^{\alpha})} \cdot \underbrace{(y^{\alpha} - f_j(\mathbf{x}^{\alpha}, \mathbf{w}_j))}_{\partial E_{\text{square}}^j / \partial \mathbf{w}_j} \cdot \frac{\partial f_j}{\partial \mathbf{w}_j} \end{aligned}$$

Wir erhalten also gerade den Gradienten bzgl. der quadratischen Energiefunktion E_{square}^j 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:

$$\begin{aligned} \Delta \mathbf{v}_i &= -\eta \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial \mathbf{v}_i} = \eta \cdot (P(i|\mathbf{x}^{\alpha}, y^{\alpha}) - P(i|\mathbf{x}^{\alpha})) \cdot \frac{\partial s_i}{\partial \mathbf{v}_i} \\ \Delta \mathbf{w}_i &= -\eta \frac{\partial E}{\partial \mathbf{w}_i} = \eta \cdot \sigma^{-2} \cdot P(i|\mathbf{x}^{\alpha}, y^{\alpha}) \cdot (y^{\alpha} - f_j(\mathbf{x}^{\alpha}, \mathbf{w}_j)) \cdot \frac{\partial f_i}{\partial \mathbf{w}_i} \end{aligned}$$

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}^{α} .

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

$$\text{Maximiere} \quad L(\Theta) = \log P(\mathbf{z}|\Theta)$$

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

$$\text{Maximiere} \quad L_{\text{complete}}(\Theta) = \log P(\tilde{\mathbf{z}}|\Theta)$$

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

EM-Algorithmus:

1. Initialisiere $t = 0$, Θ_t

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

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

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

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

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:

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

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

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

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

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

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 satisfy

$$\sum_{i=1}^N P(i) = 1, \quad 0 \leq P(i) \leq 1$$

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

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

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}^{α} 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

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

Similarly 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

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

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

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

which is the intuitive result that at the minimum the mean of the mixture component is the mean of the data weighted by the posterior probabilities that the corresponding data were generated from the respective component. Similarly, 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: $N < \infty$

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

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

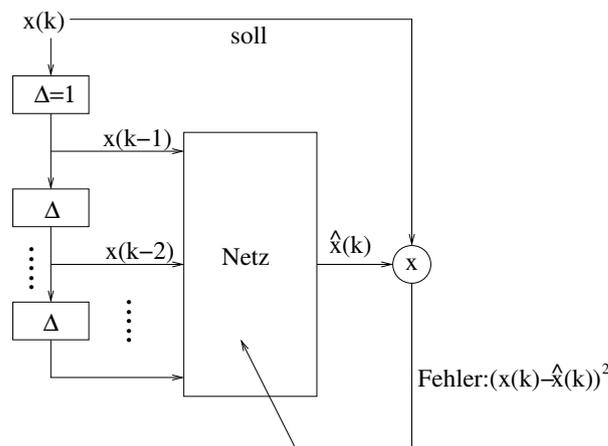
- einfachstes Modell: lineare Zeitreihen

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

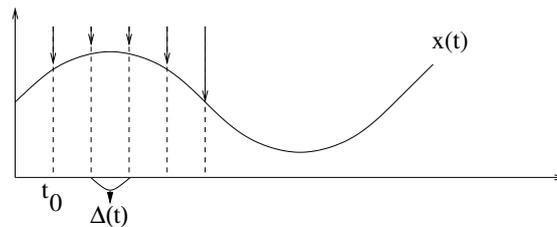
6.1 Architekturen zur Repräsentation von Kontext

6.1.1 Time-Delay-Neural-Network (TDNN)

Zeitverzögerungspuffer für die letzten N Eingaben:

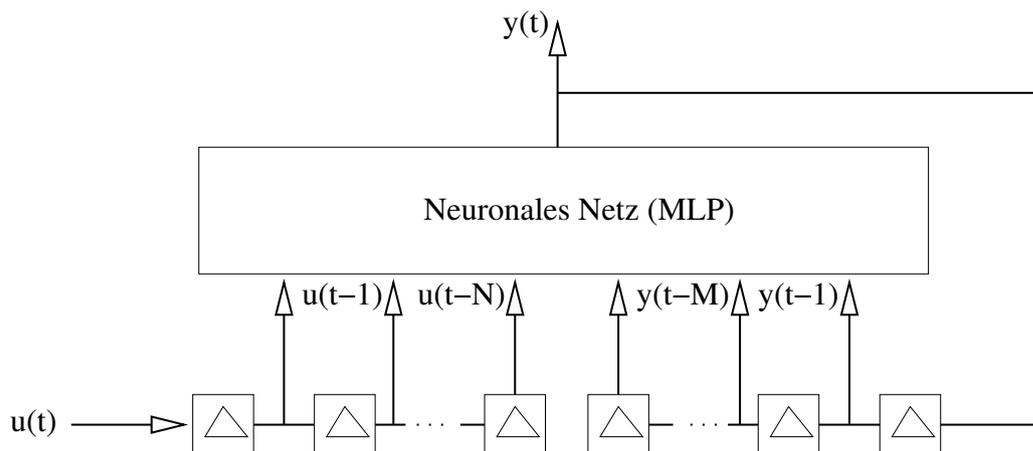


- Standard-Netz (z.B. MLP) mit Back-Prop einsetzbar
- Time-Delay-Line stellt ein Zeitfenster der Länge N dar
- optimale Breite N des Zeitfensters unbekannt (bestimmt durch Ordnung der zugrundeliegenden Funktion)
- mit N wächst Anzahl der Gewichte $O(H * N)$
- kontinuierliche Signale müssen diskretisiert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



- Vorverarbeitung und Analyse der geg. Zeitreihen notwendig

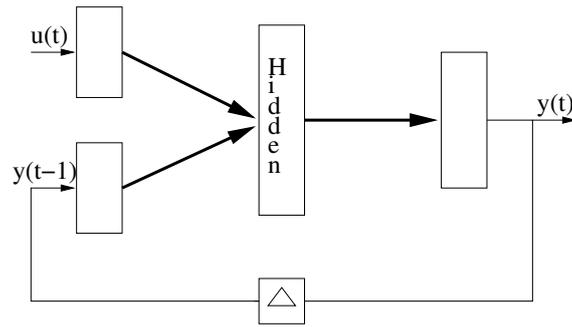
6.1.2 NARX recurrent neural networks (Nonlinear AutoRegressive with eXogenous inputs)



- 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:



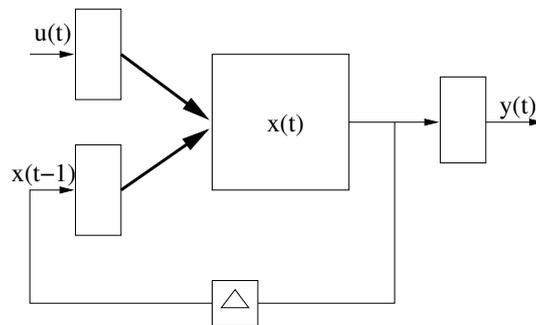
$$\begin{aligned}
 y(t) &= f(u(t), y(t-1)) \\
 &= f(u(t), f(u(t-1), y(t-2))) \\
 &= \dots
 \end{aligned}$$

- 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.

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

6.1.4 Elman-Netze (1988-1991)

- Rückkopplung der Hidden-States



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

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

$$\mathbf{x}(t) = \sigma(W^{\text{in}} u(t) + W^{\text{rek}} x(t-1))$$

$$\mathbf{y}(t) = \sigma(W^{\text{out}} x(t))$$

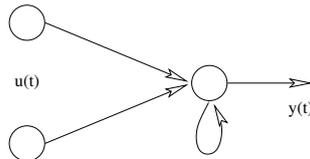
- Netz kann flexibel eine Repräsentation des Kontextes in den hidden units lernen
- vollständiges rekurrentes Netzwerk
- Vereinfachung des Lernprozesses durch Beschränkung (Truncation) auf MLP-Lernen, d.h. ignoriere rekurrente Verbindung beim Lernen. Fehler werden nicht rekursiv rückpropagiert.

6.1.5 Cascade-Corellation Network (CCN)

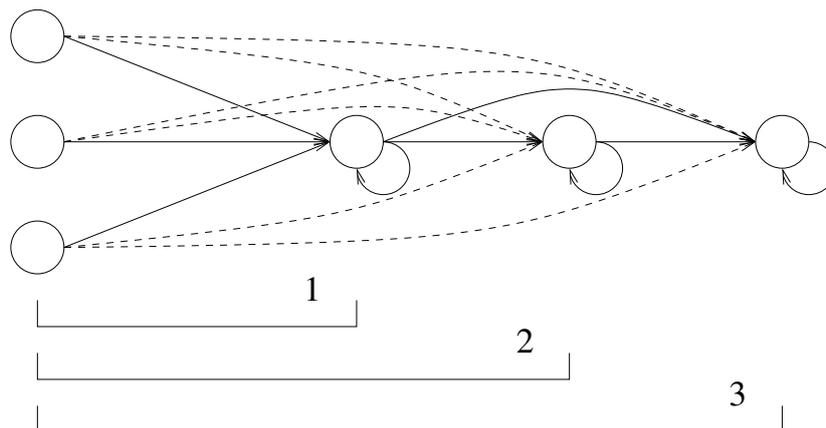
- die Wahl der Netzarchitektur bestimmt die Kapazität des Netzes
- "richtige" Architektur ist noch schwieriger als bei MLP zu finden, da rekurrente Gewichte W^{rec} hinzukommen

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".

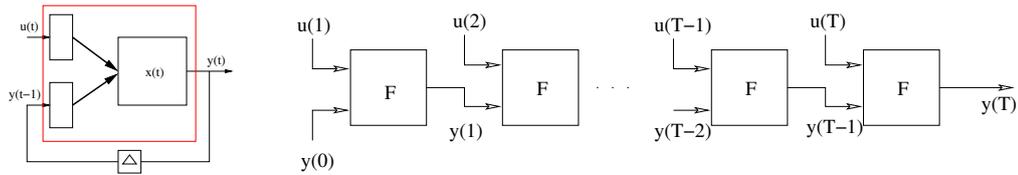
Bemerkungen:

- 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:

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

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:

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

- Mittele die Gewichtsupdates aller Schichten:

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

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

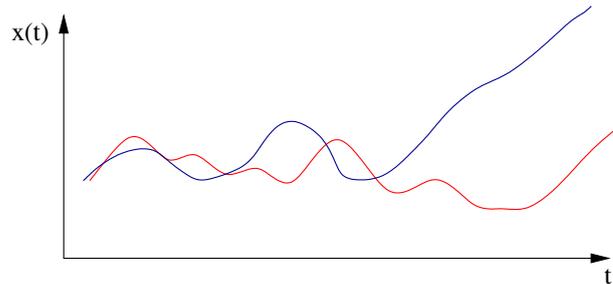


Figure 12: Stabilität eines RNN ist nicht explizit gegeben

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

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

mit den Variablen:

- \mathbf{x} - Zustandsvektor \cong Ausgabe aller Neuronen $\mathbf{x} = (x_1, \dots, x_n)^t$
- 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 : $I(t) = W_u \cdot u(t)$
- σ - sigmoide Aktivierungsfunktion

Im diskreten Fall:

$$x(t + 1) = W \cdot \sigma(x(t)) + I(t) \quad t \in \mathbb{Z} \quad (1)$$

$$y(t + 1) = \sigma(W \cdot y(t) + I(t)) \quad t \in \mathbb{Z} \quad (2)$$

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 erfordert 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).

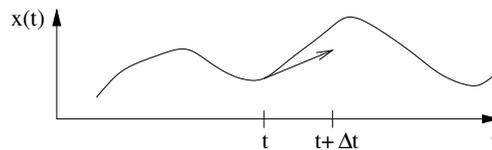


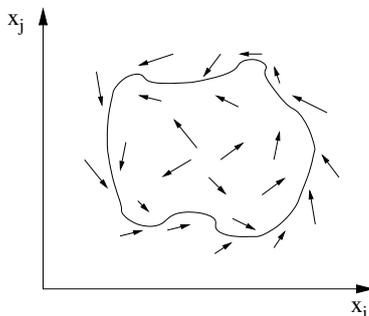
Figure 13: Zeit-Diskretisierung mittels Euler-Approximation

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

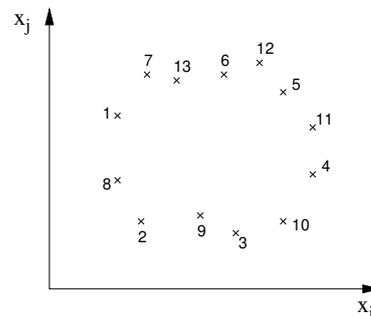
Visualisierung: Phasenplots und Vektorfelder

Phasenplot: Trage zwei Zustandskomponenten x_i und x_j gegeneinander auf.

Vektorfeld: Zusätzlich werden im Phasenplot die Geschwindigkeitsvektoren $\dot{\mathbf{x}}$ der DGL (auf einem Gitter) eingezeichnet.

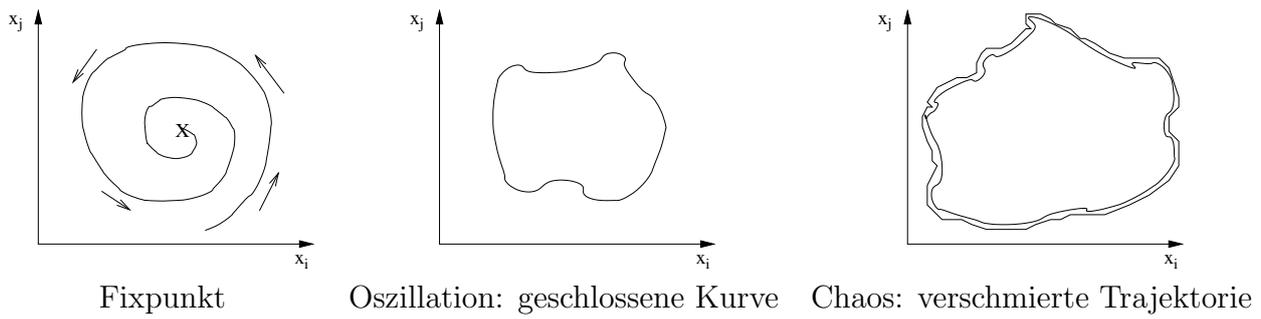


Vektorfeld eines zeit-kontinuierlichen Systems



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:



Stabilität von Fixpunkten

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

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

$$\mathbf{x}(t + 1) = W \cdot \sigma(\mathbf{x}) + I = F(\mathbf{x}, I)$$

Die Fixpunktbedingungen lauten demnach:

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

Fixpunkte werden nach ihrer lokalen und globalen Stabilität klassifiziert.

Lokal: Lokale Linearisierung

Global: Lyapunov-Funktion

Definition Stabilität:

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

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

”‘Eine kleine Störung (um δ) bleibt klein (um ε).”

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

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

- (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.

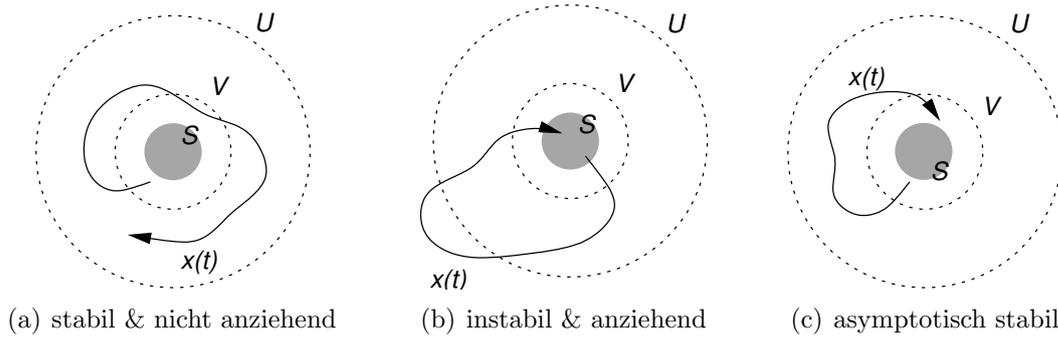


Figure 14: Stabilität eines Fixpunktes S .

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} .

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

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

Mittels Variablentransformation $z = \Delta x = x - \bar{x}$ gilt:

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

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

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

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

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

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

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

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

Um uns die zeitliche Entwicklung einer Trajektorie anzuschauen, die bei z_0 startet, entwickeln wir z_0 nach der Eigenvektor-Basis U :

$$z^0 = \sum_{i=1}^n a_i^0 \cdot \mathbf{v}_i \quad \text{bzw. in Vektornotation:} \quad z_0 = U \mathbf{a}_0$$

und erhalten für die kontinuierliche bzw. diskrete Dynamik:

$$\begin{aligned} \mathbf{z}(t) &= U e^{\Lambda t} U^{-1} U \mathbf{a}_0 = U e^{\Lambda t} \mathbf{a}_0 = U \mathbf{a}(t) \\ \mathbf{z}(t) &= U \Lambda^t U^{-1} U \mathbf{a}_0 = U \Lambda^t \mathbf{a}_0 = U \mathbf{a}(t) \end{aligned}$$

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

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

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

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

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

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

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

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

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 $J(\bar{\mathbf{x}})$ diese Grenze überschreitet, unterscheidet man folgende Fixpunkt-Bifurkationen:

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

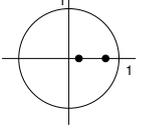
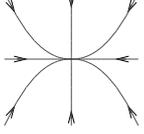
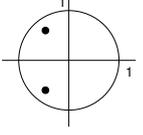
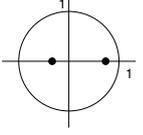
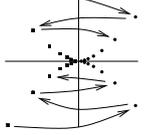
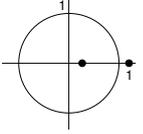
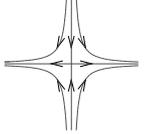
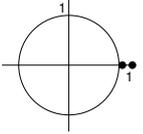
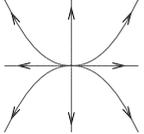
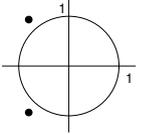
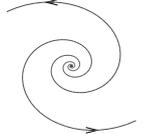
stability	eigenvalues	phase portrait	J	$\ln J$
stable		 node	$\begin{pmatrix} \lambda_1 & \\ & \lambda_2 \end{pmatrix}$ $0 < \lambda_{1,2} < 1$	$\begin{pmatrix} \ln \lambda_1 & \\ & \ln \lambda_2 \end{pmatrix}$ $\ln \lambda_{1,2} < 0$
		 focus	$\lambda \begin{pmatrix} \cos \omega & \sin \omega \\ -\sin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

Table 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 $V(\mathbf{x}(t))$ heißt Lyapunov-Funktion innerhalb einer Umgebung $U(\bar{\mathbf{x}}) \in X$ falls

- (i) $\dot{V}(\mathbf{x}(t)) = \frac{d}{dt}V(\mathbf{x}(t))$ existiert und ist stetig entlang der Trajektorien $\mathbf{x}(t) \in U$,
- (ii) $V(\mathbf{x}(t))$ ist positiv definit auf U : $V(\mathbf{x}(t)) > 0$ für alle $\mathbf{x}(t) \neq \bar{\mathbf{x}}$ und $V(\bar{\mathbf{x}}) = 0$
- (iii) $\dot{V}(\mathbf{x}(t)) \leq 0$ entlang jeder Trajektorie $\mathbf{x}(t) \in U$.

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

Satz: Ein Fixpunkt $\bar{\mathbf{x}}$ ist stabil, falls eine Lyapunov-Funktion in einer Umgebung $U(\bar{\mathbf{x}})$ 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.

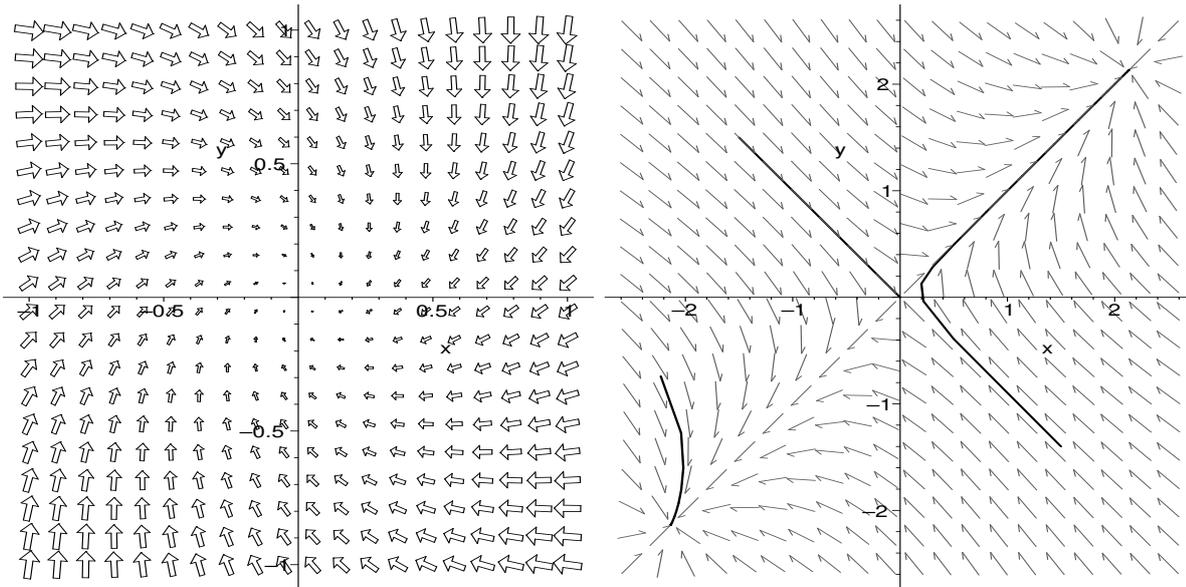


Figure 15: Vektorfeld einer zwei-dimensionalen Hopfield-Dynamik mit einem anziehenden Fixpunkt (links) bzw. zwei anziehenden Fixpunkten (rechts). Im rechten Phasendiagramm sind drei exemplarische Trajektorien gezeigt.

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:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= -\eta \nabla_{\mathbf{x}} E(\mathbf{x}(t)) \\ \dot{E}(\mathbf{x}(t)) &= \nabla_{\mathbf{x}} E \cdot \dot{\mathbf{x}}(t) = -\eta (\nabla_{\mathbf{x}} E)^2 < 0 \end{aligned}$$

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

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

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

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

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

$$\begin{aligned} \frac{\partial}{\partial y_k} \left(\sum_i y_i \sum_j w_{ij} y_j \right) &= \sum_j w_{kj} y_j + \sum_i y_i w_{ik} = \sum_j y_j \overbrace{(w_{kj} + w_{jk})}^{2w_{kj}} \\ \Rightarrow \frac{\partial E}{\partial y_k} &= - \sum_j w_{kj} \cdot y_j + \sigma^{-1}(y_k) - I_k \end{aligned}$$

Damit können wir nun \dot{E} bestimmen:

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

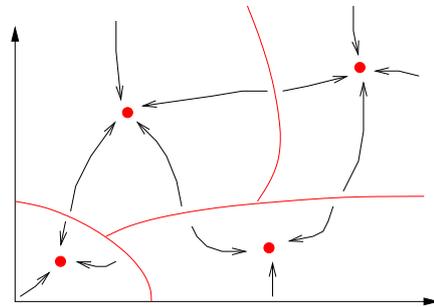
Für eine monoton wachsende Aktivierung σ , fällt $E(t)$ damit monoton entlang einer Trajektorie. Damit konvergiert die Hopfield-Dynamik immer zu einem Fixpunkt.

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 veräuschter Eingabe korrekt abrufbar, denn die Attraktorbecken der Fixpunkte sind typischerweise zusammenhängend.



Diskretes Hopfield-Modell:

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

Update:

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

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

$$\begin{aligned} \Rightarrow W &= \frac{1}{n} \cdot \sum_{\alpha=1}^m p_\alpha^T \cdot p_\alpha \\ w_{ij} &= \frac{1}{n} \cdot \sum_{\alpha=1}^m p_i^\alpha \cdot p_j^\alpha \end{aligned}$$

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

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

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^β aufgrund der sgn-Funktion erhalten. Ansonsten flippt es offenbar. Für orthogonale Muster verschwindet der Cross-Talk-Term, denn:

$$\sum_j p_j^\alpha \cdot p_j^\beta = 0 \quad \forall \alpha \neq \beta$$

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):
 $W(p^\beta + p^\gamma + p^\delta) = p^\beta + p^\gamma + p^\delta + \underbrace{C^\beta + C^\gamma + C^\delta}_{C^{\beta\gamma\delta}}$

Bemerkungen:

- 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

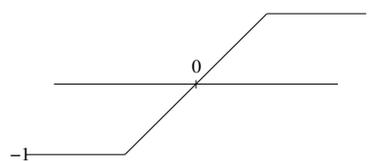
- Nehme an, $p_j^\alpha \in \{-1, 1\}$ seien statistisch unabhängig.
- Cross-talk-Term wird zu Binomialverteilung mit Mittelwert $\mu = 0$ und Varianz $\sigma^2 = M/N = \alpha$.
- Im Limes großer Netzwerke ($n \rightarrow \infty$) wirkt der Zentrale Grenzwertsatz.
- Die Binomialverteilung kann durch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

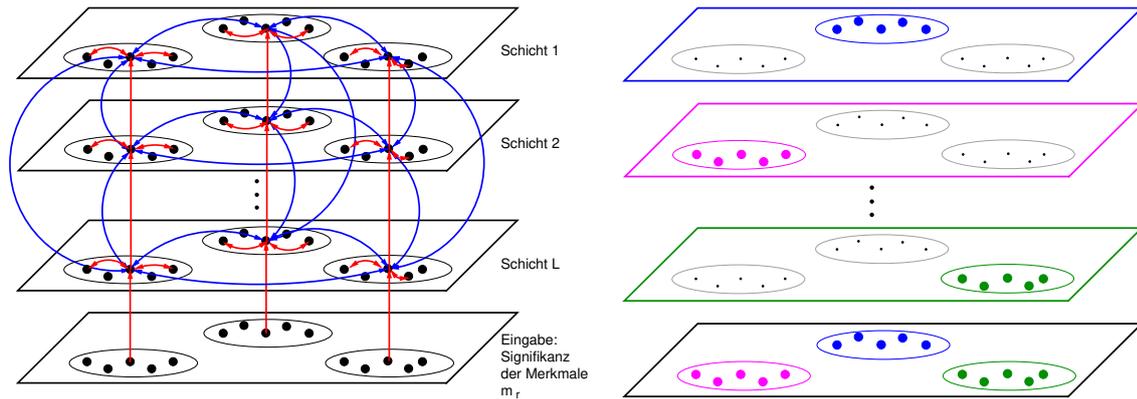


Figure 16: Schichtweiser Aufbau des CLM: Die Eingabeschicht versorgt jedes Neuron in einer Spalte mit der konstanten Eingabe h_r , die als Signifikanz des Merkmals \mathbf{m}_r interpretiert werden kann. Innerhalb jeder Schicht wird über die Kompatibilität der Merkmale eine laterale Wechselwirkung definiert, 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:

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

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:

$$E = - \sum_{r\alpha} Jh_r x_{r\alpha} + \frac{1}{2} \sum_{r\alpha\beta} J x_{r\alpha} x_{r\beta} - \frac{1}{2} \sum_{\alpha r r'} f_{rr'} x_{r\alpha} x_{r'\alpha} \quad (7.4)$$

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:

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

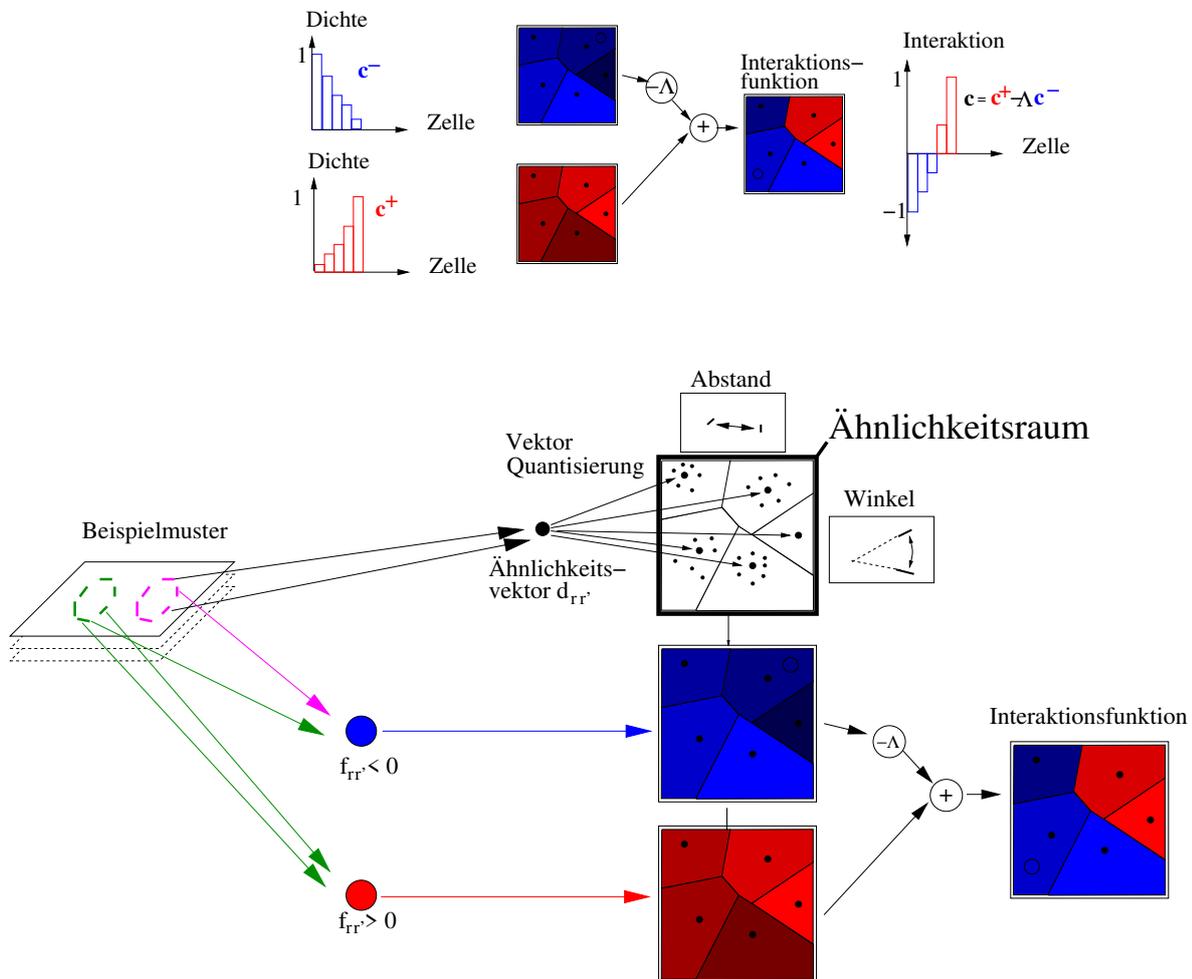


Figure 17: Zusammenfassung CLM-Lernen

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 $\{\mathbf{u}(t)^\alpha, d(t)^\alpha\}_{t=0 \dots T-1}$
diskretes Netzwerkmodell:

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

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

$$\begin{aligned} \mathbf{x} &= [\mathbf{y}, \mathbf{u}] & W &= [W_{rec}, W_u] \\ \Rightarrow \mathbf{x}(t+1) &= \sigma(W \mathbf{x}(t)) \end{aligned}$$

Fehlerfunktion:

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

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

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

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

$$p_{ij}^l(t+1) \equiv \frac{\partial x_l(t+1)}{\partial w_{ij}} \quad - \text{Empfindlichkeit des Neurons } l \text{ auf Änderung des Gewichts } w_{ij}$$

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

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

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

8.3 Virtual Teacher Forcing

Atiya & Parlos, 1990:

Vereinigender Ansatz zur Herleitung verschiedener rekurrenter Lernverfahren.

Lernen als allg. Optimierungsproblem: Minimiere $E(W)$ unter Nebenbedingungen $\mathbf{g} \equiv 0$.

Die Nebenbedingungen \mathbf{g} achten dabei auf die Einhaltung der Dynamikgleichungen:

$$\mathbf{g}(t+1) \equiv -\mathbf{x}(t+1) + W\sigma(\mathbf{x}(t)) = 0 \quad t = 0, \dots, T-1$$

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

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

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

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

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

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

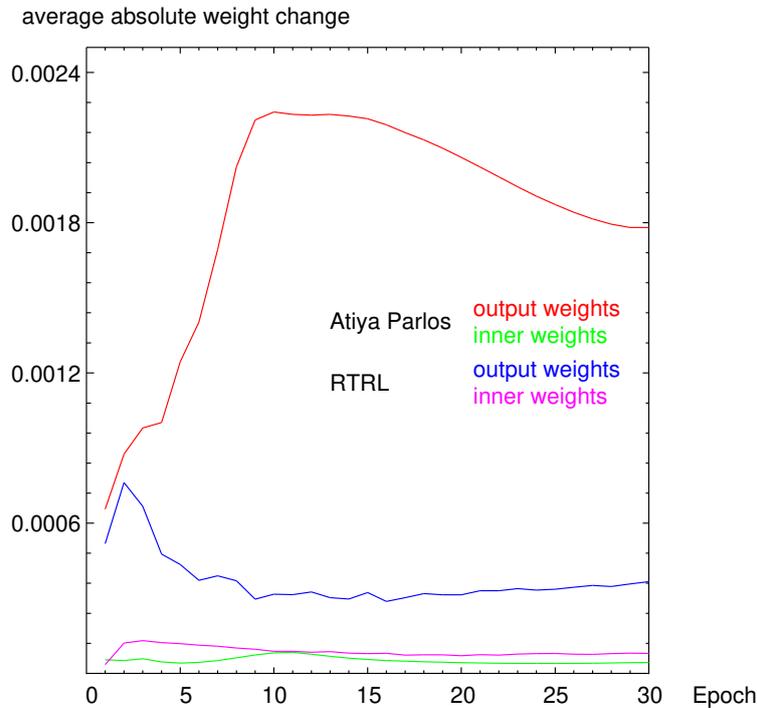
Diese Technik heißt *virtual teacher forcing*, weil die (vom Lehrer) angestrebten Zielzustände $\mathbf{x} + \Delta \mathbf{x}$ die eigentlichen Gewichtsänderungen bewirken aber niemals wirklich in das Netz eingespeist werden.

Dieses unterbestimmte LGS ($\frac{\partial \mathbf{g}}{\partial \mathbf{w}} \in \mathbb{R}^{nT \times n^2}$) wird mittels der Pseudo-Inversen gelöst:

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

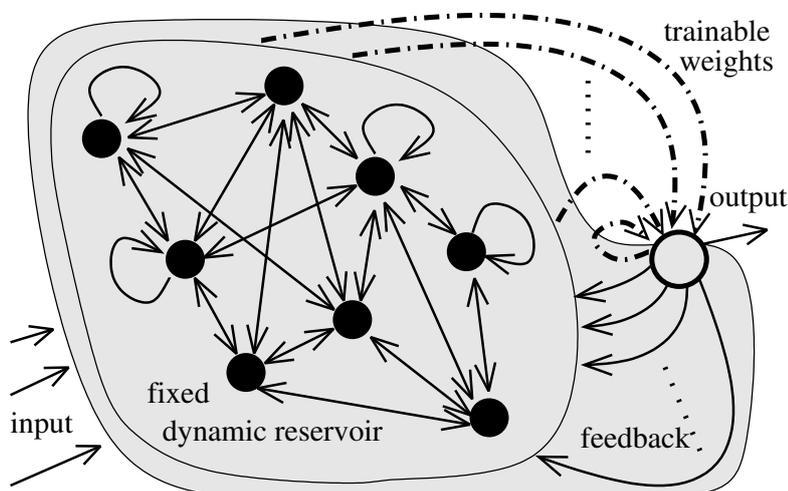
Bemerkungen zum Atiya-Parlos-Lernen

- entspricht nicht exaktem Gradienten $\frac{\partial E}{\partial w}$
- 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 $\mathbf{u}(t)$ – ”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

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

where $x_i, i=1, \dots, N$ are the states, $\mathbf{W} \in \mathbb{R}^{N \times N}$ is the weight matrix, \mathbf{W}_u the input weight matrix and $k = \hat{k}\Delta t, \hat{k} \in N_+$ is a discretized time variable³. For small Δt we obtain an approximation of the continuous time dynamics $d\mathbf{x}/dt = -\mathbf{x} + \mathbf{W}\mathbf{f}(\mathbf{x})$ and for $\Delta t = 1$ the standard discrete dynamics. We assume that \mathbf{f} is a standard sigmoidal differentiable activation function with $\mathbf{f}' \leq 1$ and is applied component wise to the vector \mathbf{x} . We further assume that \mathbf{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 $O \subset \{1, \dots, N\}$ 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

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

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

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 $y_s(k)$.

³With 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 $\Delta w_{ij}^{APbatch}$ (second term in (8.1)); (iii) by replacing the full autocorrelation matrix C_k by the instantaneous autocorrelation matrix $C(k) = \epsilon I + \mathbf{f}_k \mathbf{f}_k^T$ in the first term and use the small rank adjustment matrix inversion lemma

$$\begin{aligned} C(k)^{-1} \mathbf{f}_k &= \left[\frac{1}{\epsilon} \mathbf{1} - \frac{[\epsilon^{-1} \mathbf{f}_k][\epsilon^{-1} \mathbf{f}_k]^T}{1 + \epsilon^{-1} \mathbf{f}_k^T \mathbf{f}_k} \right] \mathbf{f}_k = \mathbf{f}_k \left[\frac{1}{\epsilon} - \frac{1}{\epsilon^2} \frac{\mathbf{f}_k^T \mathbf{f}_k}{1 + \frac{1}{\epsilon} \|\mathbf{f}_k\|^2} \right] = \frac{\mathbf{f}_k}{\sum \mathbf{f}(x_s(k))^2 + \epsilon} \\ &\Rightarrow \Delta w_{ij}^{BPDC}(k+1) = \frac{\eta}{\Delta t} [C(k)^{-1} \mathbf{f}_k]_j \gamma_i(k+1) \end{aligned} \quad (8.5)$$

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 :

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

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

$$\mathbf{y} = \mathbf{G}_I(\mathbf{W}_u \mathbf{u} + \mathbf{W} \Phi(\mathbf{y})) = \mathbf{G}_I \mathbf{W}_u \mathbf{u} + \mathbf{G} \Phi(\mathbf{y}), \quad (\mathbf{G} = \mathbf{G}_I \mathbf{W}).$$

The network acts as nonlinear feedback system implementing a loop operator $\mathbf{H} \doteq ((\mathbf{G}_I, \mathbf{W}), \Phi)$ which transforms L_2^n signals⁵ $\mathbf{W}_u \mathbf{u}$ into L_2 output signals \mathbf{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 $\mathbf{u} \equiv 0$), if the operator gains $\gamma(\mathbf{G}_I), \gamma(\mathbf{G}) = \gamma(\mathbf{G}_I \mathbf{W})$ and $\gamma(\Phi)$ are finite and the

$$\text{small gain condition:} \quad \gamma(\mathbf{G})\gamma(\Phi) < 1 \quad (8.6)$$

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

$$\gamma(\mathbf{H}) \leq \frac{\gamma(\mathbf{G}_I)}{1 - \gamma(\mathbf{G})\gamma(\Phi)} \quad \text{where} \quad \gamma(\mathbf{H}) = \sup_{\mathbf{u}'} \frac{\|\mathbf{H}(\mathbf{u}')\|_2}{\|\mathbf{u}'\|_2} = \|\mathbf{H}\|_2 \quad (8.7)$$

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

⁴We 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).

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

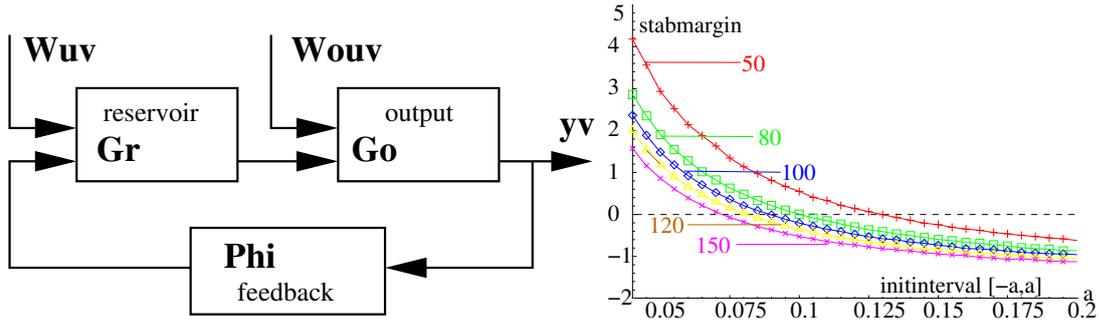


Figure 18: Left: System composed of inner reservoir subsystem, output subsystem and feedback. Right: Stability margin vs. weight initialization intervals different network sizes (averaged over 50 network initializations.)

neurons. Then $((G_I, W_r^r), \Phi)$ denotes the reservoir subsystem while $((G_I, w_{11}), \Phi)$ 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 $\gamma(G_I) = \gamma(\Phi) = 1$ and thus $\gamma(G) = \gamma(G_I W) = \|W\|$ we obtain for the subsystems the inequalities $\|W_r^r\| < 1$ and $w_{11} < 1$ and, as proved in the Appendix,

$$\|\mathbf{w}_{x_r}^o\| < (1 - \|W_r^r\|)(1 - |w_{11}|)\|\mathbf{w}_o^{x_r}\|^{-1} \quad (8.8)$$

for the overall network. Here $\mathbf{w}_{x_r}^o$ is the vector of trainable weights and $\mathbf{w}_o^{x_r}$ 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 $\|\mathbf{w}_{x_r}^o\|$ 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 $W^+ := \lambda W$ obeys the stability constraint. Given the current $\|\mathbf{w}_{x_r}^o\|$ not fulfilling (8.8), solve

$$\lambda \|\mathbf{w}_{x_r}^o\| = \frac{(1 - \lambda \|W_r^r\|)(1 - \lambda |w_{11}|)}{\lambda \|\mathbf{w}_o^{x_r}\|^{-1}} \quad (8.9)$$

for λ to exactly obey the stability condition with the scaled matrix λW . This is a quadratic form in λ and allows for the closed form solution

$$\lambda = -1/2 \frac{\|W_r^r\| + |w_{11}| \pm \sqrt{\|W_r^r\|^2 - 2|w_{11}|\|W_r^r\| + |w_{11}|^2 + 4\|\mathbf{w}_{x_r}^o\|\|\mathbf{w}_o^{x_r}\|}}{\|\mathbf{w}_{x_r}^o\|\|\mathbf{w}_o^{x_r}\| - |w_{11}|\|W_r^r\|}, \quad (8.10)$$

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:

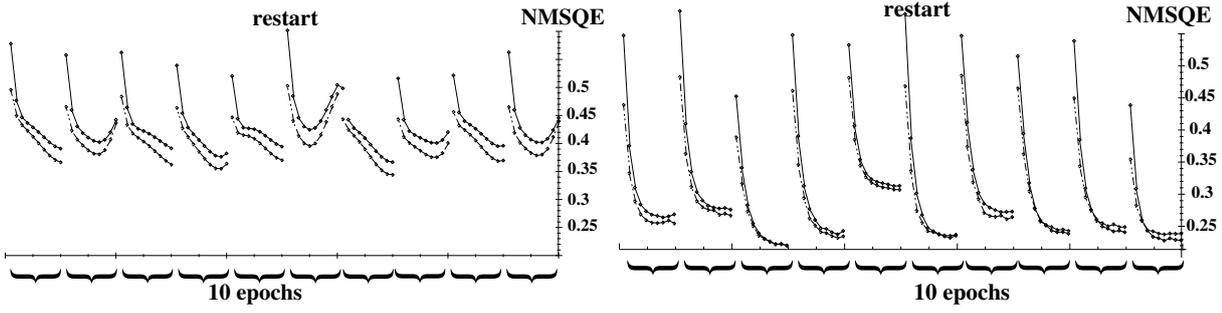


Figure 19: Left: Train and test errors for fixed stable initialization. Right Train and test errors for identical parameters, but stable online rescaling. (Ten runs of ten epochs learning of laser data with 50 nodes, 20 inputs, learning rate 0.03, 1000 points training, 1000 points test data).

- 0) initialize network with weights in $[-a, a]$, compute $\|\mathbf{W}_r^r\|$, $\|\mathbf{w}_o^{x_r}\|$, $|w_{11}|$;
- 1) check stability condition (8.8) for $\|\mathbf{w}_{x_r}^o\|$;
- 2) if (8.8) does not hold
 - ◇ compute scaling factor λ according to (8.10),
 - ◇ rescale weight matrix $\mathbf{W}^+ := \lambda \mathbf{W}$,
 - ◇ rescale $\|\mathbf{W}_r^r\|^+ := \lambda \|\mathbf{W}_r^r\|$, $\|\mathbf{w}_o^{x_r}\|^+ := \lambda \|\mathbf{w}_o^{x_r}\|$, $\|w_{11}\|^+ := \lambda \|w_{11}\|$,
 - ◇ replace the norms by the updated $\|\cdot\|^+$ values in (8.8);
- 3) iterate network;
- 4) apply learn step $\Delta \mathbf{w}^{BPDC}$;
- 5) goto 1) (unless stopping criterion is met).