# Promotionskolloquium:

# Reinforcement Lernen mit Regularisierungsnetzen

Tobias Jung

**Betreuer:**

**Prof. Dr. Thomas Uthmann**

**Prof. Dr. Elmar Schömer**

**Dr. Daniel Polani**

Fachbereich Physik, Mathematik & Informatik

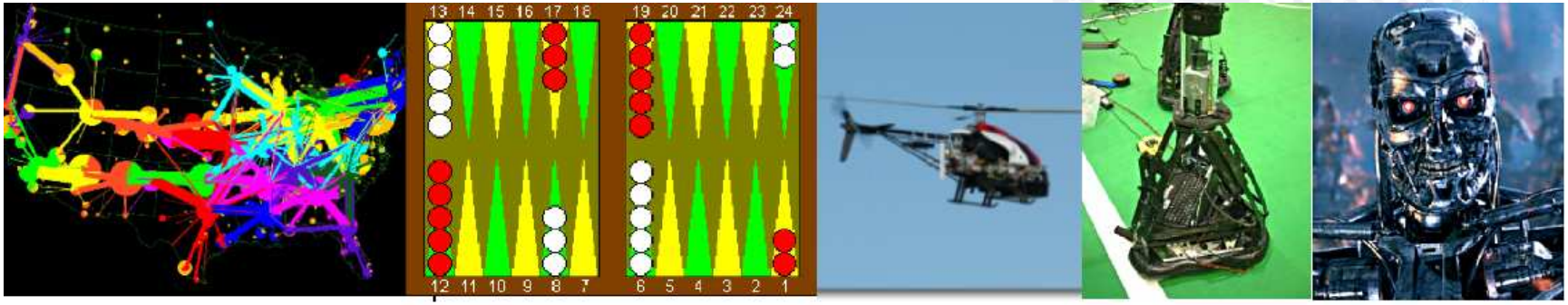
Johannes Gutenberg-Universität Mainz

# Worum geht es?

## Ein universelles Problem:

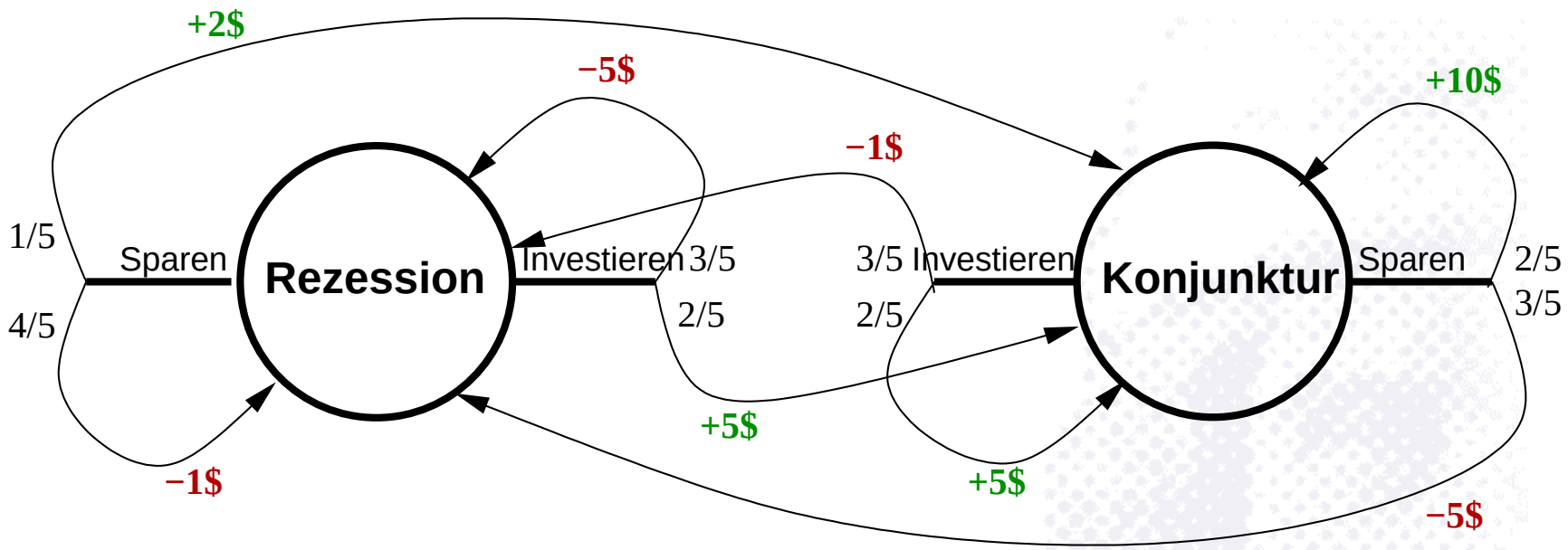
- Wie können wir uns optimal in einer komplexen Umwelt verhalten?
  (Zeitliche Abfolge von Entscheidungen, ungewisse Auswirkungen, langfristige Konsequenzen ...)
- Wie kann das ein Computer automatisch tun? (⇐ Kernfrage von KI!!)

## Wofür braucht man das?



# Teil I:
# Reinforcement Lernen

# Ein einfaches Beispiel




**Bestandteile des Systems:** –

- diskret in der Zeit $t = 0, 1, 2, 3 \ldots$
- Zustände $\mathcal{S} = \{$'Rezession','Konjunktur'$\}$, Aktionen $\mathcal{A} = \{$'Investieren','Sparen'$\}$.
- Übergangswahrscheinlichkeiten $P(s_{t+1}|s_t, a_t)$ (Markov)
- 1-Schritt Auszahlungen $R(s_{t+1}, s_t, a_t)$ ("Belohnung","Kosten")

**Ziel:** maximiere **Gesamtgewinn**!

# Dynamisches Programmieren

- Wir betrachten eine **Politik** $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (deterministisch).
- Zu jeder Politik $\pi$ die **Wertefunktion** (mit $\gamma \in (0, 1)$ Diskontfaktor)

$$\forall s : \qquad V^{\pi}(s) := \mathbb{E}\{\sum_{t \geq 0} \gamma^t R(s_{t+1}, s_t, a_t) | s_0, a_i = \pi(s_i)\}$$

**Ziel:** gesucht ist $\pi^* := \text{argmax}_{\pi} V^{\pi}$, die **optimale** Politik.

**Wie kann man das erreichen?**

1. Es gilt die **Bellman Gleichung**:

$$\forall s : \qquad V^{\pi}(s) = \mathbb{E}_{s'|s,\pi(s)}\{R(s', s, \pi(s)) + \gamma V^{\pi}(s')\}$$

2. Es gilt das **Politikverbesserungs-Theorem**: gegeben $\pi_k$, $V^{\pi_k}$. Dann gilt: für

$$\pi_{k+1}(s) := \underset{a}{\text{argmax}}\, \mathbb{E}_{s'|s,a}\{R(s', s, a) + \gamma V^{\pi_k}(s')\}$$

ist $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$, $\forall s$. Gilt Gleichheit $\forall s$, dann $\pi_{k+1} = \pi^*$ optimal.

$\Longrightarrow$ **Lösungsalgorithmus:** Politik-Iteration (Howard 1960). Alternativ: direkte Werte-Iteration (Bellman 1957).

# Notation

Zur besseren Übersicht drücken wir das kompakt mit Matrizen/Vektoren aus:

**Definiere:**

- Zustandsraum $\mathcal{S} = \{1, \ldots, N\}$
- $\mathbf{v}$, ein $N \times 1$ Vektor
- $\mathbf{P}^\pi$, eine $N \times N$ Matrix, mit $[\mathbf{P}^\pi]_{ij} = P(s' = j | s = i, a = \pi(s))$
- $\mathbf{R}^\pi$, ein $N \times 1$ Vektor, mit $[\mathbf{R}^\pi]_i = \sum_{s'} P(s'|i, \pi(i)) R(s', i, \pi(i))$
- und schließlich

$$\mathbf{v}^\pi := \begin{bmatrix} V^\pi(1) \\ \vdots \\ V^\pi(N) \end{bmatrix}$$

$\Longrightarrow$ Bellman-Gleichung in Matrix-Schreibweise: $\mathbf{v}^\pi$ ist Lösung von

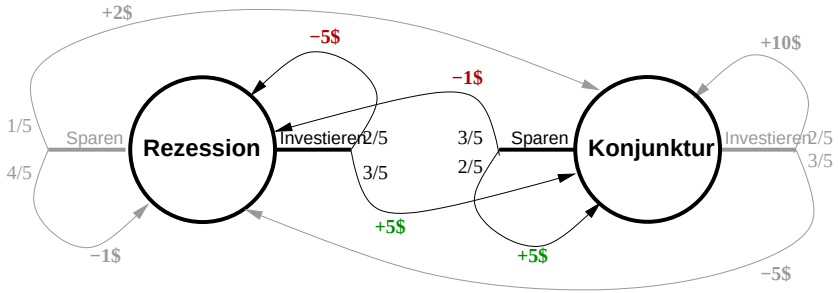
$$\mathbf{v} = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{v} \qquad \text{bzw.} \qquad (\mathbf{I} - \gamma \mathbf{P}^\pi)\mathbf{v} = \mathbf{R}^\pi$$

Wie sieht das nun konkret im Beispiel aus ...

# Beispiel (Forts.)

**Politik-Iteration:** starten mit $\pi_0 = \{\text{Falls Rez. dann Inv., Falls Konj. dann Spar.}\}$

**1. Politik-Evaluation für $\pi_0$** $(\gamma = 0.99)$




$$\mathbf{P}^{\pi 0} = \begin{bmatrix} 0.4 & 0.6 \\ 0.4 & 0.6 \end{bmatrix}, \mathbf{R}^{\pi 0} = \begin{bmatrix} 0.4 * (-5) + 0.6 * 5 \\ 0.4 * 5 + 0.6 * (-1) \end{bmatrix} = \begin{bmatrix} 1 \\ 2.6 \end{bmatrix}$$

$$\Rightarrow \mathbf{v}^{\pi 0} = \left[ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - 0.99 \begin{bmatrix} 0.4 & 0.6 \\ 0.4 & 0.6 \end{bmatrix} \right]^{-1} \begin{bmatrix} 1 \\ 2.6 \end{bmatrix} = \begin{bmatrix} 195.1 \\ 196.6 \end{bmatrix}$$

**2. Politik-Verbesserung:** Gierige Politik-Ableitung aus $V^{\pi 0}$

$$\begin{aligned} \pi_1(\text{Rez}) &:= \operatorname{argmax}\{P(\text{Rez}|\text{Rez}, \text{spa}) \; [R(\text{Rez}, \text{Rez}, \text{spa}) + 0.99 V^{\pi 0}(\text{Rez})] \\ &\quad + P(\text{Kon}|\text{Rez}, \text{spa}) \; [R(\text{Kon}, \text{Rez}, \text{spa}) + 0.99 V^{\pi 0}(\text{Kon})]; \\ &\quad P(\text{Rez}|\text{Rez}, \text{inv}) \; [R(\text{Rez}, \text{Rez}, \text{inv}) + 0.99 V^{\pi 0}(\text{Rez})] \\ &\quad + P(\text{Kon}|\text{Rez}, \text{inv}) \; [R(\text{Kon}, \text{Rez}, \text{inv}) + 0.99 V^{\pi 0}(\text{Kon})]\} \\ &= \operatorname{argmax}\{193.01; 195.04\} = \text{inv} \\ \pi_1(\text{Kon}) &:= \ldots = \operatorname{argmax}\{196.64; 194.72\} = \text{spa} \end{aligned}$$

Somit $\pi_1 = \pi_0$, also $V^{\pi 1} = V^{\pi 0}$ und damit optimal. Fertig.

# Ja, aber ...

In diesem formellen Rahmen ließe sich eine Menge von Aufgaben unterbringen und optimal lösen, z.B. Schach

Zustände=Brettposition, Aktionen=mögliche Züge, Zustandsübergang=Zug des Gegners, Belohnung={1 Sieg, -1 Niederlage, 0 sonst}

**Aber:**

- Politik-Evaluation: $\forall s: \; V^{\pi}(s) = \mathbb{E}_{s'|s,\pi(s)}\{R(s', s, \pi(s)) + \gamma V^{\pi}(s')\}$
- Politik-Verbesserung: $\forall s: \; \pi_{k+1}(s) := \text{argmax}_a \mathbb{E}_{s'|s,a}\{R(s', s, a) + \gamma V^{\pi_k}(s')\}$

**Probleme:** Zustandsraum groß? Dynamik $P(s'|s, a)$ unbekannt? (Aktionsraum groß?)

**Ansatz:** simulationsbasiertes approximatives DP (**Reinforcement Lernen**). Die Idee ist

1. Stelle $V^{\pi}$ als Linearkombination **weniger**, einfacher Basisfunktionen dar, z.B.

   $$V^{\pi}(s) = w_1 \cdot \text{Bauern}(s) + w_2 \cdot \text{Springer}(s) + w_3 \cdot \text{Läufer}(s) + w_4 \cdot \text{Turm}(s) + w_5 \cdot \text{Dame}(s)$$

2. Nähere unbekannte Systemdynamik $P(s'|s, \pi(s))$ durch Stichproben unter $\pi$ an.

# 1. Approximatives DP (mit Modell)

**Funktionsapproximation:** Sei $\mathcal{S} = \{1, \ldots, N\}$ Zustandsraum. Betrachte lineare Parametrisierung

$$V(s) \approx \tilde{V}(s; \mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(s) = \mathbf{w}^\mathsf{T} \boldsymbol{\phi}(s)$$

wobei

- $\phi_i : \mathcal{S} \to \mathbb{R}$ Basisfunktionen ($m$ Stück)
- $\boldsymbol{\phi}(s) = \big(\phi_1(s), \ldots, \phi_m(s)\big)^\mathsf{T}$ Featurevektor ($m \times 1$ Vektor)
- $\mathbf{w} = \big(w_1, \ldots, w_m\big)^\mathsf{T}$ die zu bestimmenden Gewichte sind.

Wobei die Wahl von 'richtigen' Basisfunktionen im allgemeinen Fall natürlich sehr schwierig ist ...

# Min. des Bellman Residuums (mit Modell)

## Exakte Repräsentation:

- Wertefunktion:

$$\mathbf{v} = \begin{bmatrix} V(1) \\ \vdots \\ V(N) \end{bmatrix}$$

- Politik-Evaluation: löse

$$(\mathbf{I} - \gamma \mathbf{P}^\pi)\mathbf{v} = \mathbf{R}^\pi$$

- $N$ Unbekannte, $N \times N$ GLS, $N$=Anz. Zustände

## Mit Funktionsapproximation:

- Approximierte Wertefunktion

$$\tilde{\mathbf{v}} = \begin{bmatrix} \tilde{V}(1;\mathbf{w}) \\ \vdots \\ \tilde{V}(N;\mathbf{w}) \end{bmatrix} = \mathbf{\Phi w} \text{ mit } \mathbf{\Phi} = \begin{bmatrix} \text{—} \ \boldsymbol{\phi}(1)^\mathsf{T} \ \text{—} \\ \vdots \\ \text{—} \ \boldsymbol{\phi}(N)^\mathsf{T} \ \text{—} \end{bmatrix}$$

- Approximative Politik-Evaluation: löse

$$\min_{\tilde{\mathbf{v}}=\mathbf{\Phi w}} \|(\mathbf{I} - \gamma \mathbf{P}^\pi)\tilde{\mathbf{v}} - \mathbf{R}^\pi\|^2$$

- $\Longrightarrow$ Normalgleichung für $\mathbf{w}$:

$$[(\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi})^\mathsf{T}(\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi})]\mathbf{w} = (\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi})^\mathsf{T}\mathbf{R}^\pi$$

- $m$ Unbekannte, $m \times m$ GLS, $m$=Anz. Basisfunktionen

(Und wie können wir das ohne Modell tun? ... )

**Jetzt:** betrachten wir (mit $\mathbf{D} = \mathrm{diag}(\mu(1), \ldots, \mu(N))$ stationäre Verteilung)

$$\min_{\mathbf{w}} \; \|(\mathbf{\Phi} - \gamma \mathbf{P}^{\pi} \mathbf{\Phi})\mathbf{w} - \mathbf{R}^{\pi}\|^2_{\mathbf{D}}$$

und das zugehörige Gleichungssystem für $\mathbf{w}$

$$\underbrace{(\mathbf{\Phi} - \gamma \mathbf{P}^{\pi} \mathbf{\Phi})^{\mathsf{T}} \mathbf{D} (\mathbf{\Phi} - \gamma \mathbf{P}^{\pi} \mathbf{\Phi})}_{=:\mathbf{A}} \mathbf{w} = \underbrace{(\mathbf{\Phi} - \gamma \mathbf{P}^{\pi} \mathbf{\Phi})^{\mathsf{T}} \mathbf{D} \mathbf{R}^{\pi}}_{=:\mathbf{b}}$$

**Unser Problem:** können/wollen $\mathbf{A}$ und $\mathbf{b}$ nicht ausrechnen, weil wir $\mathbf{P}^{\pi}$ nicht kennen oder $N$ zu groß ist.

**Aber:** schauen wir uns $\mathbf{A}$ und $\mathbf{b}$ genauer an (Darstellung mit äußeren Produkt)

$$\begin{aligned}
\mathbf{A} &= \sum_{i=1}^{N} \mu(i) \cdot \left(\boldsymbol{\phi}(i) - \gamma \mathbb{E}_{j|i,\pi(i)}\{\boldsymbol{\phi}(j)\}\right)\left(\boldsymbol{\phi}(i) - \gamma \mathbb{E}_{j|i,\pi(i)}\{\boldsymbol{\phi}(j)\}\right)^{\mathsf{T}} \\
\mathbf{b} &= \sum_{i=1}^{N} \mu(i) \cdot \left(\boldsymbol{\phi}(i) - \gamma \mathbb{E}_{j|i,\pi(i)}\{\boldsymbol{\phi}(j)\}\right)\mathbb{E}_{j|i,\pi(i)}\{R(j,i,\pi(i)\}
\end{aligned}$$

Jetzt machen wir folgenden Trick ...

# Simulationsbasiert (Forts.)

- Angenommen, wir können Zustandsübergänge des Systems unter $\pi$ beobachten (z.B. indem ein Roboter/Agent mit System **interagiert**), $\forall t = 0, 1, 2 \ldots$

$$s_t \xrightarrow[r_{t+1}]{\pi(s_t)} s_{t+1} \xrightarrow[r_{t+2}]{\pi(s_{t+1})} s_{t+2} \longrightarrow \cdots$$

- Bei **deterministischen** Zustandsübergängen können wir dann mit Hilfe der Stichproben (z.B. $T$ Stück) Näherungen für $\mathbf{A}$ und $\mathbf{b}$ erstellen. Es gilt (Tsitsiklis & Van Roy 97)

$$\mathbf{A}_T := \sum_{t=0}^{T-1} \big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)\big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big)^\mathsf{T} \qquad \text{(Dann } \mathbf{A}_T/T \to \mathbf{A} \text{ f.s.)}$$

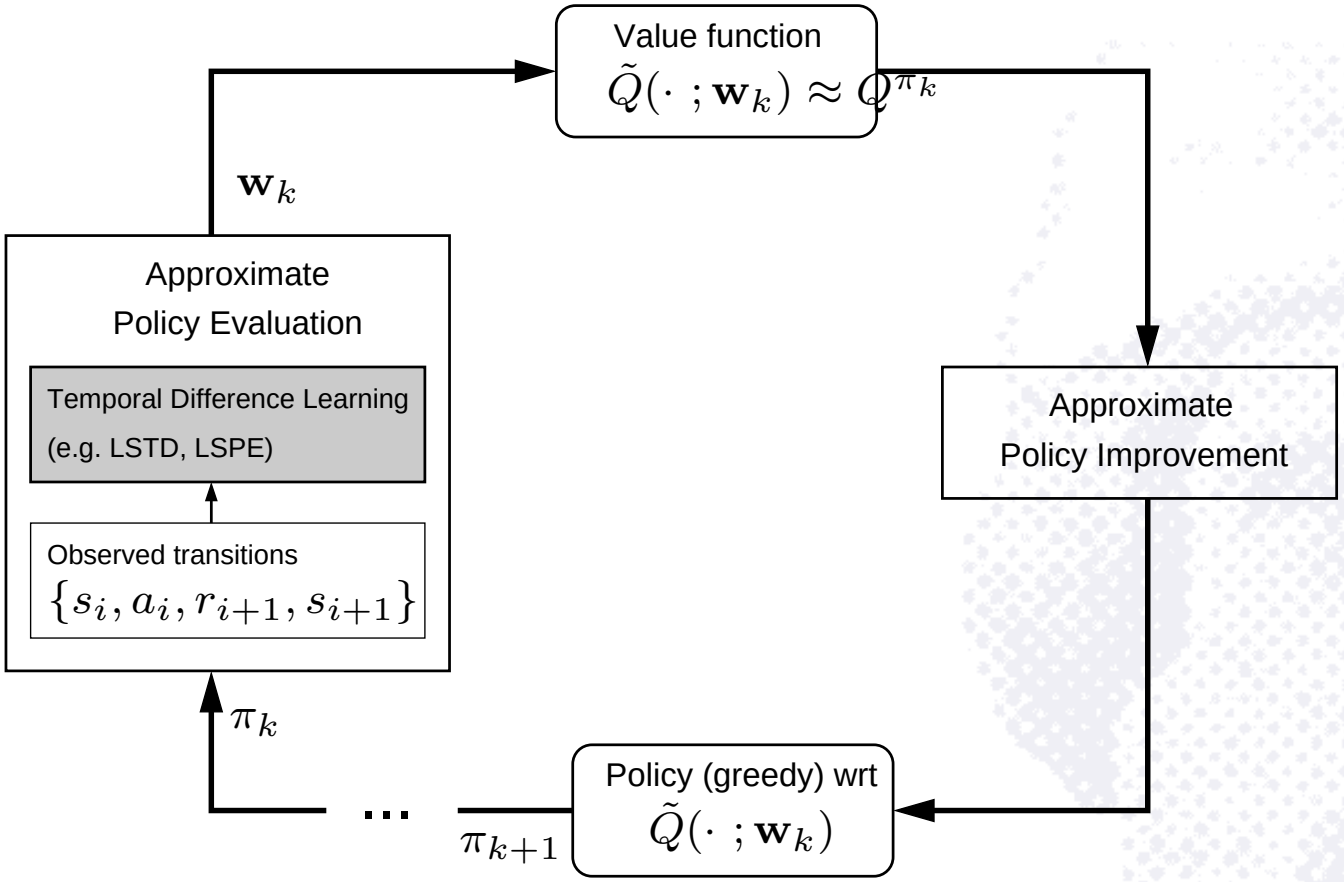
$$\mathbf{b}_T := \sum_{t=0}^{T-1} \big(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\big) r_{t+1} \qquad \text{(Dann } \mathbf{b}_T/T \to \mathbf{b} \text{ f.s.)}$$

- Statt $\mathbf{A}\mathbf{w} = \mathbf{b}$ lösen wir nun **Näherung** $\mathbf{A}_T\mathbf{w} = \mathbf{b}_T$, bzw. das zugehörige LS-Problem

$$\min_{\mathbf{w}} \left\| \begin{bmatrix} 1 & -\gamma & & \\ & \ddots & \ddots & \\ & & 1 & -\gamma \end{bmatrix} \begin{bmatrix} \text{—} & \boldsymbol{\phi}(s_0)^\mathsf{T} & \text{—} \\ & \vdots & \\ \text{—} & \boldsymbol{\phi}(s_T)^\mathsf{T} & \text{—} \end{bmatrix} \mathbf{w} - \begin{bmatrix} r_1 \\ \vdots \\ r_T \end{bmatrix} \right\|^2$$

$\Rightarrow$ Minimierung des Bellman Residuums (BRM). Alternativ: LSTD, LSPE, TD

# Zusammen: Approximative Politik-Iteration




**Im Endeffekt** reduziert sich das alles auf folgende Fragen:

1. Auf welche Weise wählen wir die Parametrisierung von $\tilde{V}$ und führen Regression durch?
2. Auf welche Weise lernen wir den zugehörigen Gewichtsvektor $\mathbf{w}$, gegeben Stichprobenübergänge (Daten)?

# Inhalt der Arbeit

**RL als LS-Problem (Regression)**

wobei Daten Stichprobenuebergaenge sind,
die von Roboter/Agent zur Laufzeit erzeugt werden.

Probleme:
- Online Lernen
- Hochdimensional
  (d.h. keine gitterbasierte Approximation)

**Regularisierungsnetze (RN) nichtparam. Funktionsapprox.**

- Gut geeignet fuer hochdim. Approx.
- Aber:
  - O(n^3) Rechenaufwand kubisch in Anz. Daten
  - Kein Online Lernen

**Subset of Regressors Approximation fuer RN**

- Selektion von m<<N relevanten Basiselementen
- Reduziert Aufwand auf O(n m^2)
- Aber: Kein OnlineLernen

**Online Lernen mit RN**

Neuer, Recursive Least-Squares-artiger Algorithmus fuer RN.
Zur Laufzeit koennen in ein bestehendes Modell
- neue Daten
- neue relevante Basiselemente hinzugenommen werden, und
- alte, irrelevant gewordene Basiselemente entfernt werden

Und das alles mit Aufwand O(m^2) pro Schritt, also unabhaengig von der Anzahl zuvor gesehener Daten.

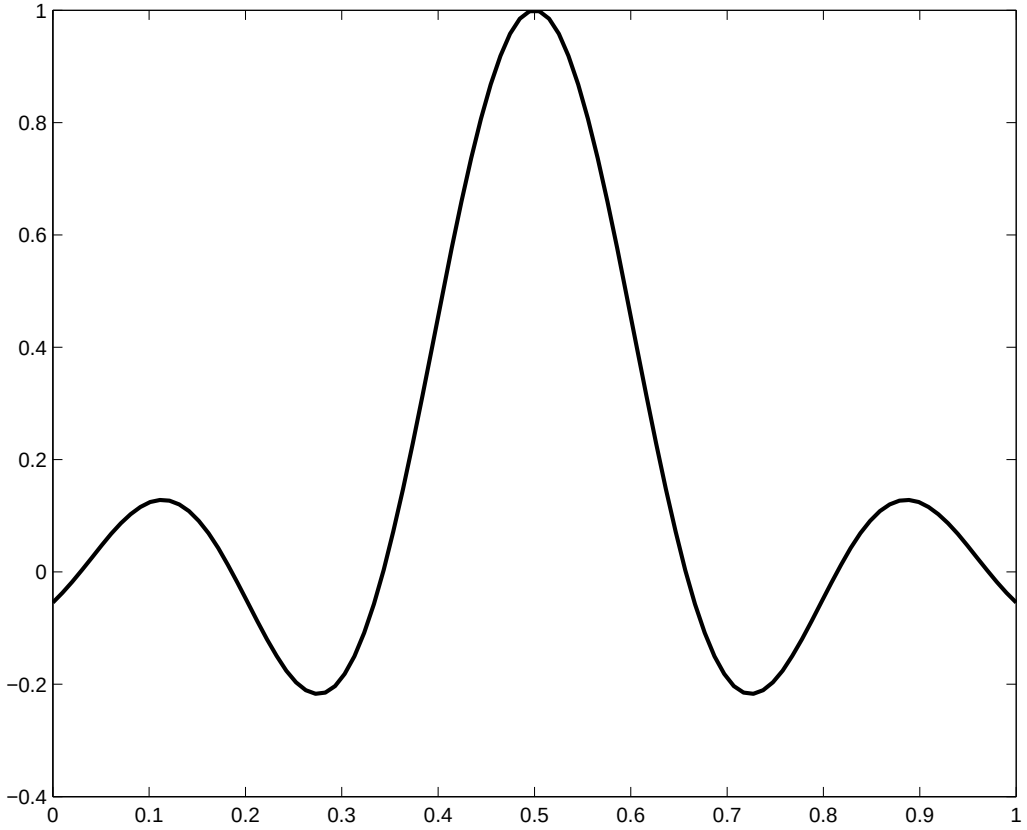
**Anwendung auf Reinforcement Lernen**

Realistische, hochdimensionale Probleme aus Robotik:
- RoboCup Keepaway
- Oktopus-Tentakel

# Teil II:
# Online Lernen mit Regularisierungsnetzen

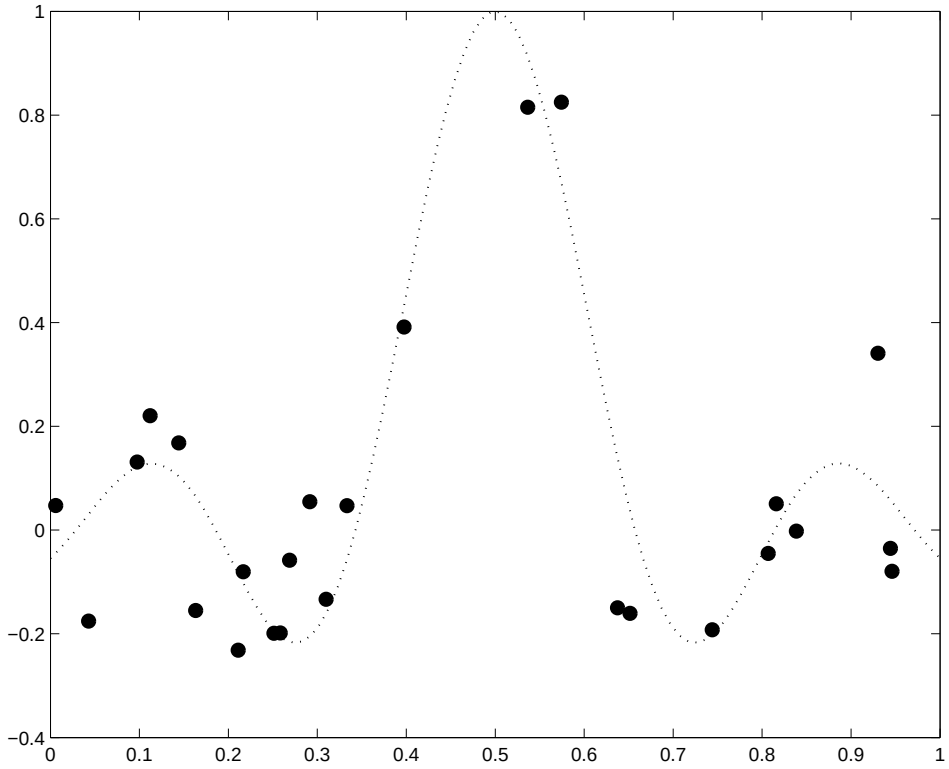
(Unabhängig von Reinforcement Lernen)

# Funktionsapproximation – Idee




Funktionaler Zusammenhang $f : \mathcal{X} \longrightarrow \mathcal{Y}$ (unbekannt)

# Funktionsapproximation – Idee

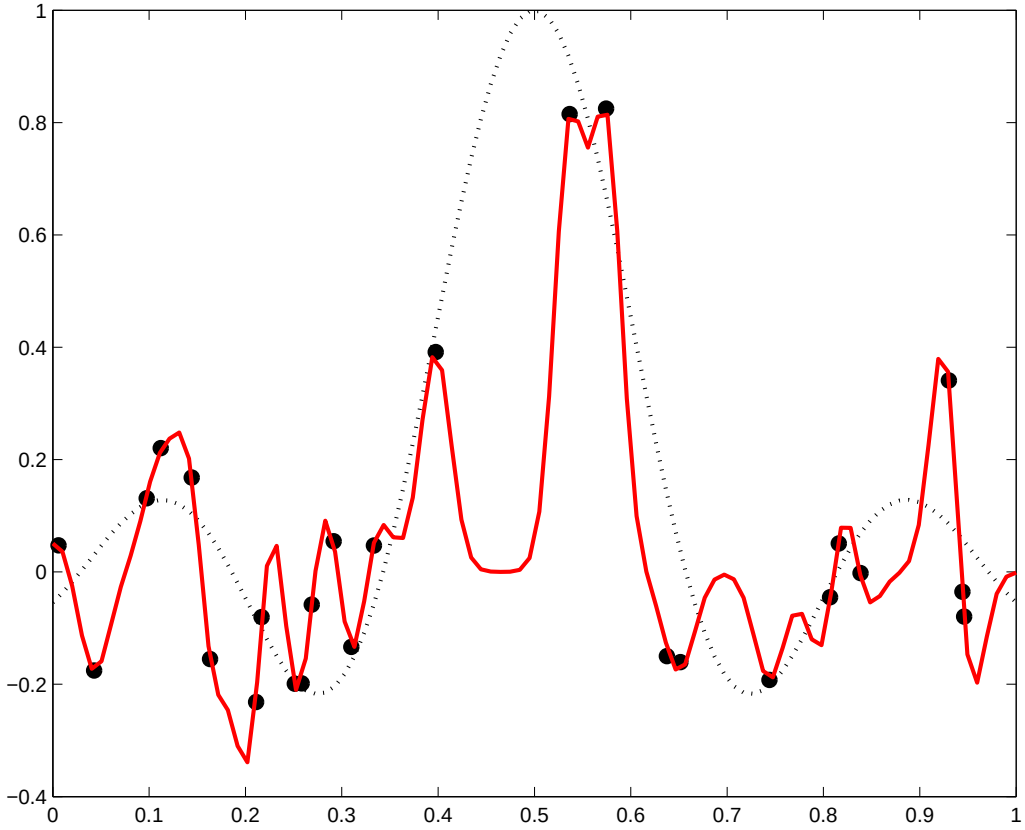



Gegeben sind nur endlich viele (verrauschte) Stichproben $y_i = f(x_i) + \varepsilon_i$.

Wie können wir aus den Daten die zugrundeliegende Funktion rekonstruieren?

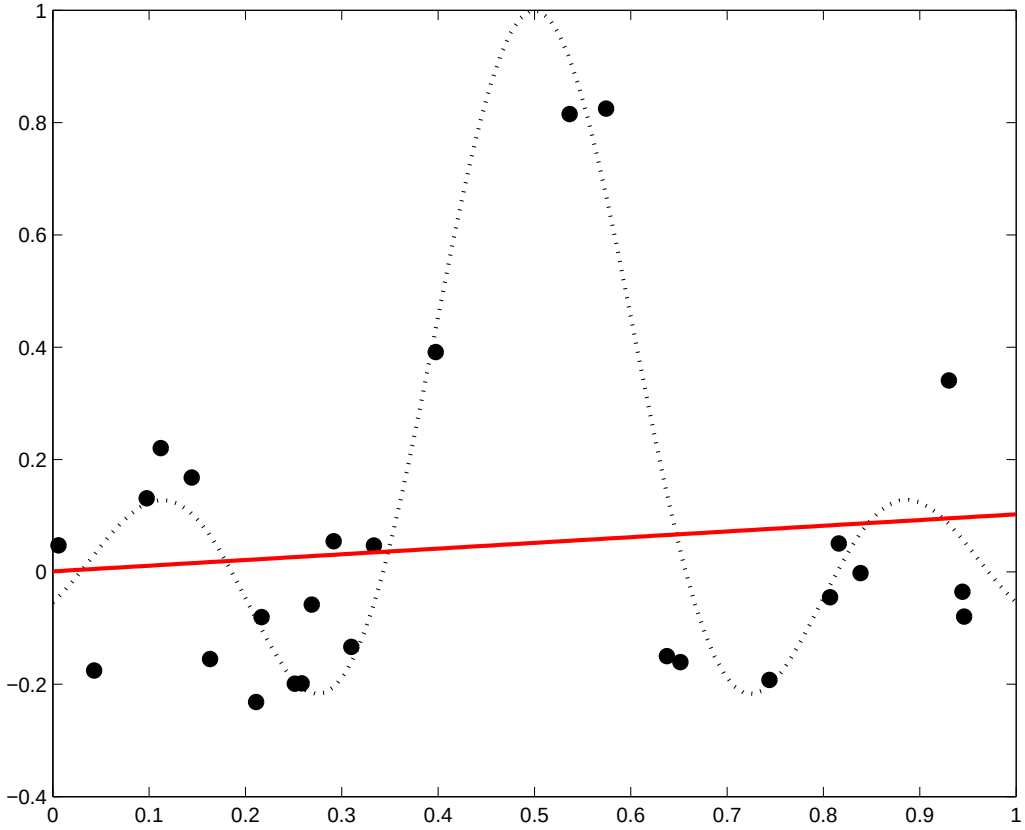
$\Longrightarrow$ Schlecht gestelltes Problem !

# Funktionsapproximation – Idee




Zu kompliziert! (Interpolation)

# Funktionsapproximation – Idee




Zu einfach!

Müssen also eine Lösung aus einer Menge von möglichen Lösungen suchen, die einerseits "reichhaltig" genug ist, um komplizierte Zusammenhänge abzubilden und zum anderen dafür sorgen, daß die Funktion nicht zu "zappelig" wird (overfitting). Eine Möglichkeit dafür $\Longrightarrow$ Regularisierungsnetze...

# Regularisierungsnetze

(auch bekannt als Kernel Ridge Regression, Gauß Prozeß Regression. Ähnlich: LS-SVM, RVM ...)

**Aufgabe:**

- **Gegeben:** Daten $\{\mathbf{x}_i, y_i\}_{i=1}^t$, Inputs $\mathbf{x}_i \in \mathbb{R}^d$, Outputs $y_i \in \mathbb{R}$
- **Ziel:** lerne zugrundeliegende Funktion $f$ aus $\mathcal{H}_k$ (RKHS) durch Lösen von

$$\min_{f \in \mathcal{H}_k} \sum_{i=1}^{t} (y_i - f(\mathbf{x}_i))^2 + \lambda \|f\|^2_{\mathcal{H}_k}$$

**Lösung:**

- **Repräsenter-Theorem:** Gesuchtes $f$ ist von der Form $f(\cdot) = \sum_i k(\mathbf{x}_i, \cdot) w_i$. Löse daher das $t \times t$ Problem

$$\min_{w \in \mathbb{R}^t} \|\mathbf{K}\mathbf{w} - \mathbf{y}\|^2 + \lambda \mathbf{w}^\mathsf{T} \mathbf{K} \mathbf{w}$$

- **Ergebnis:**

$$\mathbf{w} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$$

wobei $k(\cdot, \cdot)$ Kovarianz/Kernfunktion (z.B. **Gauß RBF**), $\mathbf{K}$ die $t \times t$ Kernmatrix $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ und $\lambda$ der Regularisierungsparameter.

Der Haken: Lösung kostet $\mathcal{O}(t^3)$. Unpraktisch für große Anzahl von Daten ...

# Subset of Regressors Approximation

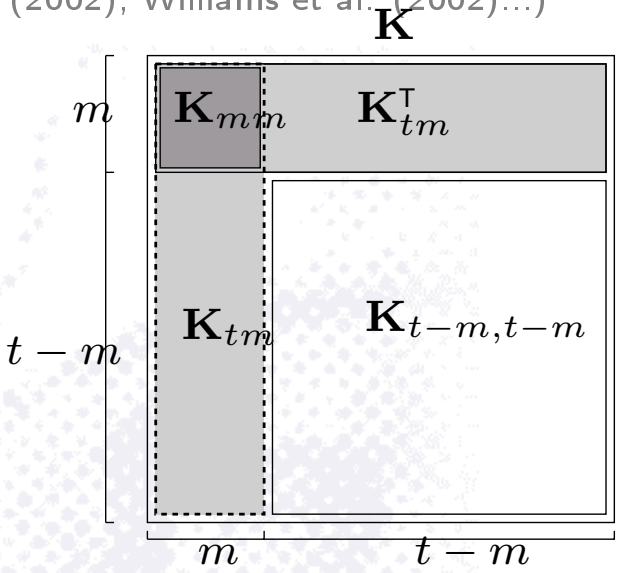
(vorgeschlagen von Poggio & Girosi (1990), Wahba (1990), Schölkopf & Smola (2002), Williams et al. (2002)...)

**Idee:**

- **Selektiere:** Teilmenge von Daten $\{\tilde{\mathbf{x}}\}_{i=1}^{m}$, mit $m \ll t$
- **Approximiere:** Kern durch

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{k}_m(\mathbf{x})^\mathsf{T} \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}') \qquad \forall \mathbf{x}, \mathbf{x}'$$

wobei $\mathbf{k}_m(\cdot) = \big(k(\tilde{\mathbf{x}}_1, \cdot), \ldots, k(\tilde{\mathbf{x}}_m, \cdot)\big)^\mathsf{T}$
und $[\mathbf{K}_{mm}]_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$.




**Was bringt uns das?**

- **Reduziertes Problem:** repräsentiere $f$ durch $f(\cdot) = \sum_i^m k(\tilde{\mathbf{x}}_i, \cdot) w_i$ und löse das $m \times m$ Problem

$$\min_{w \in \mathbb{R}^m} \; \|\mathbf{K}_{t,m}\mathbf{w} - \mathbf{y}\|^2 + \lambda \mathbf{w}^\mathsf{T} \mathbf{K}_{mm} \mathbf{w}$$

- **Ergebnis:**

$$\mathbf{w} = \big(\mathbf{K}_{t,m}^\mathsf{T} \mathbf{K}_{t,m} + \lambda \mathbf{K}_{mm}\big)^{-1} \mathbf{K}_{t,m}^\mathsf{T} \mathbf{y}$$

wobei $[\mathbf{K}_{t,m}]_{ij} = k(\mathbf{x}_i, \tilde{\mathbf{x}}_j)$. Das kostet nur noch $\mathcal{O}(tm^2)$ Operationen (anstatt $\mathcal{O}(t^3)$).

Und als nächstes kommt: wie selektieren wir die $m$ relevanten Basiselemente?

# Wie können wir diese Basiselemente auswählen?

**Frage:** wie können wir effizient die $m$ Basiselemente finden? Z.B.

- Zufallsauswahl, partielles Gram-Schmidt ... ("Unüberwacht")
- Matching Pursuit, Orthogonal Least Squares, ... ("Überwacht")

**Unser Problem:** Online Selektion

- Nehmen an, Daten sind nicht komplett sondern nur als Sequenz verfügbar, $t = 1, 2, \ldots$
- Idee: Starten mit leeren Menge von Basiselementen. Im $t$-ten Schritt beobachten wir Trainingsbeispiel $(\mathbf{x}_t, y_t)$ und entscheiden, ob wir $\mathbf{x}_t$ selektieren sollen, oder nicht.

**Unüberwacht:** ("Neuheit" des aktuellen Beispiels)

- Idee: versuche Approximationsfehler $k(\mathbf{x}, \mathbf{x}') - \mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}')$ klein zu machen.
- Kriterium: Falls Abstand zum Erzeugnis der bisher ausgewählten Elemente

$$\delta_t = k(\mathbf{x}_t, \mathbf{x}_t) - \mathbf{k}_m(\mathbf{x}_t)^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}_t)$$

  eine Toleranz TOL überschreitet, dann füge $\mathbf{x}_t$ hinzu. (Csato & Opper 02, Engel et al 03)
- Kosten $\mathcal{O}(m^2)$

**Überwacht:** ("Relevanz" des aktuellen Beispiels)

- Idee: betrachte zusätzlich noch, wie stark sich Fehler der eigentlichen Regression verändert
- Effekt: es werden nur solche Elemente selektiert, die für das konkret vorliegende Problem relevant sind.
- Vorteil: Reduktion der Basiselemente um 20-50% ohne Einbußen bei Performanz und bei vernachlässigbaren zusätzlichen Kosten.

# RLS-artiges online Lernen für RN

**Übersicht:**

- **Zu lösen war:** $\min_{\mathbf{w}} J_{tm}(\mathbf{w}) = \|\mathbf{K}_{t,m}\mathbf{w} - \mathbf{y}_t\|^2 + \lambda \mathbf{w}^T \mathbf{K}_{mm}\mathbf{w}$
- **Lösung mit Normalgleichung:** $\mathbf{w}_{tm} = \left(\mathbf{K}_{t,m}^T \mathbf{K}_{t,m} + \lambda \mathbf{K}_{mm}\right)^{-1} \mathbf{K}_{t,m}^T \mathbf{y}_t$
- **Abkürzungen:**
  - Kreuzproduktmatrix $\mathbf{P}_{tm} := \left(\mathbf{K}_{t,m}^T \mathbf{K}_{t,m} + \lambda \mathbf{K}_{mm}\right)$
  - Kosten $\xi_{tm} := J_{tm}(\mathbf{w}_{tm})$
- **Vorgehensweise:** aktualisiere rekursiv $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\}$ wenn neues Trainingsbeispiel beobachtet wird.

**Genauer:** Zeitpunkt $t+1$ (bei $m$ aktuell selektierten Basiselementen)

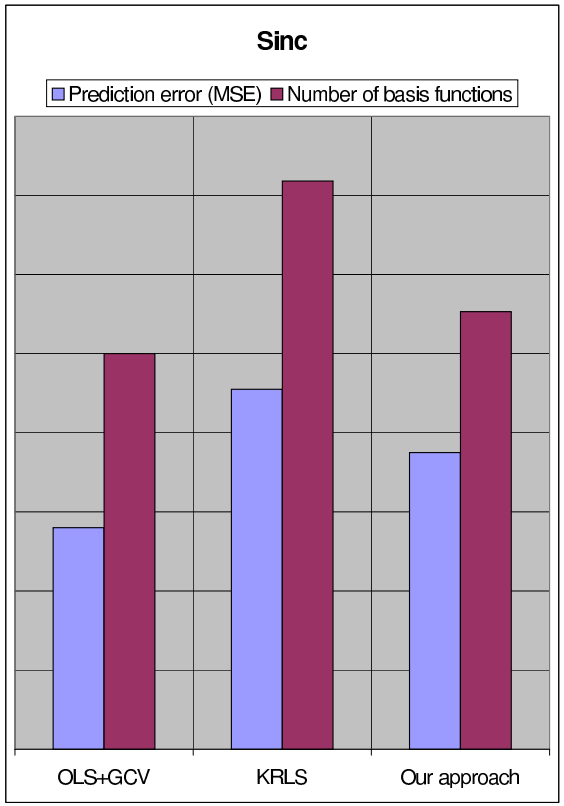
- Beobachte neues Beispiel $(\mathbf{x}_{t+1}, y_{t+1})$
- Aktualisiere $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\}$ durch die Operationen
  - **Normal:** $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t+1,m}^{-1}, \mathbf{w}_{t+1,m}, \xi_{t+1,m}\}$
  - **Basiserweiterung:** $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t,m+1}^{-1}, \mathbf{w}_{t,m+1}, \xi_{t,m+1}\}$
  - **Löschen:** $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t,m\setminus i}^{-1}, \mathbf{w}_{t,m\setminus i}, \xi_{t,m\setminus i}\}$
- Effiziente $\mathcal{O}(m^2)$ Implementation mit SMW-Formeln.

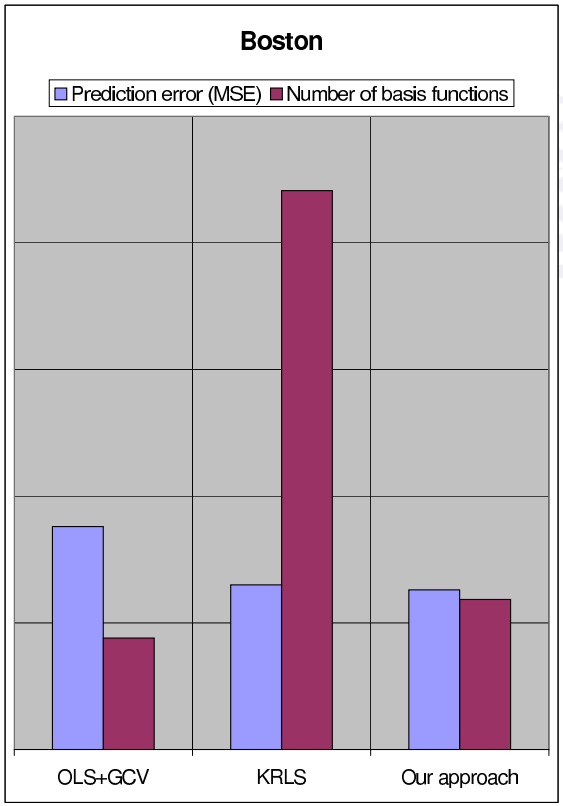
# Experimente I

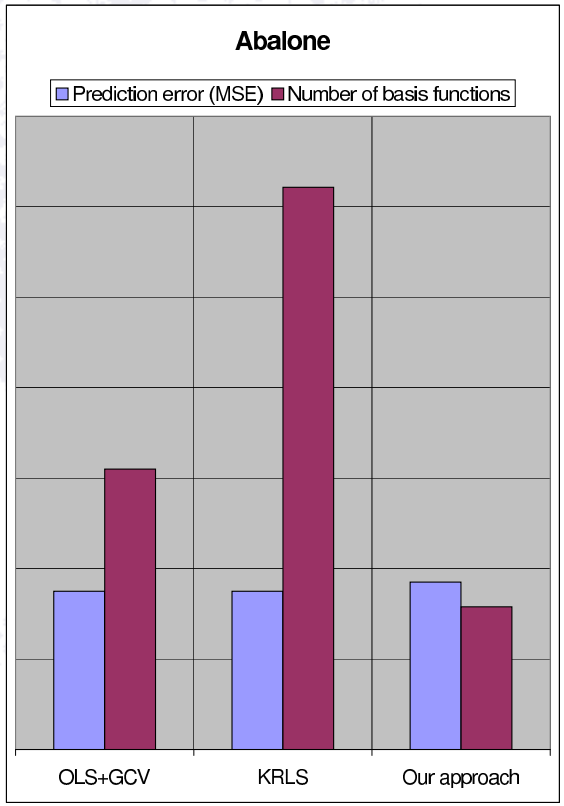
**Vergleiche Selektionsverfahren:** Vorhersagefehler vs. Anzahl selektierter Basiselemente

**Verfahren:** (alle verwenden identische Menge von Basiskandidaten)

- Orthogonal least squares + GCV Abbruchkriterium
- Kernel recursive least squares mit unüberwachter Basisselektion (Engel et al. 2004)
- Mein Ansatz mit überwachter Basisselektion



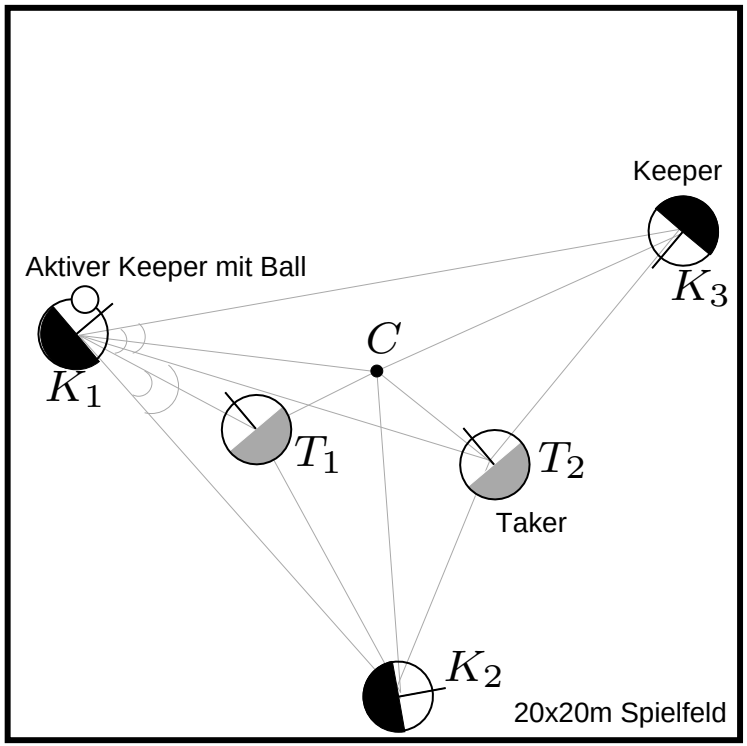

(Implementation: bis hier prototypisch in MATLAB/Octave)

# Teil III:
# Reinforcement Lernen mit Regularisierungsnetzen

# RoboCup-Keepaway (Stone et al. 2005)

(Quelle: http://www.cs.utexas.edu/users/AustinVilla/sim/keepaway/)

**Ziel:** **lerne** die Dauer zu maximieren, die die Keeper im Ballbesitz sind (3 vs. 2)



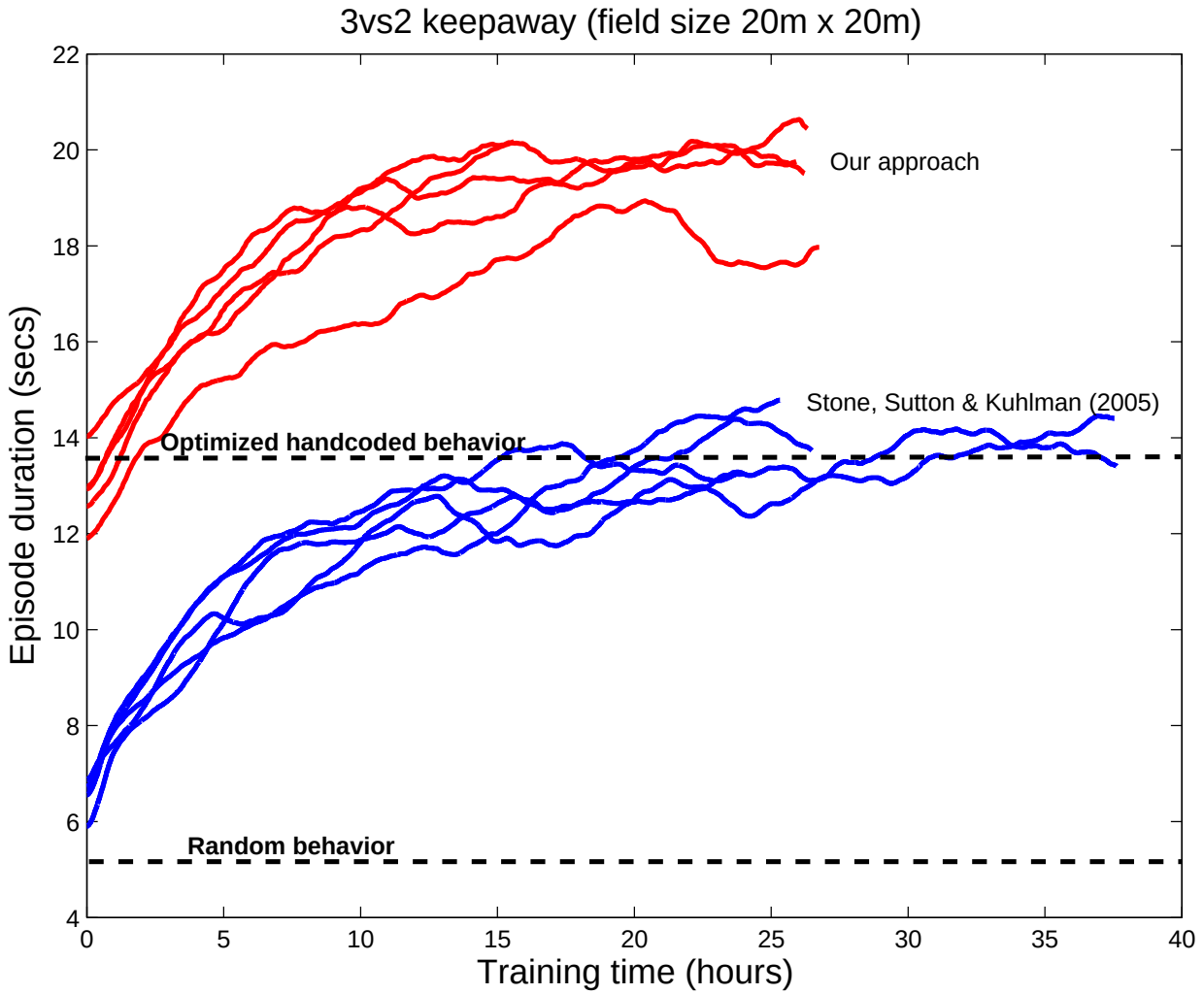

| | Zusammensetzung des Zustandsvektors |
|---|---|
| 1. | $dist(K_1, C)$ |
| 2. | $dist(K_2, C)$ |
| 3. | $dist(K_3, C)$ |
| 4. | $dist(T_1, C)$ |
| 5. | $dist(T_2, C)$ |
| 6. | $dist(K_1, K_2)$ |
| 7. | $dist(K_1, K_3)$ |
| 8. | $dist(K_1, T_1)$ |
| 9. | $dist(K_1, T_2)$ |
| 10. | $\min\{dist(K_2, T_1), dist(K_2, T_2)\}$ |
| 11. | $\min\{dist(K_3, T_1), dist(K_3, T_2)\}$ |
| 12. | $\min\{ang(K_2, K_1, T_1), ang(K_2, K_1, T_2)\}$ |
| 13. | $\min\{ang(K_3, K_1, T_1), ang(K_3, K_1, T_2)\}$ |

**Herausforderungen:**

- **Dimensionalität** des Zustandsraums (13 Dimensionen)
- **Stochastische Übergänge** (verrauschte Wahrnehmung und Aktionsausführung, mehrere **autonome** Agenten müssen kooperieren)
- **Echtzeit:** pro Simulationsschritt ca. 100 msec (basiert auf "offiziellem" Soccer Server)

# Ergebnis: RoboCup Keepaway 3vs2

**Vergleichen:** Unseren Ansatz LSTD+RN vs. Lehrbuchverfahren Sarsa($\lambda$)+Tilecoding

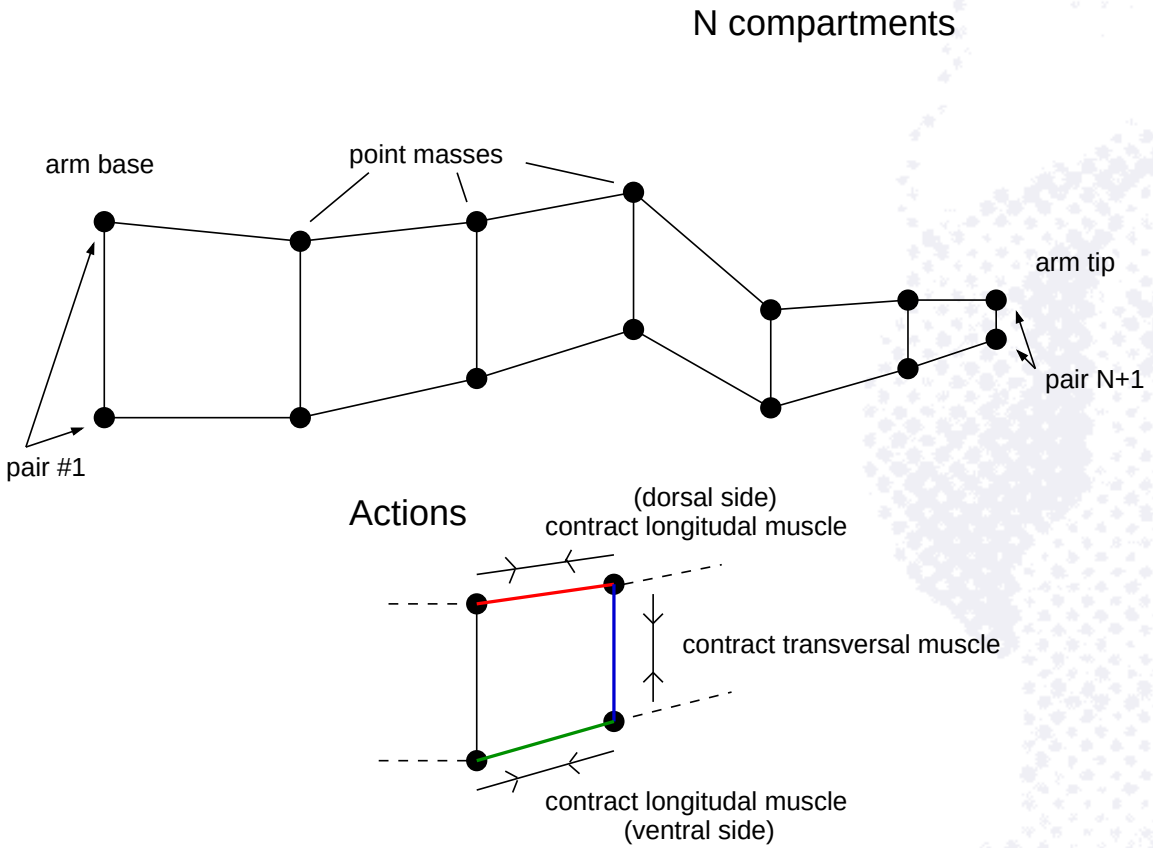



Effiziente Implementation des Verfahrens: C++ mit prozessor-optimierter BLAS (ATLAS)

# Oktopus (Engel et al. 2005)

(ICML06 RL benchmark: http://www.cs.mcgill.ca/dprecup/workshops/ICML06/octopus.html)

**Ziel:** **Erlernen** der Steuerung des Oktopus-Tentakels



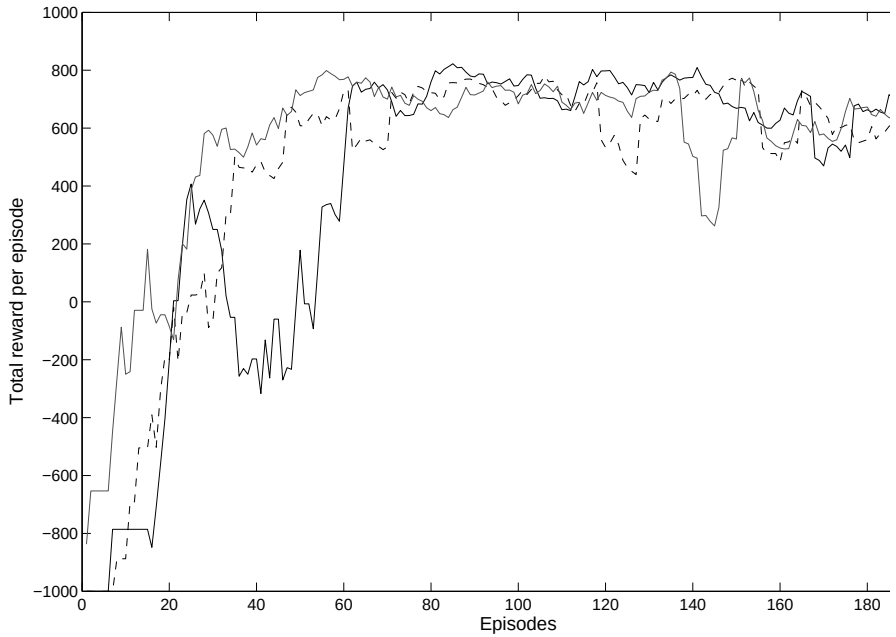

**Herausforderungen:**

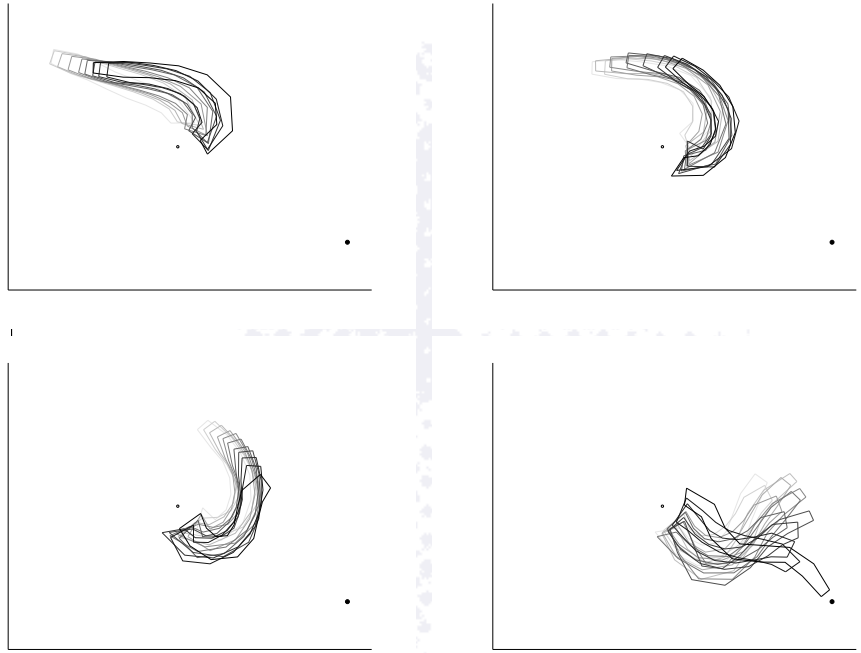
- **Hochdimensionaler** Zustandsraum $\subset \mathbb{R}^{(2N+2)\times 4}$ (z.B. N=8 Segmente $\Longrightarrow \mathbb{R}^{72}$)
- **Hochdimensionaler** und **kontinuierlicher Aktionsraum** $\subset \mathbb{R}^{N\times 3}$
- ... hier: diskretisiert in 7 Aktivierungsprofile

# Ergebnis: Oktopus

**Testbett:** 8-segmentiger Tentakel ('HardTask' vom ICML-06 benchmark)

**Verfahren:** OPI mit LSPE+RN. Kein anderer Vergleichskandidat verfügbar.



# Zusammenfassung

## I. Online Lernen mit Regularisierungsnetzen

- basierend auf Subset of Regressors Approximation
- Überwachte Online Selektion relevanter Basisfunktionen, die sowohl
  - **Fehler** bei der Approximation des Kerns, als auch
  - **Nützlichkeit** beim eigentlichen Minimierungsproblem berücksichtigt
  - $\Longrightarrow$ Keine Einbußen der Performanz bei 20-50% weniger Basiselementen
- Effiziente rekursive Implementation: $\mathcal{O}(m^2)$ pro Schritt
  (unabhängig von der Anzahl zuvor gesehener Trainingsdaten)

## II. Reinforcement Lernen mit Regularisierungsnetzen

- basierend auf Least-Squares Formulierung von Approximativer Politik Evaluation
- Key features:
  - Hochdimensionale Zustandsräume (nichtparametrisierter RN)
  - Stochastische Zustandsübergänge
  - Modellfreies Lernen (simulationsbasiert)
  - Hohe Dateneffizienz (im Vergleich zu Gradientenbasierten Standardverfahren wie Sarsa oder TD)
  - Effiziente Online Implementation für zeitkritische Echtzeitanwendungen