Experiments in Value Function Approximation with Sparse Support Vector Regression

Tobias Jung and Thomas Uthmann

{tjung,uthmann}@informatik.uni-mainz.de

Fachbereich Mathematik & Informatik

Johannes Gutenberg-Universität Mainz, Germany

Why SVR?

Why might it be a good idea?

- current state-of-the-art in many benchmark/real-world problems
- less burdened by #dim when compared with grids, tilecoding, etc.
- less unwieldy when compared to Neural Networks (no "'forgetting"', no local minima)
- better generalization when compared to local instance-based methods (e.g. LWR)

Why might it be a bad idea?

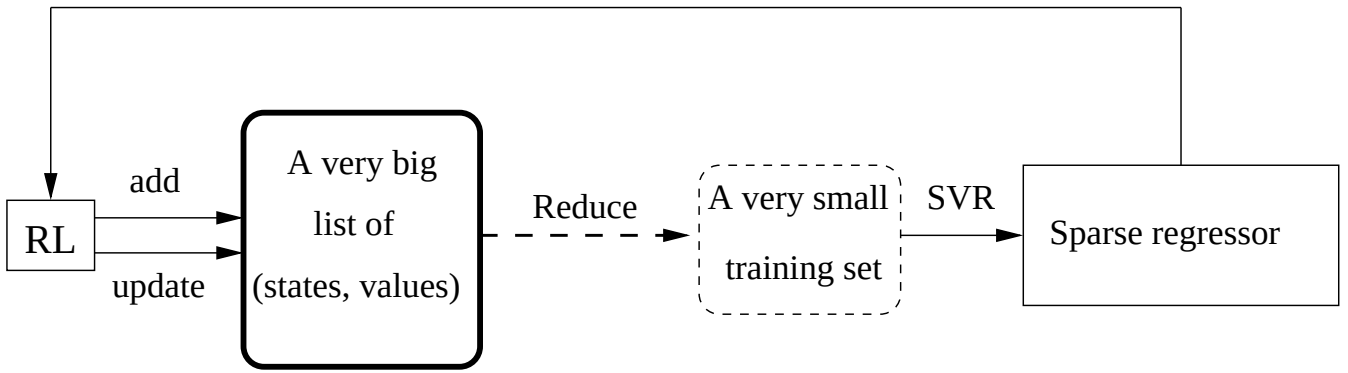
- conceptual/implementation issues: **SVR is a Batch-learner**
- on-line RL needs to
  - add new samples to the current training sequence
  - modify (update) existing ones

Contents

What is this talk about?

1. **Value Function Approximation**
   Reinforcement Learning, Temporal-Difference Learning, Function Approximation and TD(0)
2. **Support Vector Regression**
   Formulate QP, Sparse Approximation, Reduced Problem, On-line Selection of Subset (based on Engel, Mannor and Meir (2002))
3. **Experiments**
   Gridworld, Mountain Car
4. **Summary and Future Ideas**

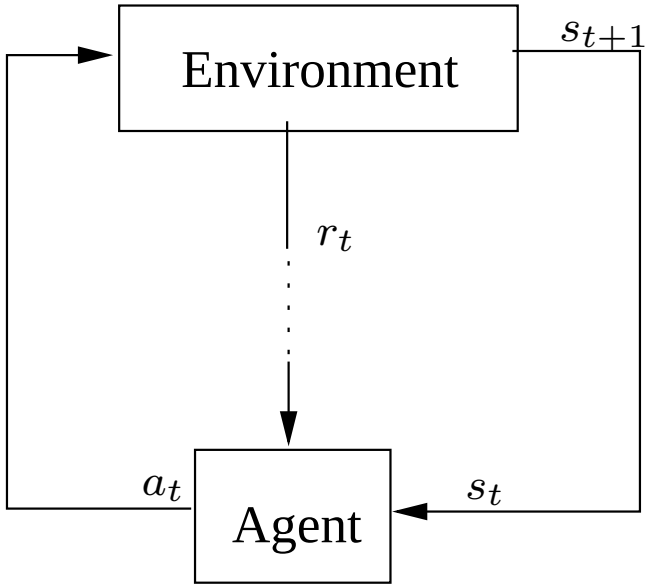
The big plan:



Reinforcement Learning I

A Markov Decision Process consists of

- States $\mathcal{S} = \{s_1, \ldots, s_N\}$
- Actions $\mathcal{A} = \{a_1, \ldots, a_M\}$
- Rewards model: $R^a(s, s')$
- Transition probabilities (Markov): $P^a(s, s')$



Hitch: Usually *delayed reward*. In RL learner does not know the *model* $P^a(s, s'), R^a(s, s')$.

Objective: choose actions to maximize long term reward.

Reinforcement Learning II

Criterion: infinite-horizon expected total discounted reward

How do we get there?

- Policy: $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (deterministic, stationary)
- Value function: (γ discount rate)

$$V^{\pi}(s) = E^{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_k \ | s_t = s, \pi\}, \quad \forall s$$

- Bellman says:

$$V^{\pi}(s) = \sum_{s'} P^{\pi(s)}(s, s')\left[R^{\pi(s)}(s, s') + \gamma V^{\pi}(s')\right], \quad \forall s$$

- **Goal:** optimal policy $\pi^* = \text{argmax}_{\pi} V^{\pi}$, optimal value function $V^*(s) = \max_{\pi} V^{\pi}(s)$, $\forall s$

Many ways to solve it:

- Methods based on Policy Iteration (e.g. Optimistic PI, Actor-Critic)
- Methods based on Value Iteration (e.g. Q-learning)

Reinforcement Learning III

Many algorithms perform policy-evaluation:

- Dynamic Programming style (model-based, use fixed policy π):

$$V_{t+1}(s) = V_t(s) + \underbrace{(\sum_{s'} P^{\pi(s)}(s,s')\left[R^{\pi(s)}(s,s') + \gamma V_t(s')\right]}_{\text{target}} - V_t(s))$$

- Temporal-Difference style (model-free, use *observed* reward r_t and next state s' using π):

$$V_{t+1}(s) = V_t(s) + \alpha(\underbrace{r_t + \gamma V_t(s')}_{\text{target (unbiased estimate)}} - V_t(s))$$

Memory-based function approximation: Basically, works by storing *(state,targets)* in a list:

- **TD-update**:
  - *add* new instance whenever current state is *far* from rest
  - else *update* target for *nearest* state
- **Query:** Build (local) approximation

Recall SVR ...

Objective: Given data $\{\mathbf{x}_i, y_i\}_{i=1}^{\ell}$. In ε-SVR we solve (bias absorbed)

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^* \in \mathbb{R}^\ell} \quad & -\tfrac{1}{2}(\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T K (\boldsymbol{\alpha}^* - \boldsymbol{\alpha}) - \varepsilon(\boldsymbol{\alpha}^* + \boldsymbol{\alpha})^T \mathbf{e} + (\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{0} \leq \boldsymbol{\alpha}, \boldsymbol{\alpha}^* \leq C\boldsymbol{e} \end{aligned}$$

Final regressor: $f(\mathbf{x}) = (\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T \boldsymbol{k}(\mathbf{x})$

where	$k(\cdot,\cdot)$	symmetric positive definite function (kernel)
	$K \in \mathbb{R}^{\ell \times \ell}$	Kernel matrix $[K]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$
	$\boldsymbol{k}(\mathbf{x}) \in \mathbb{R}^\ell$	with $\boldsymbol{k}(\mathbf{x}) = \big(k(\mathbf{x}_1, \mathbf{x}), \ldots, k(\mathbf{x}_\ell, \mathbf{x})\big)^T$
	$C \in \mathbb{R}^{\geq 0}$	Regularization parameter

Our problem: complexity scales superlinearly with # data

Support Vector Regression II

Recall the Representer Theorem: Every solution $f \in \mathcal{H}$ (RKHS) to

$$\min_{f \in \mathcal{H}} \frac{1}{\ell} \sum_i c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) + \Lambda \|f\|_{\mathcal{H}}$$

admits a representation $f(\mathbf{x}) = \sum_i^{\ell} \beta_i k(\mathbf{x}_i, \mathbf{x})$

$\Longrightarrow$ Solution lies in a subspace spanned by the $k(\mathbf{x}_i, \cdot)$ (the data!)

Observation: K's eigenvalues decay rapidly, many of them are very small

$\Longrightarrow$ This subspace can be **approximated** by just picking **some** of the $k(\mathbf{x}_i, \cdot)$

Goal: Reduce the number of coefficients β_i that we have to determine.

Eliminate linear dependence

Assume we have picked the *first* m samples (for convenience marked by $\tilde{\cdot}$) ...

Approximate: the remaining $\ell - m$ ones (in $\mathcal{H}$)

$$\min_{\boldsymbol{a}_i \in \mathbb{R}^m} \left\| k(\mathbf{x}_i, \cdot) - \sum_j^m a_{ij} k(\tilde{\mathbf{x}}_j, \cdot) \right\|_{\mathcal{H}}^2 \quad , \; i = m+1 \ldots \ell$$

.... we obtain the coefficients:

$$\boldsymbol{a}_i = \tilde{K}^{-1} \tilde{\boldsymbol{k}}(\mathbf{x}_i)$$

where $\tilde{K} \in \mathbb{R}^{m \times m}$ **Reduced Kernel matrix** $[\tilde{K}]_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$

$\tilde{\boldsymbol{k}}(\mathbf{x}_i) \in \mathbb{R}^m$ with $\tilde{\boldsymbol{k}}(\mathbf{x}_i) = \big(k(\tilde{\mathbf{x}}_1, \mathbf{x}_i), \ldots, k(\tilde{\mathbf{x}}_m, \mathbf{x}_i)\big)^T$

Define: $A \in \mathbb{R}^{\ell \times m}$ to be the matrix consisting of the rows $\boldsymbol{a}_i^T$. Then $K \approx A \tilde{K} A^T$.

Goal: Want to use the *much* smaller $\tilde{K}$ instead of the big K in our QP ...

Define a reduced problem

Consider:

- *reduced* variables $\tilde{\boldsymbol{\alpha}} = A^T \boldsymbol{\alpha}$, $\tilde{\boldsymbol{\alpha}}^* = A^T \boldsymbol{\alpha}^*$ (each in $\mathbb{R}^m$)
- transformed target values $\tilde{\boldsymbol{y}} = A^\dagger \boldsymbol{y}$
- solving the QP in the reduced variables $\tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\alpha}}^*$ with the reduced set $\{(\tilde{\boldsymbol{x}}_i, \tilde{y}_i)\}_{i=1}^m$

Obtain: the solution to the reduced problem

$$\begin{aligned} \tilde{f}(\cdot) &= \sum_{i=1}^{m} (\tilde{\alpha}_i^* - \tilde{\alpha}_i) k(\tilde{\mathbf{x}}_i, \cdot) \\ &= \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) \sum_{j=1}^{m} a_{ij} k(\tilde{\mathbf{x}}_i, \cdot) \approx \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) k(\mathbf{x}_i, \cdot) = f(\cdot) \end{aligned}$$

which is **approximately** the one we would have obtained from the full problem.

Consequence: Instead of $\{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ use the reduced data $\{(\tilde{\mathbf{x}}_i, \tilde{y}_i)\}_{i=1}^{m}$ (usually $m \ll \ell$).

How do we obtain the reduced set?

Goal: build $\{(\tilde{\mathbf{x}}_i, \tilde{y}_i)\}_{i=1}^m$ in an on-line fashion (adapted from Engel et al. (2002))

Parameter: choose TOL (approximation precision)

Book-keeping: need $\{(\tilde{\mathbf{x}}_i, \tilde{y}_i)\}_{i=1}^m$, $\tilde{K}^{-1}$, $(A^T A)^{-1}, A^T \boldsymbol{y}$

Start with an empty basis

LOOP

Get current sample $(\mathbf{x}_t, y_t)$.

Compute distance d_t to span of current basis.

IF $d_t <$ TOL then $k(\mathbf{x}_t, \cdot)$ is approximated well enough

Size of basis is unchanged. Recursively update $(A^T A)^{-1}, A^T \boldsymbol{y}$.

ELSE

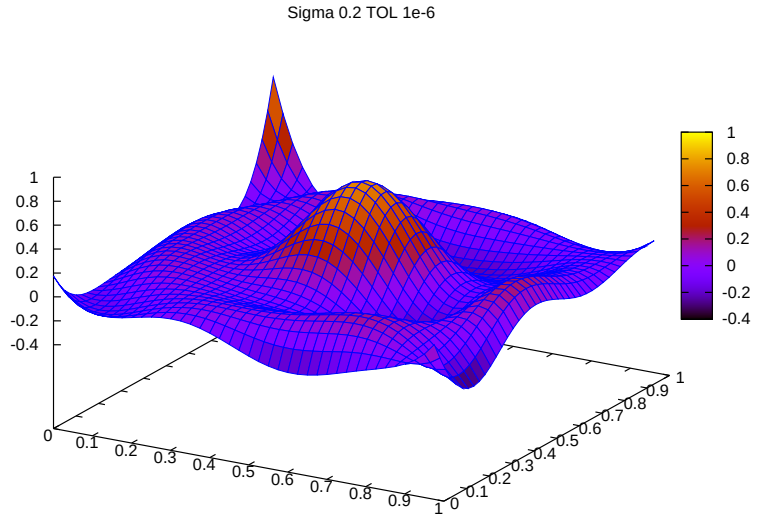
Add $\mathbf{x}_t$ to basis. Recursively update $\tilde{K}^{-1}$. Append $(A^T A)^{-1}, A^T \boldsymbol{y}$.

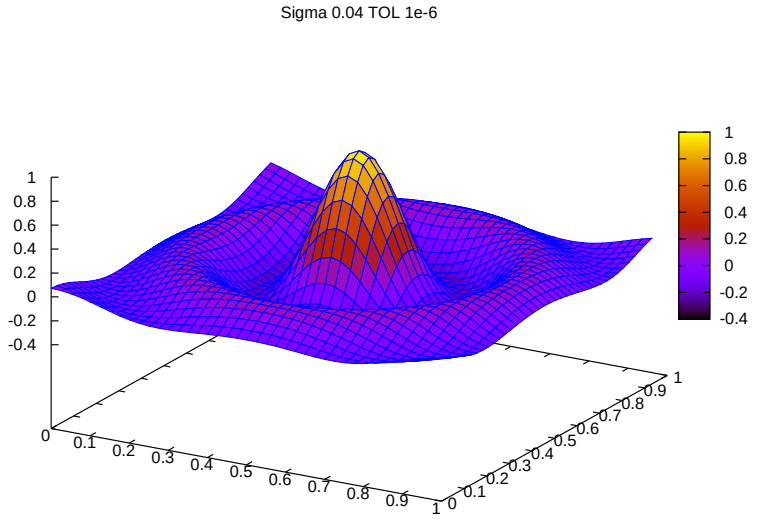
How does it scale for large sample sizes?

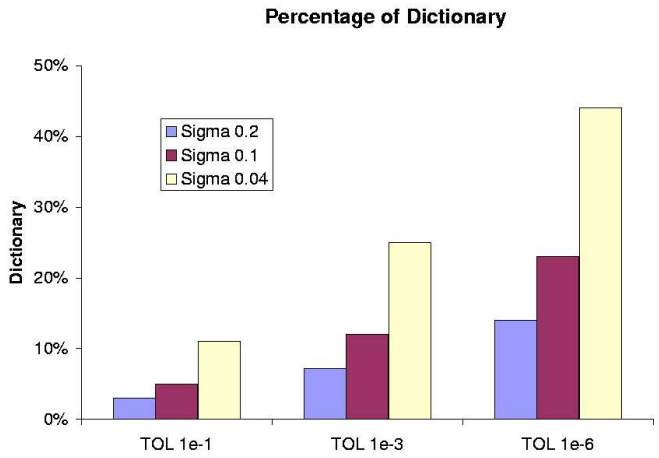
- Efficient: memory and computational complexity is $\mathcal{O}(m^2)$
- m is asymptotically independent of total # data

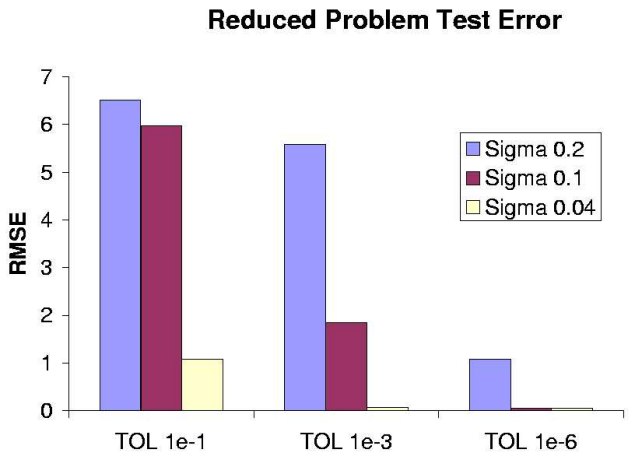
Toy Example: Sombrero

Approximating: $\sin \|\mathbf{x}\| / \|\mathbf{x}\|$, $\mathbf{x} \in [-10, 10]^2$. Training: 500 randomly drawn samples. RBF-kernel.

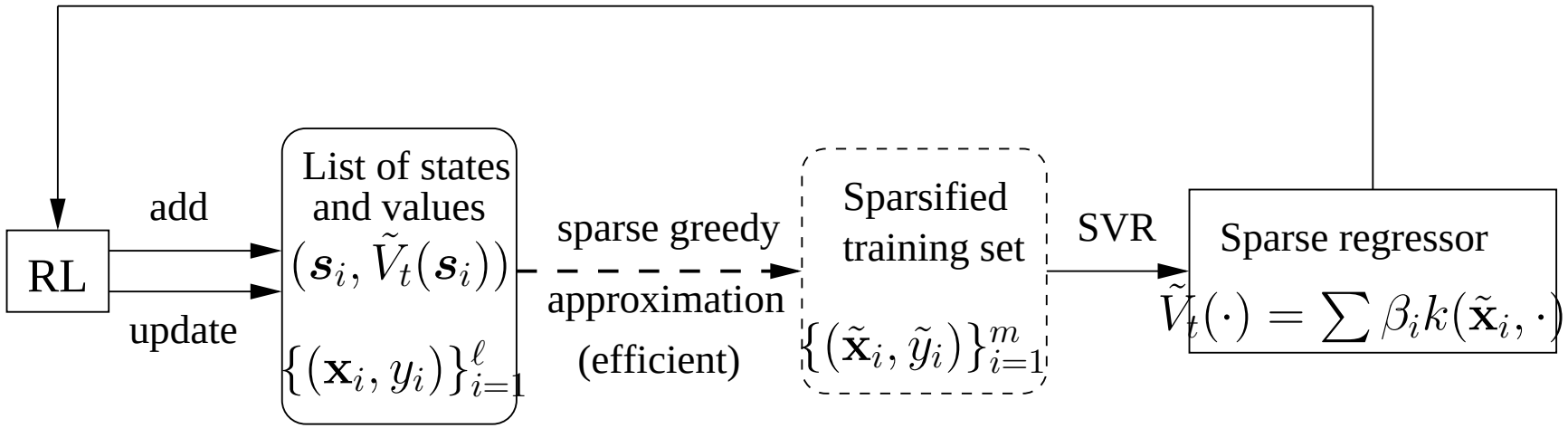






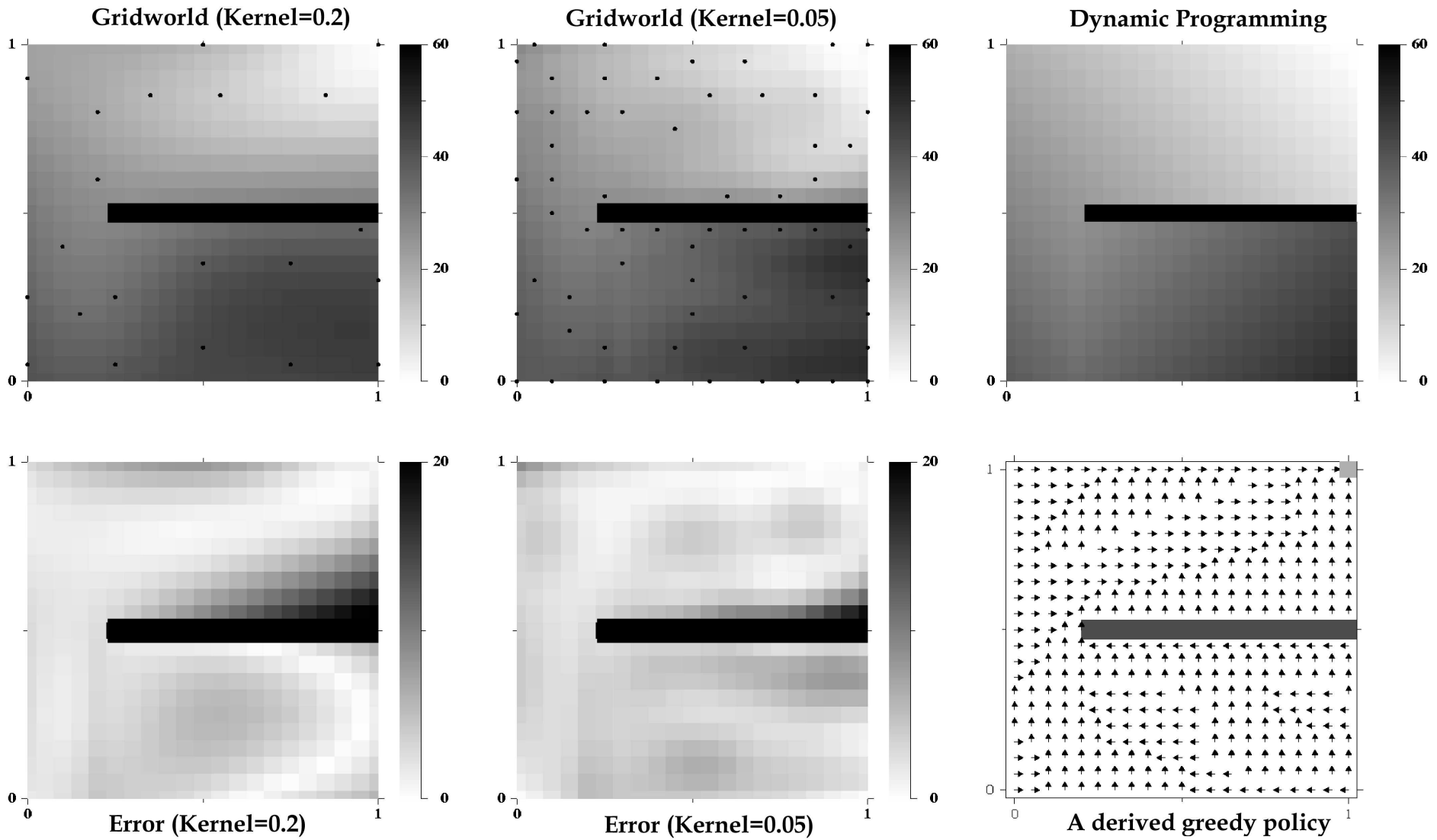


Putting everything together ...



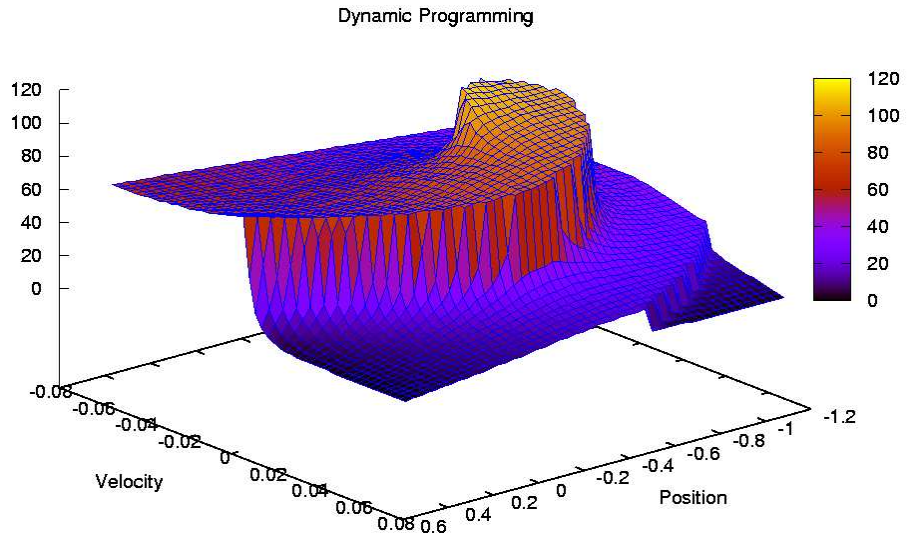
Experiment 1: Gridworld

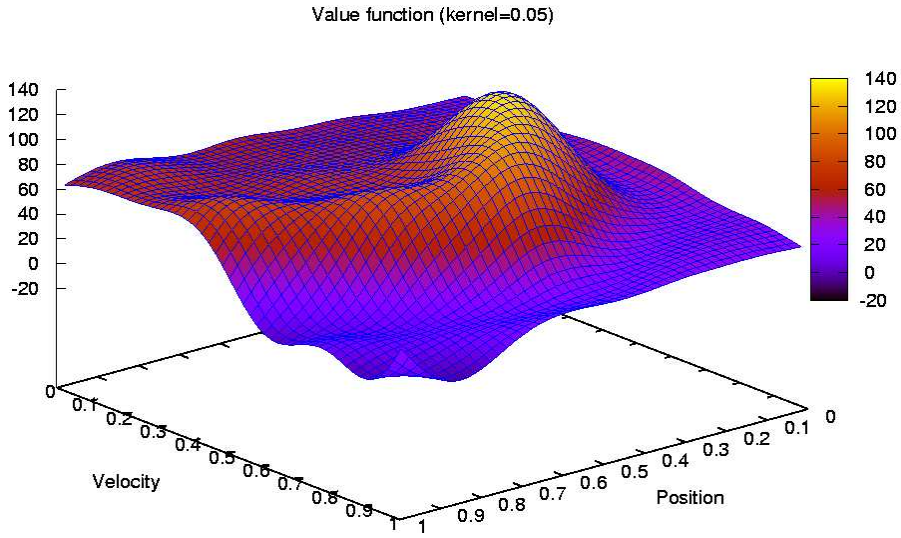
Goal: Test approximation quality in on-line RL (model-based)

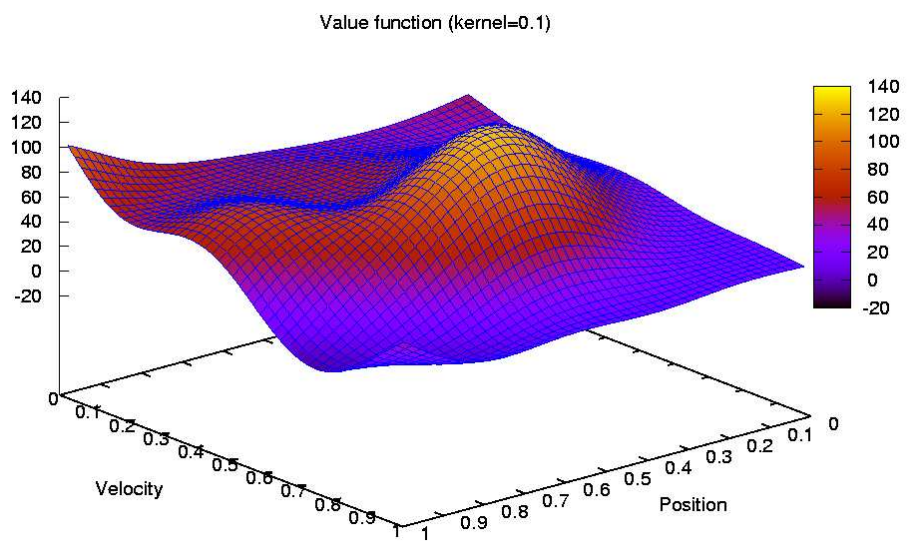


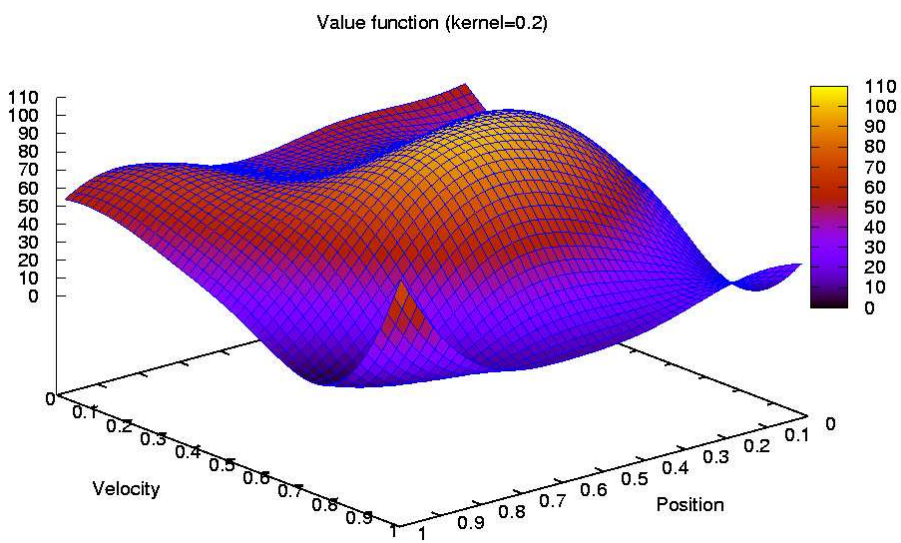
Experiment 2a: Mountain Car

Goal 1: Test approximation quality in on-line RL (model-based)



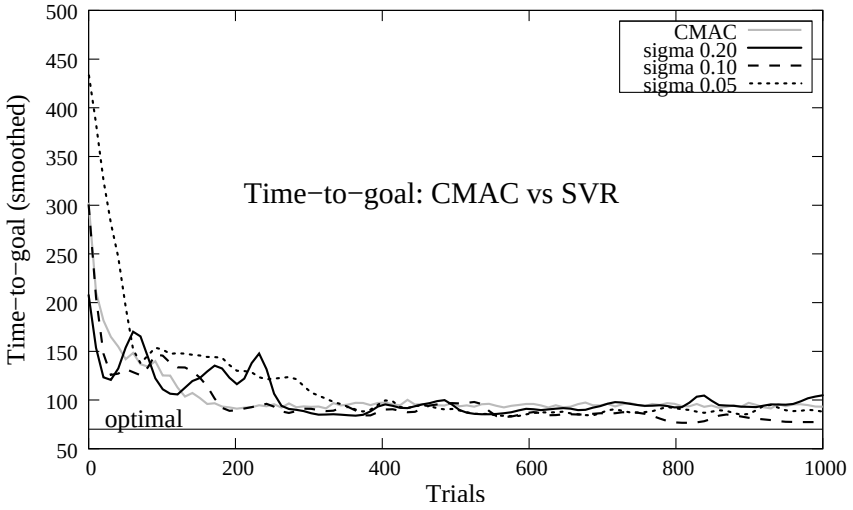


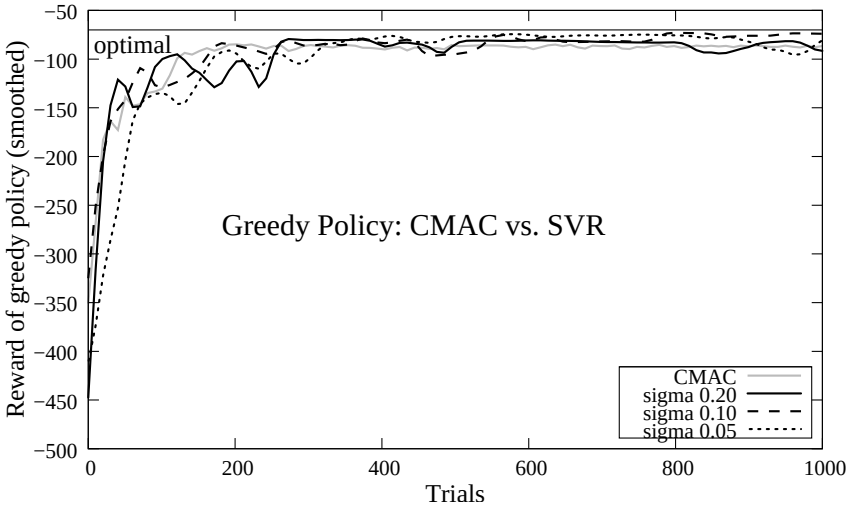


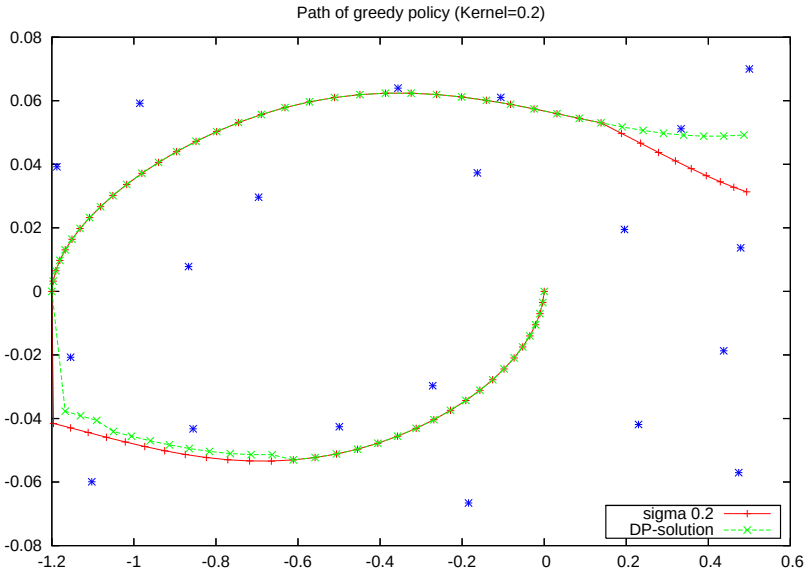


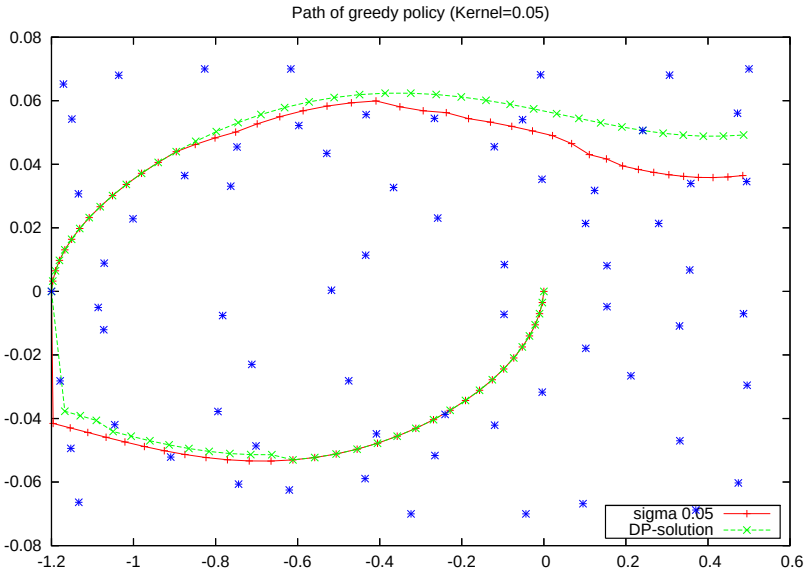
Experiment 2b: Mountain Car

Goal 2: Compare performance with tilecoding ($10 \times 10 \times 10$)









Conclusions

Summary: SVR with on-line RL is made possible by

1. Memorizing states+values as in instance-based architectures (on-line)
2. Building a sparsified training set (on-line)
3. Solving a reduced problem

Future work and some ideas:

- Other learning mechanisms, e.g. policy-iteration (batch updates to value function)
- More difficult tasks
- Minor (and major?) algorithmic improvements
- Sparsified training set could also be used in regularization networks or for placement of basis functions in RBF-networks
- Convergence?