Sparse Distributed Memories in a Bounded Metric State Space

Alexandre Bouchard-Côté

Supervisor: Doina Precup

Reasoning and Learning Lab, McGill University

Sponsored by: NSERC, McGill School of Computer Science.

Part 0

- Motivations for studying Sparse Distributed Memories (SDM) architectures [1].
- Description of the architecture.

[1] Kanerva, P. (1993). Sparse distributed memory and related models. In M. Hassoun (Ed.), Associated neural memories: Theory and implementation, 50-76. N.Y.: Oxford University Press.

The problem we are interested in:

- A Markov decision chain. Described by:
  - A specification $(S, A, \{P(s,\cdot)(a)\}_{a \in A})$ of the dynamic behavior of the environment,
  - An ordering of the policies derived with $(g : S \times A \times S \to \mathbb{R}, \gamma)$.
- Enough to find the optimal cost-to-go/state-action value functions of the problem.
- We know algorithms that can find these value functions (e.g. Q-learning).

Approximate RL

- When |S| is too large we want to approximate value functions instead.
- It brings new problems:
  - If nonlinear approximators are used, the convergence guarantees are lost.
  - If linear approximators are used, then some of the convergence results are recovered, but we get a new problem: the *curse of dimensionality*.
- SDM reconciles the best of both approaches.

What is a SDM architecture?

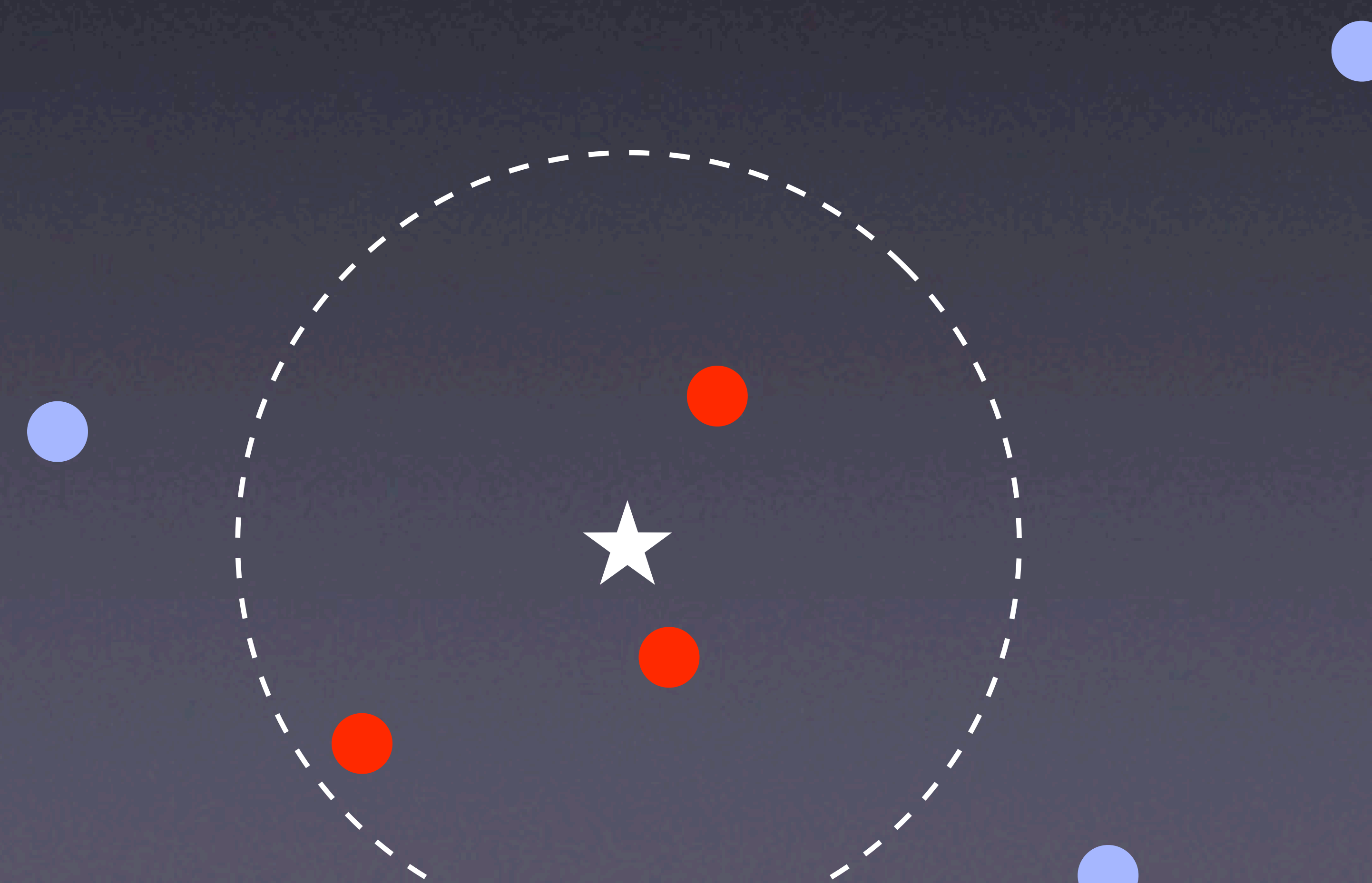
- Assumptions on S needed to define an SDM:
  - It must be a metric space:
    $d : S \times S \to \mathbb{R}^+$
  - It must be bounded (for concreteness).
- A SDM architecture is equipped with:
  - a *similarity function*
    $\sigma : S \times S \to [0, 1]$ s.t. $(1 - \sigma)$ is a metric on S
  - A set of *basis points* or *hard locations*
    $B := \{s_1, \ldots, s_K : s_i \in S\}$ with weights w_i.

Description of SDM based on: D. Precup, B. Ratitch. (2004). Sparse distributed memories for on-line, value-based reinforcement learning. ECML, 2004.

- When we want to approximate the value of the targeted function at a given point $s \in S$, we first find the set H of *actived locations*.

- Then the weights corresponding to the active locations are summed:

$$\frac{\sum \sigma(s_k, s)\ w_k}{\sum \sigma(s_k, s)}$$



- We can train the SDM architecture using the standard gradient descent update:
  $\Delta \mathbf{w} := \nabla \mathbf{g}\ [f(s) - g(s)]$
  where $g(\cdot)$ is the current approximation, $\nabla \mathbf{g}$, the gradient with respect to the weights and $f(s)$ is the training sample.

- Note that $(\nabla \mathbf{g})_i$ takes this simple form:

$$\frac{\sigma(s_i, s)}{\sum \sigma(s_k, s)}$$

Important observations:

1. It is a linear approximation architecture (good position for convergence results),
2. The *density* of the hard locations across the space need not be constant (we could put more hard locations on “important” regions of the state space).

Part 1: Convergence?

- Good surprise: a version of Q-learning converges w. pr. 1 when it uses an *interpolative* SDM architecture (approximate value iteration algorithms do not have such guarantees usually).
- Not so good surprise: an important result on the non-divergence of SARSA fails to apply in our case.
- Fortunately, TD(λ) still converges w. pr. 1.

Value iteration techniques

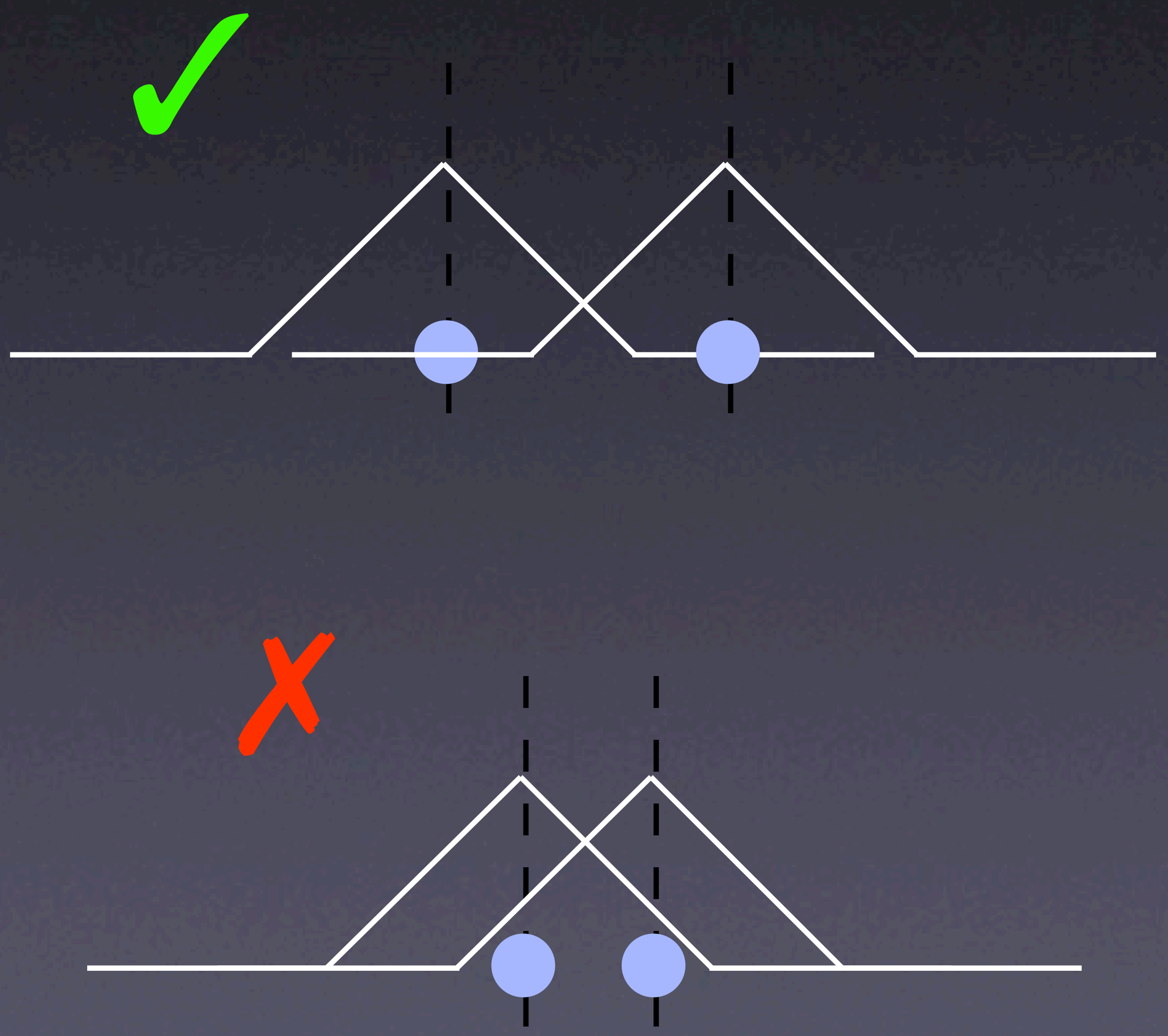
- The tabular version works by iteratively composing the bellman equation seen as a self-map in the space of value functions:
  $(T(J))(i) := \max \sum p_{ij}(a)(\ g(i, a, j) + J(j)\ )$
  $J_{k+1} := T(\ J_k\ ),\ J_k$ arbitrary element
  where the max is taken over $a \in A$, the sum is taken over $j \in S := \{1, \ldots, n\}$, $i \in S$, g is the cost function, and $p(\cdot)$, the transition probability matrices.

- By the Banach Fixed Point Theorem, the tabular version converges (in complete spaces)

- In the case of approximate versions of the value iteration algorithm we don’t have such a convergence guarantee. Even worse: it is proven that approximate Q-learning, a state-action value iteration algorithm, can diverge (even with some linear approximators).

- However, for interpolative SDM, there is a special convergence result that applies...

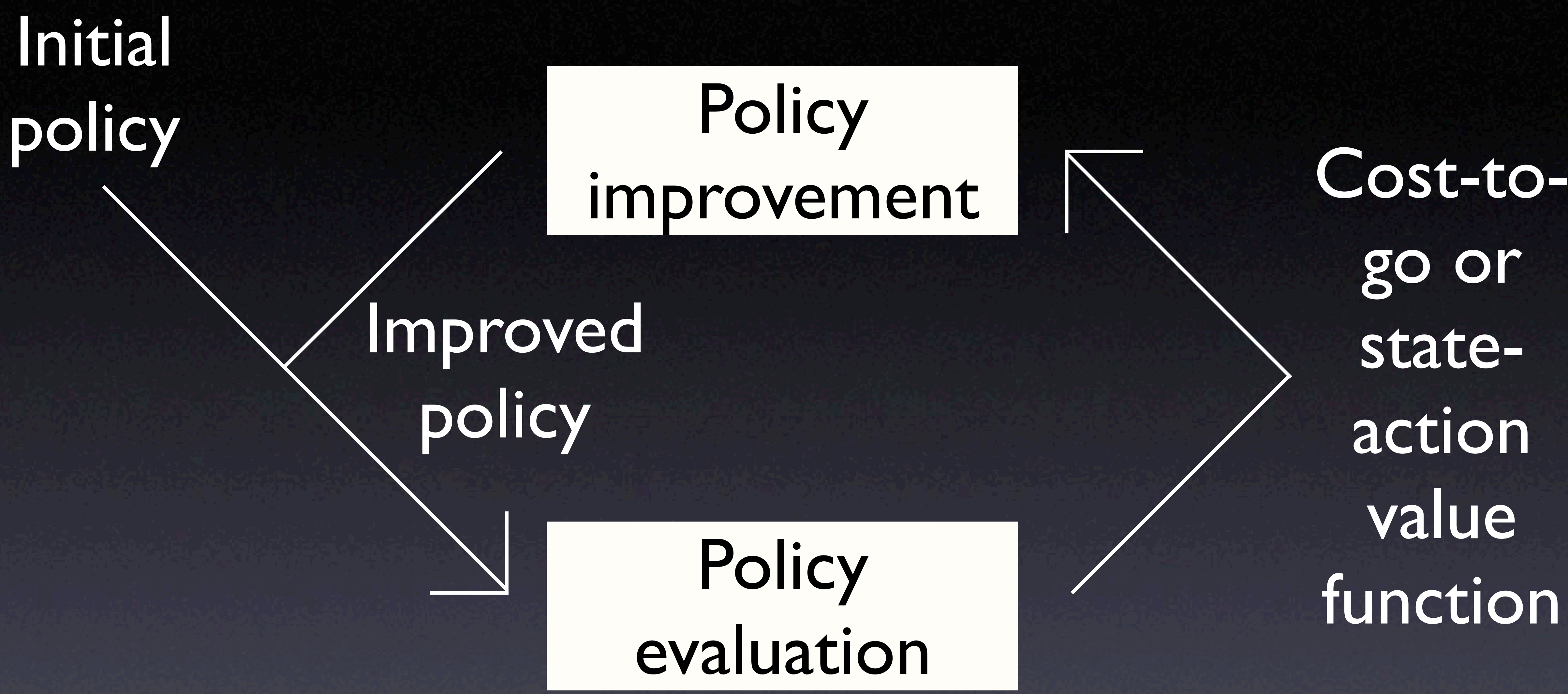
- Definition of an interpolative SDM.
- Example: symmetric triangular functions



- Theorem [3]: convergence of a form of Q-learning for interpolative approximators. The main assumptions of this theorem:
    - The approximator must be an interpolative non-expansion.
    - The set of states must be a Polish space (the homeomorphic image of a complete and separable metric space).
    - The exploration policy must be stationary... can we relax this condition?

[3] C. Szepesvári, W. Smart (2004).
Interpolation-based Q-learning. ECML, 2004.

SARSA: A Policy Iteration Method



- SARSA is an optimistic algorithm. This makes the analysis of its convergence difficult.
- There is a non-divergence result for the case of a finite state space.

- The proof of this theorem [4] uses the following facts to build a region of convergence:
  - If the policy were not changed at each iteration, the weights would converge to a fixed point,
  - There are finitely many policies the algorithm can consider.
- The second argument clearly does not hold for an arbitrary metric space.
- On the other hand, I wrote a proof based on the one given by Bertsekas and Tsitsiklis that generalizes the convergence of TD(λ) in *general* state spaces.

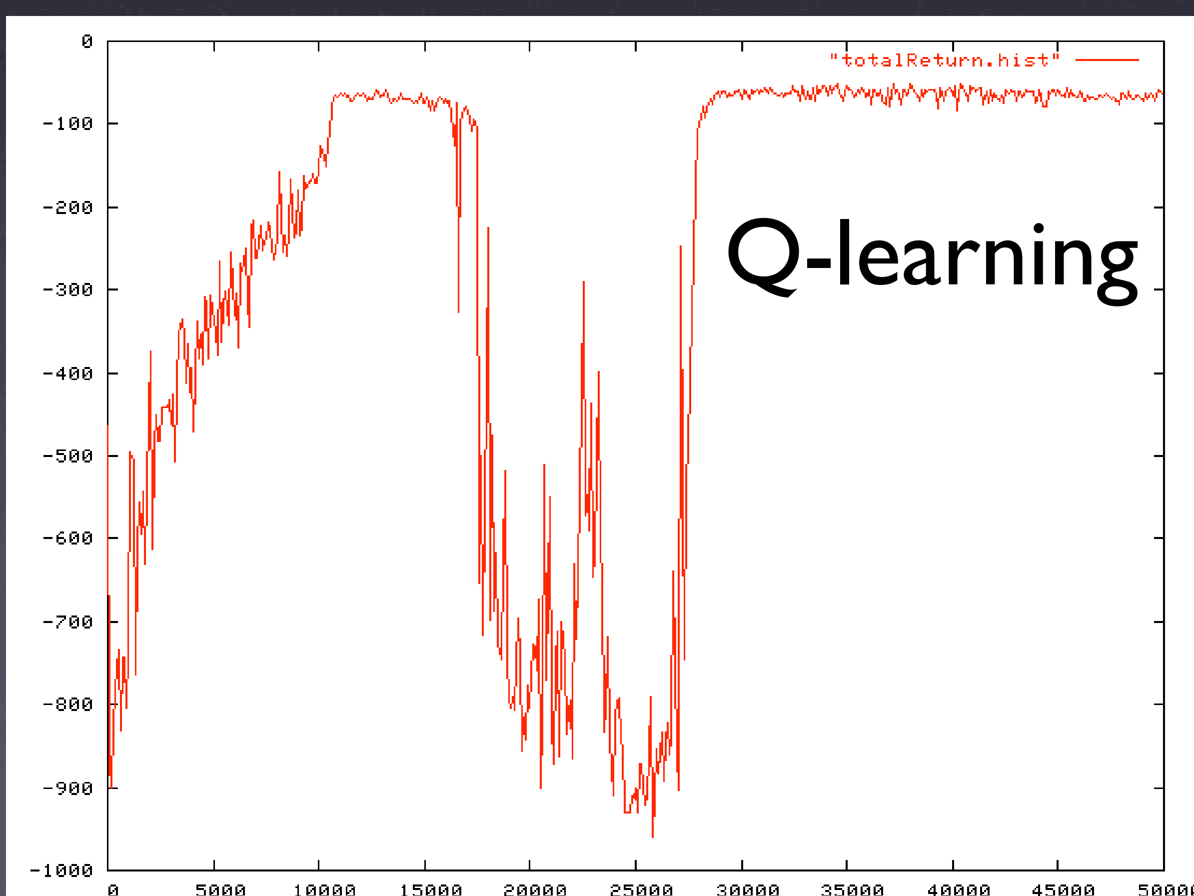
[4] G. Gordon. (2000). Reinforcement Learning with Function Approximation Converges to a Region. NIPS, 2000.

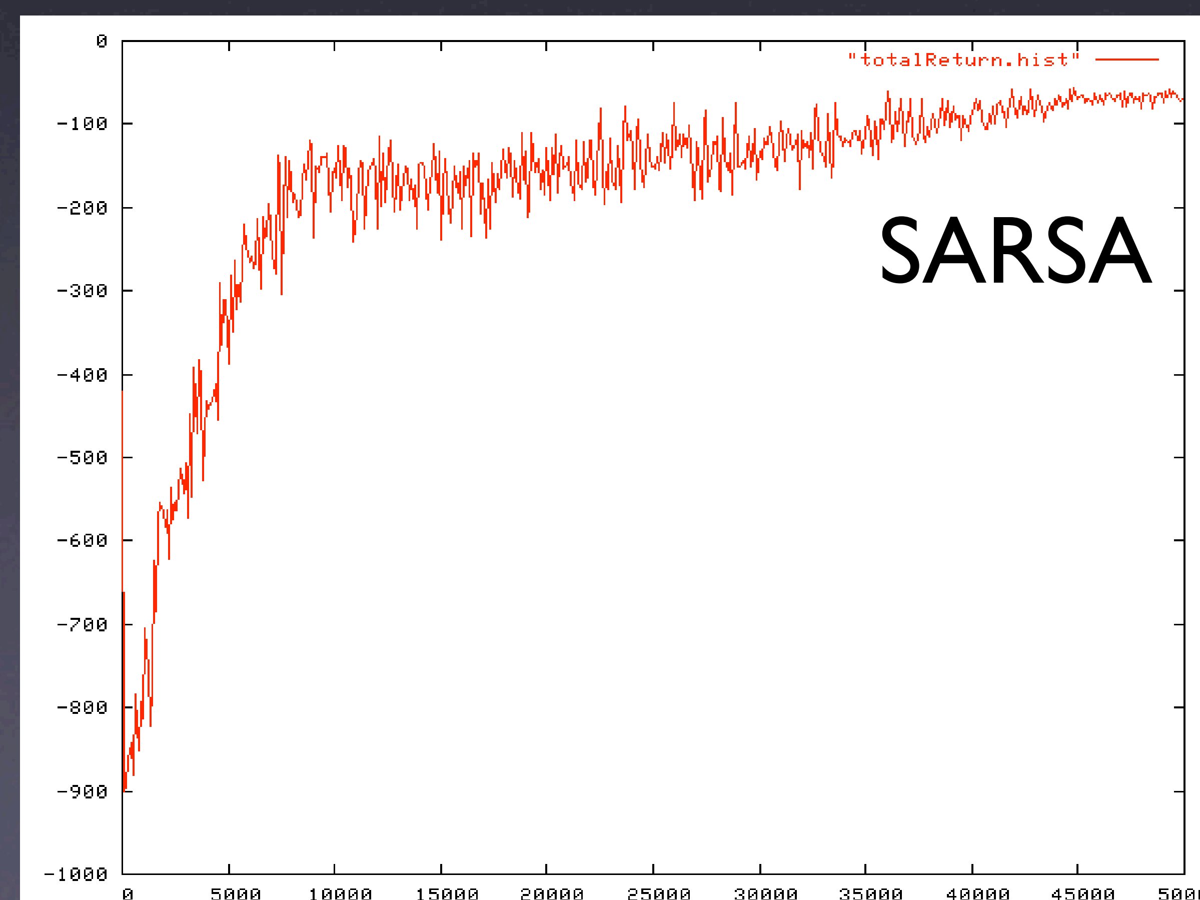
Part 2: Exorcising the Curse

- Empirical behavior on a low-dimensional problem of an implementation of the version of Q-learning and SARSA that we discussed.
- Design of a specialized data structure for the storage and retrieval of hard locations.
- Potential extensions to SDM architectures and future work.

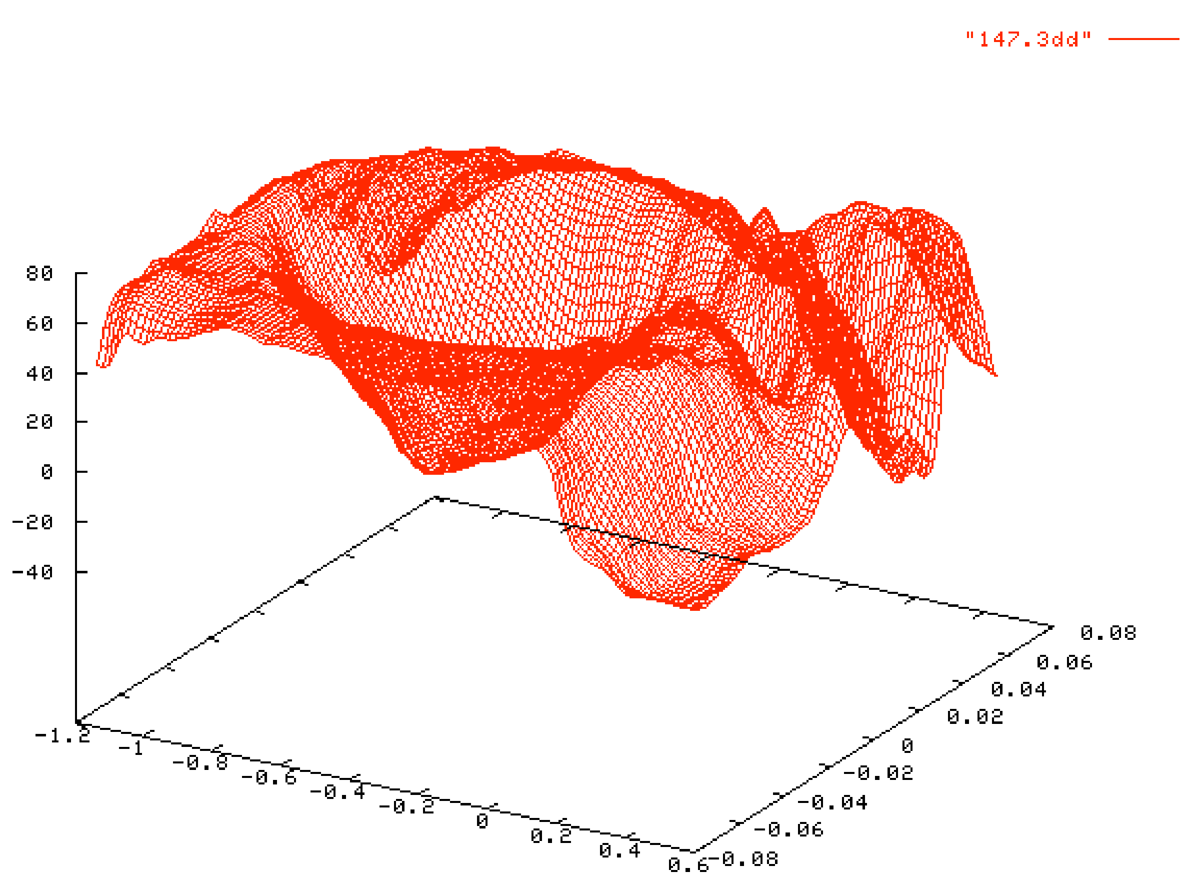
Convergence in practice of SARSA and Q-learning

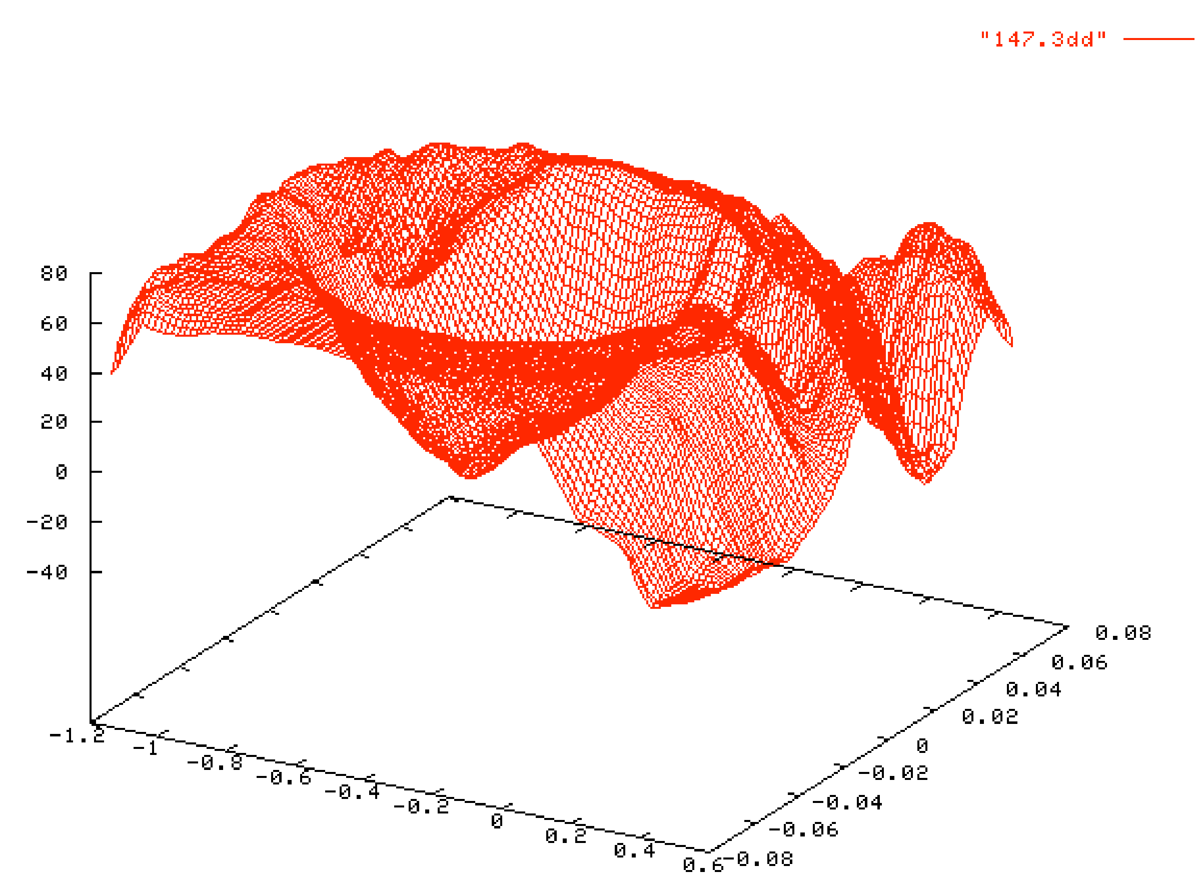
- The environment: the mountain car domain.
- However... the Q-learning algorithm suffers relatively often of dramatic instabilities:





- Potential explanations of this phenomenon:
  - Could be caused by the discontinuities introduced by the *max* operator,
  - or by the stationary exploration policy.
- Conclusion: policy iteration is preferred.





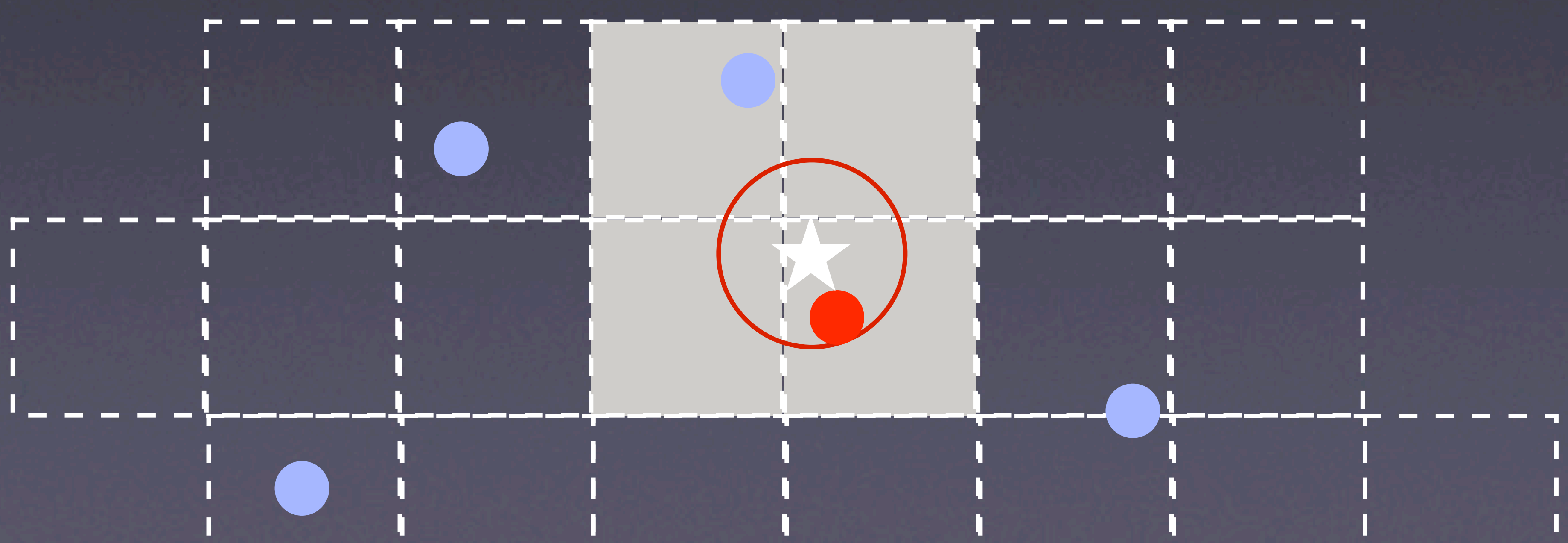
Dynamic allocation algorithms in practice

- It is possible to use the first phases of learning to construct the set of hard locations so that it has desirable properties [5].
- However, this algorithm involves a large amount of insertion/deletion of hard locations:
- A specialized data structure is necessary if we want our method to scale.

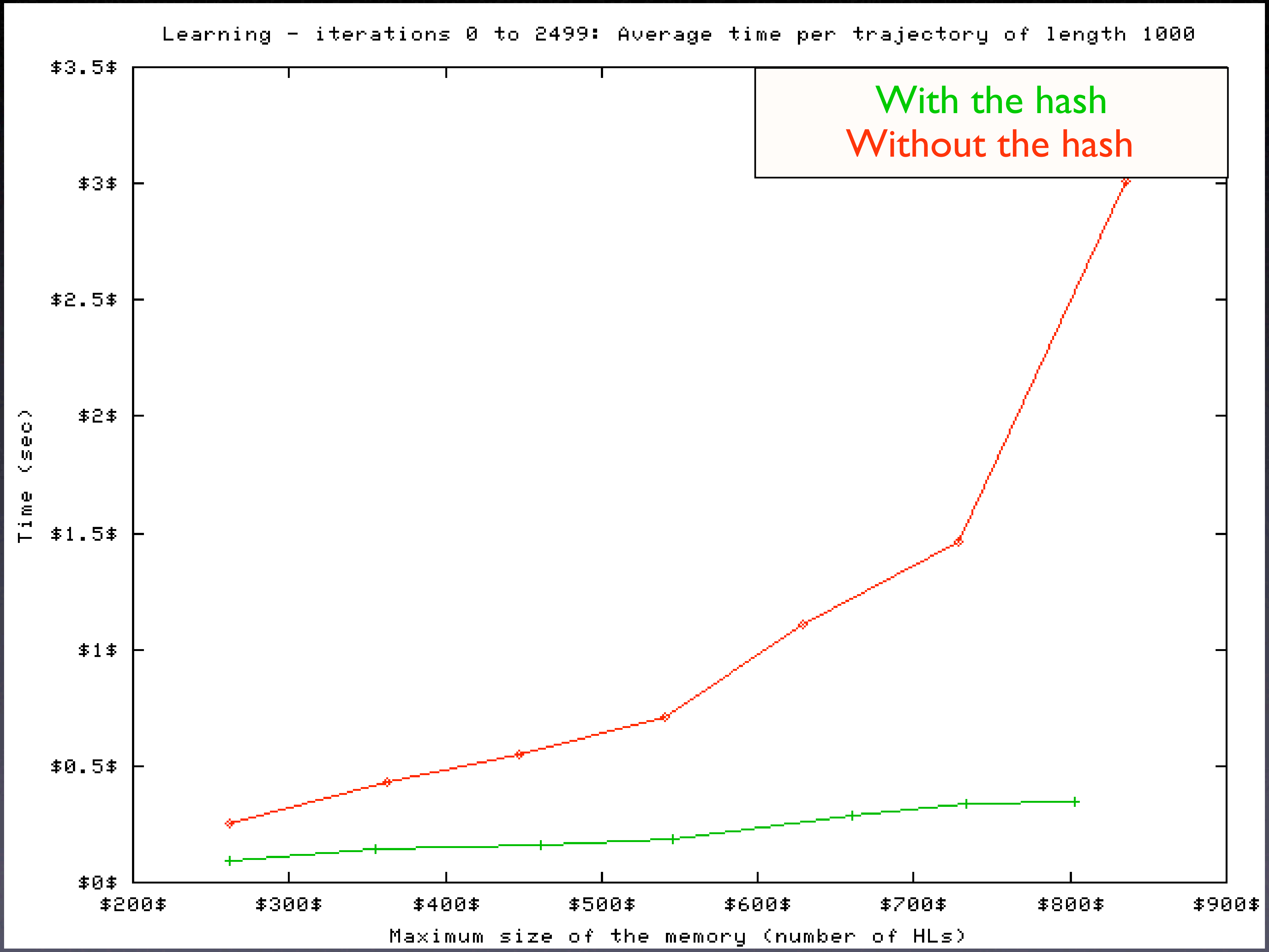
[5] D. Precup, B. Ratitch. (2004). Sparse distributed memories for on-line, value-based reinforcement learning. ECML, 2004.

- A hash-based data structure for interpolative SDM's in a finite dimensional vector space. The idea: use the fact that there is a δ such that for all set of hard locations H:
  $d(x_1, x_2) \geq \delta \Rightarrow \sigma(x_1, x_2) = 0 \quad \forall\, x_{1,2} \in H$

- Partition the space into cells of length 2δ in each dimension. Only the cells intersecting the activated region of the input location need to be examined.



- The data structure works well in practice:



Extensions to SDM:

- A sequence of *decreasing* σ_i instead of a constant σ.
  Motivation: automatic radius selection, fast learning with a good asymptotic resolution.

- Attach similarity functions σ_{x_i} to hard locations instead of having a global σ (a new data structure would be needed using reversed indexing).
  Motivation: an architecture that uses characteristics of the approximated function to distribute hard locations.