

Approximate Dynamic Programming with Gaussian Processes

Marc Deisenroth
mpd37@cam.ac.uk



CSML Seminar
University College London

February 14, 2008

Outline

- 1 Introduction to Reinforcement Learning
- 2 Gaussian Processes for Regression
- 3 Gaussian Process Dynamic Programming (GPDP)

Outline

- 1 Introduction to Reinforcement Learning
- 2 Gaussian Processes for Regression
- 3 Gaussian Process Dynamic Programming (GPDP)

What is Reinforcement Learning?

A possible description

Reinforcement learning is *learning* what to do. [...] The learner is not told which actions to take [...], but instead must *discover* which actions yield the most *reward* by *trying* them. (Sutton and Barto, 1998)

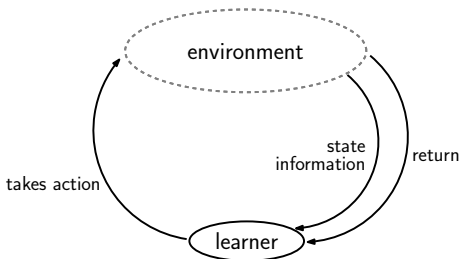


borrowed from *Science* magazine

reinforcement learning is a general framework

- to explain animal/human behavior
- to determine good strategies

RL Setup



usually

- environment **not** (exactly) known
- get return signal (loss/reward) and information about state of environment
- overall objective: optimize long-term performance
- agent should do well

A Bit More Formal

environment usually modeled as Markov Decision Process (MDP)

- set \mathcal{X} of states
- set \mathcal{U} of actions
- transition probability
- return signal g

Markov assumption:

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \dots, \mathbf{x}_0, \mathbf{u}_0) = p(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_{k-1})$$

A Bit More Formal

environment usually modeled as Markov Decision Process (MDP)

- set \mathcal{X} of states
- set \mathcal{U} of actions
- transition probability
- return signal g

Markov assumption:

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \dots, \mathbf{x}_0, \mathbf{u}_0) = p(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_{k-1})$$

objective

- minimize the expected long-term loss

$$L(\mathbf{x}_0) = \mathbb{E} \left[g_{\text{term}}(\mathbf{x}_N) + \sum_{k=0}^{N-1} g(\mathbf{x}_k, \mathbf{u}_k) \right]$$

for all $\mathbf{x}_0 \in \mathcal{X}$.

A Bit More Formal

environment usually modeled as Markov Decision Process (MDP)

- set \mathcal{X} of states
- set \mathcal{U} of actions
- transition probability
- return signal g

Markov assumption:

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \dots, \mathbf{x}_0, \mathbf{u}_0) = p(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_{k-1})$$

objective

- minimize the expected long-term loss

$$L(\mathbf{x}_0) = \mathbb{E} \left[g_{\text{term}}(\mathbf{x}_N) + \sum_{k=0}^{N-1} g(\mathbf{x}_k, \mathbf{u}_k) \right]$$

for all $\mathbf{x}_0 \in \mathcal{X}$.

- functional dependence $\mathbf{u} = \pi(\mathbf{x})$
 → find optimal **policy** π^*
- minimal expected long-term loss: **value function** V_N^*

Dynamic Programming

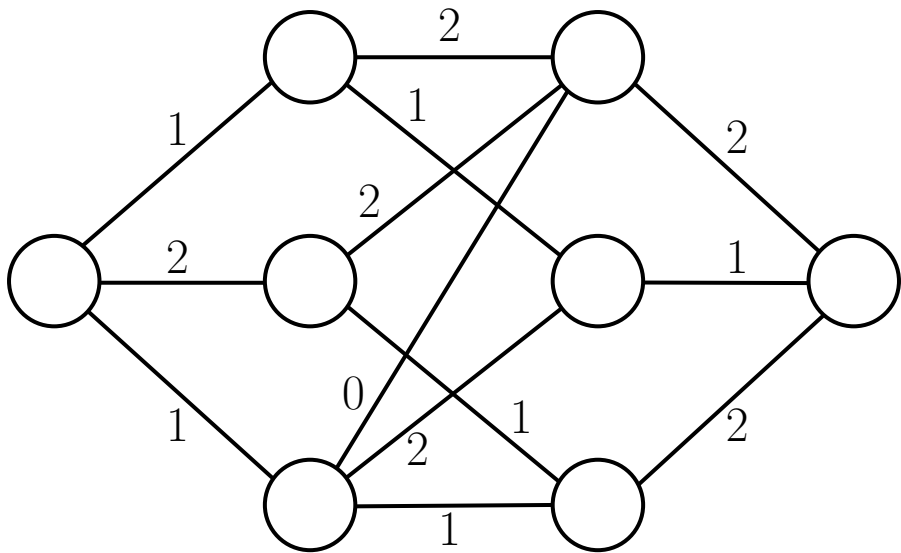
from here: planning in known MDPs

- solve the optimization problem with **dynamic programming**
- main idea: build solution to subproblem of length k based on solution to subproblem of length $k - 1$

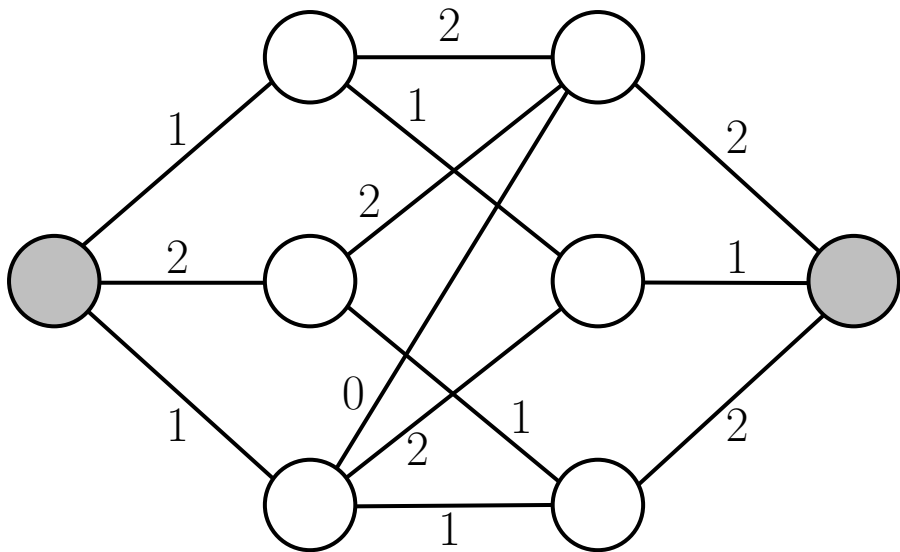
▶ show demo

▶ skip demo

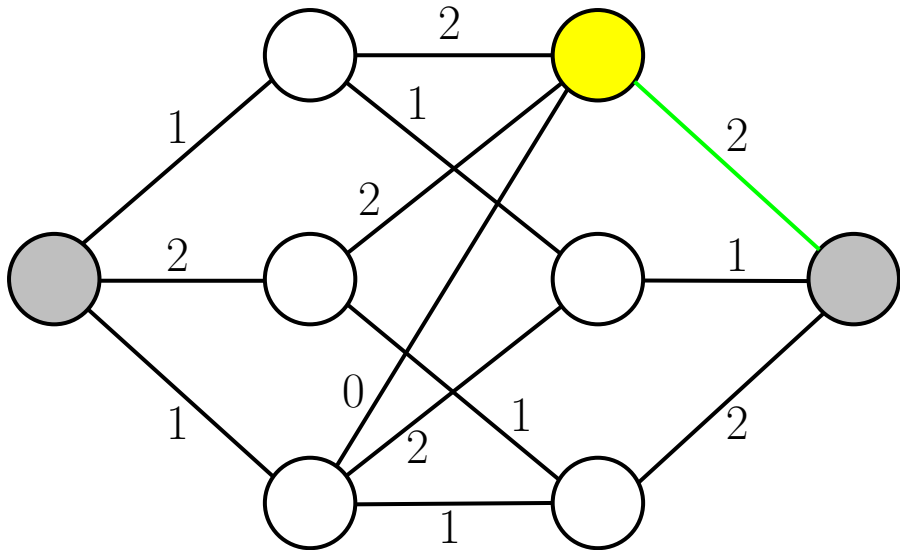
Demo: DP



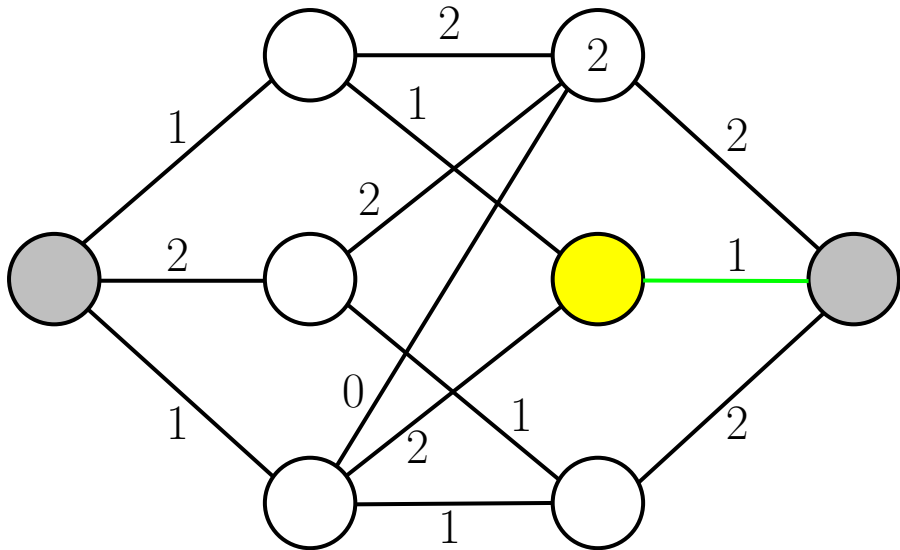
Demo: DP



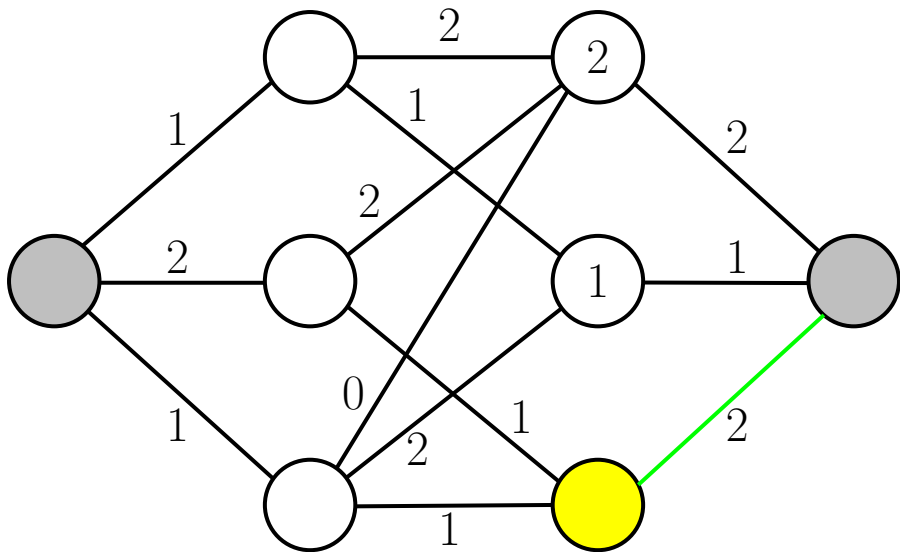
Demo: DP



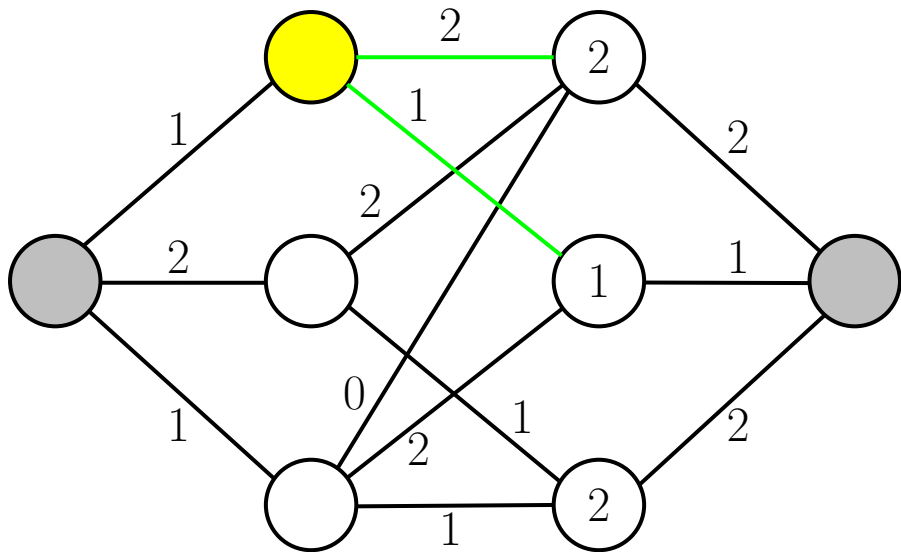
Demo: DP



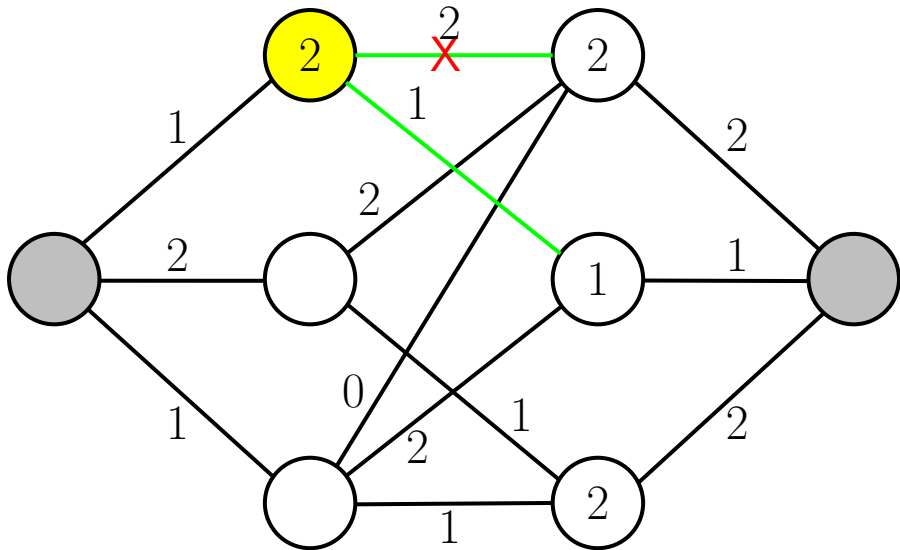
Demo: DP



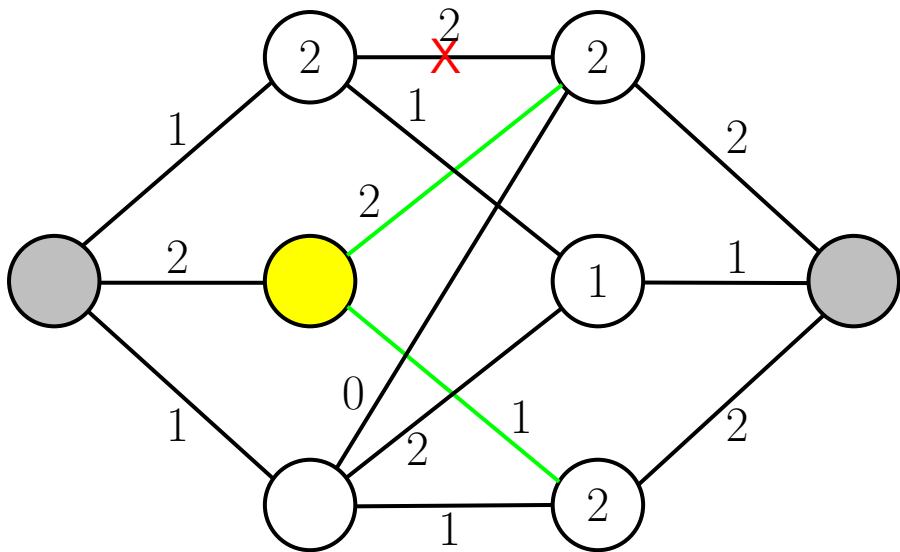
Demo: DP



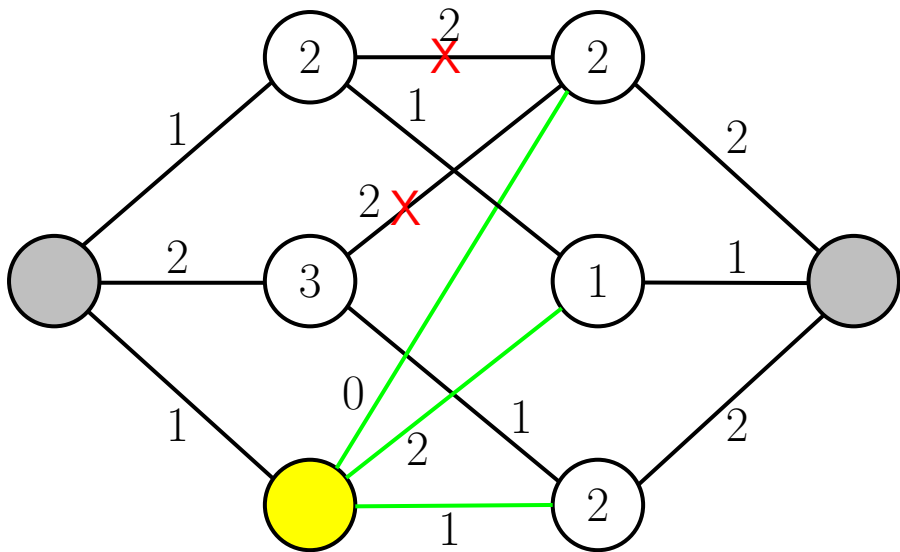
Demo: DP



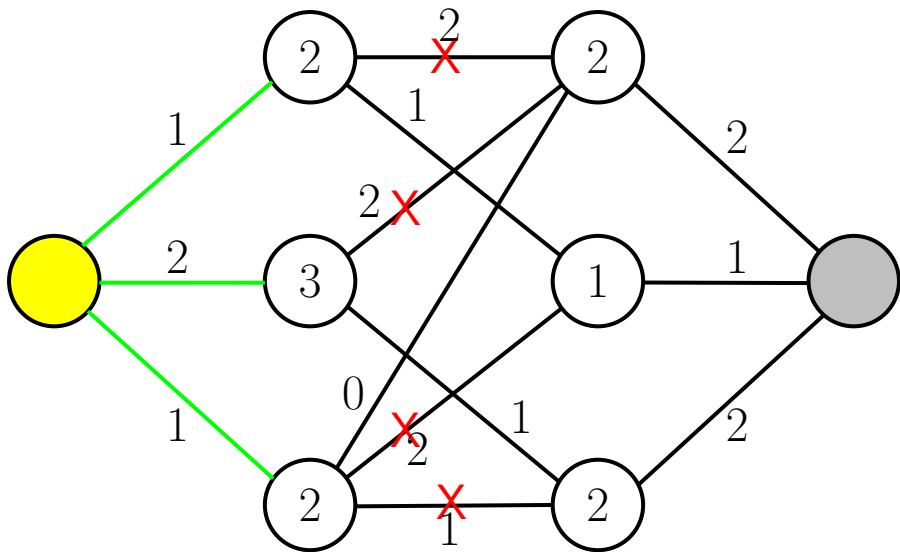
Demo: DP



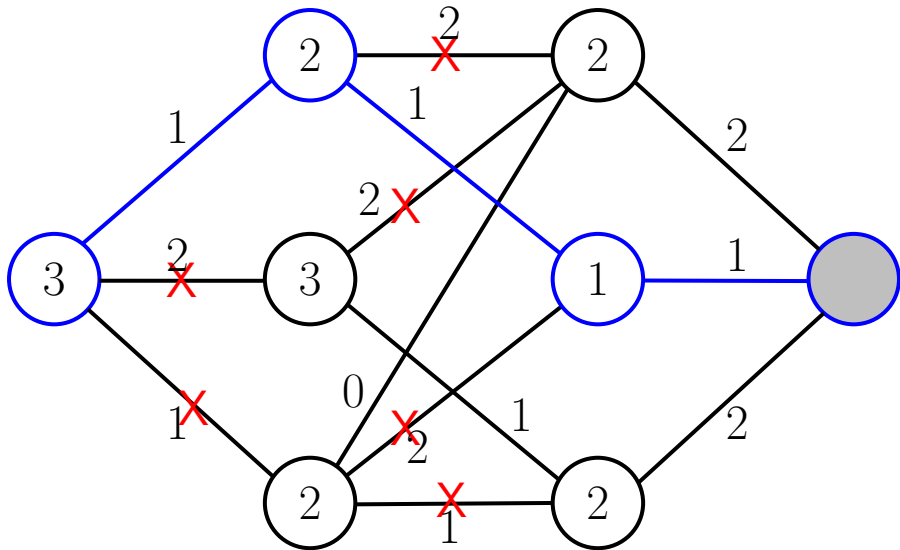
Demo: DP



Demo: DP



Demo: DP



Algorithm: Dynamic Programming

DP algorithm:

$$V_0^*(\mathcal{X}) = g_{\text{term}}(\mathcal{X})$$

▷ terminal return

for $k = 1$ to N **do**

▷ for all subproblems of length k

for all states $\mathbf{x}_i \in \mathcal{X}$ **do**

$$Q_k(\mathbf{x}_i, \mathcal{U}) = g(\mathbf{x}_i, \mathcal{U}) + \mathbb{E} [V_{k-1}^*(\mathbf{x}') | \mathbf{x}_i, \mathcal{U}]$$

$$V_k^*(\mathbf{x}_i) = \min_{\mathbf{u} \in \mathcal{U}} Q(\mathbf{x}_i, \mathbf{u})$$

$$\pi_k^*(\mathbf{x}_i) = \arg \min_{\mathbf{u} \in \mathcal{U}} Q_k(\mathbf{x}_i, \mathbf{u})$$

end for

end for

return $V_N^*, \pi_N^*(\mathcal{X})$

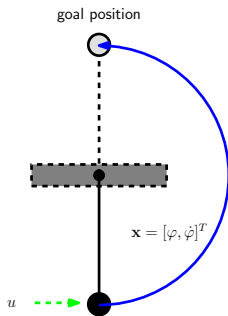
→ build solution for stage k based on solution of stage $k - 1$

Running Example: Underpowered Pendulum Swing Up

comprehensible system with 1 degree of freedom
(Atkeson and Schaal, 1997)

goal:

- 1 start from downward position $[-\pi, 0]^T$
- 2 bring the pendulum to the upright position $[0, 0]^T$
- 3 balance it there

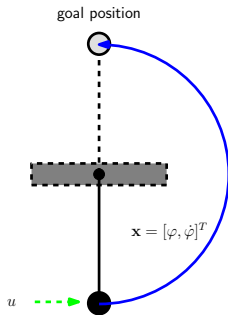


Running Example: Underpowered Pendulum Swing Up

comprehensible system with 1 degree of freedom
(Atkeson and Schaal, 1997)

goal:

- 1 start from downward position $[-\pi, 0]^T$
 - 2 bring the pendulum to the upright position $[0, 0]^T$
 - 3 balance it there
- constraint: **underpowered** system, i.e., maximum torque is insufficient to bring the pole to the upright position: $u \in [-5, 5] \text{Nm}$
 - time-discretized system (sampling every 200 ms)



Optimal Policy with Trajectory (Discretized System)

DP with discretization of state and action spaces (7.5×10^7 points)

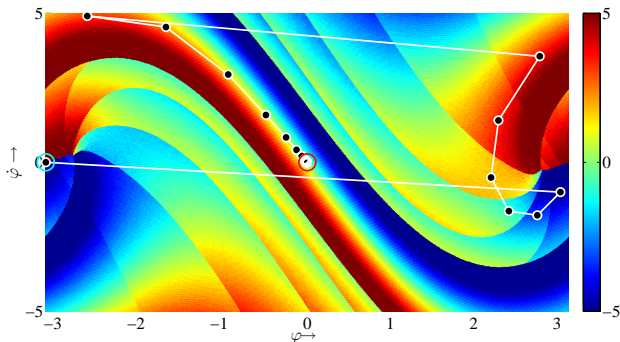


Figure: optimal policy, sampling rate 5 Hz

Trajectory (Discretized System)

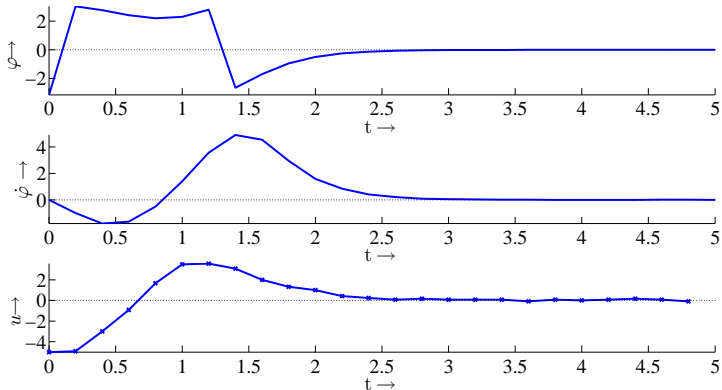


Figure: System behavior (DP solution)

Continuous Domains

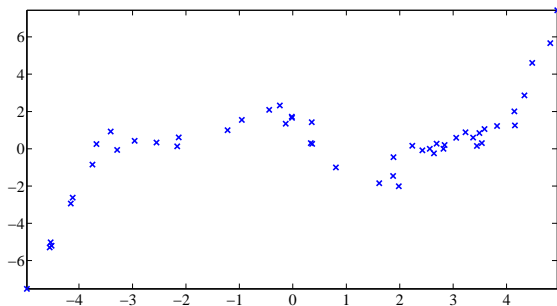
- DP for **continuous state and action spaces** only analytically tractable in special cases (e.g., LQR)
- spatial discretization \rightarrow **curse of dimensionality**
- value function approximation (NN, RBFs, linear,...)
 \rightarrow continuous state spaces
see e.g. (Bertsekas and Tsitsiklis, 1996; Ormoneit and Šen, 2002; 2005; Riedmiller, 2005)

here: **Gaussian process regression** (Rasmussen and Williams, 2006)

Outline

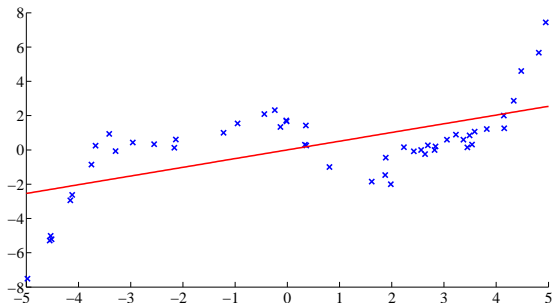
- 1 Introduction to Reinforcement Learning
- 2 Gaussian Processes for Regression
- 3 Gaussian Process Dynamic Programming (GPDP)

Regression Problem



inputs \mathbf{X} , corresponding noisy observations $\mathbf{y} = f(\mathbf{X}) + \varepsilon$
➔ infer f (everywhere)

Regression Problem



inputs \mathbf{X} , corresponding noisy observations $\mathbf{y} = f(\mathbf{X}) + \varepsilon$

➔ infer f (everywhere)

- linear regression
- NN, RBFs, polynomials, ...
- here: Gaussian processes
(nonparametric Bayesian regression method: few underlying assumptions, data-dependent model)

Gaussian Process

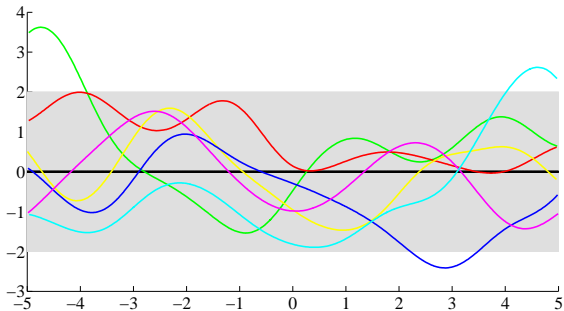
Definition (Rasmussen and Williams, 2006)

A **Gaussian process** is a collection of random variables, any finite number of which have joint Gaussian distributions.

- consider a function f as an infinitely long vector $[f(\mathbf{x}_1), f(\mathbf{x}_2), \dots]^T$
- GP as **distribution over functions** (generalization of the Gaussian distribution)
- all required computations are broken down to finite-dimensional problems
 ➔ manipulate multivariate Gaussians

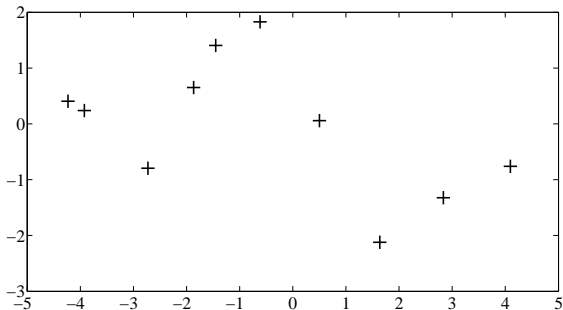
Pictorial Bayesian Inference with GPs

- 1 start with our prior beliefs



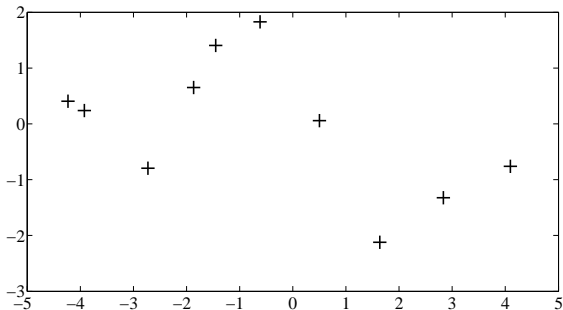
Pictorial Bayesian Inference with GPs

- 1 start with our prior beliefs
- 2 observe some (noisy) function values $y_i = f(\mathbf{x}_i) + \varepsilon$ for some input locations $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ (training set)



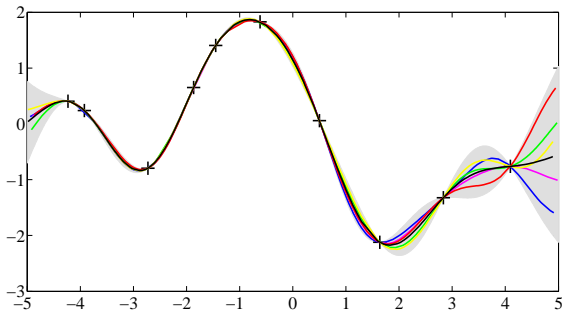
Pictorial Bayesian Inference with GPs

- 1 start with our prior beliefs
- 2 observe some (noisy) function values $y_i = f(\mathbf{x}_i) + \varepsilon$ for some input locations $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ (training set)
- 3 do Bayesian inference with GPs



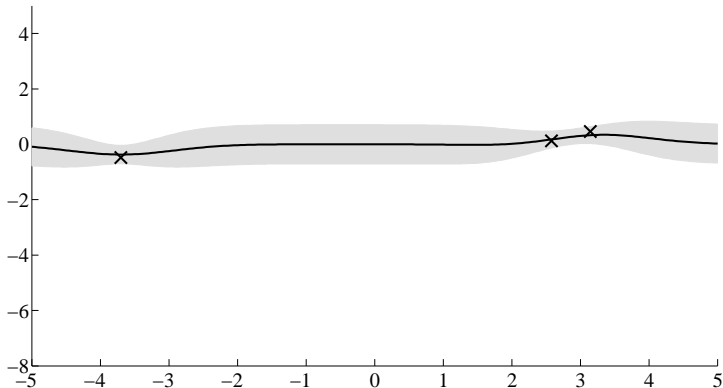
Pictorial Bayesian Inference with GPs

- 1 start with our prior beliefs
- 2 observe some (noisy) function values $y_i = f(\mathbf{x}_i) + \varepsilon$ for some input locations $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ (training set)
- 3 do Bayesian inference with GPs
- 4 infer distribution of function values for arbitrary (test) input locations \mathcal{X}_*
 → posterior specified by mean and covariance (conditioned on the training data)



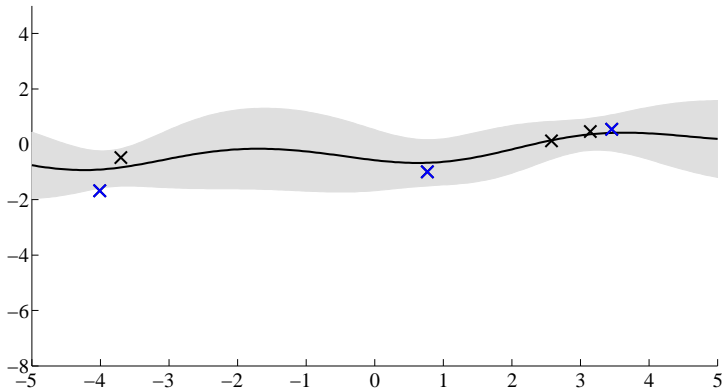
Influence of Data

- more data \rightarrow more confident predictions
- infinite data (noise-free observations): posterior mean equals latent function (if prior assumptions are not wrong)



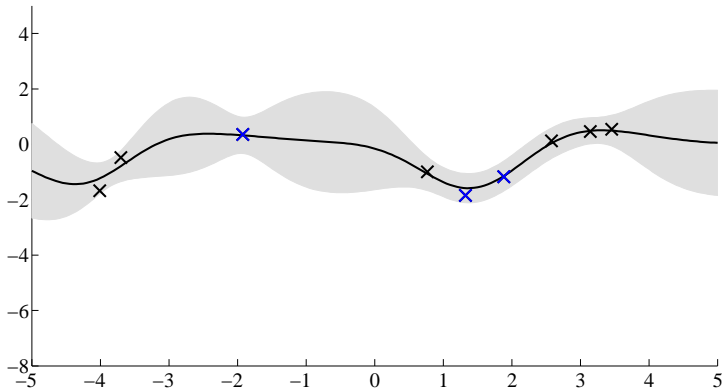
Influence of Data

- more data \rightarrow more confident predictions
- infinite data (noise-free observations): posterior mean equals latent function (if prior assumptions are not wrong)



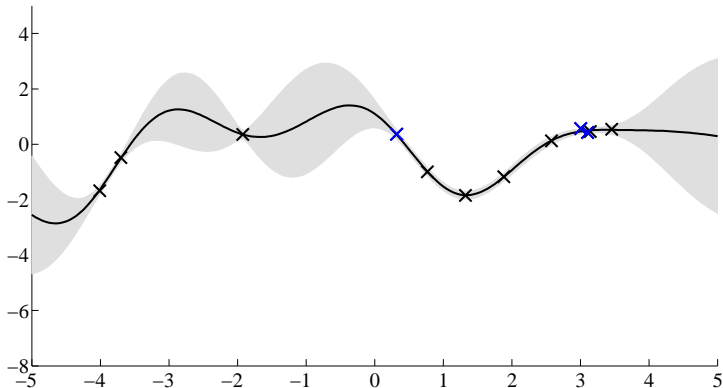
Influence of Data

- more data \rightarrow more confident predictions
- infinite data (noise-free observations): posterior mean equals latent function (if prior assumptions are not wrong)



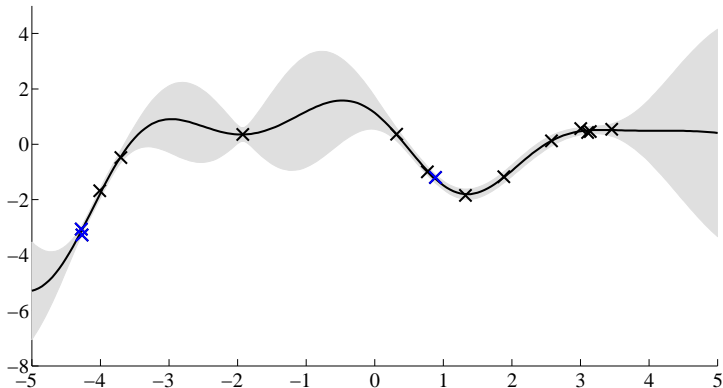
Influence of Data

- more data \rightarrow more confident predictions
- infinite data (noise-free observations): posterior mean equals latent function (if prior assumptions are not wrong)



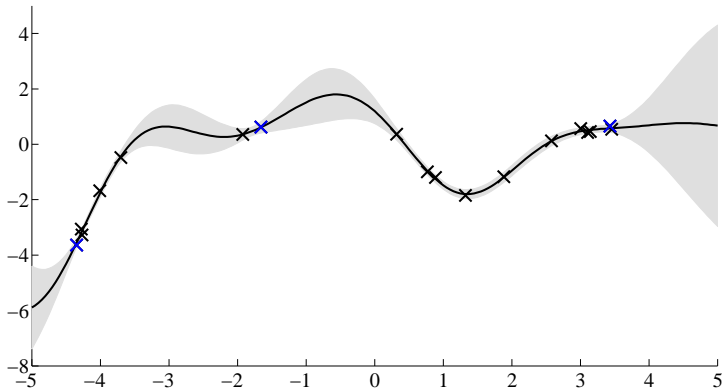
Influence of Data

- more data \rightarrow more confident predictions
- infinite data (noise-free observations): posterior mean equals latent function (if prior assumptions are not wrong)



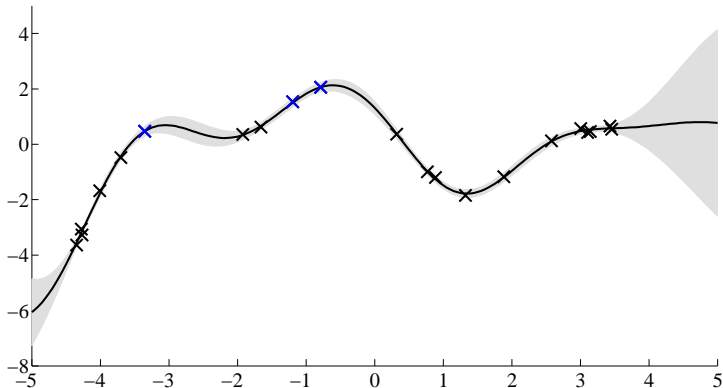
Influence of Data

- more data \rightarrow more confident predictions
- infinite data (noise-free observations): posterior mean equals latent function (if prior assumptions are not wrong)



Influence of Data

- more data \rightarrow more confident predictions
- infinite data (noise-free observations): posterior mean equals latent function (if prior assumptions are not wrong)



Outline

- 1 Introduction to Reinforcement Learning
- 2 Gaussian Processes for Regression
- 3 Gaussian Process Dynamic Programming (GPDP)

Joint Work

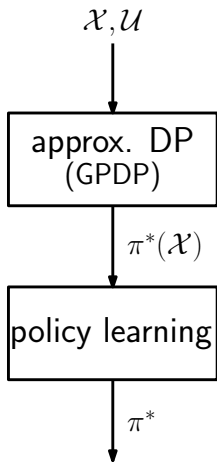


Carl Edward Rasmussen
University of Cambridge



Jan Peters
Max Planck Institute for
Biological Cybernetics

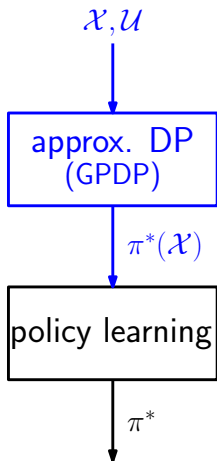
Structure



given:

- (small) set of states
 $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^{n_x}$
- (small) set of actions
 $\mathcal{U} = \{\mathbf{u}_1, \dots, \mathbf{u}_m\} \subset \mathbb{R}^{n_u}$
- ① generalize DP to continuous state spaces based on evaluations at \mathcal{X}, \mathcal{U}
- ② generalize state feedbacks to entire state space

1.: GPDP



- model value functions by means of GPs (in any iteration of DP)
- generalization to continuous domains
- get optimal set of actions $\pi^*(\mathcal{X})$ for (finite) training set \mathcal{X}

[▶ next step](#)

Algorithm: GPDP

assume known deterministic system function $\mathbf{x}' = f(\mathbf{x}, \mathbf{u})$

$$V_0^*(\mathcal{X}) = g_{\text{term}}(\mathcal{X})$$

▷ terminal return

for $k = 1$ to N **do**

▷ start recursion

for all states $\mathbf{x}_i \in \mathcal{X}$ **do**

$$Q_k(\mathbf{x}_i, \mathcal{U}) = g(\mathbf{x}_i, \mathcal{U}) + \mathbb{E}[V_{k-1}^*(\mathbf{x}') | \mathbf{x}_i, \mathcal{U}]$$

$$\boldsymbol{\pi}_k^*(\mathbf{x}_i) = \arg \min_{\mathbf{u} \in \mathcal{U}} Q_k(\mathbf{x}_i, \mathbf{u})$$

▷ minimize

$$V_k^*(\mathbf{x}_i) = \min_{\mathbf{u} \in \mathcal{U}} Q_k(\mathbf{x}_i, \mathbf{u})$$

end for

end for

return $V_N^*, \boldsymbol{\pi}_N^*(\mathcal{X})$

Algorithm: GPDP

assume known deterministic system function $\mathbf{x}' = f(\mathbf{x}, \mathbf{u})$

$$V_0^*(\mathcal{X}) = g_{\text{term}}(\mathcal{X})$$

$$V_0^* \sim \mathcal{GP}_v(m_v, k_v)$$

▷ terminal return

▷ GP models V_0^*

for $k = 1$ to N **do**

▷ start recursion

for all states $\mathbf{x}_i \in \mathcal{X}$ **do**

$$Q_k(\mathbf{x}_i, \mathcal{U}) = g(\mathbf{x}_i, \mathcal{U}) + \mathbb{E}[V_{k-1}^*(\mathbf{x}') | \mathbf{x}_i, \mathcal{U}]$$

$$\pi_k^*(\mathbf{x}_i) = \arg \min_{\mathbf{u} \in \mathcal{U}} Q_k(\mathbf{x}_i, \mathbf{u})$$

▷ minimize

$$V_k^*(\mathbf{x}_i) = \min_{\mathbf{u} \in \mathcal{U}} Q_k(\mathbf{x}_i, \mathbf{u})$$

end for

end for

return $V_N^*, \pi_N^*(\mathcal{X})$

Algorithm: GPDP

assume known deterministic system function $\mathbf{x}' = f(\mathbf{x}, \mathbf{u})$

$$V_0^*(\mathcal{X}) = g_{\text{term}}(\mathcal{X})$$

$$V_0^* \sim \mathcal{GP}_v(m_v, k_v)$$

▷ terminal return

▷ GP models V_0^*

for $k = 1$ to N **do**

▷ start recursion

for all states $\mathbf{x}_i \in \mathcal{X}$ **do**

$$Q_k(\mathbf{x}_i, \mathcal{U}) = g(\mathbf{x}_i, \mathcal{U}) + m_v(\mathbf{x}')$$

$$Q_k(\mathbf{x}_i, \cdot) \sim \mathcal{GP}_q(m_q, k_q)$$

▷ GP models Q_k

$$\pi_k^*(\mathbf{x}_i) = \arg \min_{\mathbf{u} \in \mathbb{R}^{n_u}} m_q(\mathbf{u})$$

▷ minimize

$$V_k^*(\mathbf{x}_i) = \min_{\mathbf{u} \in \mathbb{R}^{n_u}} m_q(\mathbf{u})$$

end for

end for

return $V_N^*, \pi_N^*(\mathcal{X})$

Algorithm: GPDP

assume known deterministic system function $\mathbf{x}' = f(\mathbf{x}, \mathbf{u})$

$$V_0^*(\mathcal{X}) = g_{\text{term}}(\mathcal{X})$$

$$V_0^* \sim \mathcal{GP}_v(m_v, k_v)$$

▷ terminal return

▷ GP models V_0^*

for $k = 1$ to N **do**

▷ start recursion

for all states $\mathbf{x}_i \in \mathcal{X}$ **do**

$$Q_k(\mathbf{x}_i, \mathcal{U}) = g(\mathbf{x}_i, \mathcal{U}) + m_v(\mathbf{x}')$$

$$Q_k(\mathbf{x}_i, \cdot) \sim \mathcal{GP}_q(m_q, k_q)$$

▷ GP models Q_k

$$\pi_k^*(\mathbf{x}_i) = \arg \min_{\mathbf{u} \in \mathbb{R}^{n_u}} m_q(\mathbf{u})$$

▷ minimize

$$V_k^*(\mathbf{x}_i) = \min_{\mathbf{u} \in \mathbb{R}^{n_u}} m_q(\mathbf{u})$$

end for

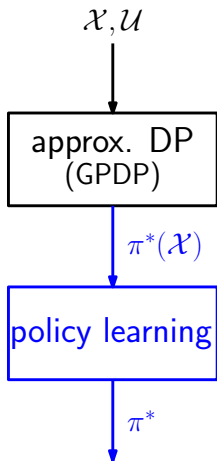
$$V_k^* \sim \mathcal{GP}_v(m_v, k_v)$$

▷ GP models V_k^*

end for

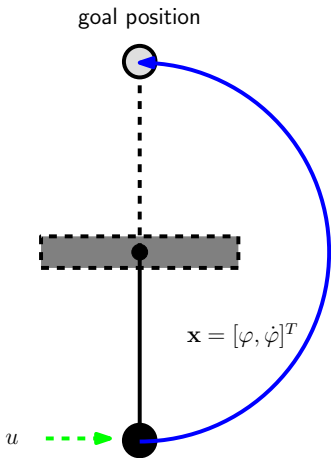
return $V_N^*, \pi_N^*(\mathcal{X})$

2.: Learning the Policy

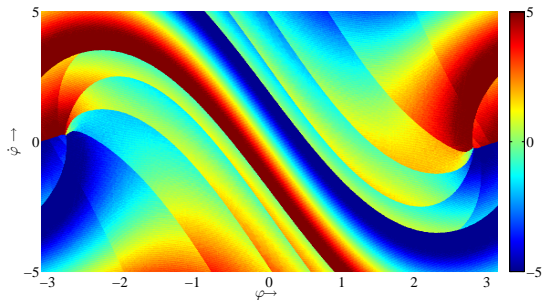


- find optimal actions for any state (not only for training set \mathcal{X})
- use GP to model latent optimal policy π^*

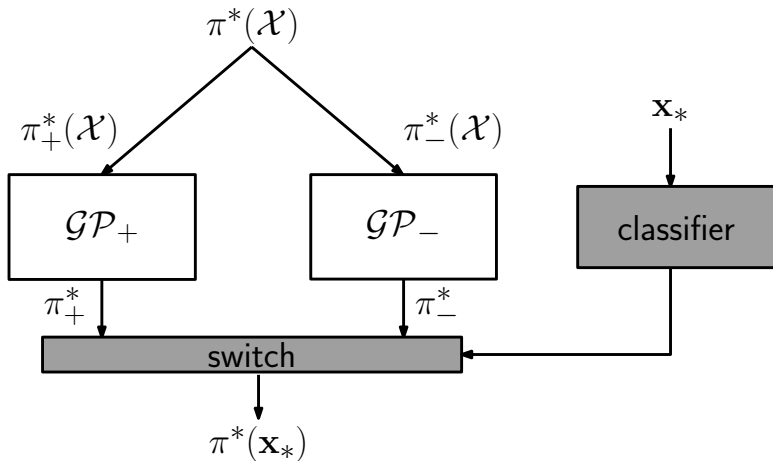
Learning a Discontinuous Policy



- in principle: smooth policy
- problem: system is underpowered
 → no direct swing-up, discontinuities
- how to treat in principle smooth policies in underpowered systems (discontinuities)?

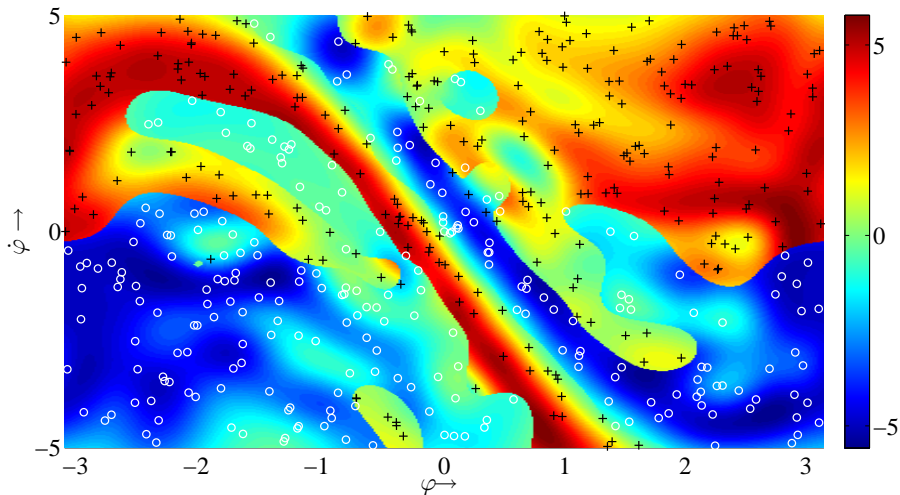


Switching GP Models

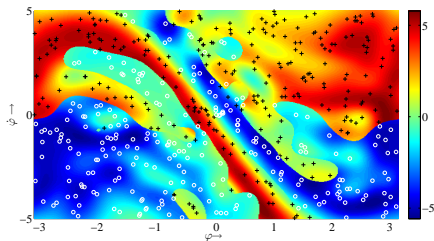
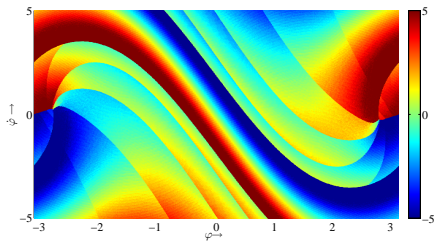


similar to mixture of experts (Jacobs et al., 1991)

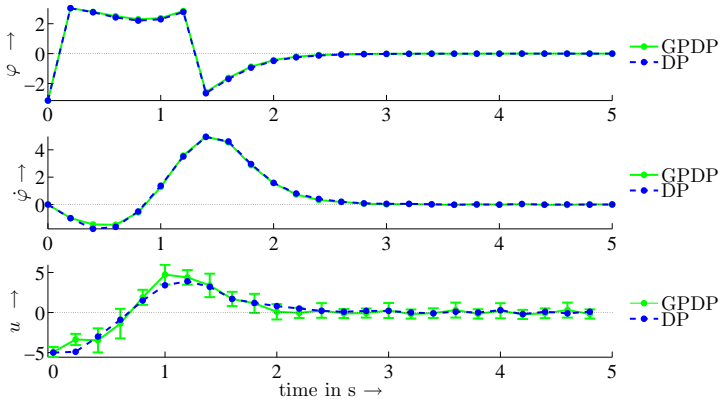
Results: Learned Policy



Results: Learned Policy (2)



Results: Trajectories



Computational Demand

- computational complexity
 - GPDP: $\mathcal{O}(|\mathcal{X}|^3 + |\mathcal{U}|^3|\mathcal{X}|)$ + nonlinear optimization
 - standard DP: $\mathcal{O}(|\mathcal{X}_{\text{DP}}|^2 * |\mathcal{U}_{\text{DP}}|)$

Computational Demand

- computational complexity
 - GPDP: $\mathcal{O}(|\mathcal{X}|^3 + |\mathcal{U}|^3|\mathcal{X}|)$ + nonlinear optimization
 - standard DP: $\mathcal{O}(|\mathcal{X}_{DP}|^2 * |\mathcal{U}_{DP}|)$
- pendulum example:

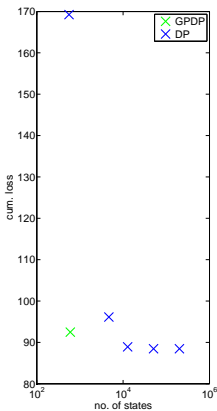


Figure: cumulative loss

Computational Demand

- computational complexity
 - GPD: $\mathcal{O}(|\mathcal{X}|^3 + |\mathcal{U}|^3|\mathcal{X}|)$ + nonlinear optimization
 - standard DP: $\mathcal{O}(|\mathcal{X}_{DP}|^2 * |\mathcal{U}_{DP}|)$
- pendulum example:

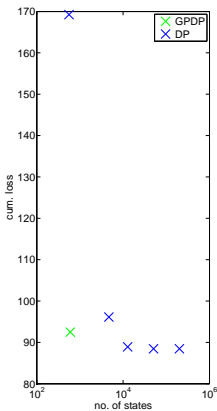


Figure: cumulative loss

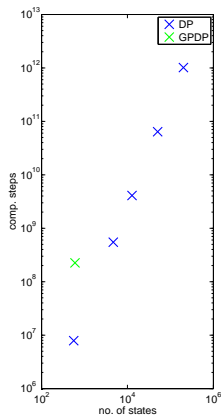


Figure: computations

Sampling Rate

OK, but if we increase sampling frequency

- in standard DP we need more discretization points
- in GDPD: we don't care! Problem becomes even easier (nasty discretization effects vanish)

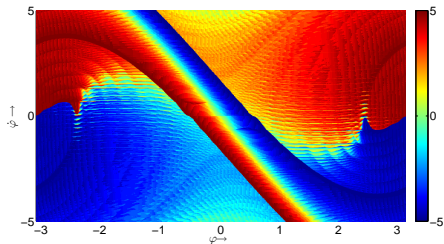


Figure: opt. policy, sampling rate: 50 Hz

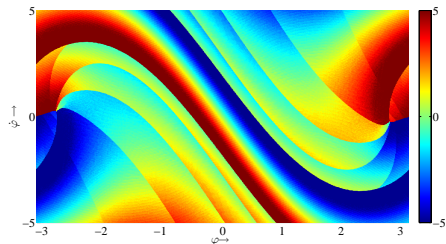


Figure: opt. policy, sampling rate: 5 Hz

Properties of GPDP

- approximation to deal with continuous-valued state spaces
- GP models provide information about uncertainty of the model
- training set for value functions independent of sampling frequency
- works well for 2D examples
- run into problems for much more data → approximations

Possible Extensions toward “Real” RL

What happens if we have some unknowns?

- ▶ noisy immediate returns (already in GPDP, not mentioned)
- ▶ unknown transition dynamics \rightarrow model them with a GP (offline)
- ▶ active learning for state and action selection (efficient use of data)
- ▶ online learning

Wrap-up

- ▶ DP suffers from curse of dimensionality
- ▶ generalize DP by means of GPs (GPDP) to continuous states and actions
- ▶ policy in entire state space through switching GPs
- ▶ extensions we were/are/will be working on

References



Christopher G. Atkeson and Stefan Schaal.

Robot Learning from Demonstration.

In D. H. Fisher Jr., editor, *Proceedings of the 14th International Conference on Machine Learning (ICML-1997)*, pages 12–20, Nashville, TN, USA, July 1997. Morgan Kaufmann.



Marc P. Deisenroth, Jan Peters, and Carl E. Rasmussen.

Approximate Dynamic Programming with Gaussian Processes.

In *Proceedings of the 2008 American Control Conference (ACC 2008)*, Seattle, WA, USA, June 2008.



Marc P. Deisenroth, Carl E. Rasmussen, and Jan Peters.

Model-Based Reinforcement Learning with Continuous States and Actions.

In *Proceedings of the 17th European Symposium on Artificial Neural Networks (ESANN 2008)*, Bruges, Belgium, April 2008.



Dirk Ormoneit and Šaunak Sen.

Kernel-Based Reinforcement Learning.

Machine Learning, 49(2–3):161–178, November 2002.



Martin Riedmiller.

Neural Fitted Q Iteration—First Experiences with a Data Efficient Neural Reinforcement Learning Method.

In *Proceedings of the 16th European Conference on Machine Learning (ECML)*, Porto, Portugal, 2005.



Carl E. Rasmussen and Christopher K. I. Williams.

Gaussian Processes for Machine Learning.

Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA, 2006.



Richard S. Sutton and Andrew G. Barto.

Reinforcement Learning: An Introduction.

Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA, 1998.