

Gaussian Process Dynamic Programming

Marc Peter Deisenroth, Carl Edward Rasmussen,
Jan Peters

2009

Presented by Evan Beachly

Motivation: Better Reinforcement Learning

TD learning and Sarsa have to interact with the environment for a loooonnnnnggg time before they learn an optimal policy.

Model the environment for faster learning.

Handle continuous state and action spaces.

Outline

- Day 1: Background
 - Gaussian Process Regression
 - Reinforcement Learning
- Day 2: Gaussian Process Dynamic Programming

Gaussian Process Regression

Multivariate Gaussian Probability Distribution

$$\text{PDF of } N(\mu, \Sigma) = \frac{1}{2\pi|\Sigma^{-1}|} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)}$$

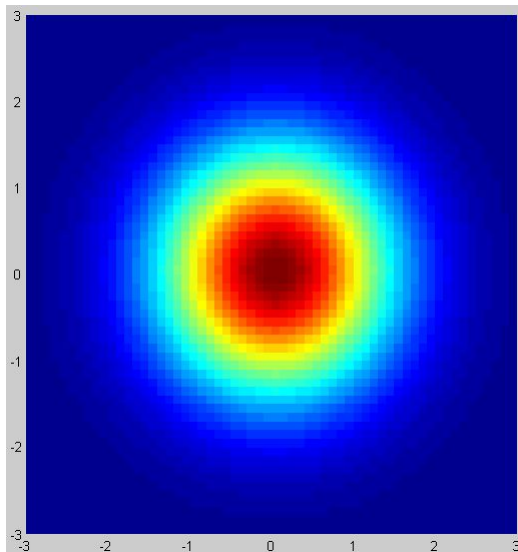
μ = mean

Σ = covariance matrix

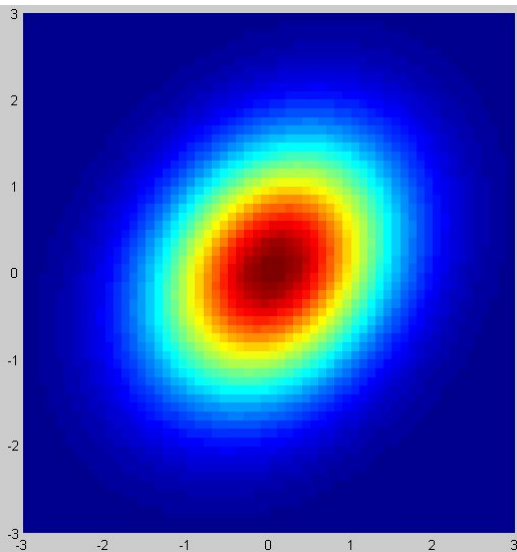
variance

$$\Sigma = \begin{vmatrix} \sigma_1^2 & \sigma_1\sigma_2 \\ \sigma_1\sigma_2 & \sigma_2^2 \end{vmatrix}$$

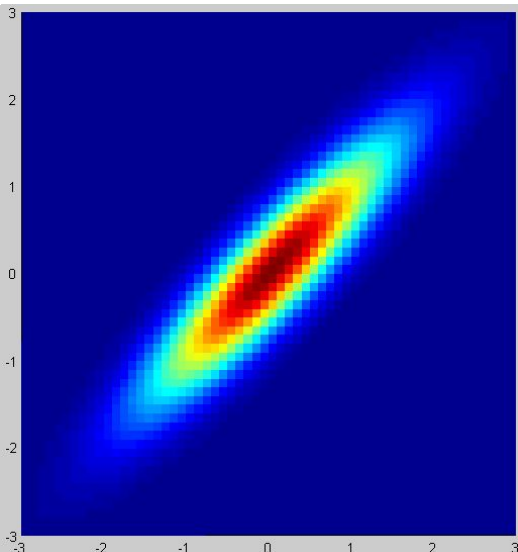
covariance



$$\Sigma = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$



$$\Sigma = \begin{vmatrix} 1 & 0.3 \\ 0.3 & 1 \end{vmatrix}$$



$$\Sigma = \begin{vmatrix} 1 & 0.9 \\ 0.9 & 1 \end{vmatrix}$$

Regression

Given:

Independent Variables Observations:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \dots \\ \mathbf{x}_n \end{pmatrix} = \begin{pmatrix} x0_1 & x1_1 & x2_1 \\ x0_2 & x1_2 & x2_2 \\ \dots & \dots & \dots \\ x0_n & x1_n & x2_n \end{pmatrix}$$

Dependent Variable Observations:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

Measurement error: σ_y^2

Predict y^* for \mathbf{x}^*

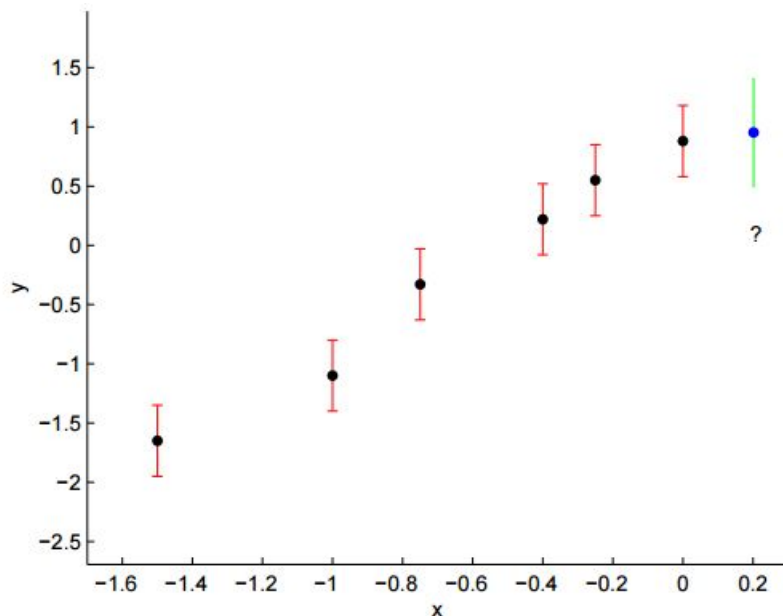


Figure 1: Given six noisy data points (error bars are indicated with vertical lines), we are interested in estimating a seventh at $x_* = 0.2$.

Gaussian Process Regression

$$\begin{bmatrix} \mathbf{y} \\ y^* \end{bmatrix} = N(0, \mathbf{K} + \sigma_y^2 \mathbf{I})$$

measurement error * Identity matrix

correlation between observations

$k(\mathbf{x}_a, \mathbf{x}_b)$ = Some symmetric function. Positive if \mathbf{x}_a and \mathbf{x}_b are close. ~ 0 if \mathbf{x}_a and \mathbf{x}_b are far away.

$$\mathbf{K} = \begin{pmatrix} \mathbf{K} & \mathbf{K}_*^T \\ \mathbf{K}_* & \mathbf{K}_{**} \end{pmatrix}$$

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \dots & \dots & \dots & \dots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

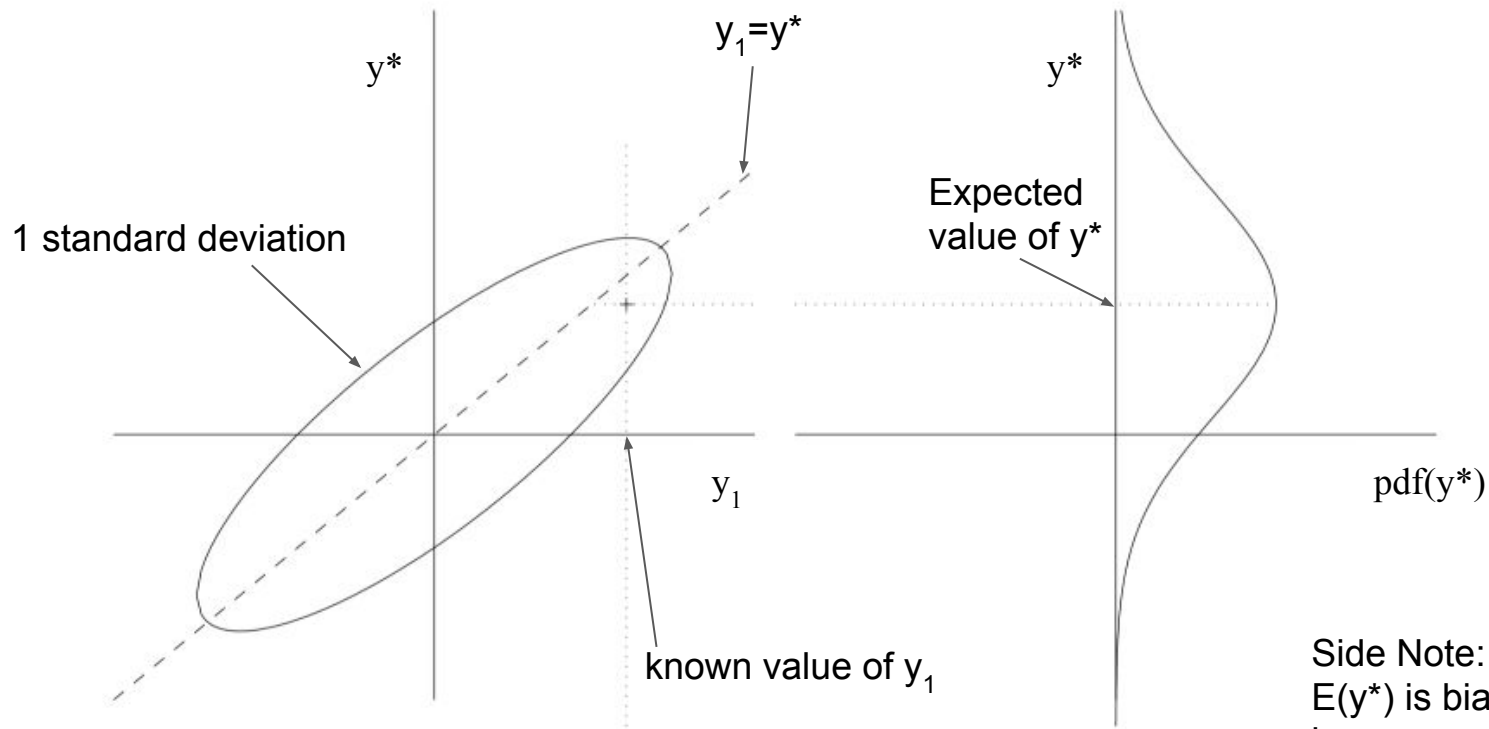
large, but doesn't depend on \mathbf{x}^*

$$\mathbf{K}_* = \begin{bmatrix} k(\mathbf{x}^*, \mathbf{x}_1) & k(\mathbf{x}^*, \mathbf{x}_2) & \dots & k(\mathbf{x}^*, \mathbf{x}_n) \end{bmatrix}$$

$$\mathbf{K}_{**} = \begin{bmatrix} k(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix}$$

\mathbf{x}^* nearby $\mathbf{x}_1 \rightarrow k(\mathbf{x}^*, \mathbf{x}_1)$ will be large $\rightarrow y^*$ and y_1 will be highly correlated by $N(0, \mathbf{K} + \sigma_y^2 \mathbf{I}) \rightarrow y^*$ will have a value similar to y_1 .

Plot of $N(0, \mathbf{K} + \sigma_y^2 \mathbf{I})$



Side Note:
 $E(y^*)$ is biased toward 0,
because we set μ to that.

Regression Algorithm

$$E(y^*) = K_*(K + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y}$$

$$\text{var}(y^*) = (K_{**} + \sigma_y^2) - K_*(K + \sigma_y^2 \mathbf{I})^{-1} K_*$$

$O(n^3)$ operations to invert K .

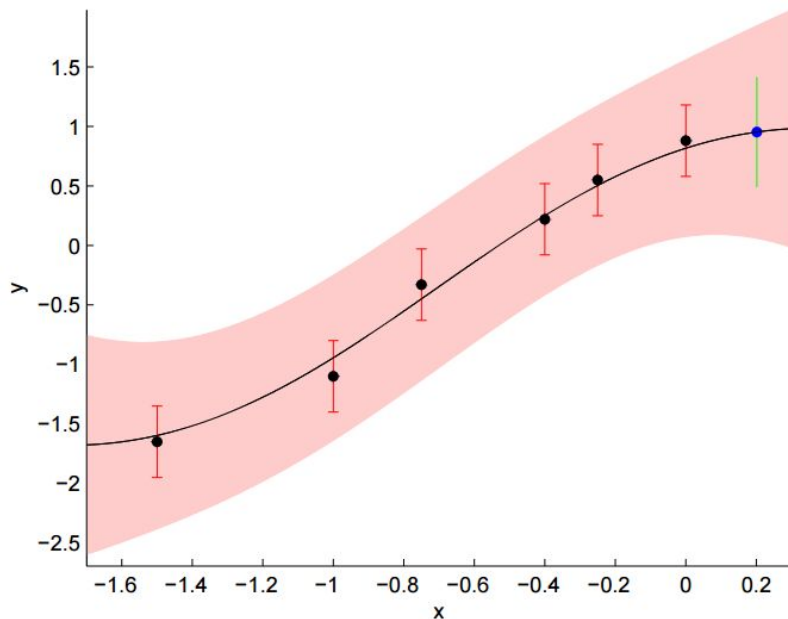


Figure 2: The solid line indicates an estimation of y_* for 1,000 values of x_* . Pointwise 95% confidence intervals are shaded.

Choosing $k(\mathbf{x}_a, \mathbf{x}_b)$

Squared Exponential: $SE(\mathbf{x}_a - \mathbf{x}_b) = \sigma^2 e^{-\frac{1}{2}(\mathbf{x}_a - \mathbf{x}_b)^T \Sigma^{-1} (\mathbf{x}_a - \mathbf{x}_b)}$ (Basic)

Cosine: $\sigma^2 (\cos(2\pi f(\mathbf{x}_a - \mathbf{x}_b)) + 1)$ (Eg. Seasonal data, Traffic)

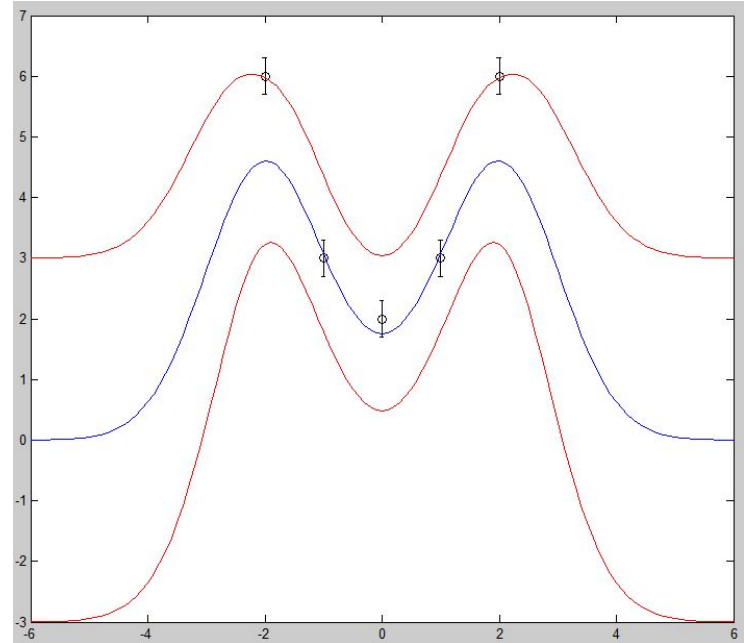
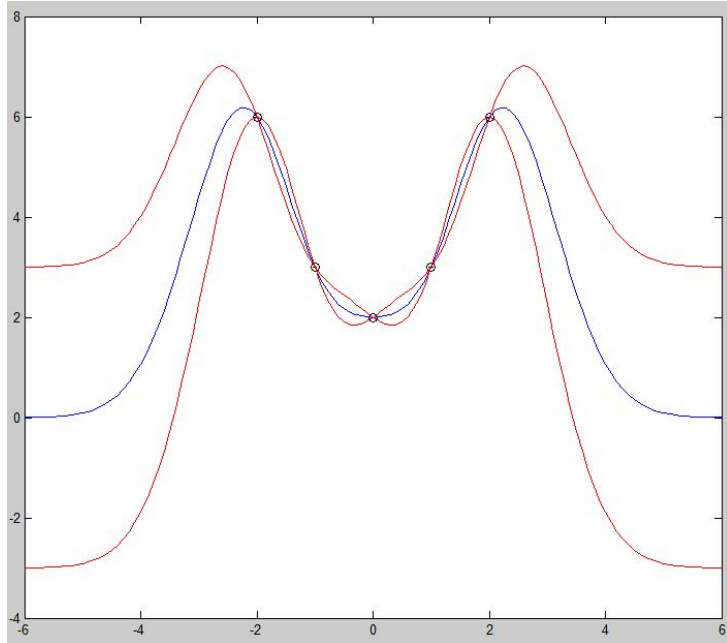
Wraparound: $SE(\tan^{-1}(\sin(\mathbf{x}_a - \mathbf{x}_b) / \cos(\mathbf{x}_a - \mathbf{x}_b)))$ (Eg. Angles)

Choosing hyperparameters $\Theta = (\sigma^2, \Sigma, f, \text{etc.})$:

Maximize log likelihood: $\log p(\mathbf{y}|\mathbf{x}, \Theta) = \underbrace{-\frac{1}{2} \mathbf{y}^T \mathbf{K}^{-1} \mathbf{y}}_{\text{data fit term}} - \underbrace{\frac{1}{2} \log |\mathbf{K}|}_{\text{complexity penalty}} - \frac{1}{2} n \log(2\pi)$

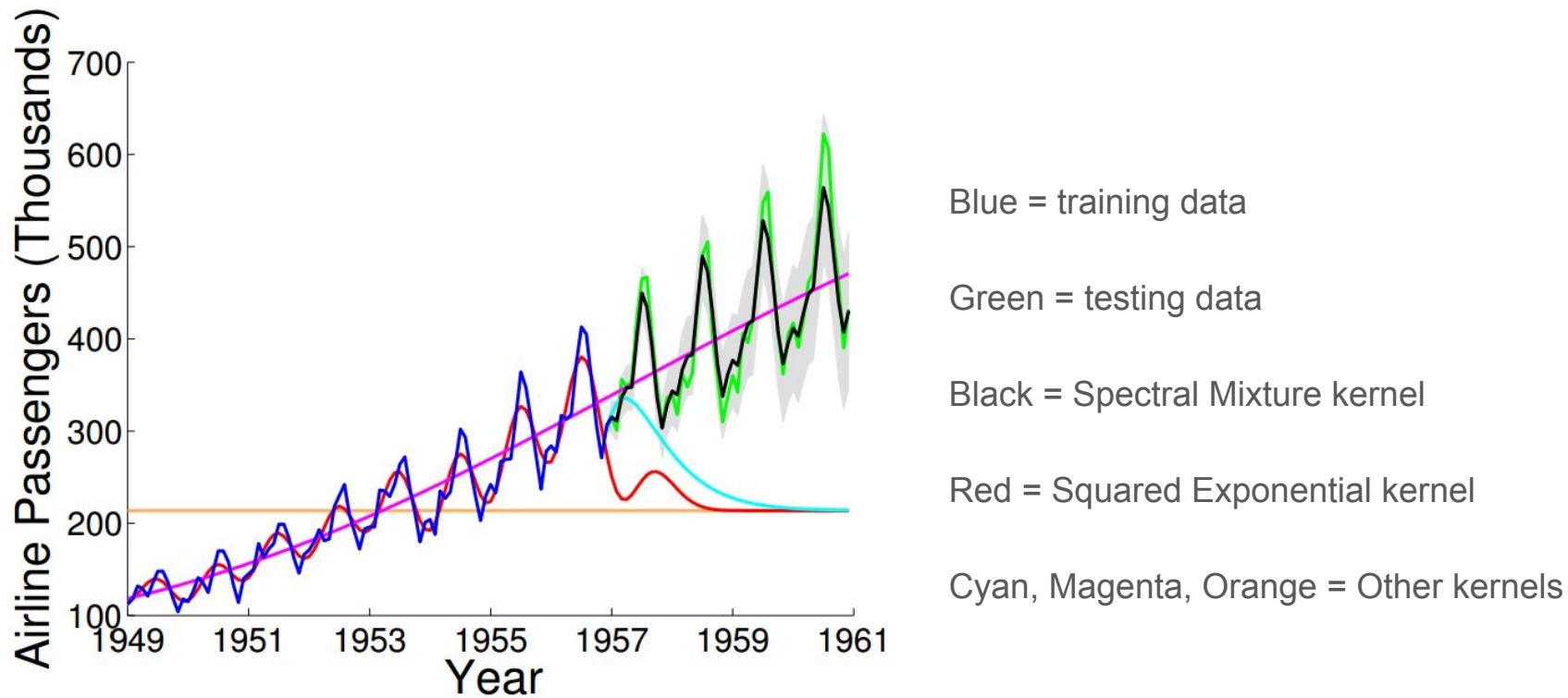
Conjugate Gradient Descent

Extrapolation - not so good



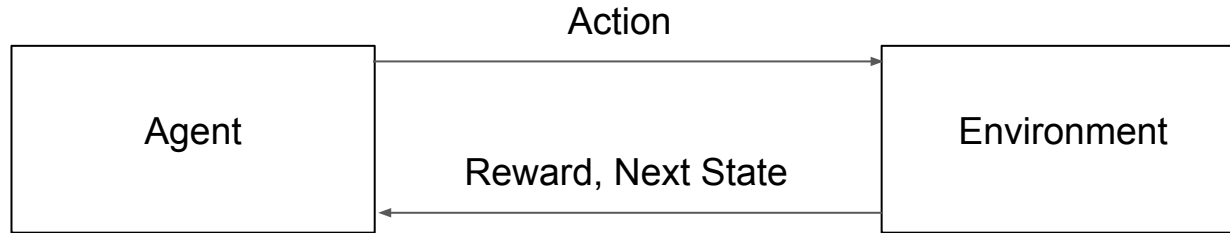
Extrapolation - maybe there's hope

Wilson et al. (2013) Gaussian Process Kernels for Pattern Discovery and Extrapolation



Model-based Reinforcement Learning in a Discrete Environment

Reinforcement Learning System



Optimal Policy

Maximize expected cumulative future reward

1. Finite Horizon $E\left(\sum_{t=0}^h r_t\right)$

2. Infinite Horizon $E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$

Discount Factor



3. Long-term Average $\lim_{h \rightarrow \infty} E\left(\frac{1}{h} \sum_{t=0}^h r_t\right)$

Value functions

$Q(s,a)$: Expected cumulative future reward that results from taking action a in state s , and taking an optimal policy afterwards.

$V(s)$: $\max_{a \in A}(Q(s,a))$: Maximum expected cumulative future reward that can be achieved in a particular state. How valuable being in this state is.

Defined Recursively:

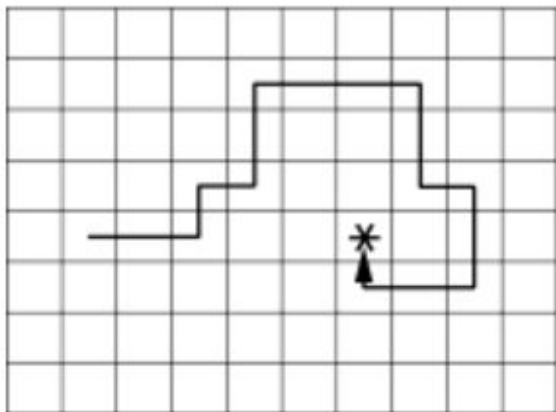
$$Q(s,a) = r(s,a) + \gamma V(s'(s,a))$$

If we know these functions, then we can take the optimal action by maximizing $Q(s,a)$ over a .

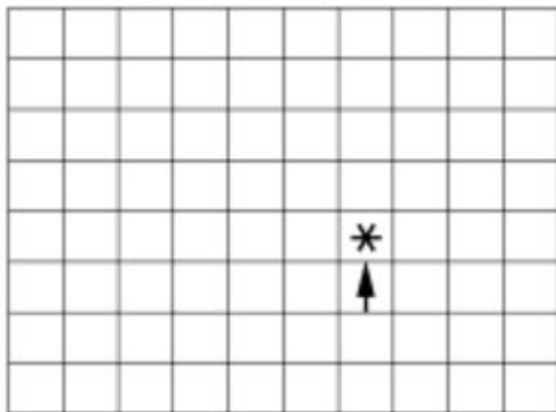
Model-free RL

Wander around until you receive a reward. Incrementally update Q Table.

Path taken



Action values increased
by one-step Sarsa



Action values increased
by Sarsa(λ) with $\lambda=0.9$

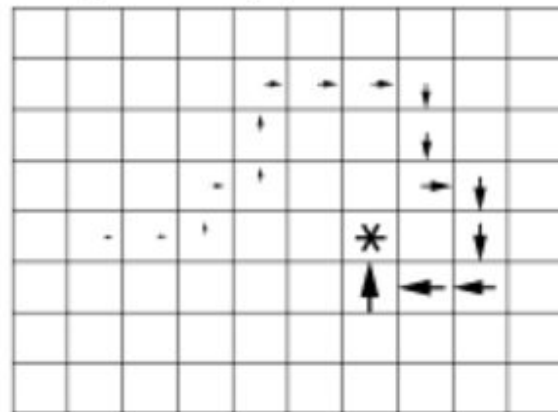
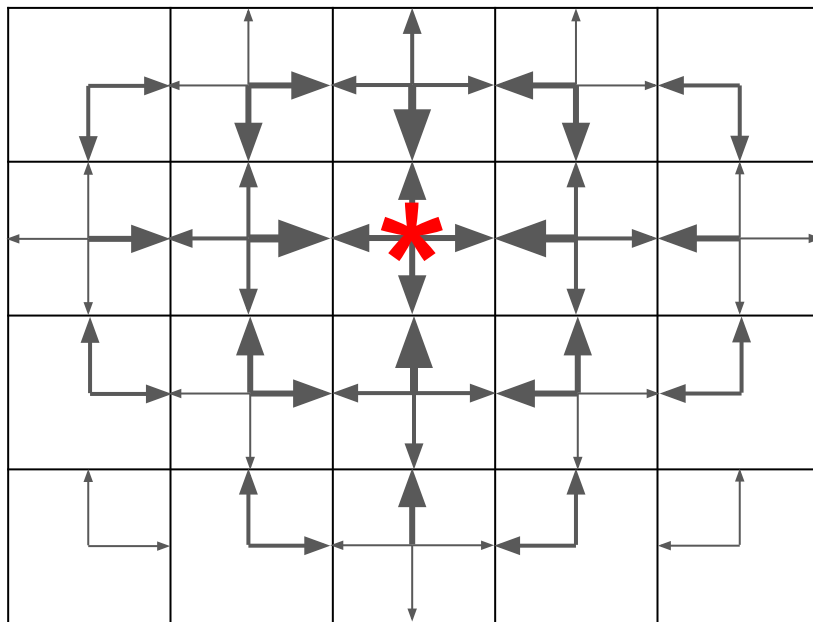


Figure 7.12: Gridworld example of the speedup of policy learning due to the use of eligibility traces.

Model-based RL

Interact with the environment until you learn models of the reward function and the transition function. Use these to compute the entire Q and V functions.



Model Based RL Algorithm Pseudocode

Algorithm 1 classic DP, known transition dynamics f

1: **input:** $f, \mathcal{X}_{\text{DP}}, \mathcal{U}_{\text{DP}}$ O(NSA)

2: $V_N^*(\mathcal{X}_{\text{DP}}) = g_{\text{term}}(\mathcal{X}_{\text{DP}})$ ▷ terminal cost

3: **for** $k = N - 1$ to 0 **do** For each time step in the horizon ▷ recursively

4: **for** all $\mathbf{x}_i \in \mathcal{X}_{\text{DP}}$ **do** For each state ▷ for all states

5: **for** all $\mathbf{u}_j \in \mathcal{U}_{\text{DP}}$ **do** For each action ▷ for all actions

6: $Q_k^*(\mathbf{x}_i, \mathbf{u}_j) = g(\mathbf{x}_i, \mathbf{u}_j) + \gamma \mathbb{E}_{\mathbf{x}_{k+1}} [V_{k+1}^*(\mathbf{x}_{k+1}) | \mathbf{x}_i, \mathbf{u}_j, f]$

7: **end for**

8: $\pi_k^*(\mathbf{x}_i) \in \arg \min_{\mathbf{u} \in \mathcal{U}_{\text{DP}}} Q_k^*(\mathbf{x}_i, \mathbf{u})$

9: $V_k^*(\mathbf{x}_i) = Q_k^*(\mathbf{x}_i, \pi_k^*(\mathbf{x}_i))$

10: **end for**

11: **end for**

12: **return** $\pi^*(\mathcal{X}_{\text{DP}}) := \pi_0^*(\mathcal{X}_{\text{DP}})$ ▷ return optimal controls for \mathcal{X}_{DP}

Model-based RL Algorithm Basics

Q and V are defined recursively

Use Dynamic Programming to compute $Q(s,a)$ and $V(s)$ for all s,a

Finite Horizon gives a base case so you don't recurse forever.

Need to model:

- $r(s,a)$: Reward function that depends on state and action
- $r_{\text{term}}(s)$: Terminal Reward function (Reward at the end of the horizon)
- $s'(s,a)$: Transition function (Forward Model)

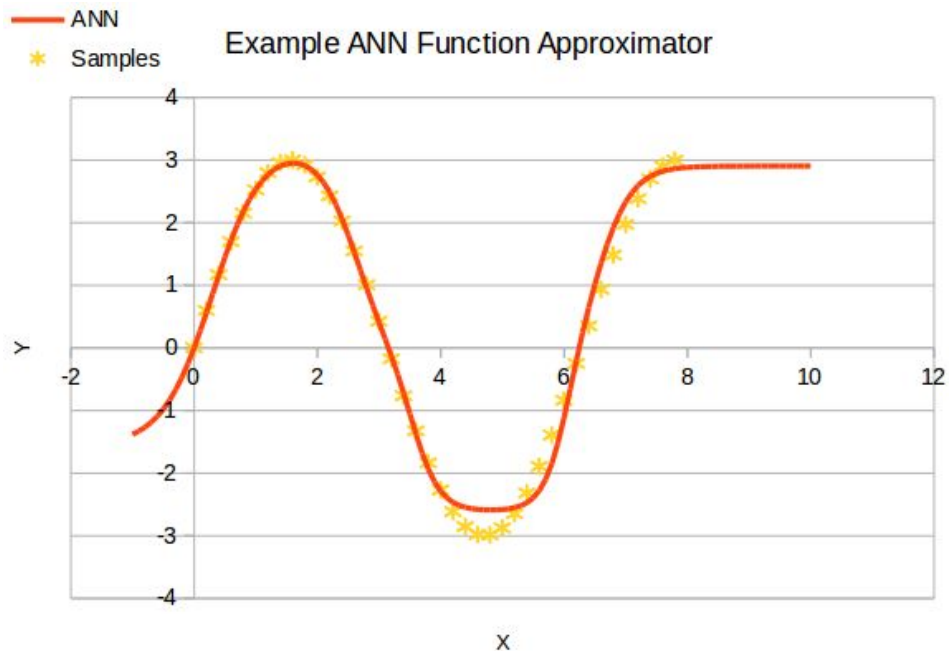
If you don't know $r_{\text{term}}(s)$, just set it to 0 and use a large horizon

How do you model the reward and transition function?

Interpolate between observations of inputted state action pairs and their outputted rewards and next states.

Function Approximators:

- Linear Regression
- Artificial Neural Network
- Gaussian Process Regression



Model-based RL Dynamic Programming Example

Input:

$$\gamma = 0.5$$

$$N = 3$$

$r(s,a)$

0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0

$r_{\text{term}}(s)$

0	0	8
0	0	0
0	0	0

Model-based RL Dynamic Programming Example

Initialization:

$$Q(s,a) = 0$$

$$V(s) = r_{\text{term}}(s)$$

Q(s,a)

0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0

V(s)

0	0	8
0	0	0
0	0	0

Model-based RL Dynamic Programming Example

Iteration 1a

$$Q(s,a) = r(s,a) + \gamma V(s'(s,a))$$

Q(s,a)

0 0 0 0	0 0 4 0	0 0 0 0
0 0 0 0	0 0 0 0	4 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0

V(s)

0	0	8
0	0	0
0	0	0

Model-based RL Dynamic Programming Example

Iteration 1b

$$V(s) = \max_{a \in A} (Q(s,a), V(s))$$

Q(s,a)

0 0 0 0	0 0 4 0	0 0 0 0
0 0 0 0	0 0 0 0	4 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0

V(s)

0	4	8
0	0	4
0	0	0

Model-based RL Dynamic Programming Example

Iteration 2a

$$Q(s,a) = r(s,a) + \gamma V(s'(s,a))$$

Q(s,a)

0 0 2 0	0 0 4 0	0 2 0 2
0 0 0 0	2 0 2 0	4 0 0 0
0 0 0 0	0 0 0 0	2 0 0 0

V(s)

0	4	8
0	0	4
0	0	0

Model-based RL Dynamic Programming Example

Iteration 2b

$$V(s) = \max_{a \in A} (Q(s,a), V(s))$$

Q(s,a)

0 0 2 0	0 0 4 0	0 2 0 2
0 0 0 0	2 0 2 0	4 0 0 0
0 0 0 0	0 0 0 0	2 0 0 0

V(s)

2	4	8
0	2	4
0	0	2

Model-based RL Dynamic Programming Example

Iteration 3a

$$Q(s,a) = r(s,a) + \gamma V(s'(s,a))$$

Q(s,a)

0 0 2 0	0 1 4 1	0 2 0 2
1 0 1 0	2 0 2 0	4 1 0 1
0 0 0 0	1 0 1 0	2 0 0 0

V(s)

2	4	8
0	2	4
0	0	2

Model-based RL Dynamic Programming Example

Iteration 3b

$$V(s) = \max_{a \in A} (Q(s,a), V(s))$$

$$\pi(s) = a_{\max}$$

Q(s,a)

0 0 2 0	0 1 4 1	0 2 0 2
1 0 1 0	2 0 2 0	4 1 0 1
0 0 0 0	1 0 1 0	2 0 0 0

V(s)

2	4	8
1	2	4
0	1	2

Outline

Day 1:

- Background
 - Gaussian Process Regression
 - Model-based Reinforcement Learning in discrete state and action spaces

Day 2:

- Gaussian Process Dynamic Programming
- "Online Learning"

Gaussian Process Dynamic Programming

High-level algorithm

Move randomly and observe reward and transition functions

Stop

Compute optimal policy

Execute optimal policy until we reach the goal.

Model-based RL in continuous domains

Need to represent $Q(s,a)$, $V(s)$, and $\pi(s)$.

Instead of dividing the domain into discrete chunks, choose a finite set S of states and A of actions.

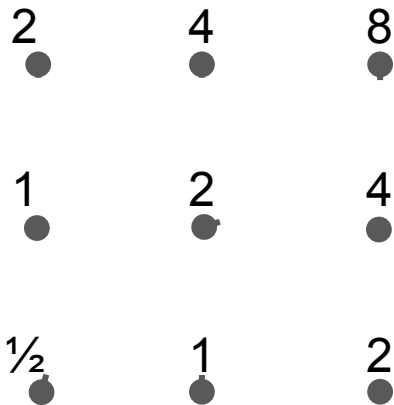
Compute Q and V for these points, and interpolate between for the rest using GPR.

$$Q(s,a) = r(s,a) + \gamma V(s'(s,a))$$

$$V(s) = \max_a (Q(s,a), V(s))$$

$$\pi(s) = a_{\max}$$

Use gradient descent to find $\max_a (Q(s,a))$



Gaussian Process Dynamic Programming (GPDP)

Algorithm 2 GPDP, known deterministic system dynamics

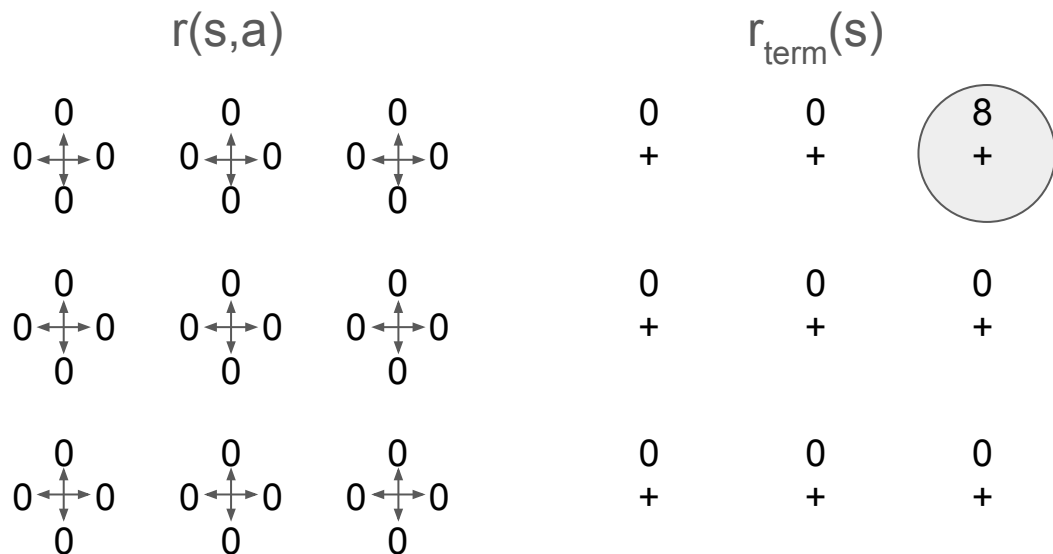
```
1: input:  $f, \mathcal{X}, \mathcal{U}$ 
2:  $V_N^*(\mathcal{X}) = g_{\text{term}}(\mathcal{X}) + w_g$   $O(|S||A|^3 + |S|^3)$   $\triangleright$  terminal cost
3:  $V_N^*(\cdot) \sim \mathcal{GP}_v$   $\triangleright$  GP model for  $V_N^*$ 
4: for  $k = N - 1$  to 0 do For each time step in the horizon  $\triangleright$  recursively
5:   for all  $\mathbf{x}_i \in \mathcal{X}$  do For each state  $\triangleright$  for all support states
6:     for all  $\mathbf{u}_j \in \mathcal{U}$  do For each action  $\triangleright$  for all support actions
7:        $Q_k^*(\mathbf{x}_i, \mathbf{u}_j) = g(\mathbf{x}_i, \mathbf{u}_j) + w_g + \gamma \mathbb{E}_V[V_{k+1}^*(f(\mathbf{x}_i, \mathbf{u}_j))]$ 
8:     end for Need to build GP models of Q and V
9:      $Q_k^*(\mathbf{x}_i, \cdot) \sim \mathcal{GP}_q$   $\triangleright$  GP model for  $Q_k^*$ 
10:     $\pi_k^*(\mathbf{x}_i) \in \arg \min_{\mathbf{u} \in \mathbb{R}^{n_u}} Q_k^*(\mathbf{x}_i, \mathbf{u})$ 
11:     $V_k^*(\mathbf{x}_i) = Q_k^*(\mathbf{x}_i, \pi_k^*(\mathbf{x}_i))$ 
12:  end for
13:   $V_k^*(\cdot) \sim \mathcal{GP}_v$   $\triangleright$  GP model for  $V_k^*$ 
14: end for
15: return  $\mathcal{GP}_v, \mathcal{X}, \pi^*(\mathcal{X}) := \pi_0^*(\mathcal{X})$ 
```

Continuous Dynamic Programming RL Example

Input:

$\gamma = 0.5$

$N = 1$

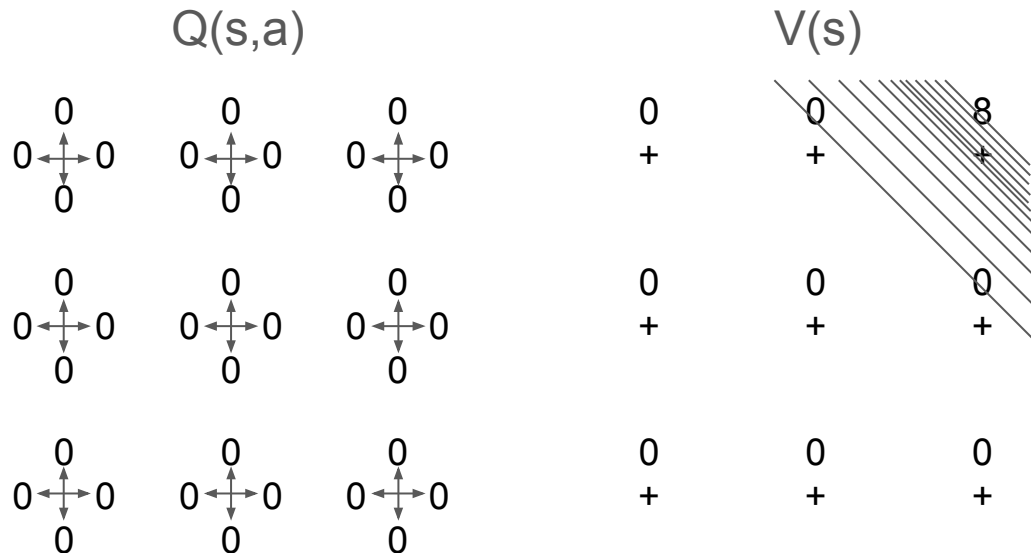


Continuous Dynamic Programming RL Example

Initialization:

$$Q(s,a) = 0$$

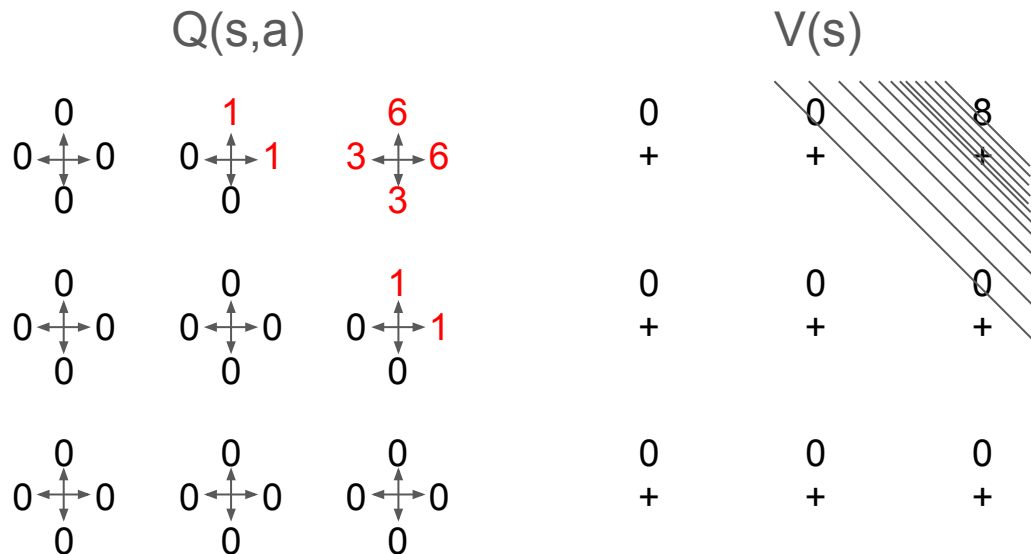
$$V(s) = r_{\text{term}}(s)$$



Continuous Dynamic Programming RL Example

Iteration 1a

$$Q(s,a) = r(s,a) + \gamma V(s'(s,a))$$

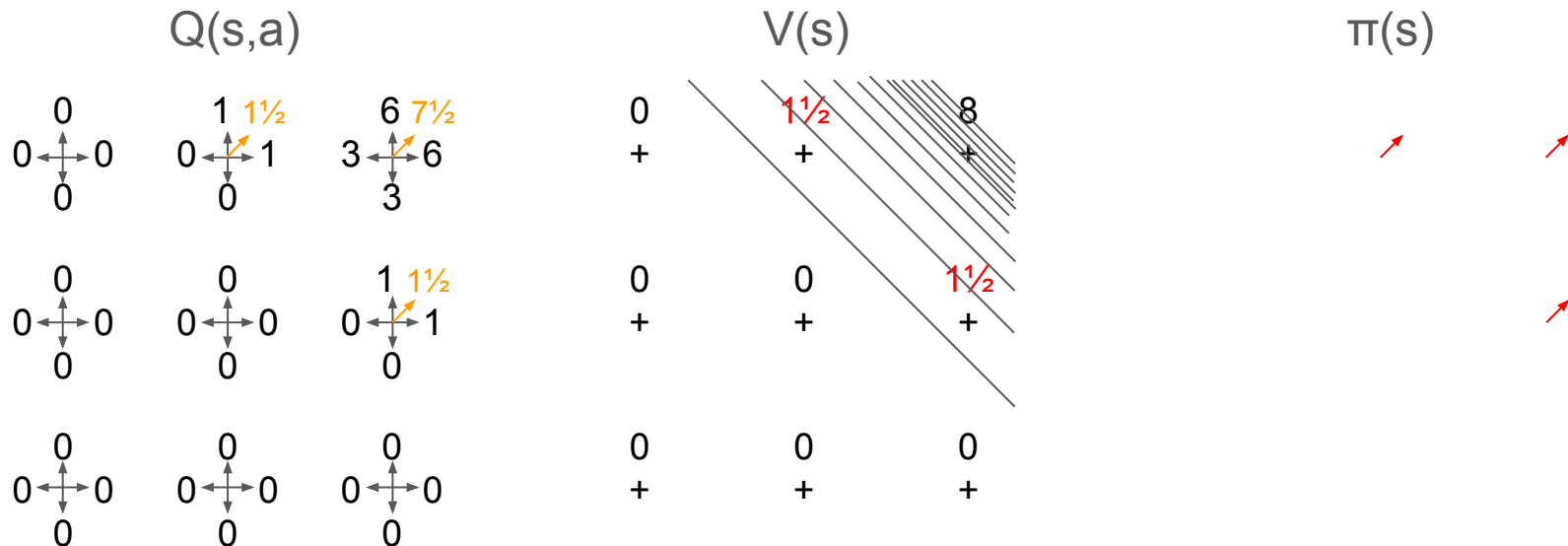


Continuous Dynamic Programming RL Example

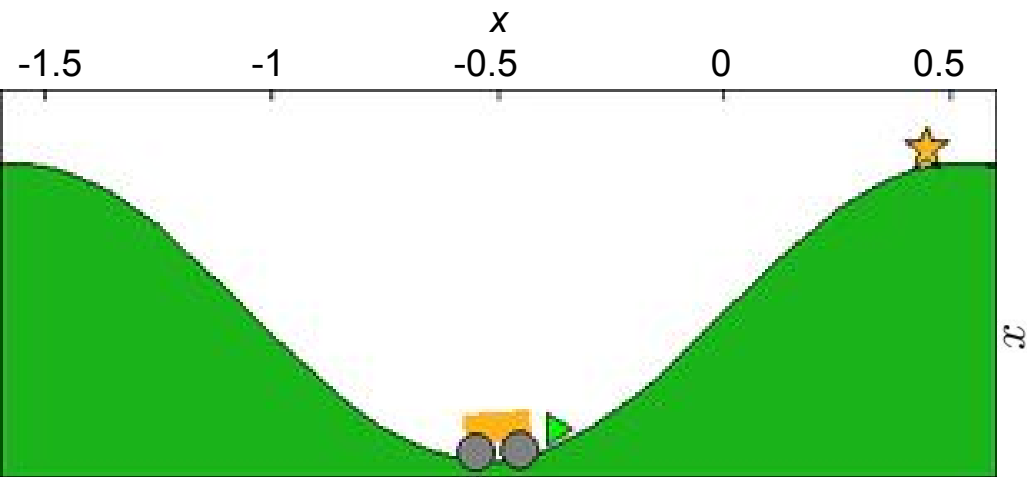
Iteration 1b

$$V(s) = \max_{a \in A} (Q(s,a), V(s))$$

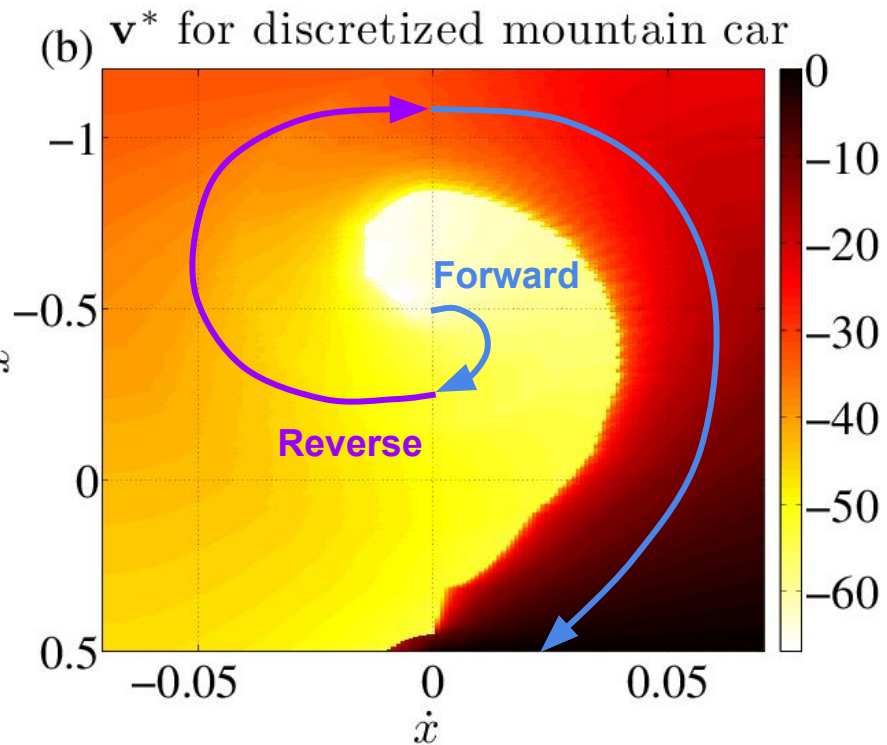
$$\pi(s) = a_{\max}$$



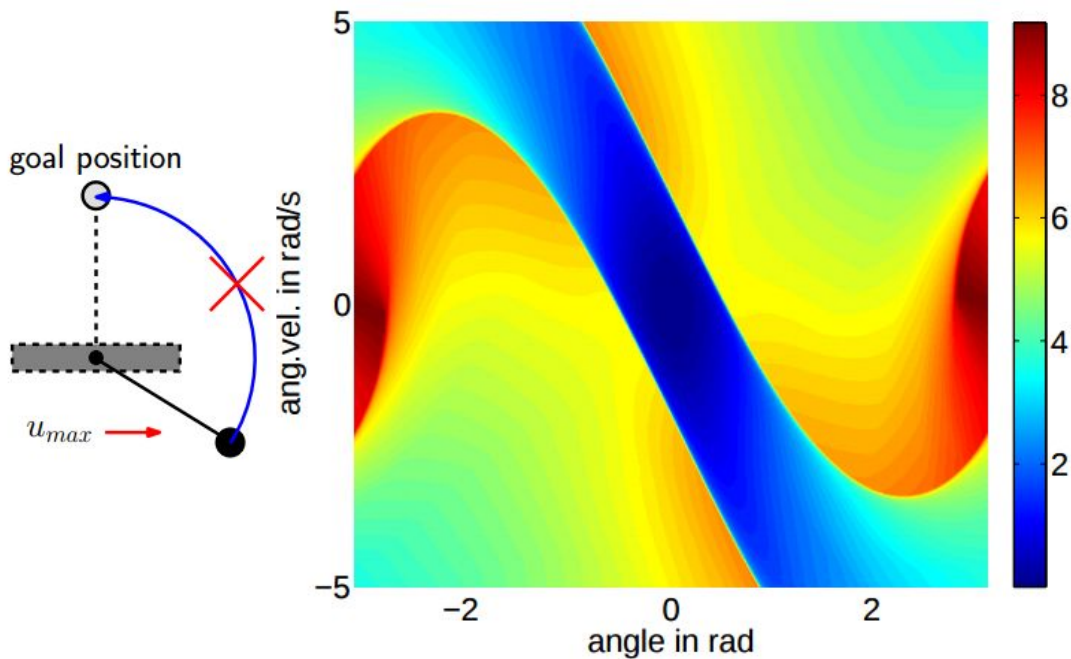
Problems



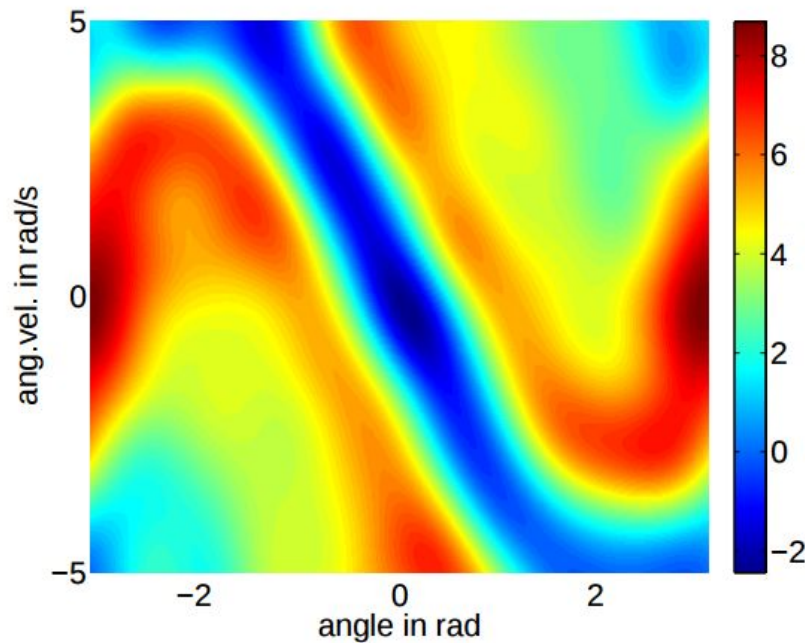
Discontinuous value functions
Discontinuous policy



Discontinuous Value Function



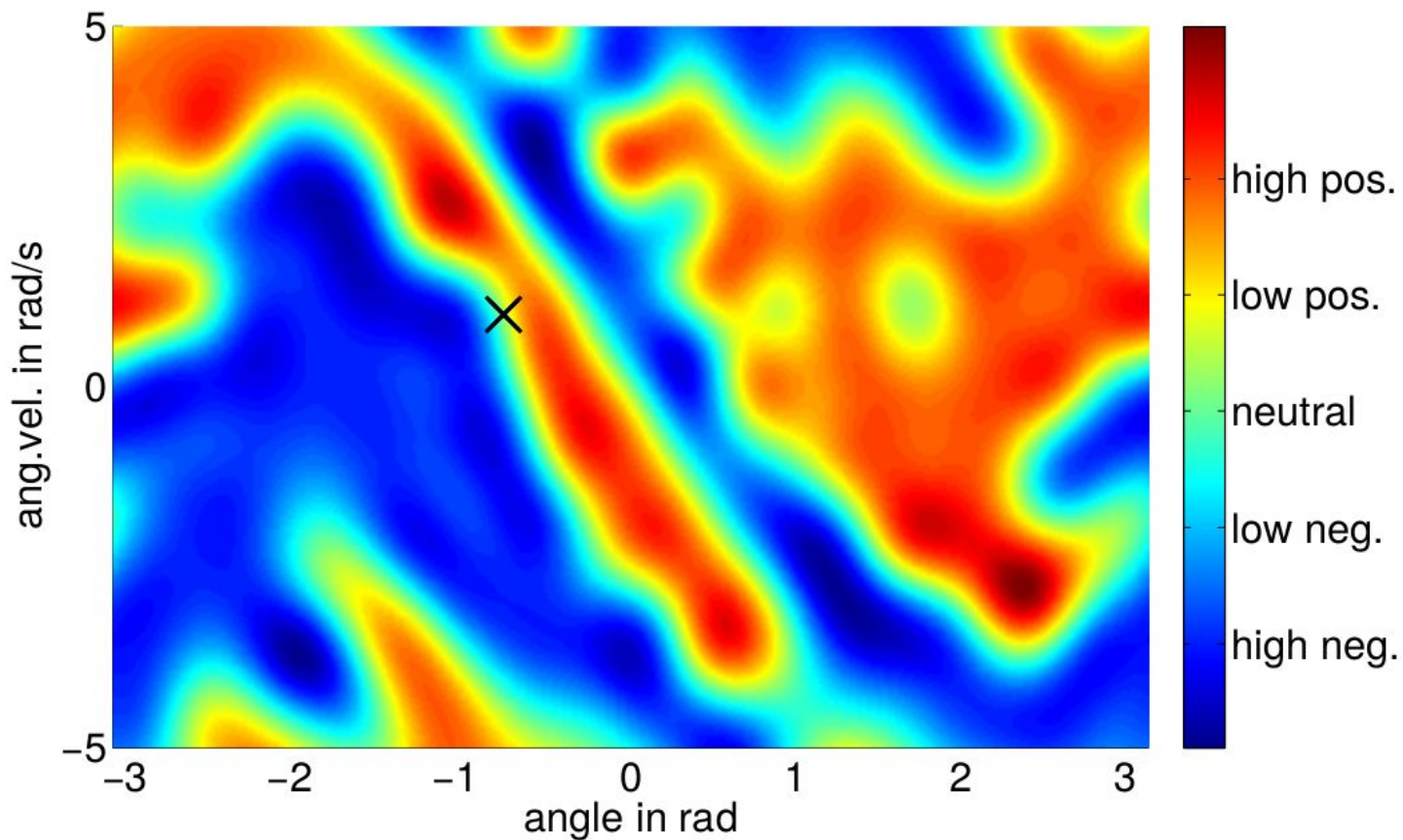
(a) Optimal DP value function.



(b) Mean of value function model (GPDP).

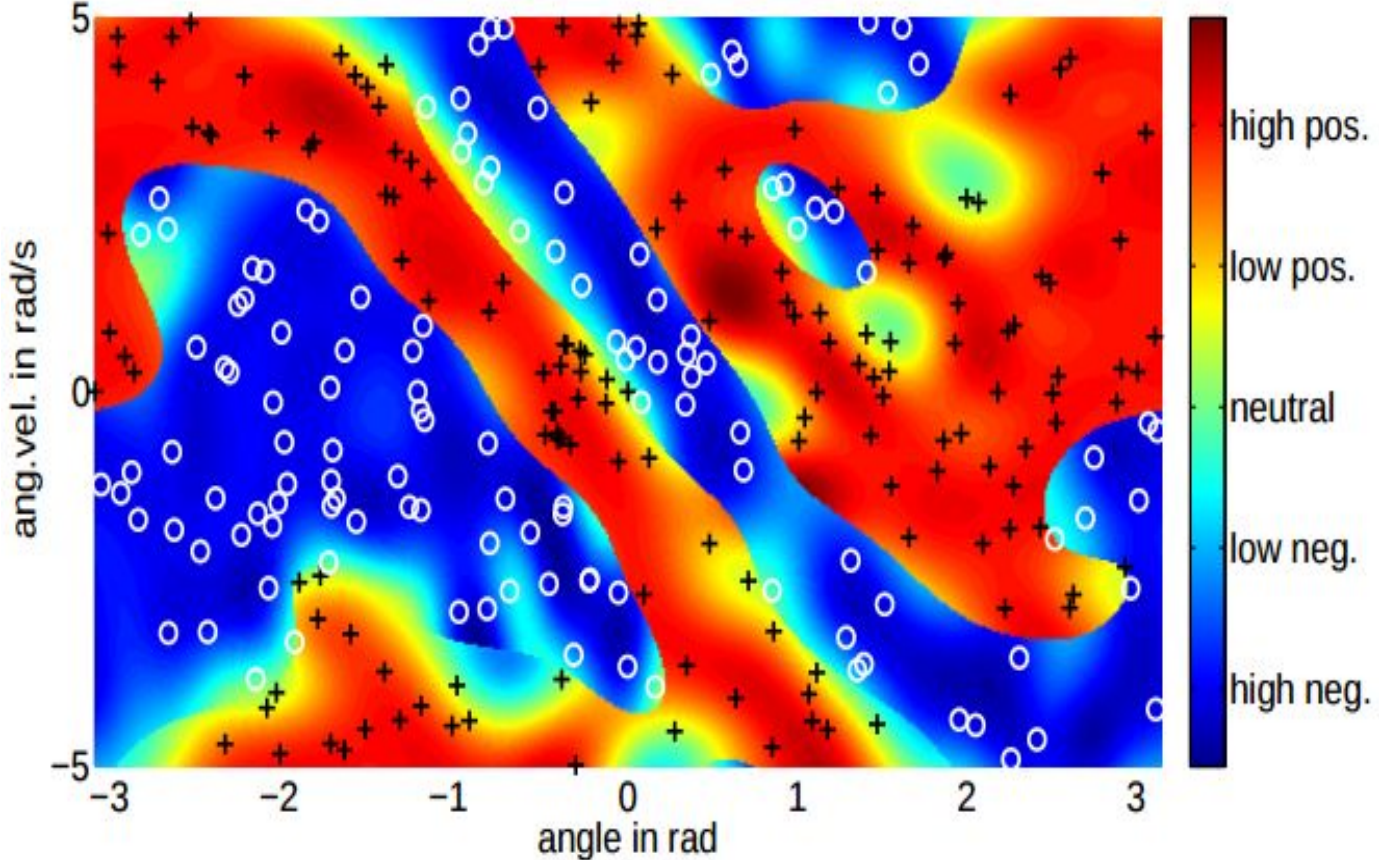
Smoothed value function may trick Dynamic Programming during the iterations

Continuous Policy



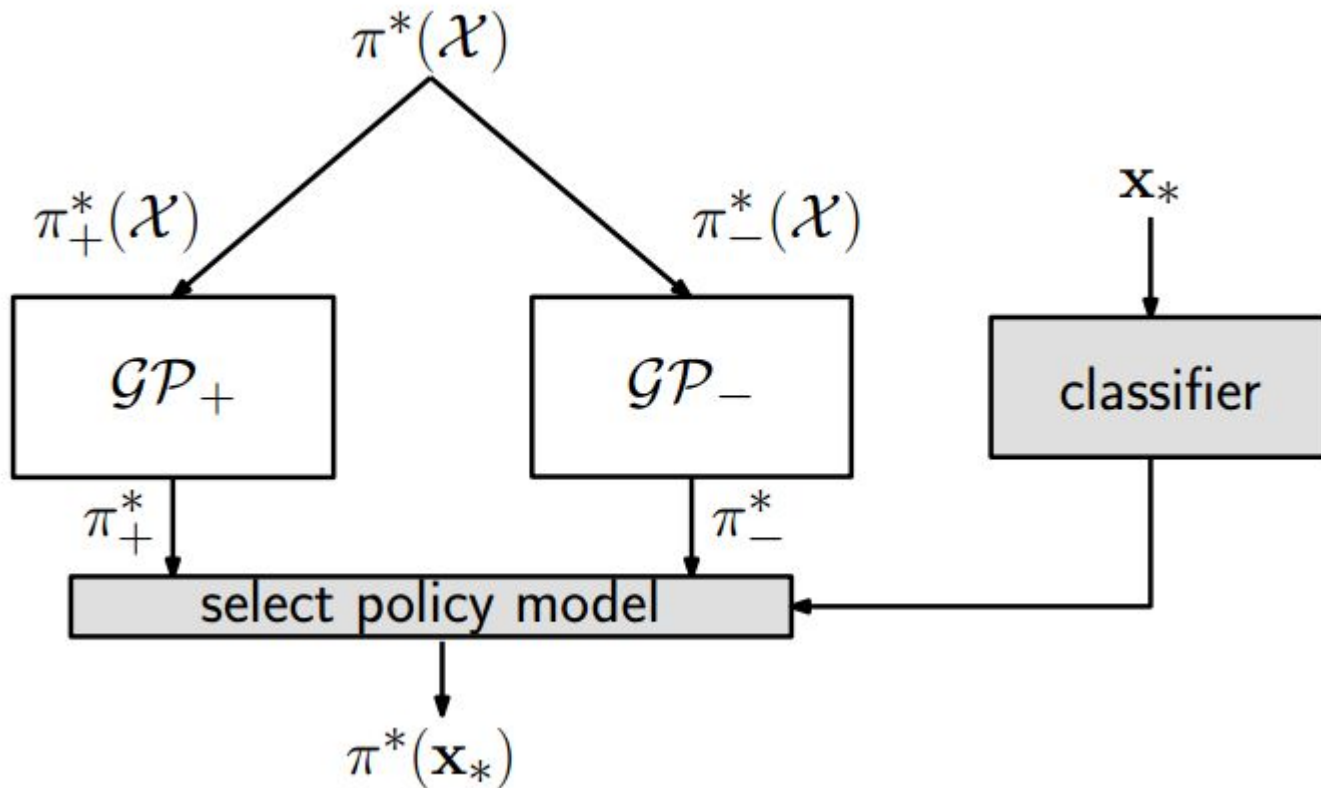
Optimal controls may be discontinuous.

Discontinuous policy.



Need a classifier to decide which policy to use.

Switching between Policies



Gaussian Process Classification

Set one class to have a value of +1, the other -1.

Do GPR on the data.

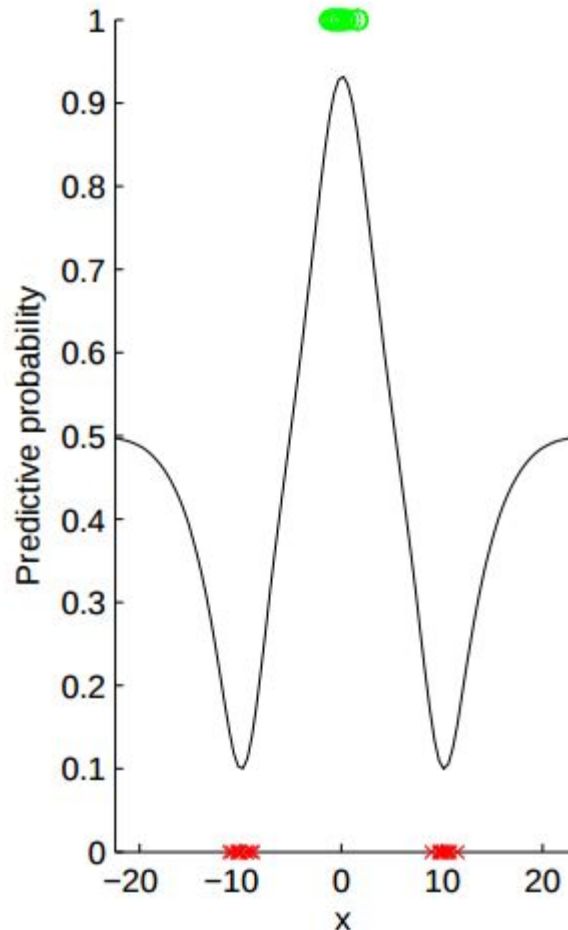
Squash it through a sigmoid function:

$$s(-\infty) = 0$$

$$s(0) = 0.5$$

$$s(\infty) = 1$$

Use that as the predictive probability



Evaluation

Underactuated Pendulum

Motor has insufficient torque to directly swing the pendulum up.

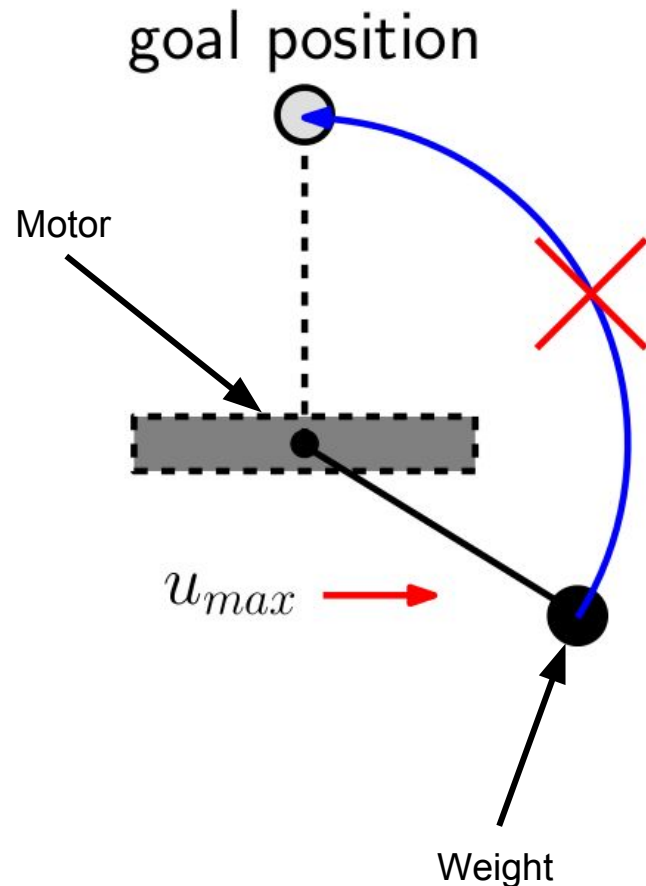
Must swing back and forth first to build up energy.

5 Hz control.

2-D State (angular position and velocity)

1-D Action (torque)

At each step, receive penalty that is a function of distance from goal + small noise.



GPDP Configuration

$\gamma = 1$ (undiscounted)

$N = 10$ steps (2 seconds)

$k(x_i, x_j) = SE(x_i, x_j) + \sigma_y$ (if $i == j$)

-They did not specify Σ or σ_y , so I assume they did some kind of optimization.

400 states randomly selected as support points

Tried Continuous policy and Switching policy

"Optimal" Configuration: Discrete DP

$\gamma = 1$ (undiscounted)

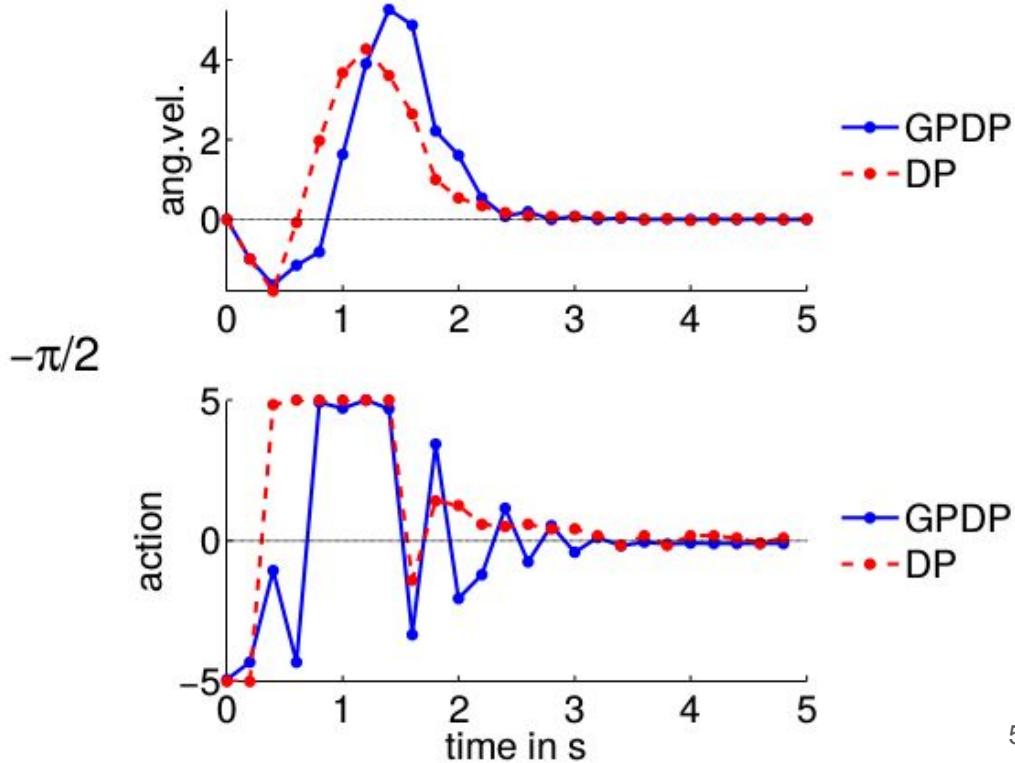
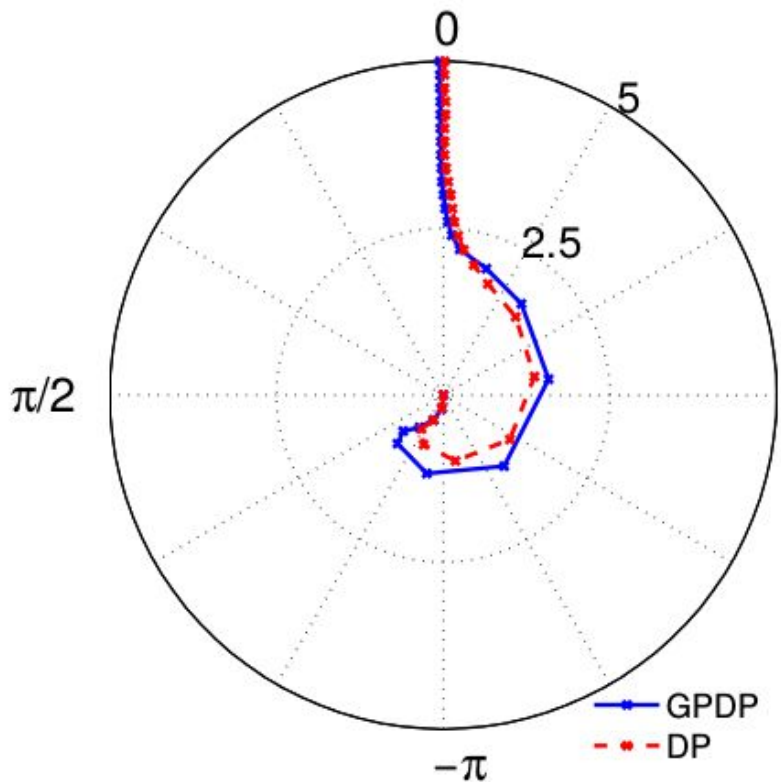
$N = 10$ steps (2 seconds)

620,000 States.

121 possible control inputs.

Results

Example Trajectory: Switching GPDP vs DP



Switching GPDP vs DP

Switching GPDP:

- incurred 15% higher cost than DP on average

- is more aggressive

- reaches upright angle earlier

- sometimes needed an additional swing (incurring a substantially higher cost)

- always succeeded within the time limit

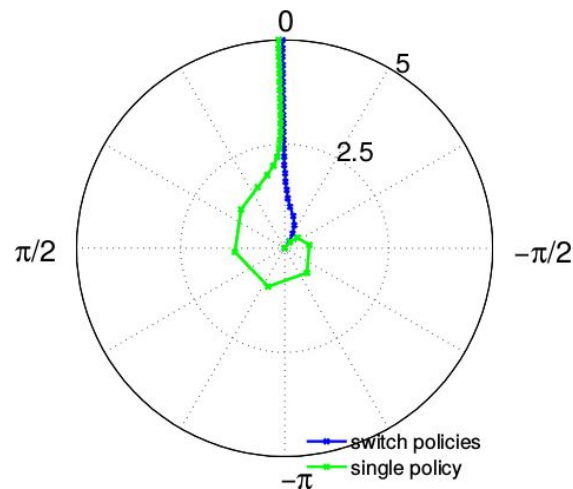
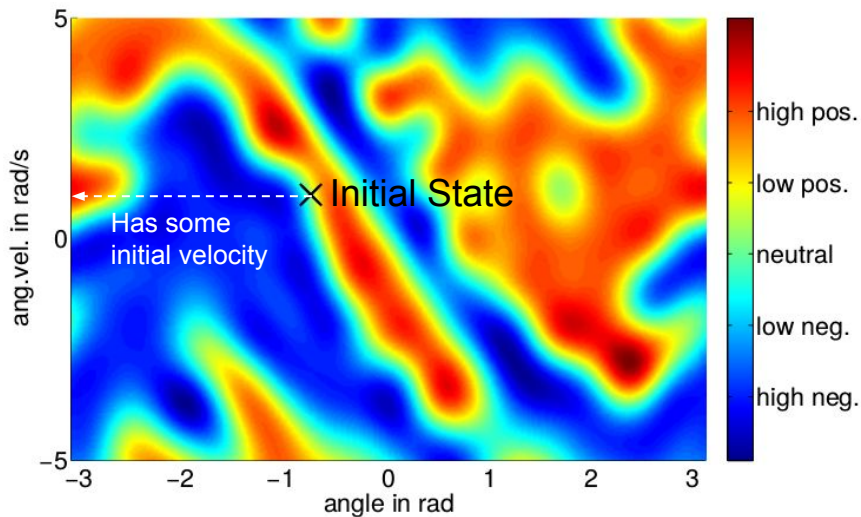
Continuous GPDP vs Switching GPDP

Continuous GPDP:

incurred 20% higher cost than DP on average

almost identical global performance to Switching GPDP

performs poorly when state trajectory reaches boundary of discontinuity



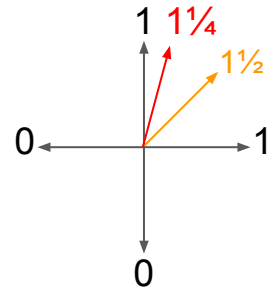
Discussion

Most interested in $Q(s,a)$ near optimal a . Use these a 's as support points in the GP.

When maximizing $Q(s,a)$ over a , subtract a fraction of the predictive variance to penalize uncertain actions.

Might not always be intuitive to divide actions into + and -

Placing support points randomly in state space might not be the best idea. Goal of next section is to place support points intelligently.



"Online Learning"

Challenges of Online Learning

Want to learn as fast as possible, with limited prior understanding of task.

Minimize interactions with environment.

Exploration vs Exploitation

Restrict state space to task-relevant portion for speed while maintaining accuracy

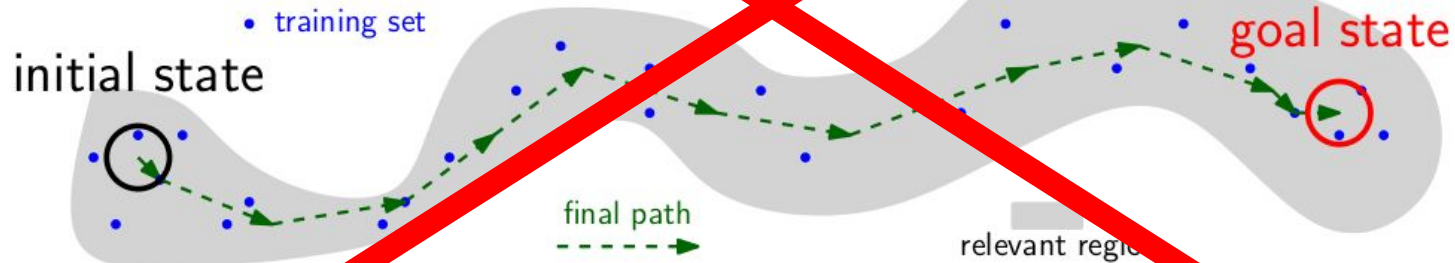


Figure 7: Starting from an initial state, the algorithm iteratively finds a solution to the RL problem without searching the entire state space, but by placing the training set in relevant regions (shaded area) of the state space only.

High Level Algorithm

Move randomly and observe reward and transition functions

Stop

S (support states) = set of states visited (one of which is the current state)


A (support actions) = random set of Actions

For each time step in the horizon:

Look at what states are reachable from S

Try reaching a few of the more useful ones.

Requires a
time machine



Add resulting states to S

Compute $Q(S,A)$ based on the previous time step's $V(S)$

Compute $V(S)$ and $\pi(S)$ based on $Q(S,A)$

Update reward and transition function with observation

Active Learning GPDP (ALGPDP)

Algorithm 4 Online Learning with GPDP

- 1: train \mathcal{GP}_f around initial states \mathcal{X}_N ▷ initialize dynamics model
- 2: $V_N^*(\mathcal{X}_N) = g_{\text{term}}(\mathcal{X}_N) + w_g$ ▷ terminal cost
- 3: $V_N^*(\cdot) \sim \mathcal{GP}_v$ ▷ GP model for V_N^*
- 4: **for** $k = N - 1$ to 0 **do** ▷ DP recursion (in time)
- 5: determine \mathcal{X}_k through Bayesian active learning ▷ Add new support states
- 6: update \mathcal{GP}_f ▷ Update Transition Model
- 7: **for** all $\mathbf{x}_i \in \mathcal{X}_k$ **do** ▷ GP transition model
- 8: **for** all $\mathbf{u}_j \in \mathcal{U}$ **do** ▷ for all support states
- 9: $Q_k^*(\mathbf{x}_i, \mathbf{u}_j) = g(\mathbf{x}_i, \mathbf{u}_j) + w_g + \gamma \mathbb{E}[V_{k+1}^*(\mathbf{x}_{k+1}) | \mathbf{x}_i, \mathbf{u}_j, \mathcal{GP}_f]$ ▷ for all support actions
- 10: **end for**
- 11: $Q_k^*(\mathbf{x}_i, \cdot) \sim \mathcal{GP}_q$ ▷ GP model for Q_k^*
- 12: $\pi_k^*(\mathbf{x}_i) \in \arg \max_{\mathbf{u} \in \mathbb{R}^{n_u}} Q_k^*(\mathbf{x}_i, \mathbf{u})$
- 13: $V_k^*(\mathbf{x}_i) = Q_k^*(\mathbf{x}_i, \pi_k^*(\mathbf{x}_i))$
- 14: **end for**
- 15: $V_k^*(\cdot) \sim \mathcal{GP}_v$ ▷ GP model for V_k^*
- 16: **end for**
- 17: **return** $\mathcal{GP}_v, \mathcal{X}, \pi^*(\mathcal{X}_0) := \pi_0^*(\mathcal{X}_0)$

Bayesian Active Learning

Determines the expected utility (outcomes or information gain) of an experiment.

Don't take explorative actions that aren't likely to give you new information, or have great cost.

Solely maximizing an expected information gain tends to just select states far away from the current state set.

Use Bayesian Active Learning to determine which parts of the state space are relevant.

Bayesian Active Learning

S = initial support states (from random movement)

For each time step

\hat{S} = Candidate Support States = all states reachable from S by action set A

For $i = 1 \dots l$:

s_i^* = the state in \hat{S} with the highest $U(s)$

Take the action that is supposed to reach s_i^*

How do we go back in time?

Observe the state we ended up in, and add it to S

Use S as support points in this step's Q and V

Is this state valuable?

$$U(s) = \rho E[V_k(s)|S_k] + 0.5 \beta \log(\text{var}[V_k(s)|S_k])$$

Is this state unique?

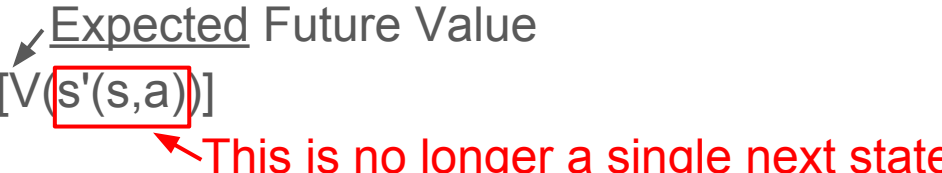
$V(s'(s,a))$ with uncertainty about s'

Previously, we assumed the transition model was known and deterministic.

GPR gives you a probability distribution of next states:

Gaussian with analytically computable mean and variance.

$$Q(s,a) = r(s,a) + \gamma E[\underbrace{V(s'(s,a))}_{\text{Expected Future Value}}]$$



This is no longer a single next state

In a discretized case we do a weighted sum of all possible next states and the probability of seeing them.

In a continuous case we would have to do a weighted integral.

Some math

How to solve: $E[V(s'(s,a))]$		
	known, deterministic $s'(s,a)$	GP $s'(s,a)$
known $V(s)$	$V(s'(s,a))$	$\int V(s'(s,a))p(s')ds'$
GP $V(s)$	$\text{mean}(V(s'(s,a)))$	$\mathbf{1}\beta$

If you use Gaussian Processes for $V(s)$ and $s'(s,a)$, you can compute the integral.

[Goto Slide 9](#)

$$E(V(s^*)) = \mathbf{K}_* \overset{\beta}{\underbrace{(\mathbf{K} + \sigma_y^2 \mathbf{I})^{-1} \mathbf{y}}}$$

Replace this with $\mathbf{1}$

$$\mathbf{l}_i = \int k_v(s_i, s'(s_i, a_j)) p(s'(s_i, a_j)) ds'(s_i, a_j) \quad \leftarrow \text{Depends on your choice of } k(x_i, x_j)$$

Evaluation

Underactuated pendulum

Maximize reward instead of Minimize Cost

$r(s,a)$ depends only on state:

upright = 0

straight down = -1

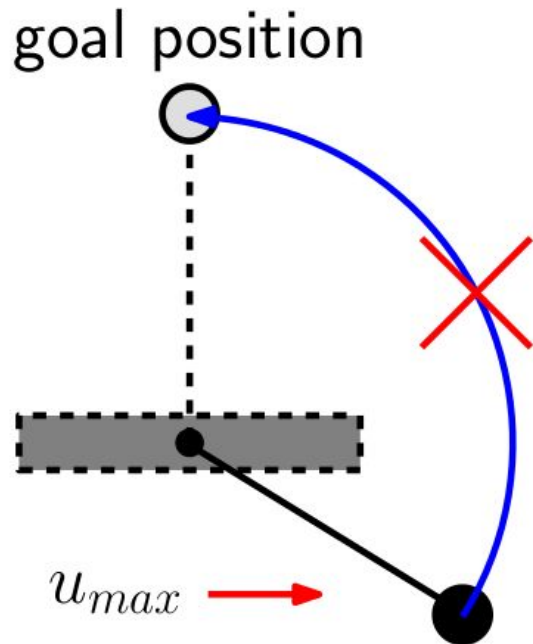
everything else $\in (-1,0)$

$\rho = 1$ $\beta = 2$

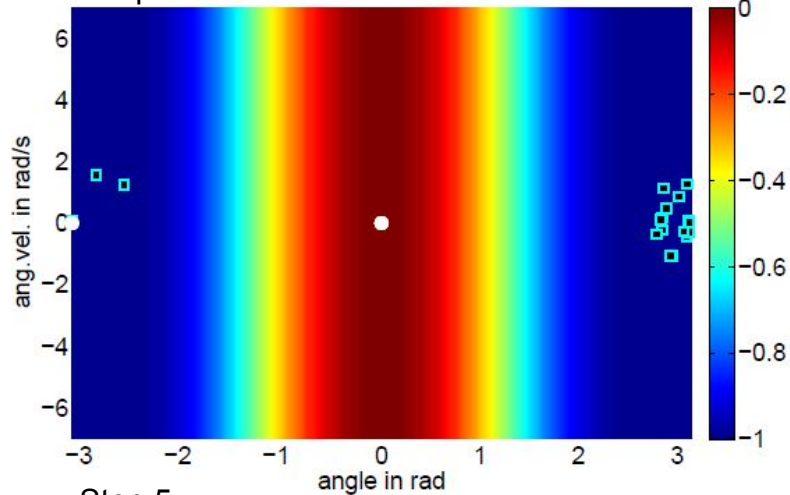
Observe two trajectories of 10 random actions

$l = 13$

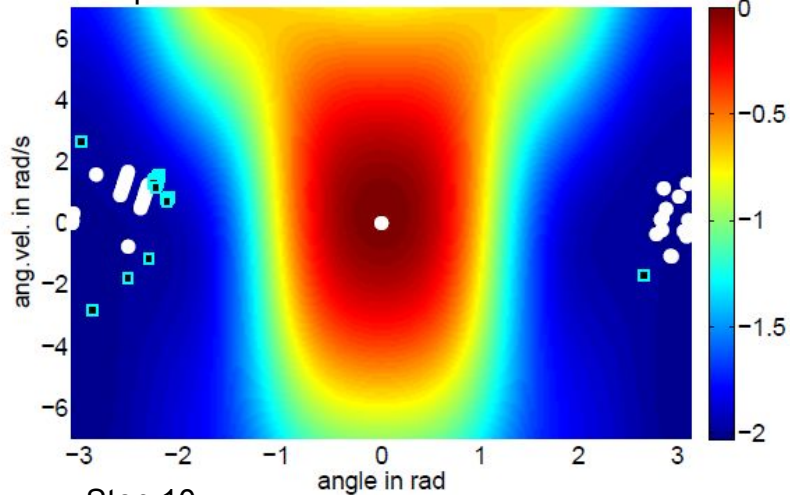
$V(s)$ initialized with two training points: one at start and one at goal.



Step 0



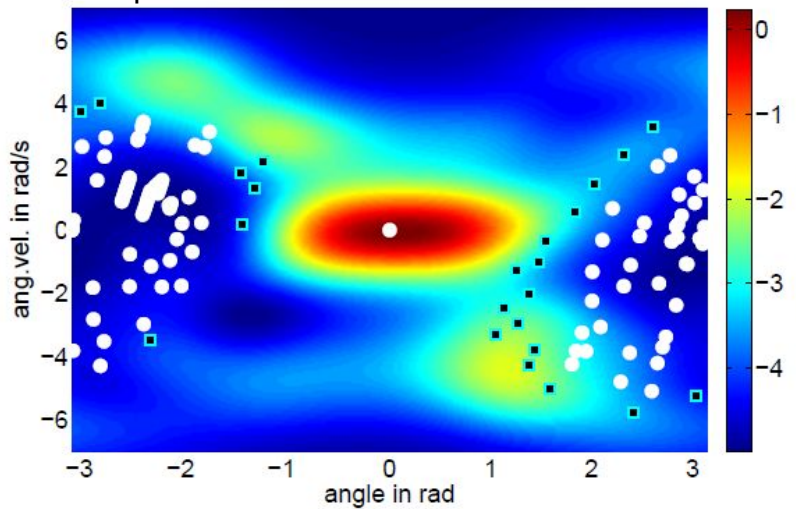
Step 2



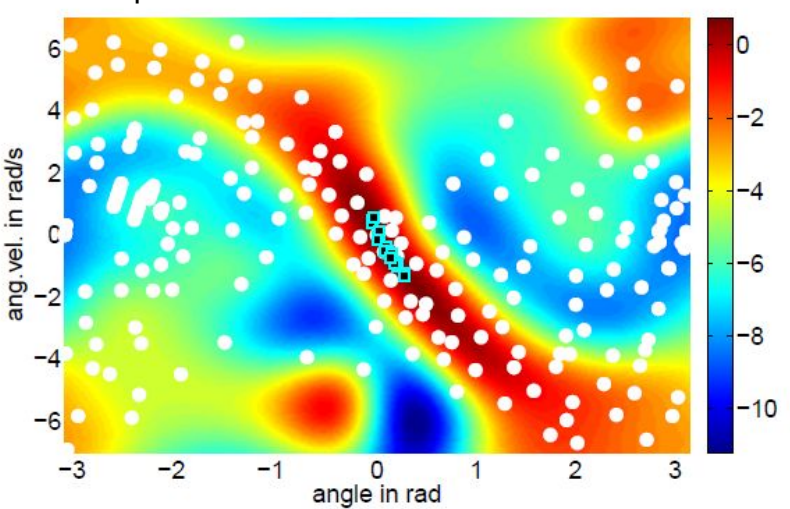
■ = new
support
points

○ = old
support
points

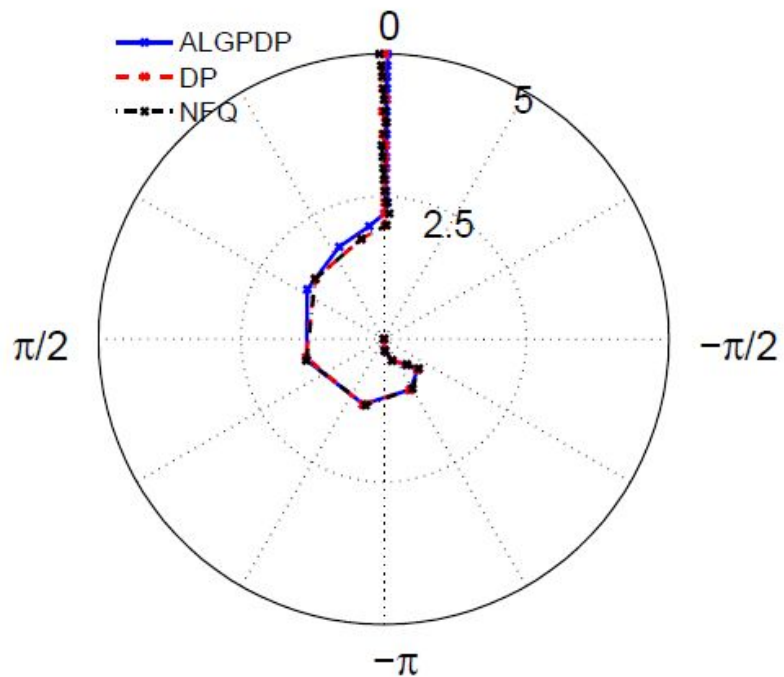
Step 5



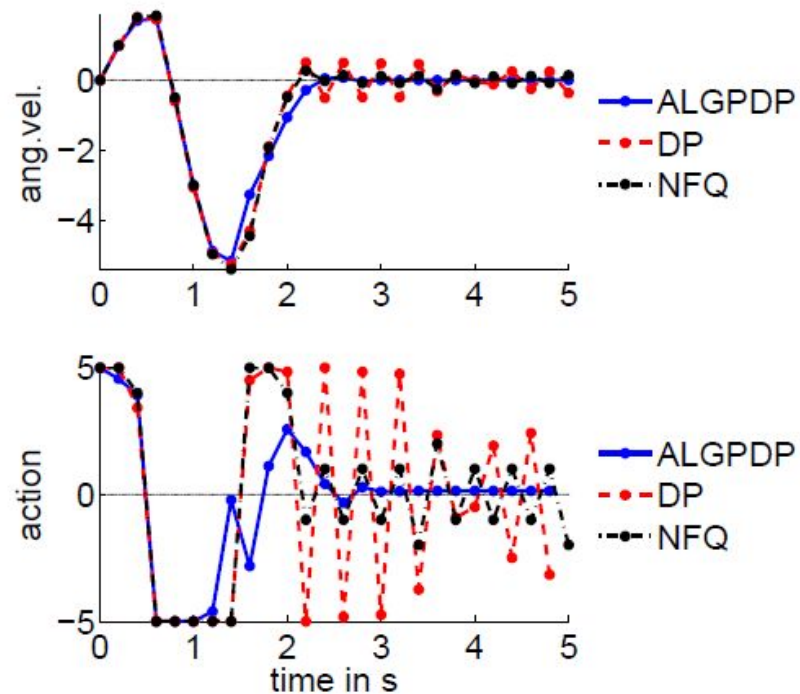
Step 10



Comparison of final policy



(a) Angle trajectories.



(b) Angular velocities and applied actions.

Comparison of final policy

	Reward	Computation Time
DP	-9.60	??? s
Neural Fitted Q Iteration	-9.66	1560 s
ALGPDP	-10.25	256 s

Conclusion

"A major shortcoming of ALGPDP is that it cannot directly be applied to a dynamic system: If we interact with a real dynamic system such as a robot, it is often not possible to experience arbitrary state transitions." - page 32