SARSA (λ) In RKHS

Matthew W. Robards, Peter Sunehag, Scott Sanner

THE AUSTRALIAN NATIONAL UNIVERSITY
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

June 16, 2010

Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection

Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection
- Hand engineering features results in poor generalization in an agent across domains

Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection
- Hand engineering features results in poor generalization in an agent across domains
- We use kernels to automatically linearize a non-linear problem

Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection
- Hand engineering features results in poor generalization in an agent across domains
- We use kernels to automatically linearize a non-linear problem
- We introduce the first memory efficient kernel TD algorithm which allows for eligibility traces

Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection
- Hand engineering features results in poor generalization in an agent across domains
- We use kernels to automatically linearize a non-linear problem
- We introduce the first memory efficient kernel TD algorithm which allows for eligibility traces **with sparsification**
- Furthermore, this is a surprisingly easy to implement algorithm which gives a nice interpretation of the eligibility trace.

Kernel Reinforcement Learning

- Several previous methods have been proposed.

Kernel Reinforcement Learning

- Several previous methods have been proposed.
- KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.

Kernel Reinforcement Learning

- Several previous methods have been proposed.
- KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.
- KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.

Kernel Reinforcement Learning

- Several previous methods have been proposed.
- KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.
- KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.
- Gaussian Processes TD learning was proposed to do online kernel TD learning.

Kernel Reinforcement Learning

- Several previous methods have been proposed.
- KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.
- KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.
- Gaussian Processes TD learning was proposed to do online kernel TD learning.
- These works proposed novel kernel algorithms with novel tricks for memory efficiency.

Kernel Reinforcement Learning

- Several previous methods have been proposed.
- KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.
- KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.
- Gaussian Processes TD learning was proposed to do online kernel TD learning.
- These works proposed novel kernel algorithms with novel tricks for memory efficiency.
- They do not allow for eligibility trace.

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$
- State space $\mathcal{S}$, action space $\mathcal{A}$

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$
- State space $\mathcal{S}$, action space $\mathcal{A}$
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state s to s' through action a

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$
- State space $\mathcal{S}$, action space $\mathcal{A}$
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state s to s' through action a
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a (possibly stochastic) reward function

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$
- State space $\mathcal{S}$, action space $\mathcal{A}$
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state s to s' through action a
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a (possibly stochastic) reward function
- $r_t = R(s_t, a_t, s'|s' = s_{t+1})$ defines the reward when action a in state s results in transition to state s'

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$
- State space $\mathcal{S}$, action space $\mathcal{A}$
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state s to s' through action a
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a (possibly stochastic) reward function
- $r_t = R(s_t, a_t, s' | s' = s_{t+1})$ defines the reward when action a in state s results in transition to state s'
- R_t denotes *return* at time t which gives expected infinite discounted total reward given by $\sum_{i=t}^{\infty} \gamma^{i-t} r_t$, and $0 < \gamma < 1$

Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$
- State space $\mathcal{S}$, action space $\mathcal{A}$
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state s to s' through action a
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a (possibly stochastic) reward function
- $r_t = R(s_t, a_t, s' | s' = s_{t+1})$ defines the reward when action a in state s results in transition to state s'
- R_t denotes *return* at time t which gives expected infinite discounted total reward given by $\sum_{i=t}^{\infty} \gamma^{i-t} r_t$, and $0 < \gamma < 1$
- Assume first order Markov property. ie. $(s_{t+1}, a_{t+1}, r_{t+1})$ is independent of $(s_{t-1}, a_{t-1}, r_{t-1})$ given (s_t, a_t, r_t)

SARSA(λ)

- For larger MDPs, SARSA (λ) is performed using linear function approximation:

SARSA(λ)

- For larger MDPs, SARSA (λ) is performed using linear function approximation:

$$Q(s,a) = \langle w, \phi(s,a) \rangle \tag{1}$$

SARSA(λ)

- For larger MDPs, SARSA (λ) is performed using linear function approximation:

$$Q(s,a) = \langle w, \phi(s,a) \rangle \tag{1}$$

- Traditional update rule for SARSA (λ) using function approximation with regularizer is

SARSA(λ)

- For larger MDPs, SARSA (λ) is performed using linear function approximation:

$$Q(s,a) = \langle w, \phi(s,a) \rangle \tag{1}$$

- Traditional update rule for SARSA (λ) using function approximation with regularizer is

$$w_{t+1} = w_t - \eta_t \left[err(s_t, a_t, R_t) e_t - \xi w_t \right] \tag{2}$$

SARSA(λ)

- For larger MDPs, SARSA (λ) is performed using linear function approximation:

$$Q(s,a) = \langle w, \phi(s,a) \rangle \tag{1}$$

- Traditional update rule for SARSA (λ) using function approximation with regularizer is

$$w_{t+1} = w_t - \eta_t \Big[err(s_t, a_t, R_t) e_t - \xi w_t \Big] \tag{2}$$

- Where $err(s_t, a_t, R_t) = (Q(s_t, a_t) - R_t)$ and $R_t = r_t + \gamma Q(s_{t+1}, a_{t+1})$

SARSA(λ)

- Where e_t is updated through

SARSA(λ)

- Where e_t is updated through

$$e_t := \gamma\lambda e_{t-1} + \phi(s_t, a_t),\ \phi(s,a) = k((s,a),\cdot) \tag{3}$$

SARSA(λ)

- Where e_t is updated through

$$e_t := \gamma\lambda e_{t-1} + \phi(s_t, a_t), \ \phi(s, a) = k((s, a), \cdot) \tag{3}$$

- And is set to $\vec{0}$ at the beginning of each episode

SARSA(λ)

- Where e_t is updated through

$$e_t := \gamma\lambda e_{t-1} + \phi(s_t, a_t),\ \phi(s, a) = k((s, a), \cdot) \tag{3}$$

- And is set to $\vec{0}$ at the beginning of each episode
- Equivalently

SARSA(λ)

- Where e_t is updated through

$$e_t := \gamma\lambda e_{t-1} + \phi(s_t, a_t),\ \phi(s,a) = k((s,a),\cdot) \tag{3}$$

- And is set to $\vec{0}$ at the beginning of each episode
- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i}\phi(s_i, a_i). \tag{4}$$

SARSA(λ)

- Where e_t is updated through

$$e_t := \gamma\lambda e_{t-1} + \phi(s_t, a_t),\ \phi(s,a) = k((s,a),\cdot) \tag{3}$$

- And is set to $\vec{0}$ at the beginning of each episode
- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i}\phi(s_i, a_i). \tag{4}$$

- Where t_0 is the time at which the current episode began

SARSA(λ)

- Where e_t is updated through

$$e_t := \gamma\lambda e_{t-1} + \phi(s_t, a_t), \ \phi(s,a) = k((s,a), \cdot) \tag{3}$$

- And is set to $\vec{0}$ at the beginning of each episode
- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i} \phi(s_i, a_i). \tag{4}$$

- Where t_0 is the time at which the current episode began
- Typically such a representation would be undesirable since it requires storing all past samples

SARSA(λ)

- Where e_t is updated through

$$e_t := \gamma\lambda e_{t-1} + \phi(s_t, a_t),\ \phi(s,a) = k((s,a),\cdot) \tag{3}$$

- And is set to $\vec{0}$ at the beginning of each episode
- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i}\phi(s_i, a_i). \tag{4}$$

- Where t_0 is the time at which the current episode began
- Typically such a representation would be undesirable since it requires storing all past samples
- For now lets assume that kernalizing our algorithm means storing all previously visited state action pairs anyway!

RKHS-SARSA(λ)

- We now do two things:

RKHS-SARSA(λ)

- We now do two things:
  - We substitute the the summed form of the eligibility trace into the update equation, and

RKHS-SARSA(λ)

- We now do two things:
  - We substitute the the summed form of the eligibility trace into the update equation, and
  - We note that by similarly summing the updates of θ we get $\theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot)$

RKHS-SARSA(λ)

- We now do two things:
  - We substitute the the summed form of the eligibility trace into the update equation, and
  - We note that by similarly summing the updates of θ we get $\theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot)$
- By doing this we get nice update equations for the new dual parameters α:

RKHS-SARSA(λ)

- We now do two things:
  - We substitute the the summed form of the eligibility trace into the update equation, and
  - We note that by similarly summing the updates of θ we get $\theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot)$
- By doing this we get nice update equations for the new dual parameters α:

$$\alpha_i' = (1 - \eta\xi)\alpha_i \, i = 1, \ldots, t_0 - 1 \tag{5}$$

$$\alpha_i' = (1 - \eta\xi)\alpha_i - \eta_t err(s_t, a_t, R_t)(\gamma\lambda)^{t-i-1}, i = t_0, \ldots, t - 1 \tag{6}$$

$$\alpha_t' = \eta_t err(s_t, a_t, R_t). \tag{7}$$

RKHS-SARSA(λ)

- We now do two things:
  - We substitute the the summed form of the eligibility trace into the update equation, and
  - We note that by similarly summing the updates of θ we get $\theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot)$
- By doing this we get nice update equations for the new dual parameters α:

$$\alpha_i' = (1 - \eta\xi)\alpha_i \, i = 1, \ldots, t_0 - 1 \tag{5}$$

$$\alpha_i' = (1 - \eta\xi)\alpha_i - \eta_t err(s_t, a_t, R_t)(\gamma\lambda)^{t-i-1}, i = t_0, \ldots, t - 1 \tag{6}$$

$$\alpha_t' = \eta_t err(s_t, a_t, R_t). \tag{7}$$

where t_0 is the time at which the current episode began

Controlling The Memory!

- This provides the foundations for a powerful kernel based reinforcement learning algorithm.

Controlling The Memory!

- This provides the foundations for a powerful kernel based reinforcement learning algorithm.
- Number of samples grows linearly with time. PROBLEM!!!

Controlling The Memory!

- This provides the foundations for a powerful kernel based reinforcement learning algorithm.
- Number of samples grows linearly with time. PROBLEM!!!
- We use ideas from the projectron method of Orabona et. al to make our algorithm more efficient in memory

Controlling The Memory!

- Before adding new sample, we ask ourselves:

Controlling The Memory!

- Before adding new sample, we ask ourselves:
  - How well can this new sample be represented as a linear combination of old ones

Controlling The Memory!

- Before adding new sample, we ask ourselves:
  - How well can this new sample be represented as a linear combination of old ones
  - For poly kernels, in fact, we will eventually span the RKHS and never need to add new samples

Controlling The Memory!

- Before adding new sample, we ask ourselves:
  - How well can this new sample be represented as a linear combination of old ones
  - For poly kernels, in fact, we will eventually span the RKHS and never need to add new samples
- Rather than storing all new samples, consider projecting the newest hypothesis in $\mathcal{H}_t$ onto $\mathcal{H}_{t-1}$

Projectron RKHS-SARSA(λ)

- Now rather than updating the Q function immediately, we consider the projection of Q_{t+1} onto $\mathcal{H}_{t-1}$

Projectron RKHS-SARSA(λ)

- Now rather than updating the Q function immediately, we consider the projection of Q_{t+1} onto $\mathcal{H}_{t-1}$
- Take "temporal hypothesis" $Q'_t = Q_{t+1}$ and its projection $Q''_t = P_{t-1}Q'_t$

Projectron RKHS-SARSA(λ)

- Now rather than updating the Q function immediately, we consider the projection of Q_{t+1} onto $\mathcal{H}_{t-1}$
- Take "temporal hypothesis" $Q'_t = Q_{t+1}$ and its projection $Q''_t = P_{t-1}Q'_t$
- Using linear projection operator P_{t-1}

Projectron RKHS-SARSA(λ)

- Now rather than updating the Q function immediately, we consider the projection of Q_{t+1} onto $\mathcal{H}_{t-1}$
- Take "temporal hypothesis" $Q'_t = Q_{t+1}$ and its projection $Q''_t = P_{t-1}Q'_t$
- Using linear projection operator P_{t-1}

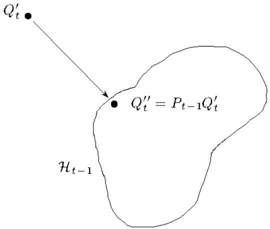


Figure: Projection of temporal hypothesis onto lower RKHS.

Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.

Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility *function* in $\mathcal{H}_k$ given by

Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility *function* in $\mathcal{H}_k$ given by

$$e_t := \sum_{i=t_0}^{t} \beta_i k((s_i, a_i), \cdot) \tag{8}$$

Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility *function* in $\mathcal{H}_k$ given by

$$e_t := \sum_{i=t_0}^{t} \beta_i k((s_i, a_i), \cdot) \tag{8}$$

- Where β is a second set of dual variables.

Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility *function* in $\mathcal{H}_k$ given by

$$e_t := \sum_{i=t_0}^{t} \beta_i k((s_i, a_i), \cdot) \tag{8}$$

- Where β is a second set of dual variables.
- Now we can also perform the projectron method on the eligibility trace.

Dealing With the Eligibility Trace

- Our new update equations are given by

$$\alpha_i' = (1 - \eta\xi)\alpha_i - \eta err(s_t, a_t, R_t)\gamma\lambda\beta_i, \qquad \text{for } i = 1, \ldots, |\mathbb{S}| \tag{9}$$

Dealing With the Eligibility Trace

- Our new update equations are given by

$$\alpha_i' = (1 - \eta\xi)\alpha_i - \eta err(s_t, a_t, R_t)\gamma\lambda\beta_i, \qquad \text{for } i = 1, \ldots, |\mathbb{S}| \tag{9}$$

- and

Dealing With the Eligibility Trace

- Our new update equations are given by

$$\alpha_i' = (1-\eta\xi)\alpha_i - \eta err(s_t, a_t, R_t)\gamma\lambda\beta_i, \qquad \text{for } i = 1, \ldots, |\mathbb{S}| \tag{9}$$

- and

$$\beta_i' = \gamma\lambda\beta_i + \mathbf{d}_i, \qquad \text{for } i = 1, ..., |\mathbb{S}|. \tag{10}$$

- If $\delta_t < \epsilon$ where δ is the norm of the difference between the temporal hypothesis and its projection.
- Moreover d_i's are the parameters of the projection and $|\mathbb{S}|$ is the support set of stored basis functions.

Projectron RKHS-SARSA(λ) Updates

- If $\delta_t > \epsilon$ we use the old updates for α

Projectron RKHS-SARSA(λ) Updates

- If $\delta_t > \epsilon$ we use the old updates for α

$$\alpha_i' = (1 - \eta\xi)\alpha_i \, i = 1, \ldots, t_0 - 1 \tag{11}$$

$$\alpha_i' = (1 - \eta\xi)\alpha_i - \eta_t err(s_t, a_t, R_t)(\gamma\lambda)\beta_i, i = t_0, \ldots, |\mathbb{S}| \tag{12}$$

$$\alpha_{|\mathbb{S}|+1}' = \eta_t err(s_t, a_t, R_t). \tag{13}$$

and simply update β through $\beta_i' = \gamma\lambda\beta_i$ for $i = 1, \ldots, |\mathbb{S}|$ and $\beta_{|\mathbb{S}|+1} = 1$

Mountain Car

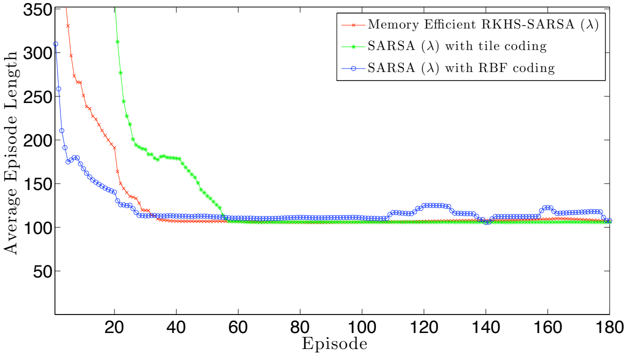


Figure: Moving average time per episode with window 10 evaluated for various algorithms at the end of each episode on mountain car.

Mountain Car

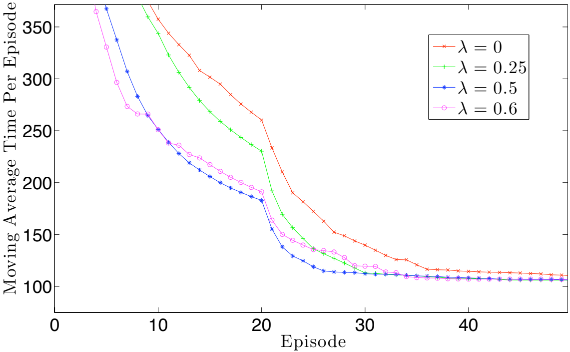
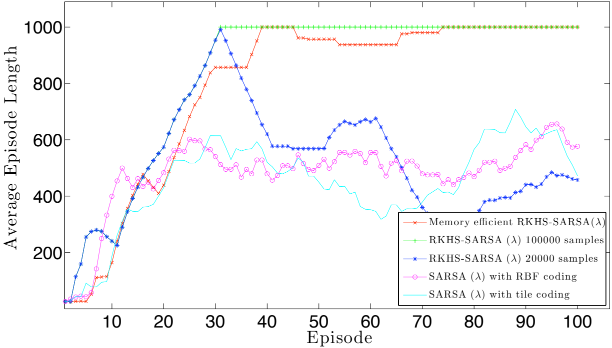


Figure: Moving average time per episode with window 10 evaluated for our algorithm with various values of λ on the mountain car problem 2.

Cart Pole



(a)

Figure: Moving average time per episode with window 10 evaluated for various algorithms at the end of each episode on the cart pole problem.

Cart Pole

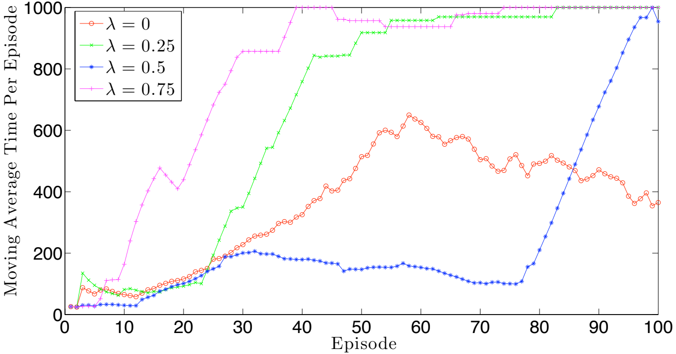


Figure: Moving average time per episode with window 10 evaluated for our algorithm with various values of λ on the cart pole problem.

Memory Efficiency

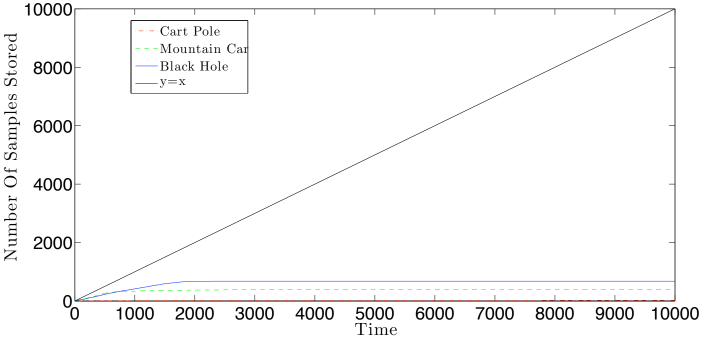


Figure: Number of samples stored by the memory efficient version of our algorithm on each problem.

Algorithm In Summary

- Novel easy to implement algorithm with nice update equations

Algorithm In Summary

- Novel easy to implement algorithm with nice update equations
- Nice way to constrain memory growth

Algorithm In Summary

- Novel easy to implement algorithm with nice update equations
- Nice way to constrain memory growth
- First online kernel TD algorithm to incorporate eligibility traces.

Questions

QUESTIONS???