Quantum Algorithm for Apprenticeship Learning

Andris Ambainis, Debbie Lim

QTML 2024

QUANTERA

UNIVERSITY OF LATVIA

Apprenticeship Learning

Apprenticeship learning

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.
- Useful in applications where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off.

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.
- Useful in applications where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off.
- E.g. driving

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.
- Useful in applications where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off.
- E.g. driving
    - Maintaining a safe distance

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.
- Useful in applications where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off.
- E.g. driving
    - Maintaining a safe distance
    - Staying away from the kerbs

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.
- Useful in applications where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off.
- E.g. driving
  - Maintaining a safe distance
  - Staying away from the kerbs
  - Maintaining a reasonable speed

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.
- Useful in applications where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off.
- E.g. driving
  - Maintaining a safe distance
  - Staying away from the kerbs
  - Maintaining a reasonable speed
- Assign a set of weights.

Apprenticeship learning

- Apprenticeship learning - the task of learning from an expert
- Learning in a setting where we can “observe” an expert demonstrating the task that we want to learn to perform.
- Useful in applications where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off.
- E.g. driving
    - Maintaining a safe distance
    - Staying away from the kerbs
    - Maintaining a reasonable speed
- Assign a set of weights.
- Reward function is tweaked until the desired behaviour is obtained.

Markov Decision Processes (MDPs)

Markov Decision Process (MDP)

Markov Decision Process (MDP)

- Represented by a five-tuple

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathcal{S}$: state space with $|\mathcal{S}| = S$;

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathscr{S}$: state space with $|\mathscr{S}| = S$;
  - $\mathscr{A}$: action space with $|\mathscr{A}| = A$;

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathscr{S}$: state space with $|\mathscr{S}| = S$;
  - $\mathscr{A}$: action space with $|\mathscr{A}| = A$;
  - $R : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ is a reward function;

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathscr{S}$: state space with $|\mathscr{S}| = S$;
  - $\mathscr{A}$: action space with $|\mathscr{A}| = A$;
  - $R : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ is a reward function;
  - $P = \{p(s' | s, a)\}_{s,a}$ is a set of transition probabilities;

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathcal{S}$: state space with $|\mathcal{S}| = S$;
  - $\mathcal{A}$: action space with $|\mathcal{A}| = A$;
  - $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function;
  - $P = \boxed{\{p(s' | s, a)\}_{s,a}}$ is a set of transition probabilities;

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathscr{S}$: state space with $|\mathscr{S}| = S$;
  - $\mathscr{A}$: action space with $|\mathscr{A}| = A$;
  - $R : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ is a reward function;
  - $P = \boxed{\{p(s' \mid s, a)\}_{s,a}}$ is a set of transition probabilities;
  - $\gamma \in [0,1)$ is a discount factor;

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathscr{S}$: state space with $|\mathscr{S}| = S$;
  - $\mathscr{A}$: action space with $|\mathscr{A}| = A$;
  - $R : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ is a reward function;
  - $P = \boxed{\{p(s' | s, a)\}_{s,a}}$ is a set of transition probabilities;
  - $\gamma \in [0,1)$ is a discount factor;
- Use MDP\R to denote an MDP without a reward function.

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathcal{S}$: state space with $|\mathcal{S}| = S$;
  - $\mathcal{A}$: action space with $|\mathcal{A}| = A$;
  - $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function;
  - $P = \boxed{\{p(s' \mid s, a)\}_{s,a}}$ is a set of transition probabilities;
  - $\gamma \in [0,1)$ is a discount factor;
- Use MDP\R to denote an MDP without a reward function.
- A police π is a mapping from states to a probability distribution over actions, $\pi(a \mid s)$.

Markov Decision Process (MDP)

- Represented by a five-tuple
  - $\mathcal{S}$: state space with $|\mathcal{S}| = S$;
  - $\mathcal{A}$: action space with $|\mathcal{A}| = A$;
  - $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function;
  - $P = \boxed{\{p(s' \mid s, a)\}_{s,a}}$ is a set of transition probabilities;
  - $\gamma \in [0,1)$ is a discount factor;
- Use MDP\R to denote an MDP without a reward function.
- A police π is a mapping from states to a probability distribution over actions, $\pi(a \mid s)$.
- Basis functions, aka feature vectors $\phi : \mathcal{S} \times \mathcal{A} \to [0,1]^k$.

Our assumptions

Our assumptions

- We have query access to a feature matrix $\Phi \in \mathbb{R}^{SA \times k}$ whose rows correspond to feature vectors $\phi(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$.

Our assumptions

- We have query access to a feature matrix $\Phi \in \mathbb{R}^{SA \times k}$ whose rows correspond to feature vectors $\phi(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$.
- $\sup_{s \in \mathcal{S}, a \in \mathcal{A}} \| \phi(s, a) \|_2 \leq 1$;

Our assumptions

- We have query access to a feature matrix $\Phi \in \mathbb{R}^{SA \times k}$ whose rows correspond to feature vectors $\phi(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$.
- $\sup_{s \in \mathcal{S}, a \in \mathcal{A}} \|\phi(s, a)\|_2 \leq 1$;
- The reward function is linear, i.e. true reward $R^*(s, a) = w^* \cdot \phi(s, a)$, where $w^* \in \mathbb{R}^k$;

Our assumptions

- We have query access to a feature matrix $\Phi \in \mathbb{R}^{SA \times k}$ whose rows correspond to feature vectors $\phi(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$.

- $\sup\limits_{s \in \mathcal{S}, a \in \mathcal{A}} \|\phi(s, a)\|_2 \leq 1;$

- The reward function is linear, i.e. true reward $R^*(s, a) = w^* \cdot \phi(s, a)$, where $w^* \in \mathbb{R}^k$;

- $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, |R(s, a)| \leq 1;$

Our assumptions

- We have query access to a feature matrix $\Phi \in \mathbb{R}^{SA \times k}$ whose rows correspond to feature vectors $\phi(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$.
- $\sup_{s \in \mathcal{S}, a \in \mathcal{A}} \|\phi(s, a)\|_2 \leq 1$;
- The reward function is linear, i.e. true reward $R^*(s, a) = w^* \cdot \phi(s, a)$, where $w^* \in \mathbb{R}^k$;
- $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, |R(s, a)| \leq 1$;
- $\|w^*\|_1 \leq 1$.

Feature expectation vector

Feature expectation vector

- Expected accumulated discounted feature value vector:

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| s^{(0)} = s, , \pi\right] \in \mathbb{R}^k,$$

Feature expectation vector

- Expected accumulated discounted feature value vector:

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| s^{(0)} = s, , \pi\right] \in \mathbb{R}^k,$$

where the expectation is taken over all random sequence of states drawn by first drawing $s \sim \mathscr{D}$ and choosing actions according to π.

Problem Setting

Problem setting

Problem setting

- Assume access to demonstration by some experts π_E.

Problem setting

- Assume access to demonstration by some experts π_E.

- Obtain an estimate of the *expert's feature expectation* $\mu_E = \mu(\pi_E)$

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi\left(s_i^{(t)}, a_i^{(t)}\right).$$

Problem setting

- Assume access to demonstration by some experts π_E.

- Obtain an estimate of the *expert's feature expectation* $\mu_E = \mu(\pi_E)$

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi\left(s_i^{(t)}, a_i^{(t)}\right).$$

Given an MDP\R, Φ, $\hat{\mu}_E$, find a policy whose performance is close to that of the expert's, on the unknown reward function.

Computational Models

Computational models

Computational models

- Classical

Computational models

- Classical
    - Φ is stored in a ROM

Computational models

- Classical
  - Φ is stored in a ROM
  - Φ' is stored in a RAM

Computational models

- Classical
  - Φ is stored in a ROM
  - Φ' is stored in a RAM
- Quantum

Computational models

- Classical
  - Φ is stored in a ROM
  - Φ' is stored in a RAM
- Quantum

Constant-depth circuits for Boolean functions and quantum memory devices using multi-qubit gates

Jonathan Allcock[1], Jinge Bao[2], Joao F. Doriguello[2,3], Alessandro Luongo[2], and Miklos Santha[2,4]

Computational models

- Classical
  - Φ is stored in a ROM
  - Φ' is stored in a RAM
- Quantum
  - Φ is stored in a Quantum Memory Device (QMD)

Constant-depth circuits for Boolean functions and quantum memory devices using multi-qubit gates

Jonathan Allcock[1], Jinge Bao[2], Joao F. Doriguello[2,3], Alessandro Luongo[2], and Miklos Santha[2,4]

$$\mathcal{O}_\Phi : |s\rangle |a\rangle |\bar{0}\rangle \rightarrow |s\rangle |a\rangle |\phi(s,a)\rangle$$

Computational models

- Classical
  - Φ is stored in a ROM
  - Φ' is stored in a RAM
- Quantum
  - Φ is stored in a Quantum Memory Device (QMD)

$$\mathcal{O}_\Phi : |s\rangle|a\rangle|\bar{0}\rangle \to |s\rangle|a\rangle|\phi(s,a)\rangle$$

  - Φ' is stored in KP-trees and updated via a QMD

Constant-depth circuits for Boolean functions and quantum memory devices using multi-qubit gates

Jonathan Allcock[1], Jinge Bao[2], Joao F. Doriguello[2,3], Alessandro Luongo[2], and Miklos Santha[2,4]

Apprenticeship Learning Algorithm

Algorithm

Algorithm is based on using “inverse reinforcement learning” to try to recover the unknown reward function.

Apprenticeship Learning via Inverse Reinforcement Learning

Pieter Abbeel PABBEEL@CS.STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305, USA

Algorithm

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

Algorithm

- Compute the estimate $\hat{\mu}_E$.

Multivariate mean estimation

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

Algorithm

- Compute the estimate $\hat{\mu}_E$.
- Store $\hat{\mu}_E$ in $\Phi'(1)$.

Multivariate mean estimation

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

Algorithm

- Compute the estimate $\hat{\mu}_E$.
- Store $\hat{\mu}_E$ in $\Phi'(1)$.
- Set $i = 0$.
- Repeat until algorithm terminates:

Multivariate mean estimation

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

Algorithm

- Compute the estimate $\hat{\mu}_E$.

Multivariate mean estimation

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

- Store $\hat{\mu}_E$ in $\Phi'(1)$.
- Set $i = 0$.
- Repeat until algorithm terminates:
  - Obtain a policy $\tilde{\pi}^{(i)}$ for MDP\R augmented with $\Phi\bar{w}^{(i)}$ as the reward function. (For $i = 0$, just pick a random policy.)

RL algorithm

Algorithm

- Compute the estimate $\hat{\mu}_E$.

Multivariate mean estimation

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

- Store $\hat{\mu}_E$ in $\Phi'(1)$.
- Set $i = 0$.
- Repeat until algorithm terminates:
    - Obtain a policy $\tilde{\pi}^{(i)}$ for MDP\R augmented with $\Phi \bar{w}^{(i)}$ as the reward function. (For $i = 0$, just pick a random policy.)

      RL algorithm
    - Obtain an estimate $\mu_q^{(i)}$ of $\mu'^{(i)} := \mu\left(\tilde{\pi}^{(i)}\right)$.

      Multivariate mean estimation

Algorithm

- Compute the estimate $\hat{\mu}_E$.
- Store $\hat{\mu}_E$ in $\Phi'(1)$.
- Set $i = 0$.
- Repeat until algorithm terminates:
  - Obtain a policy $\tilde{\pi}^{(i)}$ for MDP\R augmented with $\Phi\bar{w}^{(i)}$ as the reward function. (For $i = 0$, just pick a random policy.)
  - Obtain an estimate $\mu_q^{(i)}$ of $\mu^{'(i)} := \mu\left(\tilde{\pi}^{(i)}\right)$.
  - Store $\hat{\mu}_E - \mu_q^{'(i)}$ in $\Phi'(i+1)$.

Multivariate mean estimation

RL algorithm

Multivariate mean estimation

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

Algorithm

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

- Compute the estimate $\hat{\mu}_E$. Multivariate mean estimation
- Store $\hat{\mu}_E$ in $\Phi'(1)$.
- Set $i = 0$.
- Repeat until algorithm terminates:
  - Obtain a policy $\tilde{\pi}^{(i)}$ for MDP\R augmented with $\Phi\bar{w}^{(i)}$ as the reward function. (For $i = 0$, just pick a random policy.) RL algorithm
  - Obtain an estimate $\mu_q^{(i)}$ of $\mu'^{(i)} := \mu\left(\tilde{\pi}^{(i)}\right)$. Multivariate mean estimation
  - Store $\hat{\mu}_E - \mu_q^{'(i)}$ in $\Phi'(i+1)$.
  - Obtain an estimate $\bar{w}^{(i)}$ of $w^{(i)} = \arg\max_{w:\|w\|_2 \le 1} \min_{j \in \{0,\cdots,(i-1)\}} w^T\left(\hat{\mu}_E - \mu^{(j)}\right)$. SVM solver

Algorithm

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

- Compute the estimate $\hat{\mu}_E$. Multivariate mean estimation
- Store $\hat{\mu}_E$ in $\Phi'(1)$.
- Set $i = 0$.
- Repeat until algorithm terminates:
  - Obtain a policy $\tilde{\pi}^{(i)}$ for MDP\R augmented with $\Phi\bar{w}^{(i)}$ as the reward function. (For $i = 0$, just pick a random policy.) RL algorithm
  - Obtain an estimate $\mu_q^{(i)}$ of $\mu^{'(i)} := \mu\left(\tilde{\pi}^{(i)}\right)$. Multivariate mean estimation
  - Store $\hat{\mu}_E - \mu_q^{'(i)}$ in $\Phi'(i+1)$.
  - Obtain an estimate $\bar{w}^{(i)}$ of $w^{(i)} = \arg\max_{w:\|w\|_2 \le 1} \min_{j \in \{0,\cdots,(i-1)\}} w^T\left(\hat{\mu}_E - \mu^{(j)}\right)$. SVM solver
  - If there exists some $i_{\min}$ such that $\left\| \hat{\mu}_E - \mu_q^{'(i_{\min})} \right\|_2 \le \epsilon$, then terminate and set $n = i$. Minimum finding

Algorithm

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi\left(s^{(t)}, a^{(t)}\right) \middle| \pi\right]$$

- Compute the estimate $\hat{\mu}_E$. **Multivariate mean estimation**
- Store $\hat{\mu}_E$ in $\Phi'(1)$.
- Set $i = 0$.
- Repeat until algorithm terminates:
  - Obtain a policy $\tilde{\pi}^{(i)}$ for MDP\R augmented with $\Phi\bar{w}^{(i)}$ as the reward function. (For $i = 0$, just pick a random policy.) **RL algorithm**
  - Obtain an estimate $\mu_q^{(i)}$ of $\mu^{'(i)} := \mu\left(\tilde{\pi}^{(i)}\right)$. **Multivariate mean estimation**
  - Store $\hat{\mu}_E - \mu_q^{'(i)}$ in $\Phi'(i+1)$.
  - Obtain an estimate $\bar{w}^{(i)}$ of $w^{(i)} = \arg\max_{w:\|w\|_2 \le 1} \min_{j \in \{0,\cdots,(i-1)\}} w^T\left(\hat{\mu}_E - \mu^{(j)}\right)$. **SVM solver**
  - If there exists some $i_{\min}$ such that $\left\| \hat{\mu}_E - \mu_q^{'(i_{\min})} \right\|_2 \le \epsilon$, then terminate and set $n = i$. **Minimum finding**
  - Set $i = i + 1$.

Total Number of Iterations n

Apprenticeship Learning via Inverse Reinforcement Learning

Pieter Abbeel PABBEEL@CS.STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305, USA

Algorithm terminates when

$$n = \tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

Classical subroutines

Classical subroutines

- SVM solver

Sublinear Optimization for Machine Learning

Kenneth L. Clarkson
IBM Almaden Research Center
San Jose, CA

Elad Hazan*
Department of Industrial Engineering
Technion - Israel Institute of Technology
Haifa 32000 Israel

David P. Woodruff
IBM Almaden Research Center
San Jose, CA

Classical subroutines

- SVM solver

Sublinear Optimization for Machine Learning

Kenneth L. Clarkson
IBM Almaden Research Center
San Jose, CA

Elad Hazan*
Department of Industrial Engineering
Technion - Israel Institute of Technology
Haifa 32000 Israel

David P. Woodruff
IBM Almaden Research Center
San Jose, CA

- Multivariate Monte Carlo estimation

Classical subroutines

- SVM solver

Sublinear Optimization for Machine Learning

Kenneth L. Clarkson
IBM Almaden Research Center
San Jose, CA

Elad Hazan*
Department of Industrial Engineering
Technion - Israel Institute of Technology
Haifa 32000 Israel

David P. Woodruff
IBM Almaden Research Center
San Jose, CA

- Multivariate Monte Carlo estimation
- RL algorithm

Breaking the Sample Size Barrier in Model-Based Reinforcement Learning with a Generative Model

Gen Li Tsinghua
Yuting Wei CMU
Yuejie Chi CMU
Yuantao Gu Tsinghua
Yuxin Chen Princeton

Time Complexity Per Iteration

Time Complexity Per Iteration

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Time Complexity Per Iteration

Dimension of feature vector

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Time Complexity Per Iteration

Dimension of feature vector

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

Error

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Time Complexity Per Iteration

Dimension of feature vector

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

Error

Discount factor

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Time Complexity Per Iteration

Dimension of feature vector

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

Error

Discount factor

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

iterations

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Time Complexity Per Iteration

Dimension of feature vector

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

Error

Discount factor

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

iterations

Size of state space

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Time Complexity Per Iteration

Dimension of feature vector

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

Error

Discount factor

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

iterations

Size of state space

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Size of action space

Time Complexity Per Iteration

Dimension of feature vector

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

Error

Discount factor

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

iterations

Size of state space

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Size of action space

- Total number of iterations $n = \tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$

Time Complexity Per Iteration

Dimension of feature vector

Size of state space

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$$

SVM solver

$$O\left(\frac{n+k}{\epsilon^2}\log n\right)$$

Minimum finding

$$O(n)$$

RL algorithm

$$\tilde{O}\left(\frac{SA}{\epsilon^2(1-\gamma)^3}\right)$$

Error

Discount factor

iterations

Size of action space

- Total number of iterations $n = \tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$
- Total per-iteration time complexity: $\tilde{O}\left(\frac{k+SA}{\epsilon^4(1-\gamma)^3}\right)$

Quantum subroutines

Quantum subroutines

- Quantum SVM solver

Sublinear quantum algorithms for training linear and kernel-based classifiers

Tongyang Li[*] Shouvanik Chakrabarti[†] Xiaodi Wu[‡]

Quantum subroutines

- Quantum SVM solver

Sublinear quantum algorithms for training linear and kernel-based classifiers

Tongyang Li* Shouvanik Chakrabarti† Xiaodi Wu‡

- Quantum multivariate Monte Carlo estimation

Quantum subroutines

- Quantum SVM solver

Sublinear quantum algorithms for training linear and kernel-based classifiers

Tongyang Li[*] Shouvanik Chakrabarti[†] Xiaodi Wu[‡]

- Quantum multivariate Monte Carlo estimation
- Quantum minimum finding

A quantum algorithm for finding the minimum[*]

Christoph Dürr[†] Peter Høyer[‡]

Quantum subroutines

- Quantum SVM solver

Sublinear quantum algorithms for training linear and kernel-based classifiers

Tongyang Li[*] Shouvanik Chakrabarti[†] Xiaodi Wu[‡]

- Quantum multivariate Monte Carlo estimation
- Quantum minimum finding

A quantum algorithm for finding the minimum[*]

Christoph Dürr[†] Peter Høyer[‡]

- Quantum RL algorithm

Quantum Algorithms for Reinforcement Learning with a Generative Model

Daochen Wang [1] **Aarthi Sundaram** [2] **Robin Kothari** [2] **Ashish Kapoor** [3] **Martin Roetteler** [2]

Time Complexity Per Iteration

Time Complexity Per Iteration

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{\sqrt{k}}{\epsilon(1-\gamma)}\right)$$

SVM solver

$$\tilde{O}\left(\frac{\sqrt{n}}{\epsilon^4}+\frac{\sqrt{k}}{\epsilon^8}\right)$$

Minimum finding

$$O\left(\frac{\sqrt{k}}{\epsilon}\right)$$

RL algorithm

$$\tilde{O}\left(\frac{S\sqrt{A}}{\epsilon(1-\gamma)}\right)$$

Time Complexity Per Iteration

Multivariate Monte Carlo

$$\tilde{O}\left(\frac{\sqrt{k}}{\epsilon(1-\gamma)}\right)$$

SVM solver

$$\tilde{O}\left(\frac{\sqrt{n}}{\epsilon^4}+\frac{\sqrt{k}}{\epsilon^8}\right)$$

Minimum finding

$$O\left(\frac{\sqrt{k}}{\epsilon}\right)$$

RL algorithm

$$\tilde{O}\left(\frac{S\sqrt{A}}{\epsilon(1-\gamma)}\right)$$

- Total number of iterations $n = \tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$.

Time Complexity Per Iteration

Multivariate Monte Carlo	SVM solver	Minimum finding	RL algorithm
$\tilde{O}\left(\frac{\sqrt{k}}{\epsilon(1-\gamma)}\right)$	$\tilde{O}\left(\frac{\sqrt{n}}{\epsilon^4}+\frac{\sqrt{k}}{\epsilon^8}\right)$	$O\left(\frac{\sqrt{k}}{\epsilon}\right)$	$\tilde{O}\left(\frac{S\sqrt{A}}{\epsilon(1-\gamma)}\right)$

- Total number of iterations $n = \tilde{O}\left(\frac{k}{\epsilon^2(1-\gamma)^2}\right)$.
- Total per-iteration time complexity: $\tilde{O}\left(\frac{\sqrt{k}+S\sqrt{A}}{\epsilon^8(1-\gamma)^{1.5}}\right)$.

Conclusion

Summary of results

Algorithm	Per-iteration time complexity
Classical approximate algorithm	$\tilde{O}\left(\frac{k + SA}{\epsilon^4(1-\gamma)^3}\right)$
Quantum algorithm	$\tilde{O}\left(\frac{\sqrt{k} + S\sqrt{A}}{\epsilon^8(1-\gamma)^{1.5}}\right)$

Future Directions

Future Directions

- Apprenticeship learning in a setting where the reward function is expressed as a nonlinear function of feature vectors?

Future Directions

- Apprenticeship learning in a setting where the reward function is expressed as a nonlinear function of feature vectors?
- Apply our quantum algorithm as a subroutine to solve learning problems

Future Directions

- Apprenticeship learning in a setting where the reward function is expressed as a nonlinear function of feature vectors?
- Apply our quantum algorithm as a subroutine to solve learning problems
  - The Hamiltonian learning problem


Thank you