Offline Reinforcement Learning:
Neural Function Approximation, Randomization and Sample Complexity

Thanh Nguyen-Tang

Department of Computer Science, Johns Hopkins University

Joint work with

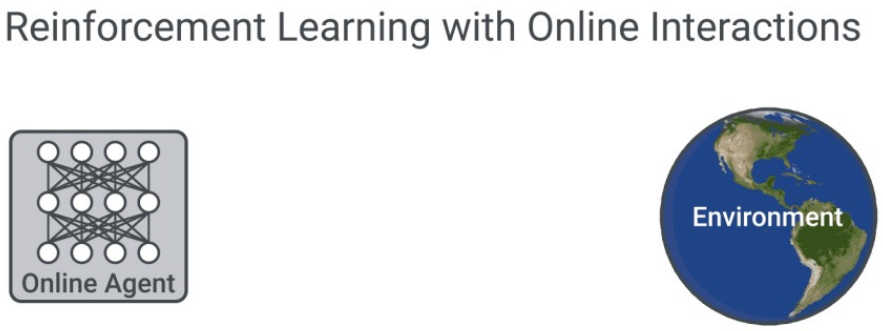
Raman Arora
JHU CS

Preprint: https://thanhnguyentang.github.io/assets/iclr23_submission.pdf

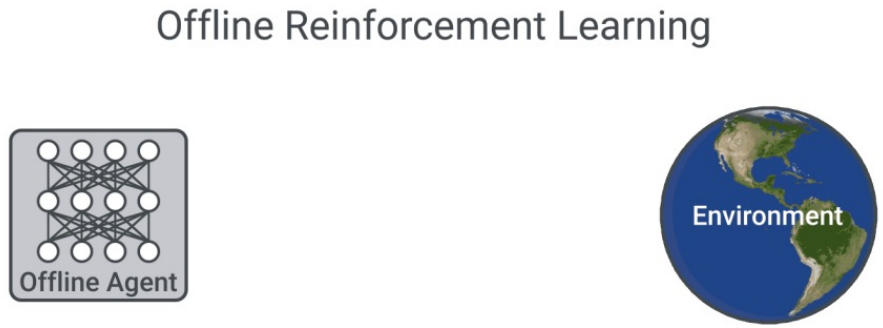
Outline

- **Introduction and research question**
- Background and literature
- Algorithm: Randomized Value Iteration
- Why Randomized Value Iteration works? Key Results
- Experiments
- Future Directions

Why Offline RL?



- Can be extremely costly to run
- Can lead to unsafe/unethical behaviors



- Can be more feasible in many domains
- Can enable better generalization by utilizing large datasets & diverse prior experiences

Image credit: Agarwal and Norouzi (2020)

Why (Neural) Function Approximation?

- RL in large state spaces: we will never get enough data to learn each states individually
  - Game of Go has $> 10^{172}$ states
  - Neural networks: can be efficiently trained by (S)GD
- Crucial challenges:
  - How do we generalize to unseen states and actions?
    - Statistical learning theory: if $\mathbb{E}[Y|X = x] \in \mathcal{F}$, then ERM learns an ϵ-suboptimal predictor using $\mathcal{O}(\frac{\log|\mathcal{F}|}{\epsilon^2})$ samples
    - Offline RL requires new techniques and ideas beyond classical statistical learning
      - Due to non-i.i.d. data and distributional shifts!
  - How do we design computationally efficient algorithms?

Question: Is it possible for offline RL in large state spaces and under partial coverage to be both runtime- and sample-efficient?

- Sample-efficient: the required # of samples is independent of the state space size (and polynomial in other problem factors)
- Runtime-efficient: the algorithm runs in polynomial time
- Partial coverage: the offline data only needs to partially cover the state-action space

Outline

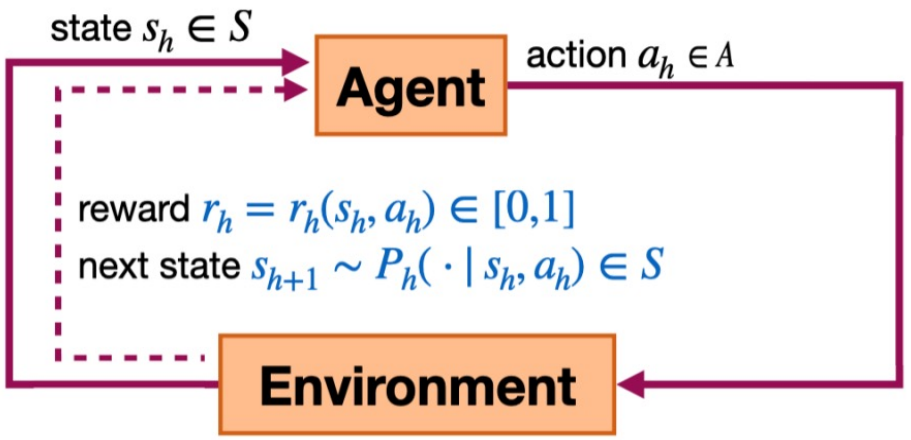
- Introduction and research question
- **Background and literature**
- Algorithm: Randomized Value Iteration
- Why Randomized Value Iteration works? Key Results
- Experiments
- Future Directions

Episodic MDP

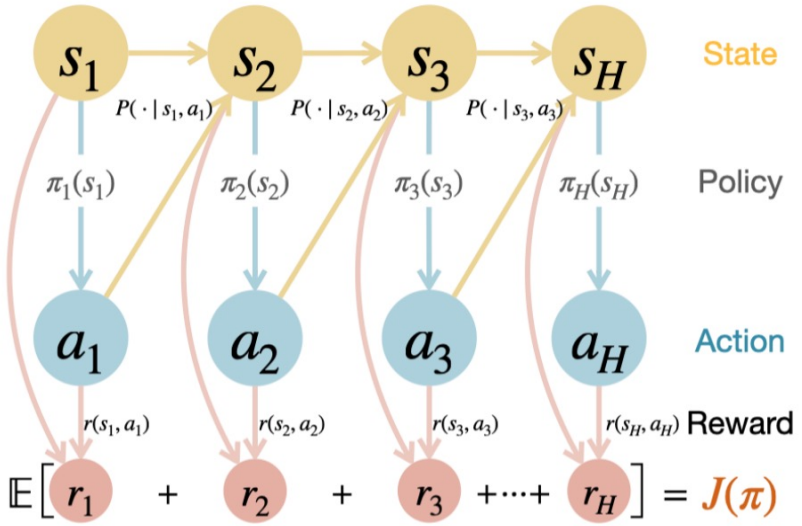
- Episodic time-inhomogeneous Markov decision process

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, H, P, r, d_1)$$

  - State space $\mathcal{S}$
  - Action space $\mathcal{A}$
  - Episode length H:
    - Agent interacts with MDP for H steps and then restart the episode
  - Transition kernels $P = (P_1, \dots, P_H)$, where $P_h : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$
  - Reward functions $r = (r_1, \dots, r_H)$, where r_h
  - Initial state distribution $d_1 \in \Delta(\mathcal{S})$



Episodic MDP



- A policy $\pi = \{\pi_h\}_{h\in[H]}$ where $\pi_h : \mathcal{S} \to \Delta(\mathcal{A})$
- Action-value functions:

$$Q_h^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{i=1}^H r_i | (s_h, a_h) = (s,a)\right]$$

- Value functions

$$V_h^\pi(s) = \mathbb{E}_\pi\left[\sum_{i=1}^H r_i |\, s_h = s\right]$$

- The optimal policy π^* maximizes V_1^π

Dynamic Programming and Bellman Equation

- Optimal action-value functions $Q^* = \{Q_h^*\}_{h\in[H]}$
- Optimal value functions $V^* = \{V_h^*\}_{h\in[H]}$, $V_h^*(s) = \max_a Q_h^*(s,a)$
- Optimal policy π^* is greedy wrt Q^*

$$Q_h^*(s,a) = \underbrace{r_h(s,a) + \mathbb{E}_{s'\sim P_h(\cdot|s,a)}[V_{h+1}^*(s')]}_{\text{Bellman operator } \mathbb{B}_h:\quad \mathbb{B}_h V_{h+1}^*}$$

- Bellman equation

$$Q_h^* = \mathbb{B}_h Q_{h+1}^* \text{ , } Q_{H+1}^* = 0$$

Offline RL with Value Function Approximation

- Offline dataset: collected a priori, $\mathcal{D} = \{(s_h^t, a_h^t, r_h^t)\}_{h\in[H]}^{t\in[K]}$
  - $a_h^t \sim \mu_h(\cdot\,|s_h^t)$, $s_{h+1}^t \sim P_h(\cdot\,|s_h^t, a_h^t)$
  - μ is the behavior policy
  - K: # number of episodes
- No further interactions with MDP
- Learning objective:

$$\mathrm{SubOpt}(\hat{\pi}; s_1) = V_1^*(s_1) - V_1^{\hat{\pi}}(s_1)$$

where $\hat{\pi} = \mathrm{OfflineRLAlgo}(\mathcal{D}, \mathcal{F})$, $\mathcal{F}$ is some function class (e.g., neural networks)

Naïve Value Iteration

- Hope: $\hat{Q}_h \approx Q_h^*$, thus $\hat{\pi} \approx \pi^*$
- Classical and well-known offline RL algorithms is FQI (Ernst, Geurts, and Wehenkel, 2005)
- Classical analysis of FQI (Munos and Szepesvári, 2005)

- End of Episode: $\hat{Q}_{H+1} \leftarrow 0$
- Bootstrapping (backward induction): $h = H, H-1, \ldots, 1$
  - Estimate via fitting a model to the offline dataset (e.g., train a neural net on the data)

$$\mathcal{D} = \{(\underbrace{s_h^k, a_h^k}_{\text{input}}, \underbrace{r_h^k + \hat{V}_{h+1}(s_{h+1}^k)}_{\text{output}})\}_{k \in [K]}$$

  - Obtain predicted action-value functions $\hat{Q}_h(s,a)$
  - Optimize $\hat{\pi}_h(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} \ \hat{Q}_h(s,a)$, and $\hat{V}_h(s) = \max_a \hat{Q}_h(s,a)$

Naïve Value Iteration

- The good ☺
  - Implementation-friendly: Estimate via regression is amenable to neural function approximation with guarantees [Nguyen-Tang et al., 2021]
- The bad ☹
  - doesn't work, even in the simplest case of multi-arm bandits unless …
  - a very strong assumption about data coverage is made!
    - E.g., Classical analysis of FQI (Munos and Szepesvári, 2005; Nguyen-Tang et al. 2021) requires sufficiently large K and

Data coverage assumption

Uniform concentrability coefficient:

$$\kappa = \sup_{\pi} \sup_{h,s,a} \left| \frac{d_h^\pi(s,a)}{\mu_h(s,a)} \right| < \infty$$

Too strong to be useful in practice

$d_h^\pi(s,a)$: the visitation density function at stage h under policy π

[Nguyen-Tang et al., 2021] "Sample Complexity of Offline Reinforcement Learning with Deep ReLU Networks"

Why Naïve Value Iteration fail?

Example: Multi-arm bandits

NaïveVI simply estimates the sample means of all arms from the offline data and act greedily with the sample means

- Arm a were pulled $N(a)$ times in the offline data $\mathcal{D} = \{(a_i, r_i)\}_{i \in [K]}$
- NaïveVI performs:
  - Estimate by sample mean: $\hat{\mu}_a = \frac{\sum_{i:a_i=a} r_i}{N(a)}$
  - Optimize by greedy: $\hat{a} = \text{argmax}_{a \in \mathcal{A}} \hat{\mu}_a$

Why Naïve Value Iteration fail?

Example: Multi-arm bandits

For small $N(a)$, a bad arm might appear good by chance

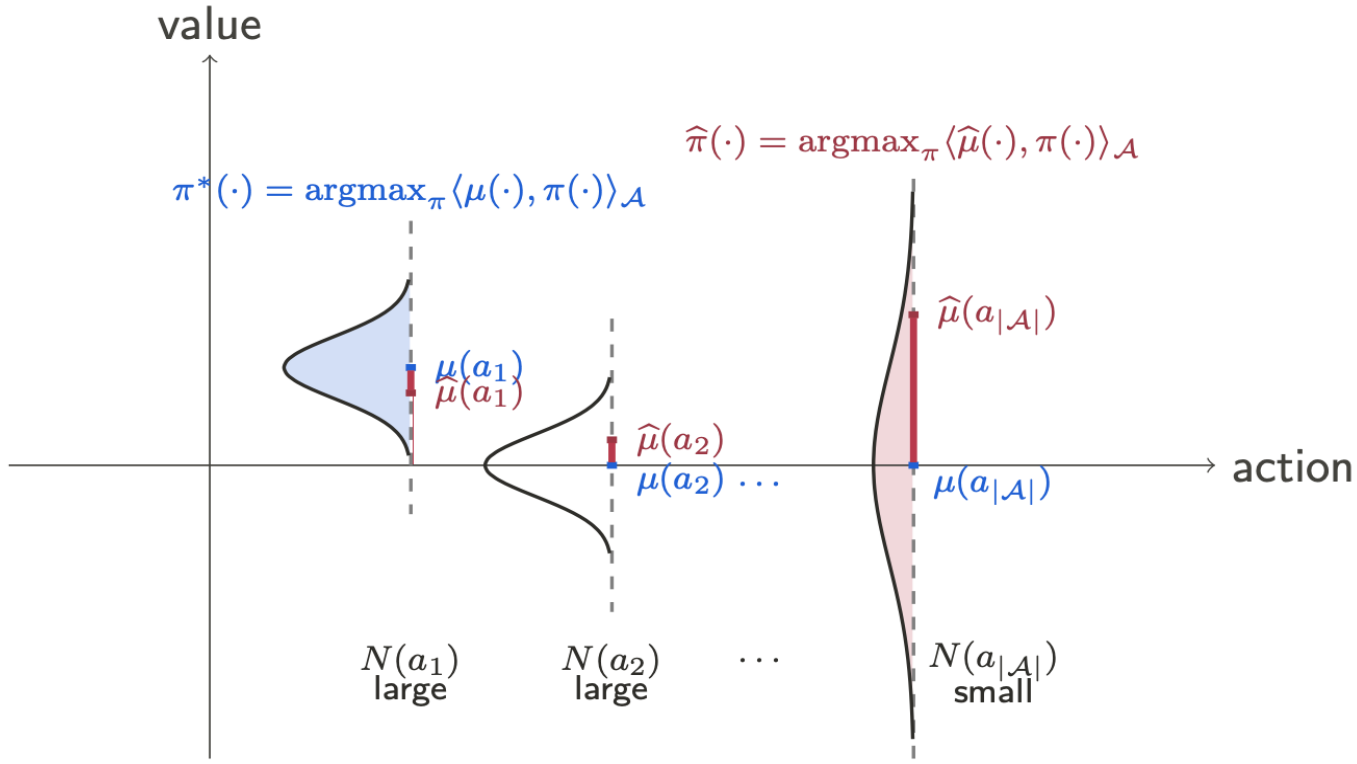
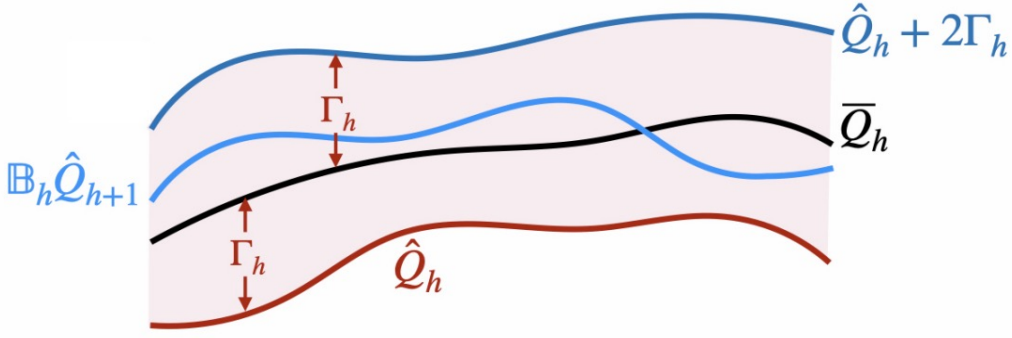


Figure credit: [Jin et al., ICML'21] "Is pessimism provably efficient for offline rl?"

Pessimism

- Intuition: Penalize large epistemic uncertainty
  - High estimated value + high uncertainty: not prefer
  - High estimated value + low uncertainty: prefer
- Algorithmic framework: Pessimistic value iteration via Lower Confidence Bounds

$$\underbrace{\hat{Q}_{\mathrm{h}}(\mathrm{s},\mathrm{a})}_{LCB} = \underbrace{\bar{Q}_h(s,a)}_{\text{NaïveVI}} - \underbrace{\Gamma_h(s,a)}_{\text{Uncertainty quantifier}}$$



[Jin et al., ICML'21] "*Is pessimism provably efficient for offline rl?*"

Pessimistic Value Iteration: The Good ☺

LCB-type algorithms enjoy well-established theoretical guarantees

under mild data coverage assumption

- Partial data coverage assumption: $\kappa := \sup_{h,s,a} \frac{d_h^*(s,a)}{d_h^\mu(s,a)} < \infty$
- E.g., Minimax guarantee in linear MDPs:

$$\text{SubOpt}(\hat{\pi}) = \tilde{\mathcal{O}}(\kappa H^2 d^{\frac{3}{2}} \cdot K^{-\frac{1}{2}})$$

- E.g., Gap-dependent guarantee in linear MDPs:

$$\text{SubOpt}(\hat{\pi}) = \tilde{\mathcal{O}}(\kappa^3 H^2 d^3 \Delta_{min}^{-1} \cdot K^{-1})$$

Uniform data coverage

$$\sup_{\pi} \sup_{h,s,a} \left| \frac{d_h^\pi(s,a)}{\mu_h(s,a)} \right| < \infty$$

[Nguyen-Tang et al., AAAI'23] "*On Instance-Dependent Bounds for Offline Reinforcement Learning with Linear Function Approximation*"

Pessimistic Value Iteration: The Bad ☹

- Uncertainty quantifier $\Gamma_h(s,a)$ is infeasible for many practical models such as neural networks
- E.g. [Nguyen-Tang et al., ICLR'22]

$$\Gamma_h(s,a) = \beta||\nabla_W f(s,a;W)||_{\Lambda_h^{-1}}$$

  - f$(s,a;W)$: a trained neural network with parameter W
  - $\nabla_W f(s,a;W)$: the gradient of the neural network w.r.t. its parameters
  - $\Lambda_h = \lambda I + \sum_{k=1}^{K} \nabla_W f\left(s_h^k, a_h^k; W\right)\left(\nabla_W f\left(s_h^k, a_h^k; W\right)\right)^T$ is the empirical covariance matrix
  - Computation complexity: (# of network parameters)^2

[Nguyen-Tang et al., ICLR'22] "Offline neural contextual bandits: Pessimism, Optimization, and Generalization"

Outline

- Introduction and research question
- Background and literature
- **Algorithm: Randomized Value Iteration**
- Why Randomized Value Iteration works? Key Results
- Experiments
- Future Directions

Randomized Value Iteration

Implicit Pessimism via Reward Perturbing

Best of both worlds

- Runtime-efficient like naïve VI ☺
  - Amenable to neural function approximation trained by (S)GD ☺
    - No need to construct uncertainty quantifier during training ☺
  - Perform favorably or better than Pessimistic Value Iteration in various benchmarks in practice ☺
- Sample-efficient like Pessimistic Value Iteration ☺
  - Guarantees under partial data coverage ☺

Randomized Value Iteration

Implicit Pessimism via Reward Perturbing

- End of Episode: $\tilde{Q}_{H+1} \leftarrow 0$
- Bootstrapping (backward induction): for $h = H, H-1, \dots, 1$
  - Perturb the rewards with independent Gaussian noises $\xi_h^{k,i} \sim \mathcal{N}(0, \sigma^2)$ to create M perturbed datasets $\widetilde{\mathcal{D}}_h^1, \widetilde{\mathcal{D}}_h^2, \dots, \widetilde{\mathcal{D}}_h^M$

$$\widetilde{\mathcal{D}}_h^i = \{(\underbrace{s_h^k, a_h^k}_{\text{input}}, \underbrace{r_h^k + \tilde{V}_{h+1}(s_{h+1}^k) + \xi_h^{k,i}}_{\textbf{perturbed output}})\}_{k \in [K]}$$

  - Train a different neural network in each $\widetilde{\mathcal{D}}_h^i$ to get $f(s, a; W^i)$
  - Pessimistically estimate

$$\tilde{Q}_h(s, a) = \min_{i \in [M]} f(s, a; W^i)$$

  - Optimize $\tilde{\pi}_h(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} \ \tilde{Q}_h(s, a)$ and $\tilde{V}_h(s) = \max_a \tilde{Q}_h(s, a)$
  - Move to step h-1 and repeat

Algorithmic benefit highlights

- No need to maintain any statistical confidence regions ☺
- Train efficiently like the naïve value iteration method ☺

Why such Randomized Value Iteration help offline RL? 🧐

Outline

- Introduction and research question
- Background and literature
- Algorithm: Randomized Value Iteration
- **Why Randomized Value Iteration works? Key Results**
- Experiments
- Future Directions

Why Randomized Value Iteration works? 🧐

- Define the model evaluation error (i.e., Bellman error)

$$\mathrm{err}_{\mathrm{h}}(s,a) = (\mathbb{B}_h \tilde{V}_h - \tilde{Q}_h)(s,a)$$

- Error decomposition:

$$\mathrm{SubOpt}(\tilde{\pi}; s_1) = -\sum_{h=1}^{H} \mathbb{E}_{\tilde{\pi}}[\mathrm{err}_{\mathrm{h}}(\mathrm{s}_{\mathrm{h}}, \mathrm{a}_{\mathrm{h}})] + \sum_{h=1}^{H} \mathbb{E}_{\pi^*}[\mathrm{err}_{\mathrm{h}}(s_h, a_h)]$$

$$\underbrace{+\sum_{h=1}^{H} \mathbb{E}_{\pi^*}\left[\langle \tilde{Q}_h(s_h,\cdot), \pi_h^*(\cdot\,|s_h) - \tilde{\pi}_h(\cdot\,|s_h)\rangle_{\mathcal{A}}\right]}_{\leq 0}$$

Why Randomized Value Iteration works? 🧐

We need to control the model evaluation error

$$\mathrm{err}_{\mathrm{h}}(s,a) = (\mathbb{B}_h \tilde{V}_h - \tilde{Q}_h)(s,a)$$

The key technical challenges:

- There is no explicit uncertainty quantifier (e.g., LCB)
- $\tilde{Q}_h$ were obtained by training a deep neural network using (S)GD

Overparameterized neural networks

- Action-value functions are approximated by a two-layer neural net

$$f(x; W) = \frac{1}{\sqrt{m}} \sum_{i=1}^{m} b_i \cdot \mathrm{ReLU}(w_i^T x)$$

  - $x = (s, a) \in \mathcal{X} = \mathcal{S} \times \mathcal{A} \in \mathbb{R}^d$
  - m: network width
  - $b_i \sim \mathrm{Unif}(\{-1, 1\})$: fixed during training
  - $W = (w_1, w_2, \ldots, w_m) \in \mathbb{R}^{md}$: trainable
  - W_0: initialization of W
    - $w_i \sim \mathcal{N}(0, \frac{I_d}{d})$

Overparameterized neural networks

- overparameterized in the sense that

$$\text{the width } m \gg \#\text{number of samples } K$$

- Overparameterization: effective to study the convergence and interpolation of neural networks (Arora et al., 2019; Allen-Zhu et al., 2019; Hanin & Nica, 2019; Cao & Gu, 2019; Belkin, 2021)
- In the overparameterization regime, the dynamics of the training of the neural network can be captured by the framework of **neural tangent kernel (NTK)** (Jacot et al., 2018).

Neural Tangent Kernel (NTK) regime

- Taylor expansion around the init W_0

$$f(x; W) \approx f(x; W_0) + \nabla_W f(x; W_0) \cdot (W - W_0) + \underbrace{o(||W - W_0||_2^2)}_{\text{small when m gets large!}}$$

- Overparameterized neural nets behave like kernel regression, with the init NTK kernel

$$\mathrm{K}_{\mathrm{int}}(x_1, x_2) = \langle \nabla_W f(x_1; W_0), \nabla_W f(x_2; W_0) \rangle$$

- In the infinite width limit, K_{int} apporaches the NTK kernel

$$K_{ntk}(x_1, x_2) = \mathbb{E}_{w \sim \mathcal{N}(0, \frac{I_d}{d})} \langle x_1 \cdot 1\{w^T x_1 \geq 0\}, x_2 \cdot 1\{w^T x_2 \geq 0\} \rangle$$

Neural Tangent Kernel (NTK) regime

- The NTK kernel K_{ntk} induces a reproducing kernel Hilbert space (RKHS) $\mathcal{H}_{ntk}$
- Effective dimension

$$\tilde{d}_h = \frac{\mathrm{logdet}(I_K + \mathcal{K}_h/\lambda)}{\log(1 + K/\lambda)}$$

where $\mathcal{K}_h = \left[K_{ntk}\left(x_h^i, x_h^j\right)\right]_{1 \le i,j \le K}$ is the Gram matrix of the NTK kernel on the input data.

- $\tilde{d} = \max_{h \in [H]} \tilde{d}_h$

Validity of implicit uncertainty quantifier

- "De-randomize" the randomized value functions:

$$\underbrace{f(x;\widehat{W}_h)}_{\text{found by GD on non-perturbed data}} - \underbrace{\mathbb{B}_h\,\tilde{V}_{h+1}(x)}_{\text{Bellman target}} \le \beta\,\underbrace{||\nabla_W f(x;W_0)||_{\Lambda_h^{-1}}}_{\text{elliptical potential}} + o(1)$$

where $\beta = \mathcal{O}\left(H\sqrt{\tilde{d}\log K}\right)$, and $\Lambda_h = \lambda I + \sum_{k=1}^{K} \nabla_W f(x;W_0)\cdot\nabla_W f(x;W_0)^T$

  - Bellman completeness $\forall\, V \in (\mathcal{S}\to[0,H]), \mathbb{B}_h V \in \mathcal{H}_{ntk}$

- Anti-concentration of the randomized value functions

$$\tilde{Q}_h(x) \le \underbrace{\langle \nabla_w f(x;W_0), \widehat{W}_h - W_0\rangle}_{\text{linearized non-perturbed value func}} - \sigma||\nabla_W f(x;W_0)||_{\Lambda_h^{-1}} + o(1)$$

- Linearize neural net in the NTK regime $f\left(x;\widehat{W}_h\right) \approx \langle \nabla_w f(x;W_0), \widehat{W}_h - W_0\rangle$

*All inequalities hold with high probability under appropriate choices of algorithm hyperparameters

Lower bound $\mathbf{err_h(x)}$

- Choose $\sigma = \beta = \mathcal{O}\left(H\sqrt{\tilde{d}\log K}\right)$

$$\begin{aligned}\mathbb{B}_h\,\tilde{V}_h(x) &\geq f\left(x;\widehat{W}_h\right) - \beta\,||\nabla_W f(x;W_0)||_{\Lambda_h^{-1}} - o(1)\\ &\geq \left\langle \nabla_w\, f(x;W_0), \widehat{W}_h - W_0\right\rangle - \sigma||\nabla_W f(s,a;W_0)||_{\Lambda_h^{-1}} - o(1)\\ &\geq \tilde{Q}_h(x) - o(1)\end{aligned}$$

- Thus,

$$\mathrm{err_h}(x) = \mathbb{B}_h\,\tilde{V}_h(x) - \tilde{Q}_h(x) \geq -o(1)$$

Upper bound $\mathbf{err_h}(x)$

$$\mathrm{err_h}(x) = \mathbb{B}_h\, \tilde{V}_h(x) - \tilde{Q}_h(x)$$

$$= \underbrace{\mathbb{B}_h\, \tilde{V}_h(x) - f\left(x; \widehat{W}_h\right)}_{\text{de-randomize}} + \underbrace{f\left(x; \widehat{W}_h\right) - \tilde{Q}_h(x)}_{\substack{\text{linearize} \\ \text{+concentration of Gaussian}}}$$

$$\leq \beta\, ||\nabla_W f(x; W_0)||_{\Lambda_h^{-1}} + \sigma\, ||\nabla_W f(x; W_0)||_{\Lambda_h^{-1}} + o(1)$$

Bound SubOpt

- Plug the model evaluation error control in the error decomposition:

$$\begin{aligned}\mathrm{SubOpt}(\tilde{\pi}; s_1) &\leq -\sum_{h=1}^{H} \mathbb{E}_{\tilde{\pi}}[\mathrm{err_h}(s_h, a_h)] + \sum_{h=1}^{H} \mathbb{E}_{\pi^*}[\mathrm{err_h}(s_h, a_h)] \\ &\leq \quad \mathrm{o}(1) \quad + \beta \sum_{h=1}^{H} \mathbb{E}_{\pi^*}\left[||\nabla_W f(s, a; W_0)||_{\Lambda_h^{-1}} \right]\end{aligned}$$

- Control distributional shift: Assume $\kappa := \sup_{h,s,a} \frac{d_h^*(s,a)}{d_h^{\mu}(s,a)} < \infty$
- Bound the second term by marginal importance sampling and the inequality for elliptical potential:

$$\mathrm{SubOpt}(\tilde{\pi}) = \tilde{\mathcal{O}}\left(\frac{H^{2.5} \cdot \kappa \cdot \tilde{d}}{\sqrt{K}}\right)$$

Comparison

work	bound	i.i.d?	explorative data?	finite spectrum?	matrix inverse?	opt
Jin et al. (2021)	$\tilde{\mathcal{O}}\left(\frac{d_{lin}^{3/2}H^2}{\sqrt{K}}\right)$	no	yes	yes	yes	analytical
Yang et al. (2020)	$\tilde{\mathcal{O}}\left(\frac{H^2\sqrt{\tilde{d}^2+\tilde{d}\tilde{n}}}{\sqrt{K}}\right)$	no	–	no	yes	oracle
Xu & Liang (2022)	$\tilde{\mathcal{O}}\left(\frac{\tilde{d}H^2}{\sqrt{K}}\right)$	yes	yes	yes	yes	oracle
This work	$\tilde{\mathcal{O}}\left(\frac{\kappa H^{5/2}\tilde{d}}{\sqrt{K}}\right)$	**no**	**no**	**no**	**no**	**GD**

Table 1: SOTA results for offline RL with function approximation. The third and fourth columns ask if the corresponding result needs the data to be i.i.d, and well-explored, resp.; the fifth column asks if the induced RKHS needs to have a finite spectrum, the sixth column asks if the algorithm needs to inverse a covariance matrix and the last column presents the optimizer being used. Here $\tilde{n}$ is the log covering number.

Outline

- Introduction and research question
- Background and literature
- Algorithm: Randomized Value Iteration
- Why Randomized Value Iteration works? Key Results
- **Experiments**
- Future Directions

Experiment 1: Hard linear MDPs

- Construct the hard instance of linear MDPs (Yin et al., 2022a; Min et al., 2021) to see if reward perturbing works in this simple setting
- Methods:
  - LinLCB: Pessimistic Value Iteration
  - LinGreedy: Naïve Value Iteration
  - LinPER: Randomized Value Iteration

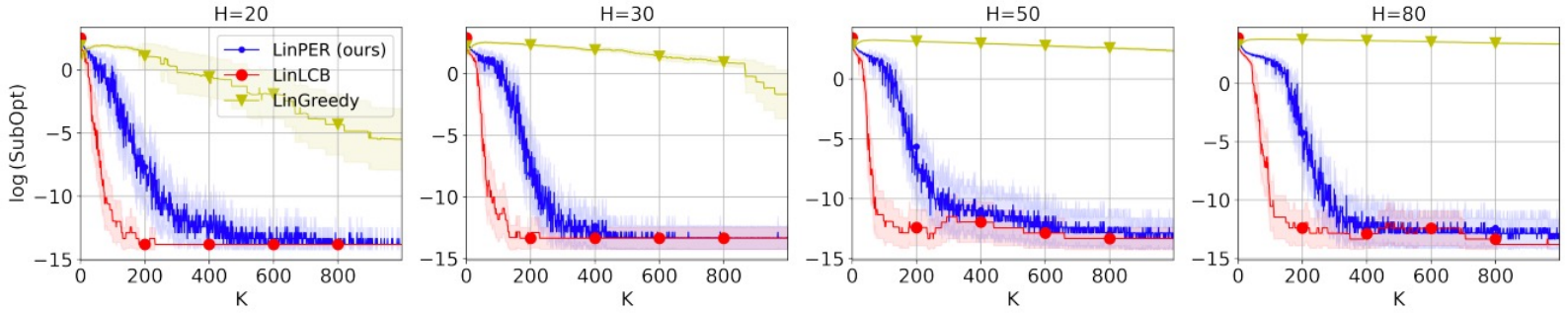


Figure 2: Empirical results of sub-optimality (in log scale) on linear MDPs.

Experiment 2: Neural Contextual Bandits

Methods:

- LinLCB: Pessimistic Value Iteration + linear representation
- LinPER: Randomized Value Iteration + linear representation
- NeuralGreedy: Naïve Value Iteration + neural representation
- NeuraLCB: Pessimistic value Iteration + neural representation
- NeuralPER: Randomized Value Iteration + neural representation

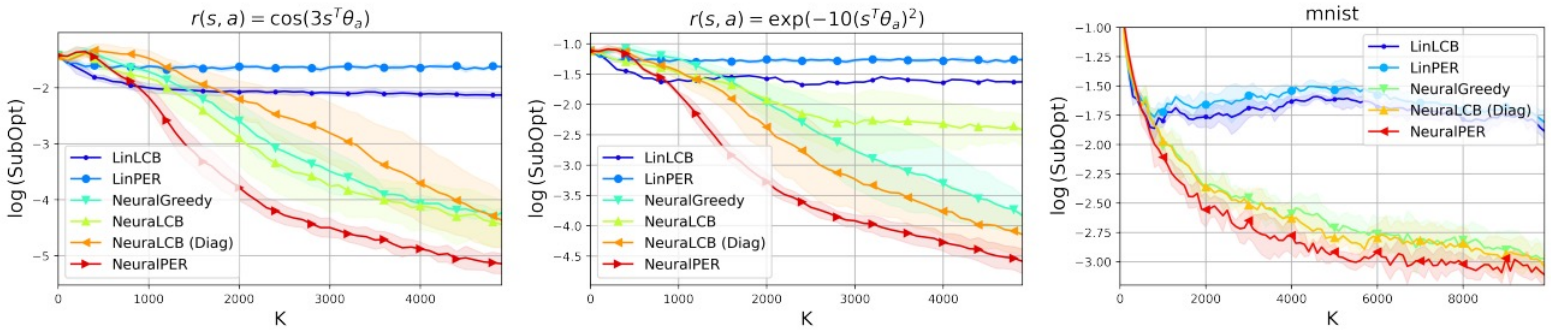


Figure 3: Empirical results of sub-optimality (in log scale) in neural contextual bandits: (a) $r(s,a) = \cos(3s^T\theta_a)$, (b) $r(s,a) = \exp(-10(s^T\theta_a)^2)$, and (c) MNIST.

Runtime efficiency

- LCB spends $O(K^2)$ time in action selection
- Randomized Value Iteration spends $O(1)$ in action selection

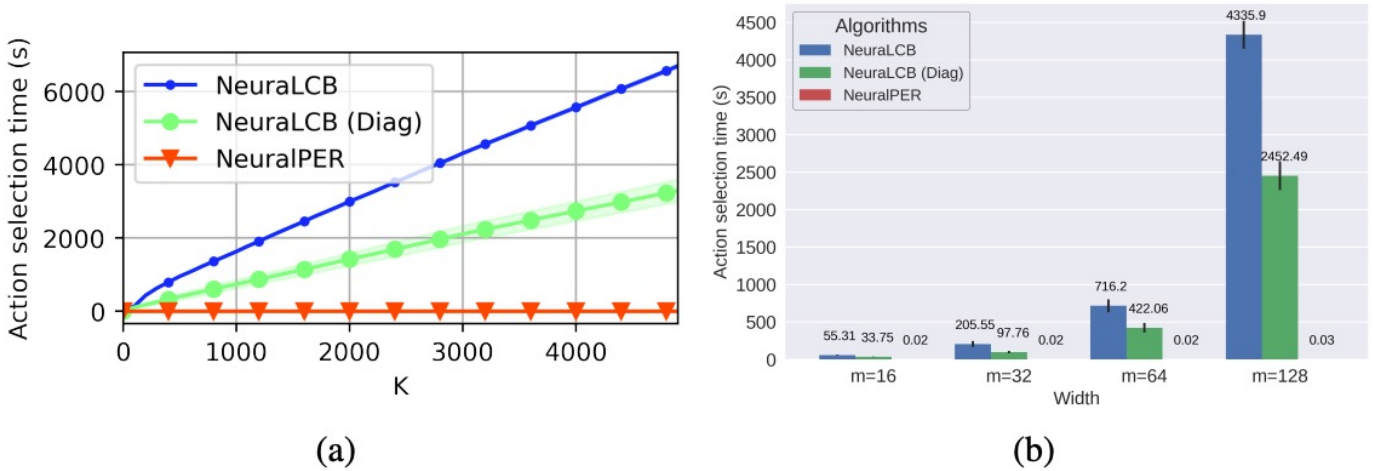


Figure 4: Elapsed time (in seconds) for action selection in the contextual bandit $r(s,a) = 10(s^T\theta_a)^2$: (a) Running time of action selection versus the number of (offline) data points K, and (b) running time of action selection versus the network width m (for $K = 500$).

Controlling pessimism

- M controls the level of pessimism and nicely correlates with the decrease of subopt in practice

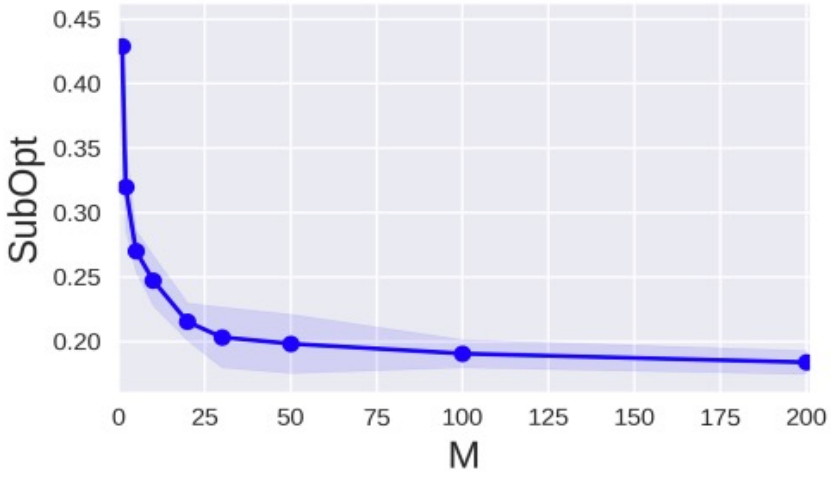
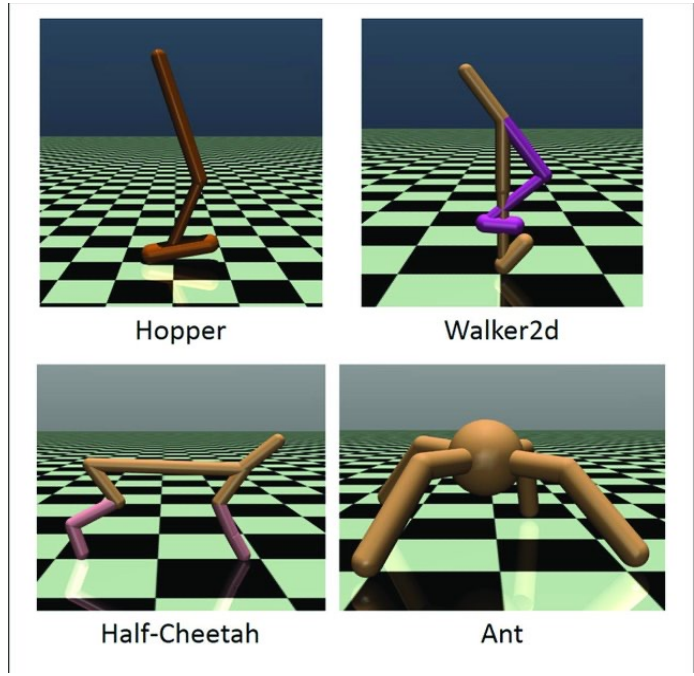


Figure 5: Sub-optimality of NeuralPER versus different values of M.

D4RL benchmark (Fu et al., 2020)

- Standard benchmark for offline RL in continuous domains
- The Gym domain
  - Three environments (HalfCheetah, Hopper, and Walker2d)
  - Five datasets (random, medium, medium-replay, medium-expert, and expert)
    - Random: unroll randomly initialized policies
    - Medium: train soft actor-critic and early stop it and then collect the data using the partially trained policy
    - Medium-replay: medium + all data in the replay buffer
    - Medium-expert: mix expert demonstrations and medium data
    - Expert: expert demonstrations

  → making up 15 different settings



How to adopt Randomized Value Iteration to continuous control?

- M critics $\{Q_{\theta^i}\}_{i\in[M]}$ and one actor π_ϕ, where $\{\theta^i\}_{i\in[M]}$ and ϕ are the parameters of neural networks
- Training the critics by minimizing the Bellman error loss and penalizing the OOD actions

$$\mathcal{L}(\theta^i;\tau) = \left(Q_{\theta^i}(s,a) - (r + \gamma Q_{\bar{\theta}^i}(s') + \xi)\right)^2 + \beta \underbrace{\mathbb{E}_{a'\sim\pi_\phi(\cdot|s)}\left[(Q_{\theta^i}(s,a') - \bar{Q}(s,a'))^2\right]}_{\text{penalization term } R(\theta^i;s,\phi)}, \quad (1)$$

- Train the actor via the soft actor update [Haarnoja et al. (2018)] with the minimum-value critic

$$\max_\phi \left\{\mathbb{E}_{s\sim\mathcal{D},a'\sim\pi_\phi(\cdot|s)}\left[\min_{i\in[M]} Q_{\theta^i}(s,a') - \log\pi_\phi(a'|s)\right]\right\},$$

Result in D4RL

- **BEAR** (Kumar et al., 2019): use MMD distance to constraint policy to the offline data
- **UWAC** (Wu et al., 2021): improves BEAR using dropout uncertainty
- **CQL** (Kumar et al., 2020) that minimizes Q-values of OOD actions
- **MOPO** (Yu et al., 2020): uses model-based uncertainty via ensemble dynamics
- **TD3-BC** (Fujimoto & Gu, 2021): uses adaptive behavior cloning
- **PBRL** (Bai et al., 2022): use uncertainty quantification via disagreement of bootstrapped Q-functions

		BEAR	UWAC	CQL	MOPO	TD3-BC	PBRL	VIPeR
Random	HalfCheetah	2.3 ±0.0	2.3 ±0.0	17.5 ±1.5	**35.9** ±2.9	11.0 ±1.1	11.0 ±5.8	14.5 ±2.1
	Hopper	3.9 ±2.3	2.7 ±0.3	7.9 ±0.4	16.7 ±12.2	8.5 ±0.6	26.8 ±9.3	**31.4** ±0.0
	Walker2d	12.8 ±10.2	2.0 ±0.4	5.1 ±1.3	4.2 ±5.7	1.6 ±1.7	8.1 ±4.4	**20.5** ±0.5
Medium	HalfCheetah	43.0 ±0.2	42.2 ±0.4	47.0 ±0.5	**73.1** ±2.4	48.3 ±0.3	57.9 ±1.5	58.5 ±1.1
	Hopper	51.8 ±4.0	50.9 ±4.4	53.0 ±28.5	38.3 ±34.9	59.3 ±4.2	75.3 ±31.2	**99.4** ±6.2
	Walker2d	-0.2 ±0.1	75.4 ±3.0	73.3 ±17.7	41.2 ±30.8	83.7 ±2.1	**89.6** ±0.7	**89.6** ±1.2
Medium Replay	HalfCheetah	36.3 ±3.1	35.9 ±3.7	45.5 ±0.7	**69.2** ±1.1	44.6 ±0.5	45.1 ±8.0	45.0 ±8.6
	Hopper	52.2 ±19.3	25.3 ±1.7	88.7 ±12.9	32.7 ±9.4	60.9 ±18.8	**100.6** ±1.0	**100.2** ±1.0
	Walker2d	7.0 ±7.8	23.6 ±6.9	81.8 ±2.7	73.7 ±9.4	81.8 ±5.5	77.7 ±14.5	**83.1** ±4.2
Medium Expert	HalfCheetah	46.0 ±4.7	42.7 ±0.3	75.6 ±25.7	70.3 ±21.9	90.7 ±4.3	92.3 ±1.1	**94.2** ±1.2
	Hopper	50.6 ±25.3	44.9 ±8.1	105.6 ±12.9	60.6 ±32.5	98.0 ±9.4	**110.8** ±0.8	**110.6** ±1.0
	Walker2d	22.1 ±44.9	96.5 ±9.1	107.9 ±1.6	77.4 ±27.9	**110.1** ±0.5	**110.1** ±0.3	**109.8** ±0.5
Expert	HalfCheetah	92.7 ±0.6	92.9 ±0.6	96.3 ±1.3	81.3 ±21.8	96.7 ±1.1	92.4 ±1.7	**97.4** ±0.9
	Hopper	54.6 ±21.0	110.5 ±0.5	96.5 ±28.0	62.5 ±29.0	107.8 ±7	**110.5** ±0.4	**110.8** ±0.4
	Walker2d	106.6 ±6.8	108.4 ±0.4	108.5 ±0.5	62.4 ±3.2	**110.2** ±0.3	108.3 ±0.3	108.3 ±0.2
	Average	38.78 ±10.0	50.41 ±2.7	67.35 ±9.1	53.3 ±16.3	67.55 ±3.8	74.37 ±5.3	**78.2** ±1.9

Table 2: Average normalized score and standard deviation of all algorithms over five seeds in the Gym domain in the "v2" dataset of D4RL (Fu et al., 2020). The scores for all the baselines are from Table 1 of Bai et al. (2022). The highest scores are highlighted.

Outline

- Introduction and research question
- Background and literature
- Algorithm: Randomized Value Iteration
- Why Randomized Value Iteration works? Key Results
- Experiments
- **Future Directions**

Future Work

- Match the lower bound in terms of κ?
- Avoid the need of training an ensemble of M models?
- General-purpose offline RL with any regression oracle?
- Can randomized value iteration obtain a horizon-free rate?
- Extension to the POMDP?
- Model-free posterior sampling for offline RL with optimal frequentist rates?

Thank you