Offline Neural Contextual Bandits: Pessimism, Optimization and Generalization

ICLR'22

Thanh Nguyen-Tang[†], Sunil Gupta[†], A.Tuan Nguyen[‡], Svetha Venkatesh[†]

March 13, 2022

[†]: Deakin University; [‡]: University of Oxford

Outline

- Background
  - Offline policy learning for contextual bandits
  - Deep neural networks
- Algorithm – NeuraLCB
  - Use a neural network to learn the reward
  - Use neural network's gradients for pessimistic exploitation
  - Lower confidence bound strategy
  - Stochastic gradient descent for optimization
  - Stream offline data for generalization and adaptive offline data
- Main theory
  - Neural tangent kernel matrix + effective dimension + single-policy concentration
  - $\tilde{\mathcal{O}}(\kappa\sqrt{\tilde{d}}n^{-1/2})$ regret, where κ is distributional shift measure, $\tilde{d}$ is effective dimension, n is the number of offline samples

Outline

- Background
  - Offline policy learning for contextual bandits
  - Deep neural networks
- Algorithm – NeuraLCB
  - Use a neural network to learn the reward
  - Use neural network's gradients for pessimistic exploitation
  - Lower confidence bound strategy
  - Stochastic gradient descent for optimization
  - Stream offline data for generalization and adaptive offline data
- Main theory
  - Neural tangent kernel matrix + effective dimension + single-policy concentration
  - $\tilde{\mathcal{O}}(\kappa\sqrt{\tilde{d}}n^{-1/2})$ regret, where κ is distributional shift measure, $\tilde{d}$ is effective dimension, n is the number of offline samples

Outline

- Background
  - Offline policy learning for contextual bandits
  - Deep neural networks
- Algorithm – NeuraLCB
  - Use a neural network to learn the reward
  - Use neural network's gradients for pessimistic exploitation
  - Lower confidence bound strategy
  - Stochastic gradient descent for optimization
  - Stream offline data for generalization and adaptive offline data
- Main theory
  - Neural tangent kernel matrix + effective dimension + single-policy concentration
  - $\tilde{\mathcal{O}}(\kappa\sqrt{\tilde{d}}n^{-1/2})$ regret, where κ is distributional shift measure, $\tilde{d}$ is effective dimension, n is the number of offline samples

Background – offline K-armed contextual bandits

- Online setting: At each round t,
  - Agent observes K d-dimensional contextual vectors $\{\boldsymbol{x}_{t,a} \in \mathbb{R}^d : a \in [K]\}$
  - Agent takes an action a_t and observe reward $r_t \sim P(\cdot|\boldsymbol{x}_{t,a_t})$
  - Value: $v^\pi(\boldsymbol{x}) = \mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{x}), r\sim P(\cdot|\boldsymbol{x}_a)}[r]$
  - Optimal value: $v^*(\boldsymbol{x}) = \max_\pi v^\pi(\boldsymbol{x})$
  - Optimal policy: $\pi^* = \arg\max_\pi v^\pi$
- Offline policy learning (OPL) setting
  - Offline data $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$: collected a priori by an unknown and possibly adaptive behaviour policy μ
  - Goal: Learn $\hat{\pi}$ from $\mathcal{D}_n$ with small sub-optimality:

$$\mathrm{SubOpt}(\underbrace{\hat{\pi}}_{\text{data-dependent}}) := \mathbb{E}_{\boldsymbol{x}\sim\rho}[\mathrm{SubOpt}(\hat{\pi}; \boldsymbol{x})]$$

$$= \underbrace{\mathbb{E}_{\boldsymbol{x}\sim\rho}}_{\text{generalize to all contexts}} [v^*(\boldsymbol{x}) - v^{\hat{\pi}}(\boldsymbol{x})].$$

Background – offline K-armed contextual bandits

- Online setting: At each round t,
  - Agent observes K d-dimensional contextual vectors $\{\boldsymbol{x}_{t,a} \in \mathbb{R}^d : a \in [K]\}$
  - Agent takes an action a_t and observe reward $r_t \sim P(\cdot|\boldsymbol{x}_{t,a_t})$
  - Value: $v^\pi(\boldsymbol{x}) = \mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{x}), r\sim P(\cdot|\boldsymbol{x}_a)}[r]$
  - Optimal value: $v^*(\boldsymbol{x}) = \max_\pi v^\pi(\boldsymbol{x})$
  - Optimal policy: $\pi^* = \arg\max_\pi v^\pi$
- Offline policy learning (OPL) setting
  - Offline data $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$: collected a priori by an <u>unknown</u> and <u>possibly adaptive</u> behaviour policy μ
  - Goal: Learn $\hat{\pi}$ from $\mathcal{D}_n$ with small sub-optimality:

$$\begin{aligned}\mathrm{SubOpt}(\underbrace{\hat{\pi}}_{\text{data-dependent}}) &:= \mathbb{E}_{\boldsymbol{x}\sim\rho}[\mathrm{SubOpt}(\hat{\pi};\boldsymbol{x})] \\ &= \underbrace{\mathbb{E}_{\boldsymbol{x}\sim\rho}}_{\text{generalize to all contexts}}[v^*(\boldsymbol{x}) - v^{\hat{\pi}}(\boldsymbol{x})].\end{aligned}$$

Background – General reward function

Reward generation

$$r_t = \boxed{h(\mathbf{x}_{t,a_t})} + \xi_t, h(\cdot) \in [0, 1],$$

$$\xi_t \sim R\text{-subgaussian}|\{(\mathbf{x}_\tau, a_\tau, r_\tau)\}_{1 \le \tau \le t-1}, \mathbf{x}_t, a_t$$

Including many contextual bandit problems:

- Linear contextual bandit: $h(\mathbf{x}) = \langle \mathbf{x}, \boldsymbol{\theta} \rangle, \|\mathbf{x}\| \le 1, \|\boldsymbol{\theta}\| \le 1$
- Generalized linear bandit: $h(\mathbf{x}) = g(\langle \mathbf{x}, \boldsymbol{\theta} \rangle), \|\mathbf{x}\| \le 1, \|\boldsymbol{\theta}\| \le 1, \|\nabla g\| \le 1$
- Kernelized bandit: $h(\mathbf{x})$ in a norm-bounded RKHS

h is unknown and can be non-linear

Use a universal function approximator, e.g. neural networks

Background – General reward function

Reward generation

$$r_t = \boxed{h(\boldsymbol{x}_{t,a_t})} + \xi_t, h(\cdot) \in [0,1],$$
$$\xi_t \sim R\text{-subgaussian}|\{(\boldsymbol{x}_\tau, a_\tau, r_\tau)\}_{1\le\tau\le t-1}, \boldsymbol{x}_t, a_t$$

Including many contextual bandit problems:

- Linear contextual bandit: $h(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle, \|\boldsymbol{x}\| \le 1, \|\boldsymbol{\theta}\| \le 1$
- Generalized linear bandit: $h(\boldsymbol{x}) = g(\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle), \|\boldsymbol{x}\| \le 1, \|\boldsymbol{\theta}\| \le 1, \|\nabla g\| \le 1$
- Kernelized bandit: $h(\boldsymbol{x})$ in a norm-bounded RKHS

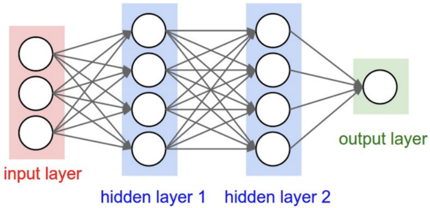
h is unknown and can be non-linear

Use a universal function approximator, e.g. neural networks

Background – Deep neural networks

$$f_{\boldsymbol{W}}(\boldsymbol{x}) = \sqrt{m}\boldsymbol{W}_L\sigma\left(\boldsymbol{W}_{L-1}\sigma\left(\dots\sigma(\boldsymbol{W}_1\boldsymbol{x})\dots\right)\right), \forall \boldsymbol{u} \in \mathbb{R}^d,$$

$$\boldsymbol{W}_l^{(0)} \sim \mathcal{N}(\boldsymbol{0}, \Theta(1/m)\boldsymbol{I}) \text{ (initialization)}$$



- $\sigma(\cdot) = \max\{\cdot, 0\}$ is ReLU function
- $\boldsymbol{W}_1 \in \mathbb{R}^{m\times d}, \boldsymbol{W}_i \in \mathbb{R}^{m\times m}, \forall i \in [2, L-1], \boldsymbol{W}_L \in \mathbb{R}^{m\times 1}$
- $\boldsymbol{W} := (\boldsymbol{W}_1, \dots, \boldsymbol{W}_L)$, $\mathrm{vec}(\boldsymbol{W}) \in \mathbb{R}^p, p = md + m + m^2(L-2)$
- Gradient: $\nabla f_{\boldsymbol{W}} \in \mathbb{R}^p$

Question

- Neural network-based offline policy learning [Nguyen-Tang et al., 2021, Uehara et al., 2021]
- But they require
  - Strong uniform data coverage: $\frac{\pi(a|\boldsymbol{x})}{\mu(a|\boldsymbol{x})} \leq C < \infty, \forall \pi, \forall \boldsymbol{x}, a$
  - Intractable optimization oracle: $\hat{f} = \arg\min_{f \in \mathcal{F}} L(f)$
  - i.i.d. data: $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$ are independent
  - functional assumption on the reward function

Can we design a computationally efficient neural network-based OPL algorithm that can

- learn a general reward function,
- require a weaker data coverage assumption, and
- work on adaptive offline data?

Question

- Neural network-based offline policy learning [Nguyen-Tang et al., 2021, Uehara et al., 2021]
- But they require
  - Strong uniform data coverage: $\frac{\pi(a|\boldsymbol{x})}{\mu(a|\boldsymbol{x})} \leq C < \infty, \forall \pi, \forall \boldsymbol{x}, a$
  - Intractable optimization oracle: $\hat{f} = \arg\min_{f \in \mathcal{F}} L(f)$
  - i.i.d. data: $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$ are independent
  - functional assumption on the reward function

Can we design a computationally efficient neural network-based OPL algorithm that can

- learn a general reward function,
- require a weaker data coverage assumption, and
- work on adaptive offline data?

Question

Can we design a computationally efficient neural network-based OPL algorithm that can

- learn a general reward function,
- require a weaker data coverage assumption, and
- work on adaptive offline data?

Yes! NeuraLCB

- neural network to model reward function, LCB strategy for pessimistic exploitation
- Stochastic gradient descent for optimization
- Stream offline data to handle generalization and adaptive data

Provable learning: $\tilde{\mathcal{O}}(\kappa \cdot \sqrt{\tilde{d}} \cdot n^{-1/2})$ sub-optimality. Compared to online counterpart [Zhou et al., 2020],

- bound improved by a factor of $\sqrt{\tilde{d}}$
- computation: from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$

Question

Can we design a computationally efficient neural network-based OPL algorithm that can

- learn a general reward function,
- require a weaker data coverage assumption, and
- work on adaptive offline data?

Yes! NeuraLCB

- neural network to model reward function, LCB strategy for pessimistic exploitation
- Stochastic gradient descent for optimization
- Stream offline data to handle generalization and adaptive data

Provable learning: $\tilde{\mathcal{O}}(\kappa \cdot \sqrt{\tilde{d}} \cdot n^{-1/2})$ sub-optimality. Compared to online counterpart [Zhou et al., 2020],

- bound improved by a factor of $\sqrt{\tilde{d}}$
- computation: from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$

NeuraLCB – Lower confidence bounds

Stream the data $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$ sequentially one by one. At each step t,

- Retrieve $(\boldsymbol{x}_t, a_t, r_t)$ from $\mathcal{D}_n$
- Compute LCB

$$L_t(\cdot) := \underbrace{f_{W^{(t-1)}}(\cdot)}_{mean} - \beta_{t-1} \underbrace{\|\nabla f_{W^{(t-1)}}(\cdot) \cdot m^{-1/2}\|_{\Lambda_{t-1}^{-1}}}_{variance}$$

- Extract policy $\hat{\pi}_t(\boldsymbol{x}) \leftarrow \arg\max_{a \in [K]} L_t(\boldsymbol{x}_a)$, where $\boldsymbol{x} = \{\boldsymbol{x}_a \in \mathbb{R}^d : a \in [K]\}$

Output: Uniformly sample $\hat{\pi}$ from $\{\hat{\pi}_1, \ldots, \hat{\pi}_n\}$

NeuraLCB – Lower confidence bounds

Stream the data $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$ sequentially one by one. At each step t,

- Retrieve $(\boldsymbol{x}_t, a_t, r_t)$ from $\mathcal{D}_n$
- Compute LCB

$$L_t(\cdot) := \underbrace{f_{\boldsymbol{W}^{(t-1)}}(\cdot)}_{mean} - \beta_{t-1} \underbrace{\|\nabla f_{\boldsymbol{W}^{(t-1)}}(\cdot) \cdot m^{-1/2}\|_{\boldsymbol{\Lambda}_{t-1}^{-1}}}_{variance}$$

- Extract policy $\hat{\pi}_t(\boldsymbol{x}) \leftarrow \arg\max_{a \in [K]} L_t(\boldsymbol{x}_a)$, where $\boldsymbol{x} = \{\boldsymbol{x}_a \in \mathbb{R}^d : a \in [K]\}$

Output: Uniformly sample $\hat{\pi}$ from $\{\hat{\pi}_1, \ldots, \hat{\pi}_n\}$

NeuraLCB – Lower confidence bounds

Stream the data $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$ sequentially one by one. At each step t,

- Retrieve $(\boldsymbol{x}_t, a_t, r_t)$ from $\mathcal{D}_n$
- Compute LCB

$$L_t(\cdot) := \underbrace{f_{\boldsymbol{W}^{(t-1)}}(\cdot)}_{\textit{mean}} - \beta_{t-1} \underbrace{\|\nabla f_{\boldsymbol{W}^{(t-1)}}(\cdot) \cdot m^{-1/2}\|_{\boldsymbol{\Lambda}_{t-1}^{-1}}}_{\textit{variance}}$$

- Extract policy $\hat{\pi}_t(\boldsymbol{x}) \leftarrow \arg\max_{a \in [K]} L_t(\boldsymbol{x}_a)$, where $\boldsymbol{x} = \{\boldsymbol{x}_a \in \mathbb{R}^d : a \in [K]\}$

Output: Uniformly sample $\hat{\pi}$ from $\{\hat{\pi}_1, \ldots, \hat{\pi}_n\}$

NeuraLCB – Lower confidence bounds

Stream the data $\mathcal{D}_n = \{(\boldsymbol{x}_t, a_t, r_t)\}_{t=1}^n$ sequentially one by one. At each step t,

- Retrieve $(\boldsymbol{x}_t, a_t, r_t)$ from $\mathcal{D}_n$
- Compute LCB

$$L_t(\cdot) := \underbrace{f_{\boldsymbol{W}^{(t-1)}}(\cdot)}_{mean} - \beta_{t-1} \underbrace{\|\nabla f_{\boldsymbol{W}^{(t-1)}}(\cdot) \cdot m^{-1/2}\|_{\boldsymbol{\Lambda}_{t-1}^{-1}}}_{variance}$$

- Extract policy $\hat{\pi}_t(\boldsymbol{x}) \leftarrow \arg\max_{a \in [K]} L_t(\boldsymbol{x}_a)$, where $\boldsymbol{x} = \{\boldsymbol{x}_a \in \mathbb{R}^d : a \in [K]\}$

Output: Uniformly sample $\hat{\pi}$ from $\{\hat{\pi}_1, \ldots, \hat{\pi}_n\}$

NeuraLCB – Update parameters

- Update the empirical covariance matrix $\boldsymbol{\Lambda}_t$, where $\boldsymbol{\Lambda}_0 = \lambda \boldsymbol{I}$:

$$\boldsymbol{\Lambda}_t \leftarrow \boldsymbol{\Lambda}_{t-1} + \underbrace{\mathrm{vec}(\nabla f_{\boldsymbol{W}^{(t-1)}}(\boldsymbol{x}_{t,a_t})) \cdot \mathrm{vec}(\nabla f_{\boldsymbol{W}^{(t-1)}}(\boldsymbol{x}_{t,a_t}))^T / m}_{\text{dynamical as } \boldsymbol{W}^{(t-1)} \text{ changes with } t}$$

- Update $\boldsymbol{W}^{(t)}$ using stochastic gradient descent:

$$\boldsymbol{W}^{(t)} \leftarrow \underbrace{\boldsymbol{W}^{(t-1)} - \eta_t \nabla \mathcal{L}_t(\boldsymbol{W}^{(t-1)})}_{\text{SGD}}$$

$$\text{where } \mathcal{L}_t(\boldsymbol{W}) = \underbrace{\frac{1}{2}(f_{\boldsymbol{W}}(\boldsymbol{x}_{t,a_t}) - r_t)^2 + \frac{m\lambda}{2}\|\boldsymbol{W} - \boldsymbol{W}^{(0)}\|_F^2}_{\text{ridge regression}}.$$

Compared to parameter update in NeuralUCB [Zhou et al., 2020],

- NeuralUCB: train a new net from scratch at each t, $\mathcal{O}(n^2)$ update steps
- NeuraLCB: re-uses the trained parameters from the prev. iter., $\mathcal{O}(n)$ update steps

NeuraLCB – Update parameters

- Update the empirical covariance matrix $\mathbf{\Lambda}_t$, where $\mathbf{\Lambda}_0 = \lambda \boldsymbol{I}$:

$$\mathbf{\Lambda}_t \leftarrow \mathbf{\Lambda}_{t-1} + \underbrace{\mathrm{vec}(\nabla f_{\boldsymbol{W}^{(t-1)}}(\boldsymbol{x}_{t,a_t})) \cdot \mathrm{vec}(\nabla f_{\boldsymbol{W}^{(t-1)}}(\boldsymbol{x}_{t,a_t}))^T / m}_{\text{dynamical as } \boldsymbol{W}^{(t-1)} \text{ changes with } t}$$

- Update $\boldsymbol{W}^{(t)}$ using stochastic gradient descent:

$$\boldsymbol{W}^{(t)} \leftarrow \underbrace{\boldsymbol{W}^{(t-1)} - \eta_t \nabla \mathcal{L}_t(\boldsymbol{W}^{(t-1)})}_{\text{SGD}}$$

$$\text{where } \mathcal{L}_t(\boldsymbol{W}) = \underbrace{\frac{1}{2}(f_{\boldsymbol{W}}(\boldsymbol{x}_{t,a_t}) - r_t)^2 + \frac{m\lambda}{2}\|\boldsymbol{W} - \boldsymbol{W}^{(0)}\|_F^2}_{\text{ridge regression}}.$$

Compared to parameter update in NeuralUCB [Zhou et al., 2020],

- NeuralUCB: train a new net from scratch at each t, $\mathcal{O}(n^2)$ update steps
- NeuraLCB: re-uses the trained parameters from the prev. iter., $\mathcal{O}(n)$ update steps

NeuraLCB – Confidence radius

- Under overparameterized setting ($m \gg 1$):

$\|\boldsymbol{W}^* - \boldsymbol{W}^{(t)}\|_{\boldsymbol{\Lambda}_t} \leq \beta_t$ where

$$\beta_t := (\underbrace{\sqrt{2}m^{-1/2}S}_{\|\boldsymbol{W}^* - \boldsymbol{W}^{(0)}\|_F} + \underbrace{t^{1/2}\lambda^{-1/2}m^{-1/2}}_{\|\boldsymbol{W}^{(t)} - \boldsymbol{W}^{(0)}\|_F \text{ via SGD}})\underbrace{\sqrt{\lambda + C_3^2 tL}}_{\sqrt{\|\boldsymbol{\Lambda}_t\|}}$$

$\boldsymbol{W}^*$: network params that interpolate h in the training contexts

Compared to NeuralUCB [Zhou et al., 2020],

$$\beta_t = \mathcal{O}(m^{-1/12}t^{7/12}L^2\lambda^{-7/12}(\sqrt{\lambda}S + \nu\sqrt{\tilde{d}\log(1 + tK/\lambda)})) + \mathcal{O}(m^{-1/6}t^{19/6}L^{9/2}\lambda^{-13/6})$$

Our confidence radius does not depend on $\tilde{d}$

- much simpler and tighter
- Key: Don't regress toward the minimizer of the least squared problem in the linear case.

NeuraLCB – Confidence radius

- Under overparameterized setting ($m \gg 1$):

$\|\boldsymbol{W}^* - \boldsymbol{W}^{(t)}\|_{\boldsymbol{\Lambda}_t} \leq \beta_t$ where

$$\beta_t := (\underbrace{\sqrt{2}m^{-1/2}S}_{\|\boldsymbol{W}^* - \boldsymbol{W}^{(0)}\|_F} + \underbrace{t^{1/2}\lambda^{-1/2}m^{-1/2}}_{\|\boldsymbol{W}^{(t)} - \boldsymbol{W}^{(0)}\|_F \text{ via SGD}})\underbrace{\sqrt{\lambda + C_3^2 tL}}_{\sqrt{\|\boldsymbol{\Lambda}_t\|}}$$

$\boldsymbol{W}^*$: network params that interpolate h in the training contexts

Compared to NeuralUCB [Zhou et al., 2020],

$$\beta_t = \mathcal{O}(m^{-1/12}t^{7/12}L^2\lambda^{-7/12}(\sqrt{\lambda}S + \nu\sqrt{\tilde{d}\log(1 + tK/\lambda)}))$$
$$+ \mathcal{O}(m^{-1/6}t^{19/6}L^{9/2}\lambda^{-13/6})$$

Our confidence radius does not depend on $\tilde{d}$

- much simpler and tighter
- Key: Don't regress toward the minimizer of the least squared problem in the linear case.

Main theorem – Assumptions

Assumption

$\exists \lambda_0 > 0$ such that $\boldsymbol{H} \succeq \lambda_0 \boldsymbol{I}$ where $\boldsymbol{H}$ is the neural tangent kernel matrix [Arora et al., 2019, Du et al., 2019b,a, Cao and Gu, 2019] on training contexts $\{\boldsymbol{x}^{(i)}\}_{i \in [nK]}$

- Satisfied if no two contexts in $\{\boldsymbol{x}^{(i)}\}_{i \in [nK]}$ are parallel.
- $\lambda_0 \geq \Omega(d)$ under mild input condition [Nguyen et al., 2021]

Main theorem – Assumptions

Assumption

- $\boldsymbol{x}_t$ is independent of $\mathcal{D}_{t-1} = \{(\boldsymbol{x}_\tau, a_\tau, r_\tau)\}_{\tau \in [t-1]}$,
- $\exists \kappa \in (0, \infty), \left\| \frac{\pi^*(\cdot|\boldsymbol{x}_t)}{\mu(\cdot|\mathcal{D}_{t-1}, \boldsymbol{x}_t)} \right\|_\infty \leq \kappa, \forall t \in [n]$.

- the first part is minimal,
  - e.g. when $\{\boldsymbol{x}_t\}_{t=1}^n \overset{i.i.d.}{\sim} \rho$.
  - a_t can still depend on $\mathcal{D}_{t-1}$ and $\boldsymbol{x}_t$
- the second part only requires that μ has sufficient coverage over only π^* only in the observed contexts
  - weaker than any other existing data coverage assumptions for OPL

Main theorem – Assumptions

Assumption

- $\boldsymbol{x}_t$ is independent of $\mathcal{D}_{t-1} = \{(\boldsymbol{x}_\tau, a_\tau, r_\tau)\}_{\tau \in [t-1]}$,
- $\exists \kappa \in (0, \infty), \left\| \frac{\pi^*(\cdot|\boldsymbol{x}_t)}{\mu(\cdot|\mathcal{D}_{t-1}, \boldsymbol{x}_t)} \right\|_\infty \leq \kappa, \forall t \in [n]$.

- the first part is minimal,
  - e.g. when $\{\boldsymbol{x}_t\}_{t=1}^n \overset{i.i.d.}{\sim} \rho$.
  - a_t can still depend on $\mathcal{D}_{t-1}$ and $\boldsymbol{x}_t$
- the second part only requires that μ has sufficient coverage over only π^* only in the observed contexts
  - weaker than any other existing data coverage assumptions for OPL

Main theorem – Definition

Definition

Effective dimension $\tilde{d} = \log\det(\boldsymbol{I} + \boldsymbol{H}/\lambda)/\log(1 + nK/\lambda)$ [Zhou et al., 2020, Valko et al., 2013, Yang and Wang, 2020, Yang et al., 2020]

- $\tilde{d}$ measures how quickly the eigenvalues of $\boldsymbol{H}$ decays
- $\tilde{d} = \mathcal{O}(\log n)$ in some typical cases

Main theorem – sub-optimality bound

Theorem

Set learning rates $\eta_t = \frac{\iota}{\sqrt{t}}$ where

$$\iota^{-1} = \Omega(n^{2/3} m^{5/6} \lambda^{-1/6} L^{17/6} \log^{1/2} m) \vee \Omega(m \lambda^{1/2} \log^{1/2}(nKL^2(10n)/\delta)),$$

under overparameterization, w.p. at least $1 - \delta$,

$$\mathbb{E}[\mathrm{SubOpt}(\hat{\pi})] = \tilde{\mathcal{O}}(\kappa \cdot \max\{\sqrt{\tilde{d}}, 1\} \cdot n^{-1/2})$$

- the bound does not depend on p
- Compared to NeuralUCB [Zhou et al., 2020]: $\sqrt{\tilde{d}}$-improved
  - NeuralUCB: $\mathcal{O}(\tilde{d} n^{-1/2})$ regret
  - Minimax lower bound regret [Chu et al., 2011]: $\Omega(\sqrt{d} n^{-1/2})$

Main theorem – sub-optimality bound

Theorem

Set learning rates $\eta_t = \frac{\iota}{\sqrt{t}}$ where

$$\iota^{-1} = \Omega(n^{2/3} m^{5/6} \lambda^{-1/6} L^{17/6} \log^{1/2} m) \vee \Omega(m \lambda^{1/2} \log^{1/2}(nKL^2(10n)/\delta)),$$

under overparameterization, w.p. at least $1 - \delta$,

$$\mathbb{E}[\text{SubOpt}(\hat{\pi})] = \tilde{\mathcal{O}}(\kappa \cdot \max\{\sqrt{\tilde{d}}, 1\} \cdot n^{-1/2})$$

- the bound does not depend on p
- Compared to NeuralUCB [Zhou et al., 2020]: $\sqrt{\tilde{d}}$-improved
    - NeuralUCB: $\mathcal{O}(\tilde{d} n^{-1/2})$ regret
    - Minimax lower bound regret [Chu et al., 2011]: $\Omega(\sqrt{d} n^{-1/2})$

Main theorem – Comparison

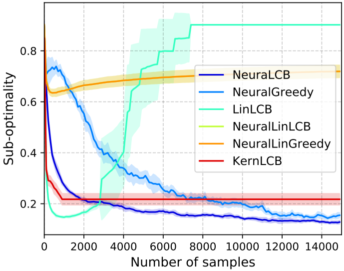
Table 1: The SOTA generalization theory of OPL with function approximation. Here the distributional shift measure κ can be defined differently in different works.

Work	Function	Type	Optimization	Sub-optimality	Data Coverage	Data Gen.
Yin & Wang (2020)[a]	Tabular	Greedy	Analytical	$\tilde{O}\left(\sqrt{\lvert\mathcal{X}\rvert \cdot K} \cdot n^{-1/2}\right)$	Uniform	I
Rashidinejad et al. (2021)	Tabular	Pessimism	Analytical	$\tilde{O}\left(\sqrt{\lvert\mathcal{X}\rvert \cdot \kappa} \cdot n^{-1/2}\right)$	SPC	I
Duan & Wang (2020)[b]	Linear	Greedy	Analytical	$\tilde{O}\left(\kappa \cdot n^{-1/2} + d \cdot n^{-1}\right)$	Uniform	I
Jin et al. (2020)	Linear	Pessimism	Analytical	$\tilde{O}\left(d \cdot n^{-1/2}\right)$	Uniform	I
Nguyen-Tang et al. (2021)	Narrow ReLU	Greedy	Oracle	$\tilde{O}\left(\sqrt{\kappa} \cdot n^{-\frac{\alpha}{2(\alpha+d)}}\right)$	Uniform	I
This work	Wide ReLU	Pessimism	SGD	$\tilde{O}(\kappa \cdot \sqrt{\tilde{d}} \cdot n^{-1/2})$	eSPC	I/D

[a,b] The bounds of these works are for off-policy evaluation which is generally easier than OPL problem.

Takehome messages

- NeuraLCB uses a neural net to learn the reward and LCB strategy for pessimistic exploitation
- Offline data can be adaptive and only needs to cover the data of the optimal policy in the training contexts
- NeuraLCB achieves $\tilde{\mathcal{O}}(\kappa \cdot \max\{\sqrt{\tilde{d}}, 1\} \cdot n^{-1/2})$, subliner rate
- More statistically efficient than NeuralUCB by a factor of $\sqrt{\tilde{d}}$ and more computationally efficient (from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$)
- NeuraLCB performs well empirically



Thank you!

References I

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. *Advances in Neural Information Processing Systems*, 32:10836–10846, 2019.

Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.

Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 1675–1685. PMLR, 2019a.

References II

Simon S. Du, Xiyu Zhai, Barnabás Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR (Poster)*. OpenReview.net, 2019b.

Quynh Nguyen, Marco Mondelli, and Guido F Montufar. Tight bounds on the smallest eigenvalue of the neural tangent kernel for deep relu networks. In *International Conference on Machine Learning*, pages 8119–8129. PMLR, 2021.

Thanh Nguyen-Tang, Sunil Gupta, Hung Tran-The, and Svetha Venkatesh. Sample complexity of offline reinforcement learning with deep relu networks, 2021.

Masatoshi Uehara, Masaaki Imaizumi, Nan Jiang, Nathan Kallus, Wen Sun, and Tengyang Xie. Finite sample analysis of minimax offline reinforcement learning: Completeness, fast rates and first-order efficiency. *arXiv preprint arXiv:2102.02981*, 2021.

References III

Michal Valko, Nathaniel Korda, Rémi Munos, Ilias Flaounas, and Nelo Cristianini. Finite-time analysis of kernelised contextual bandits. *arXiv preprint arXiv:1309.6869*, 2013.

Lin Yang and Mengdi Wang. Reinforcement learning in feature space: Matrix bandit, kernels, and regret bound. In *International Conference on Machine Learning*, pages 10746–10756. PMLR, 2020.

Zhuoran Yang, Chi Jin, Zhaoran Wang, Mengdi Wang, and Michael I Jordan. On function approximation in reinforcement learning: Optimism in the face of large state spaces. *arXiv preprint arXiv:2011.04622*, 2020.

Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*, pages 11492–11502. PMLR, 2020.