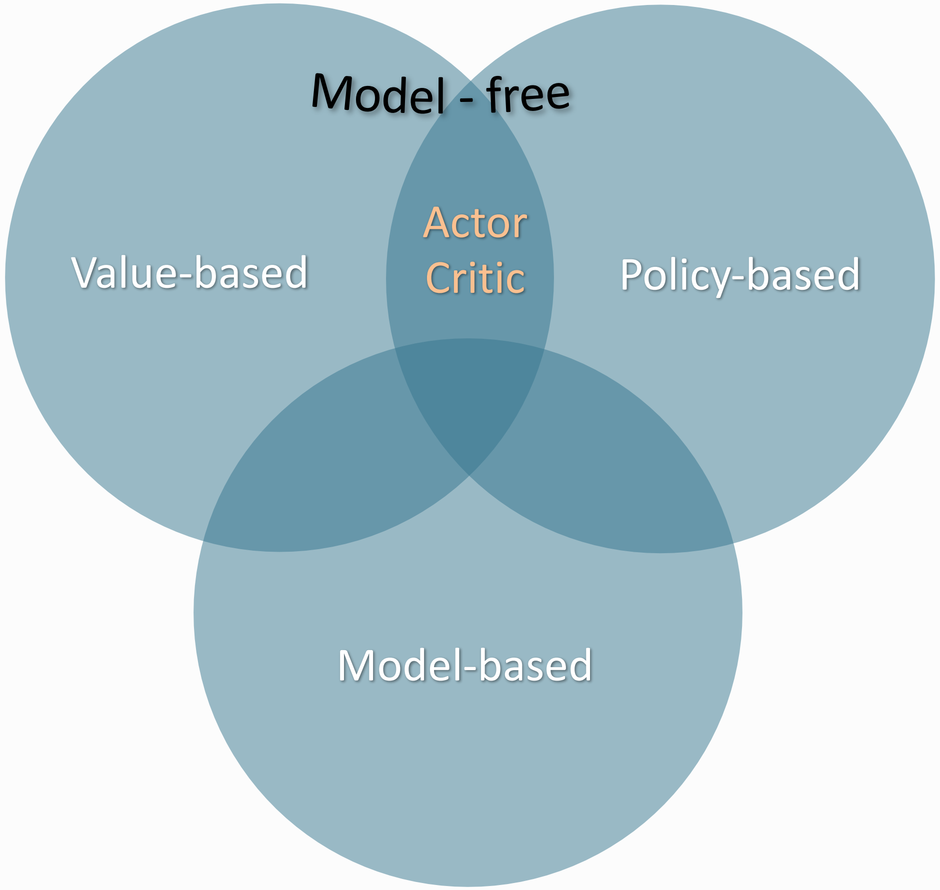
EPFL Summer School on Data Science, Optimization and Operations Research
August 15-20, 2021

Lecture 4: RL from Deep Learning Perspectives

Niao He, D-INFK, ETH Zurich

Recap: Reinforcement Learning Approaches



- **Value-based RL**
  - Estimate the optimal value function $Q^*(s, a)$
  - Example: Q-learning
- **Policy-based RL**
  - Search directly the optimal policy $\pi^*(\cdot\,|s)$
  - Example: Policy Gradient Method
- **Model-based RL**
  - First estimate the model P, R and then do planning

Outline of Lecture Series

Lecture 1	Introduction to RL
Lecture 2	RL from Control Perspectives - Value-based RL
Lecture 3	RL from Optimization Perspectives *- Policy-based RL*
Lecture 4	RL from Deep Learning Perspectives *- Deep RL*
Lecture 5	RL from Game Perspectives

Focus:

Provably convergent "deep" RL methods

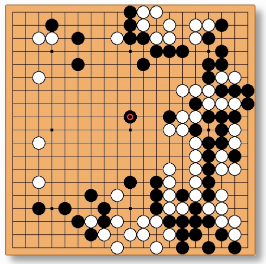
- RL with nonlinear function approximation
- RL with neural network approximation

The Grand Challenge

- **Large** state space, policy space

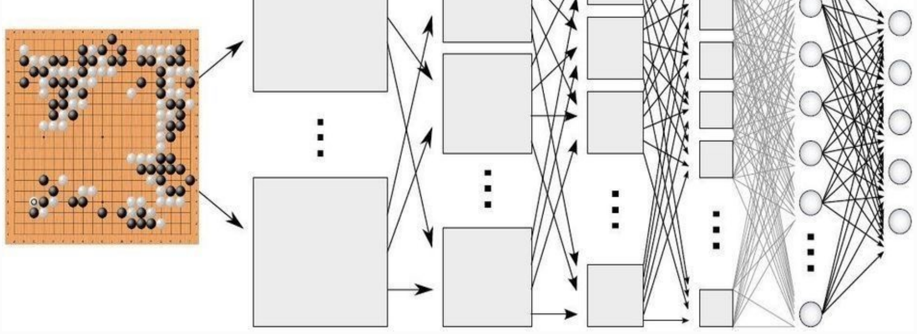
Chess: 10^{120}

Go: 3^{361}



State s

Action a



$V_\theta(s)$

$Q_\theta(s,a)$

$\pi_\theta(a|s)$

$P_\theta(s,s',a)$

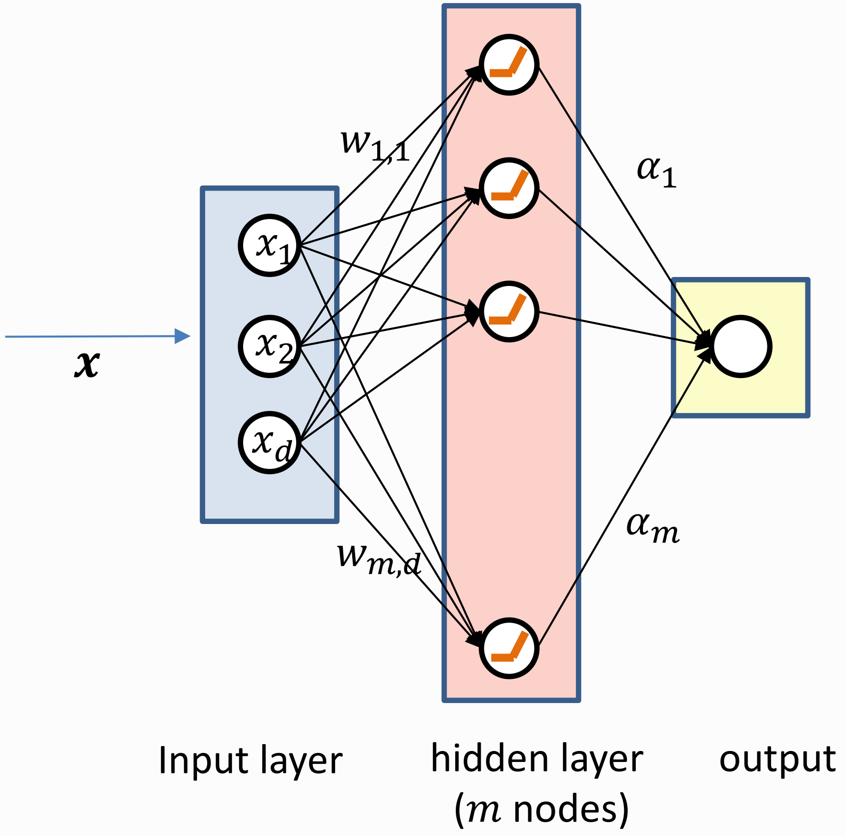
Learn parameter
$\theta \in R^d$
$(d \ll |S||A|)$

Using neural network approximation seems a must.
AI = RL + DL?

Neural Networks

- Nested composition of (learnable) linear transformation with (fixed) nonlinear activation functions

A single-hidden-layer neural network $f(x; w, \alpha) = \sum_{i=1}^{m} \alpha_i \sigma(w_i^T x)$



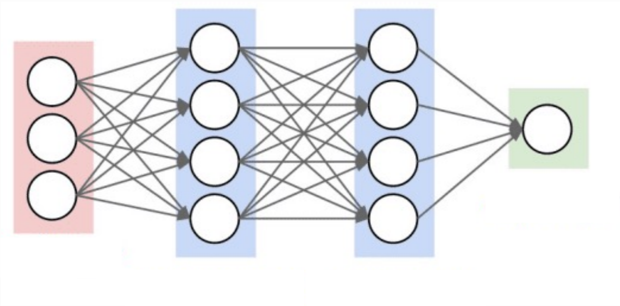
Activation function $\sigma(\cdot)$:

- Identity: $\sigma(u) = u$
- Sigmoid: $\sigma(u) = \frac{1}{1+\exp(-u)}$
- Tanh: $\sigma(u) = \tanh(u)$
- Rectified linear unit: $\sigma(u) = \max(0, u)$

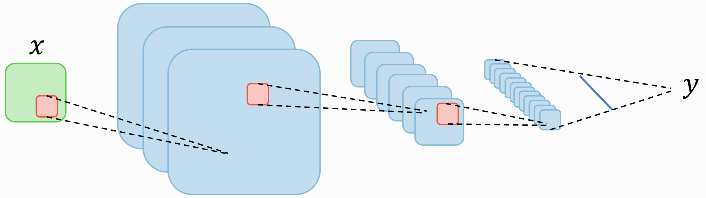
Deep Neural Networks

- More hidden layers, different activation functions, more general graph structure ....

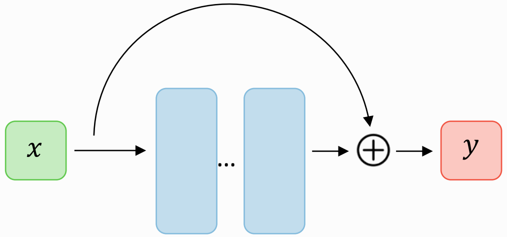
Feed forward network



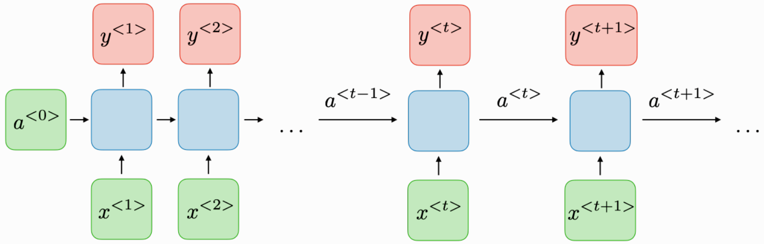
Convolutional network



Residual network



Recurrent network



Representation Power - why neural networks?

Shallow networks are universal approximators

- Any continuous function on bounded domain can be approximated arbitrarily well by a one-hidden layer network with nonconstant and increasing continuous activation function.
  [Cybenko, 1989; Hornik et al.,1989; Barron, 1993]

- Number of neurons can be large.

Benefits of depth

- A deep network cannot be approximated by a reasonably-sized shallow network.

- There exists ReLU networks with $poly(d)$ nodes in 2 hidden layers which cannot be approximated by 1-hidden-layer networks with less than 2^d nodes.
  [Eldan and Shamir, 2015]

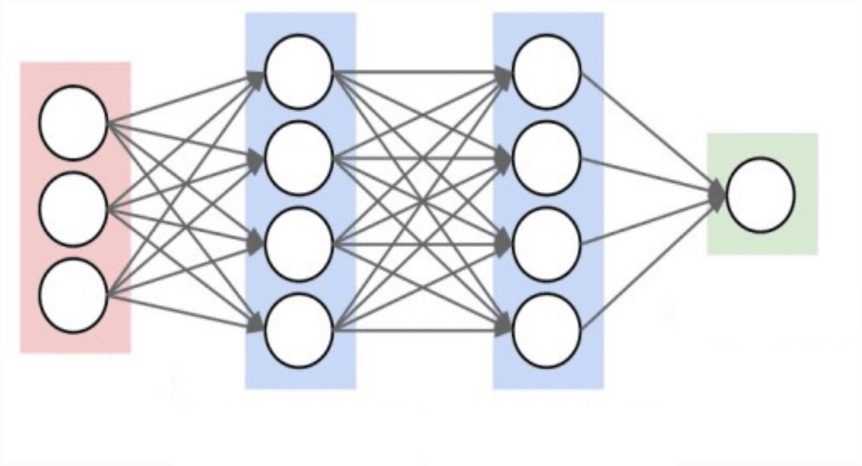
- There exists a function with $O(L^2)$ layers and width 2 which requires width $O(2^L)$ to approximate with $O(L)$ layers. [Telgarsky 2015,2016]

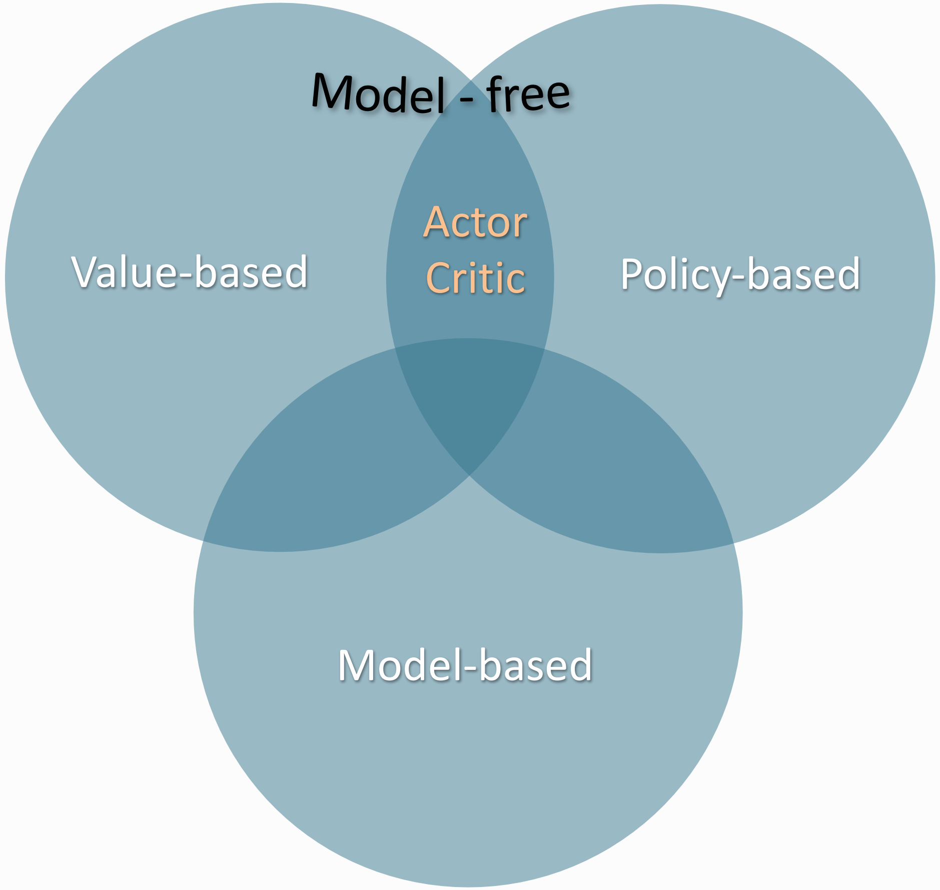
Training with Neural Networks

- **Gradient vanishing or exploding**
  - ReLU activation, gradient clipping
- **Overfitting**
  - regularization techniques (dropout, early stopping, etc.)
- **Nonconvexity**
  - noisy gradient
- **Ill-conditioning**
  - adaptive gradient methods (Adam, AdaGrad, RMSprop, etc.)

Deep Reinforcement Learning

- Using (deep) neural networks to represent
  - Value function
  - Policy
  - Model



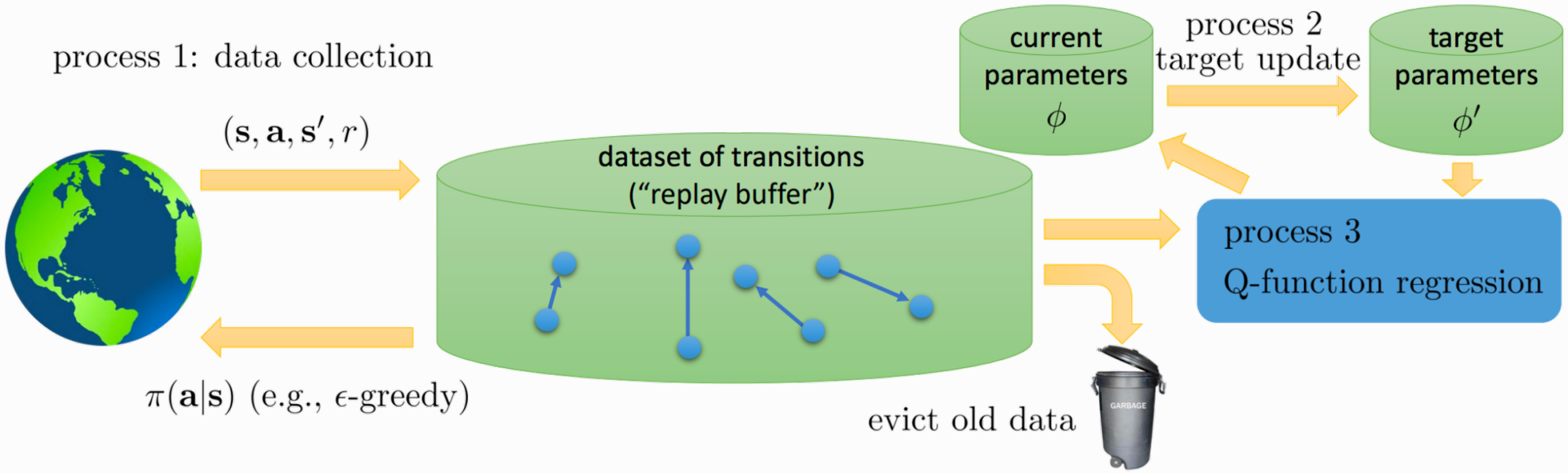


Deep Value-based and Actor-critic RL

- Deep Q-Learning

$$Q^*(s,a) \approx Q(s,a;w)$$

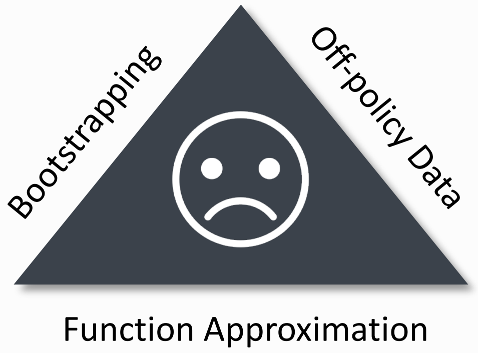
$$\min_{w} \; L(w) := \mathbb{E}_{(s,a,s',r)\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s',a';w^-) - Q(s,a;w)\right)^2\right]$$



- Stabilizing training: (prioritized) experience replay, target network, double learning, dueling network.

Figure source: EE-618 at EPFL

The Deadly Triad?



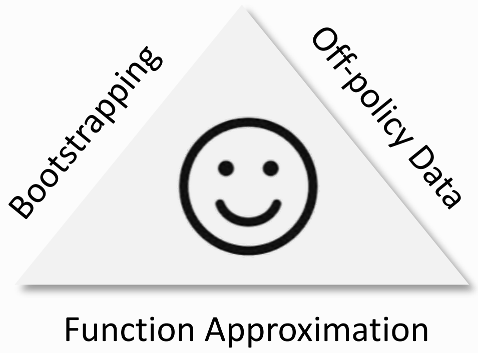
Network capacity

Target networks

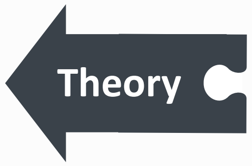
Overestimation

Multi-step returns

Prioritization

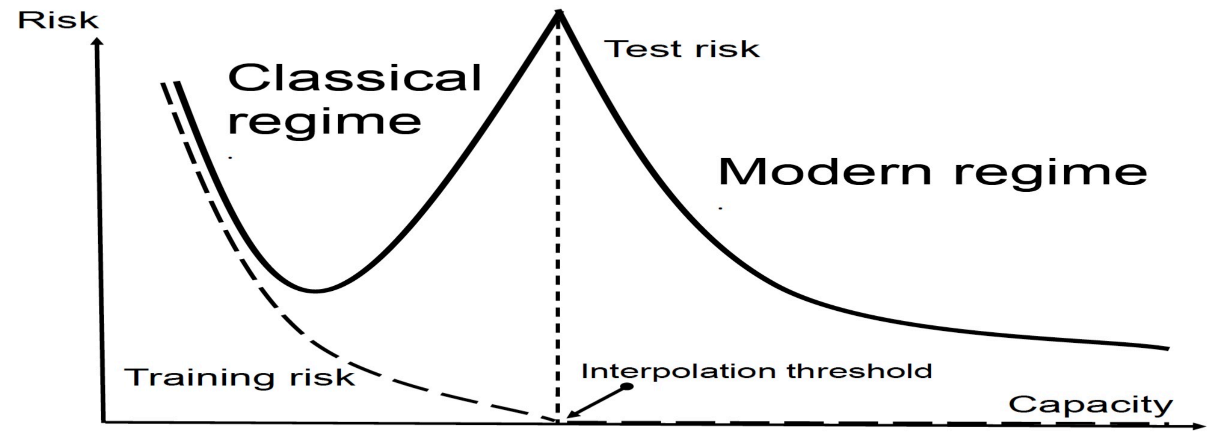


Q-learning with function approximation could diverge.
[Barto & Sutton, 2018]



DQN successfully learnt to play many Atari 2600 games.
[van Hasselt et al, 2018]

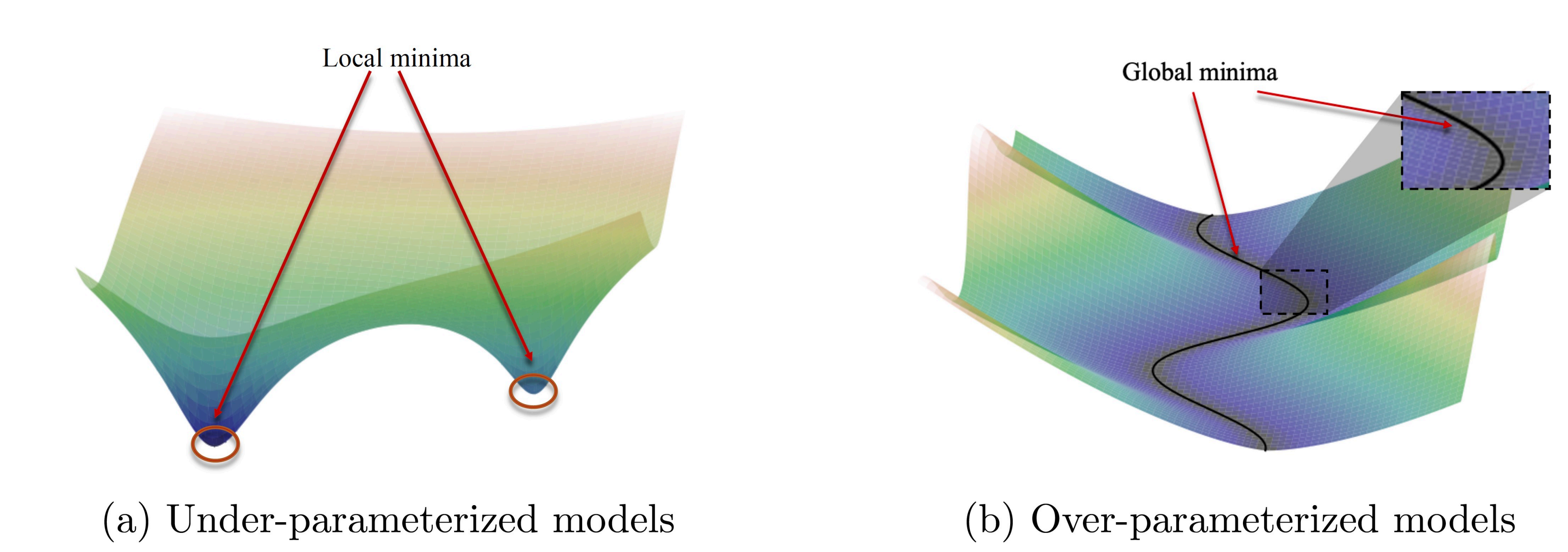
Wisdom from modern deep learning theory



	Classical (under-parameterized)	Modern (over-parameterized)
Generalization curve	U-shaped	Descending
Optimal model	Bottom of U (hard to find)	Any large model (easy to find)
Optimization landscape:	Locally convex Minimizers locally unique	Not locally convex Manifolds of minimizers Satisfies PL* condition
GD/SGD convergence	GD converges to local min SGD w. fixed learning rate does not converge	GD/SGD converge to global min SGD w. fixed learning rate converges exponentially
Adversarial examples	?	Unavoidable
Transition to linearity		Wide networks w. linear last layer

Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of Interpolation, 2021.

The optimization landscape



Extensive work:

Jacot et al. 2018; Li and Liang 2018; Du et al. 2018; Allen-Zhu et al. 2018; Oymak and Soltanolkotabi 2019; Zou et al. 2018; Chizat and Bach 2019; Ji and Telgarsky 2019a; Z. Chen et al. 2019; Arora et al. 2019; Cao and Gu 2020; ……

Liu et al. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks, 2021.

The neural tangent kernel (NTK)

- Supervised learning: find a model parameter that fits the training data

$$f(x_i; w^*) \approx y_i, \quad i = 1, \dots, n$$

$$\min_{w \in \mathbb{R}^m} \; L(w) := \frac{1}{2} \sum_{i=1}^{n} (f(x_i; w) - y_i)^2$$

- Neural tangent kernel:

$$K_{ij}(w) = \langle \nabla_w f(x_i; w), \nabla_w f(x_j; w) \rangle$$

$$K_{ij}(w_0) = \mathbb{E}_{w_0 \sim N(0, I_m)} \langle \nabla_w f(x_i; w_0), \nabla_w f(x_j; w_0) \rangle$$

- Kernel matrix: $K(w_0) \succcurlyeq 0$ if m is sufficiently large

Key insight behind the scene

- Gradient flow:

$$\frac{dw(t)}{dt} = -\nabla L(w(t))$$

$$\mathbf{u}(t) = f(w(t); \mathbf{x}) - \mathbf{y}$$

$$\frac{d\mathbf{u}(t)}{dt} = -K(w(t))\mathbf{u}(t)$$

- PL* condition

$$\|\nabla L(w)\|^2 = (f(w; \mathbf{x}) - \mathbf{y})^T K(w)(f(w; \mathbf{x}) - \mathbf{y}) \geq 2 \cdot \lambda_{min}(K(w)) \cdot L(w)$$

$K(w_0) \succcurlyeq \mu_0$, for random w_0 and large $m = poly(n)$

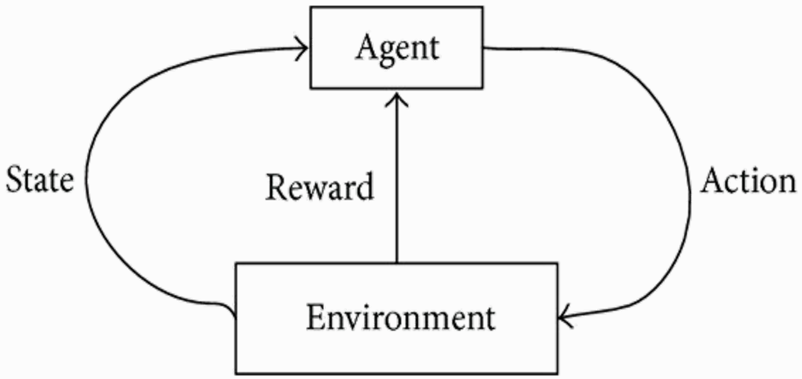
$\lambda_{min}(K(w)) - \lambda_{min}(K(w_0)) = O\left(\frac{1}{\sqrt{m}}\right)$, for $w \in B$

$K(w(t)) \succ \mu$

GD/SGD converges to global optima

Supervised Learning vs. RL

- **Common features:** learning from experience and generalize
  - SL: given $(x_i, y_i)_{i=1,\ldots,n}$, learn best f in hypothesis class
  - RL: given $(s_i, a_i, r_i)_{i=1,\ldots,n}$, learn best $Q(s,a)$ or $\pi^*(a|s) = \arg\min_a Q(s,a)$.
- **Distinguishing features of RL:**
  - Lack of supervisor, only a reward signal
  - Delayed feedback
  - Non-i.i.d. data
  - Difficulty with data reuse



Notation Recap

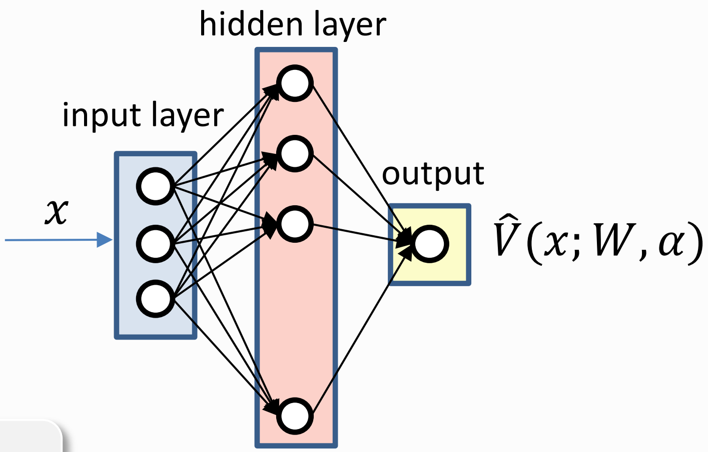
MDP $(S, \mathcal{A}, P, R, \mu, \gamma)$

State value function:	$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \vert s_0 = s\right]$
State-action value function:	$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \vert s_0 = s, a_0 = a\right]$
Optimal value function:	$V^*(s) := \max_\pi V^\pi(s), \quad Q^*(s,a) := \max_\pi Q^\pi(s,a)$
Optimal policy:	$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s,a)$
Bellman equation:	$V^\pi(s) = \mathbb{E}_{a \sim \pi(s)}\left[R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot \vert s,a)} V^\pi(s')\right]$
Bellman optimality:	$Q^*(s,a) = R(s,a) + \mathbb{E}_{s' \vert s,a}\left[\gamma \max_{a' \in \mathcal{A}} Q^*(s', a')\right]$
Policy gradient:	$\frac{\partial V^{\pi_\theta}(\mu)}{\partial \theta} = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_\mu^{\pi_\theta}, a \sim \pi_\theta(\cdot \vert s)}\left[Q^{\pi_\theta}(s,a) \nabla \log \pi_\theta(a \vert s)\right]$
State visitation distribution:	$d_\mu^\pi(s) = \mathbb{E}_{s_0 \sim \mu}\left[(1-\gamma) \sum_{k=0}^{\infty} \gamma^k P(s_k = s \vert s_0, \pi)\right]$

TD Learning with Neural Network Approximation

- **Value function approximation**: $x = \phi(s) \in R^d$

$$\hat{V}(x; W, \alpha) = \frac{1}{\sqrt{m}} \sum_{i=1}^{m} \alpha_i \left(W_i^T x\right)^+$$



- **Symmetric Initialization:**

$$\alpha_i = -\alpha_{i+m/2} \sim Unif\{-1,1\},\ W_i(0) = W_{i+m/2}(0) \sim N(0, I_d)$$

- **Neural TD Learning:**

$$W(t+1) = W(t) + \eta_t \left[r(x_t) + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)\right] \nabla_W \hat{V}_t(x_t)$$

Optimization Perspective

Minimizing mean-square Bellman error (MSBE):

$$\min_{W} E_{x\sim\mu}\left(\hat{V}(x;W,\alpha)-\left(r(x)+\gamma E_{x'|x}\,\hat{V}\,(x';W,\alpha)\right)\right)^2$$

- TD Learning can be viewed as a stochastic semi-gradient method.
- With neural network approximation, the MSBE objective becomes non-convex.
- Approximation error between $\hat{V}(x;W,\alpha)$ and true value function $V(x)$.

Goal: Can we achieve $\left\|\hat{V}_T - V\right\| \leq \epsilon$?

- Sample complexity T (required number of samples)?
- Network complexity m (required number of neurons)?

Existing Theory

- **TD Learning with linear function approximation**
  - Finite-time analysis of TD with projection: [Bhandari et al., 2019]
  - Finite-time analysis of TD without projection: [Srikant & Ying, 2019]
  - Finite-time analysis under i.i.d. setting: [Dalal et al., 2018], [Lakshminarayanan & Szepesvári, 2018]

- **(Stochastic) Gradient Descent with two-layer overparametrized neural network**
  - Infinite-width limit $(m \to \infty)$: [Jacot et al., 2018], [Chizat et al., 2019]
  - GD with polynomial width: [Du et al., 2018] , [Oymak and Soltanolkotabi, 2019], [Arora et al., 2019]
  - SGD with polylogarithmic width (classification only): [Ji & Telgarsky, 2020]

Key Challenges:

- Massive overparameterization (poly in $|S|$) is not suitable for TD Learning
- Drift of the network parameter $||W(t) - W(0)||$

Neural Tangent Kernel

- Recall $\hat{V}(x; W, \alpha) = \frac{1}{\sqrt{m}} \sum_{i=1}^{m} \alpha_i \left(W_i^T x\right)^+$

$$\hat{V}(x; W, \alpha) \approx \hat{V}(x; W(0), \alpha) + \frac{1}{\sqrt{m}} \sum_{i=1}^{m} \alpha_i I\left(W_i^T(0)x \geq 0\right) x^T [W_i - W_i(0)]$$

$$\hat{V}(x; W, \alpha) \approx \frac{1}{\sqrt{m}} \sum_{i=1}^{m} \alpha_i I\left(W_i^T(0)x \geq 0\right) x^T W_i$$

- <u>**Neural Tangent Kernel:**</u>

$$K(x, y) = E_{w_0 \sim N(0, I_d)}[I(w_0^T x \geq 0) I(w_0^T y \geq 0) x^T y]$$

  - The NTK is a universal kernel.
  - The corresponding RKHS is dense in the continuous function space defined on a compact set.

- <u>**Assumption:**</u> $V(x) = E[v^T(w_0)x \cdot I(w_0^T x \geq 0)]$, where $\sup_w \|v(w)\|_2 \leq \bar{v} < \infty$.

Neural TD Learning with Regularization

Algorithm 1: Projection-Free NTD

$$W(t+1) = W(t) + \eta \cdot g_t$$

Regularization: Early stopping

$$T = T(\bar{v}, \epsilon, \delta)$$

(Ji & Telgarsky, '19, Li et al., '20) for SL

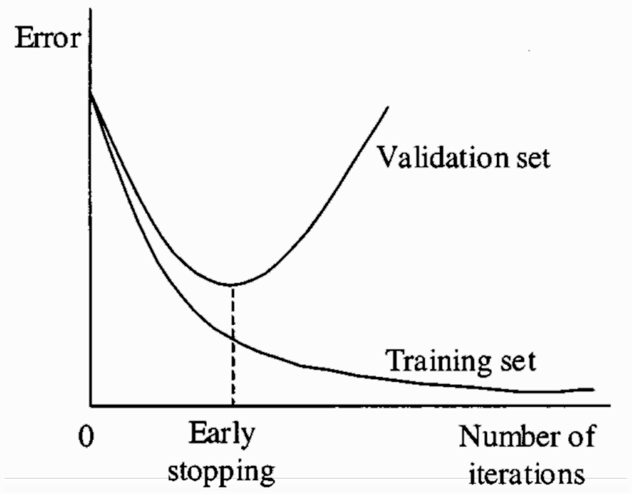
Algorithm 2: Max-Norm NTD

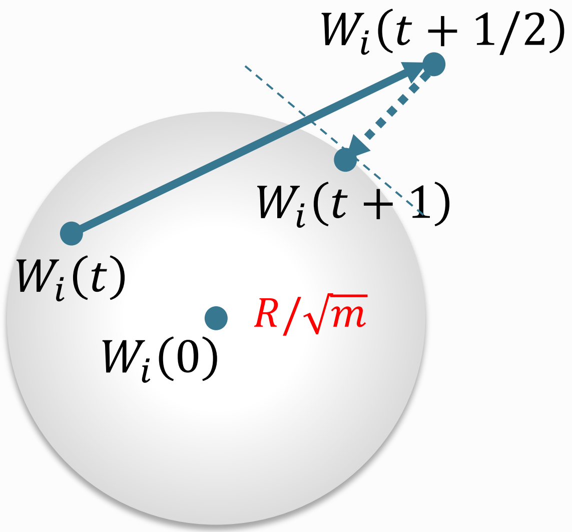
$$W_i(t+1) = \mathrm{Proj}_{B(W_i(0),R)}[W_i(t) + \eta \cdot g_t^i]$$

Regularization: Max-norm

$$||W_i(t) - W_i(0)||_2 \leq R/\sqrt{m}$$

(Srivastava, '14, Goodfellow, '13) for SL





Convergence of Neural TD

Assumption: $V(\cdot) \in F_{NTK}$ (dense in cont. functions over a **compact** state space (Ji et al., '19))

$$\boldsymbol{E}\left[\left|\hat{V}_T - V\right|_\mu 1_\mathcal{E}\right] \le \epsilon \text{ where } \boldsymbol{P}(\mathcal{E}) > 1 - \delta$$

Algorithm 1: Projection-Free NTD

Sample complexity: $T = poly(\bar{v})/\epsilon^6$

Network width: $m = poly(\bar{v})/\epsilon^6$

Algorithm 2: Max-Norm NTD

Sample complexity: $T = poly(R)/\epsilon^4$

Network width: $m = poly(R)/\epsilon^2$

Projection radius: $R > \bar{v}$

Here $\bar{v}$ is the bound of the NTK norm of $V(\cdot)$.

Highlight

- **Some regularization + modest overparameterization** → convergence to true value function

	State space	Network width	Sample complexity	Error	Regularization
Cai et al., 2019	General	$O(1/\epsilon^8)$	$O(1/\epsilon^4)$	$\epsilon + \epsilon_m$	ℓ_2-projection
Wang et al., 2019	General	$O(1/\epsilon^8)$	$O(1/\epsilon^4)$	$\epsilon + \epsilon_\infty$	ℓ_2-projection
Agazzi & Lu, 2019	Finite	$poly(\lvert\mathcal{X}\rvert)$	$O(\log(1/\epsilon))$	ϵ	$poly(\lvert\mathcal{X}\rvert)$ width
Our result	General	$\widetilde{O}(1/\epsilon^6)$	$O(1/\epsilon^6)$	ϵ	Early stopping
Our result	General	$\widetilde{O}(1/\epsilon^2)$	$O(1/\epsilon^4)$	ϵ	Max-norm projection

More expressive power ⟷ Faster convergence

Projection-free NTD
(Early stopping)

[Cai et al., '19]
(ℓ_2-reg.)

Max-norm NTD
(ℓ_∞-reg)

Lyapunov Drift Analysis

- **Minimum norm solution:**

$$\bar{W} = [W_i(0) + \alpha_i \frac{v(W_i(0))}{\sqrt{m}}]_{i\in[m]}$$

  Note that $\nabla\hat{V}(x; W(0), a)^T\, \bar{W} \to V(x), as\ m \to \infty$.

- **Lyapunov function**: $L\big(W(t)\big) = \|W(t) - \bar{W}\|_2^2$

- **Stopping time:** $\tau = \inf\left\{t > 0 \colon \|W_i(t) - W_i(0)\|_2 > \frac{\lambda}{\sqrt{m}} \text{ for some } i\right\}.$

- **Drift bound:**

$$E_t\big[L\big(W(t+1)\big) - L\big(W(t)\big)\big] \le -2\eta(1-\gamma)\big\|\hat{V}_t - V\big\|_\mu^2 + O\left(\eta^2 + \frac{\eta\|\hat{V}_t - V\|_\pi}{\sqrt{m}}\right), \text{ for } t < \tau$$

Drift Bound

- Recall $W(t+1) = W(t) + \eta g_t$,

$$\|W(t+1) - \bar{W}\|_2^2 = \|W(t) - \bar{W}\|_2^2 + 2\eta g_t^T(W(t) - \bar{W}) + \eta^2\|g_t\|_2^2$$

$$g_t = \delta_t \cdot \nabla_W \hat{V}_t(x_t; W(t)),$$
$$\delta_t = r(x_t) + \gamma\hat{V}_t(x_{t+1}) - \hat{V}_t(x_t).$$

- Bound the second term

$$E[g_t^T(W(t) - \bar{W})]$$

$$= E[\delta_t \cdot \nabla_W \hat{V}_t(x_t; W(t))^T(W(t) - \bar{W})]$$

$$= E\left[\delta_t \cdot \left(\underbrace{\hat{V}_t\big(x_t; W(t)\big) - V(x_t)}_{\le -(1-\gamma)\left\|\hat{V}_t - V\right\|_\mu^2} + \underbrace{V(x_t) - \nabla\hat{V}_t\big(x_t; W(0)\big)^T\bar{W}}_{\le O\left(\frac{\bar{v}}{\sqrt{m}}\right)} + \underbrace{\nabla\hat{V}_t\big(x_t; W(0)\big)^T\bar{W} - \nabla\hat{V}_t\big(x_t; W(t)\big)^T\bar{W}}_{\le O\left(\frac{\lambda}{\sqrt{m}}\right)}\right)\right]$$

Extensions and Open Questions

- Extensions of Neural TD Learning
  - Markovian setting
  - Extended feature vector
  - Smooth activation functions

- Open Questions
  - Beyond two-layers, can we achieve reduced overparameterization bound?
  - Beyond two-layers, under what conditions can we achieve global convergence?
  - Is early stopping or regularization necessary?
  - Extension to deep Q-learning to find optimal policy?
  - **How to integrate RL with general nonlinear function approximation in a more principled manner?**

Optimization-based RL Algorithms

- Bellman-residual-minimization methods
  - Residual gradient algorithm [Baird, 1995]
  - Gradient TD [Sutton et al., 2009]
  - Least-Squares Policy Iteration [Antos et al., 2006]
  - SBEED [Dai et al., 2018]
- Linear programming-based methods
  - Stochastic primal-dual method [Chen & Wang, 2016] [Lee & He, 2018]
  - Dual actor-critic [Dai et al., 2017]
  - Primal-dual stochastic mirror descent [Jin & Sidford, 2020]
  - Logistic Q-learning [Bas-Serrano et al., 2021]
- Policy gradient methods
  - Natural policy gradient method (NPG) [Kakade, 2001]
  - Trust region policy optimization (TRPO) [Schulman et al., 2015]
  - Proximal policy optimization algorithm (PPO) [Schulman et al., 2017]
  - Entropy-regularized policy gradient methods and actor-critic algorithms

Rich theory and gradient-based algorithms for nonconvex optimization

Exploitation of off-policy data

Adaptation to neural network approximation

Extensibility (safety, multi-agent RL, etc)

Revisit Bellman Optimality Equation

- Recall the Bellman optimality equation:

$$V^*(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \mathbb{E}_{s'|s,a}[V^*(s')] \right]$$

- Equivalently:

$$V^*(s) = \max_{\pi(\cdot|s) \in P(\mathcal{A})} \mathbb{E}_{a \sim \pi(\cdot|s)} \left[R(s,a) + \gamma \mathbb{E}_{s'|s,a}[V^*(s')] \right]$$

  - The max-operator is highly nonsmooth and causes instability when function approximation is used.

Smoothing the max-Operator

- Introduce entropic regularization to Bellman optimality equation,

$$V_\lambda(s) = \max_{\pi(\cdot|s)\in P(\mathcal{A})} \left(\mathbb{E}_{a\sim\pi(\cdot|s)}\left[R(s,a) + \gamma\mathbb{E}_{s'|s,a}[V_\lambda(s')]\right] + \lambda\cdot H(\pi,s)\right)$$

$$= \lambda \log\left(\sum_{a\in\mathcal{A}} \exp\left(\frac{R(s,a) + \gamma\mathbb{E}_{s'|s,a}[V_\lambda(s')]}{\lambda}\right)\right)$$

  - $H(\pi,s) = -\sum \pi(a|s)\log\pi(a|s)$ is the entropy, $\lambda > 0$ is the smoothness parameter

- The smoothed Bellman operator is also a γ-contraction.
- Smoothing bias: $\left|\left|V^*(s) - V_\lambda(s)\right|\right|_\infty \leq \frac{\lambda\cdot C}{1-\gamma}$.
- The corresponding (V_λ, π_λ) satisfies the smoothed Bellman equation:

$$V(s) = R(s,a) + \gamma\mathbb{E}_{s'|s,a}[V(s')] - \lambda\cdot\log\pi(a|s), \forall a \in \mathcal{A}.$$

Bellman Residual Minimization

- Minimizing mean-squared smoothed Bellman error:

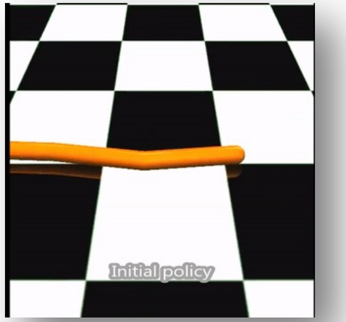
(CSO): $$\min_{V,\pi} \mathbb{E}_{s,a}\left[\left(R(s,a)+\gamma\mathbb{E}_{s'|s,a}[V(s')]-\lambda\log\pi(a|s)-V(s)\right)^2\right]$$

(Min-Max SO): $$\min_{V,\pi}\max_{\nu}\Psi(V,\pi;\nu)$$
$$:=\mathbb{E}_{s,a,s'}\left[\left(R(s,a)+\gamma V(s')-\lambda\log\pi(a|s)-V(s)\right)\cdot\nu(s,a)\right]-\frac{1}{2}\mathbb{E}_s[\nu^2(s,a)]$$

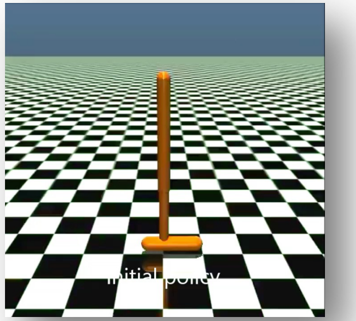
Convergent off-policy RL algorithm
with nonlinear function approximation

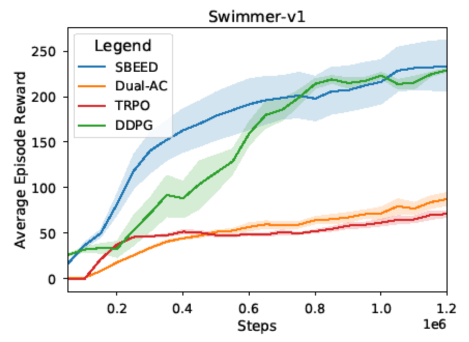
[Dai et al., ICML 2018]

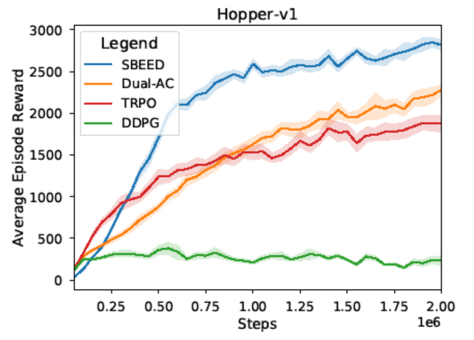
Swimmer



Hopper







Caveat: require solving nonconvex-(non)concave min-max optimization!

Linear-programming-based Method

- LP formulation:

$$\min_{V,Q} \mu^T V \quad \text{s.t.} \quad Q_a \leq V, \quad \alpha P_a V + R_a = Q_a, \quad a \in \mathcal{A}$$

(Min-Max SO): $$\min_{x=(V,Q)} \max_{y=(\lambda,\mu)} L_M(x;y)$$

$$:= \mu^T V + \mu^T M(\alpha P V + R - Q) + \lambda^T (Q - (\mathbf{1}_{|\mathcal{A}|} \otimes I_{|\mathcal{S}|})V)$$

Convergent off-policy RL algorithm w/o function approximation

[Dai et al., 2017; Donghwan and H., 2019]

(Primal policy): $\pi_p^*(s,a) = \operatorname*{argmin}_{a \in A} Q_a^*(s)$

(Dual policy): $\pi_d^*(s,a) \propto \lambda_a^*(s)$

Environment	Dual-AC	PPO	TRPO
Pendulum	**−155.45**	−266.98	−245.11
InvertedDoublePendulum	**8599.47**	1776.26	3070.96
Swimmer	234.56	223.13	232.89
Hopper	**2983.79**	2376.15	2483.57
HalfCheetah	**3041.47**	2249.10	2347.19
Walker	**4103.60**	3315.45	2838.99

Caveat: lack of duality; require solving nonconvex-(non)concave min-max optimization!

Summary

- Understanding the convergence and generalization of deep RL from modern deep learning theory
- Principled approaches for RL with neural network approximation

Value-based methods
- Neural TD learning
- Neural Q-learning

Optimization-based methods
- Bellman Residual Minimization
- Linear Programming

Policy-based methods
- Neural Policy Gradient
- Neural Actor Critic

Open Questions

- Benefits of depth and different architectures?
- Nonconvex min-max optimization?
- Regularization and sample complexity?

Reference

- **[Cayci, Satpathi, H., Srikant, 2021]** Sample Complexity and Overparameterization Bounds for Temporal Difference Learning with Neural Network Approximation. arXiv preprint arXiv:2103.01391, 2021.

- **[Dai et al., 2018]** SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation. ICML 2018.

- **[Du et al., 2019]** Gradient Descent Provably Optimizes Over-parameterized Neural Networks. ICLR 2019.

- **[Fan et al., 2020]** A Theoretical Analysis of Deep Q-Learning. arXiv: 1901.00137, 2019.