CS 287 Lecture 19 (Fall 2019)
Off-Policy, Model-Free RL:
DQN, SoftQ , DDPG, SAC

Pieter Abbeel
UC Berkeley EECS

Outline

- Motivation
- Q-learning
- DQN + variants
- Q-learning with continuous action spaces (SoftQ)
- Deep Deterministic Policy Gradient (DDPG)
- Soft Actor Critic (SAC)

Story-line

- TRPO, PPO: Importance sampling surrogate loss allows to do more than a gradient step, but still very local
- Could we re-use samples more? Could we learn more globally / off-policy?
- Yes! By leveraging the dynamic programming structure of the problem, breaking it down into 1-step pieces
  - Q-learning, DQN: 1-step (sampled) off-policy Bellman back-ups → more sample re-use → more data-efficient learning directly about the optimal policy
  - Why not always Q-learning/DQN?
    - Often less stable
    - The data doesn't always support learning about the optimal policy (even if in principle can learn fully off-policy)
  - DDGP, SAC: like Q-learning, but does off-policy learning about the current policy and how to locally improve it (vs. directly learning about the optimal policy)

Outline

- Motivation
- ***Q-learning***
- DQN + variants
- Q-learning with continuous action spaces (SoftQ)
- Deep Deterministic Policy Gradient (DDPG)
- Soft Actor Critic (SAC)

Recap Q-Values

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

(Tabular) Q-Learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:

$$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha) Q_k(s,a) + \alpha \left[\text{target}(s') \right]$$

(Tabular) Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s’

If s’ is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s’

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \left[\text{target}\right]$$

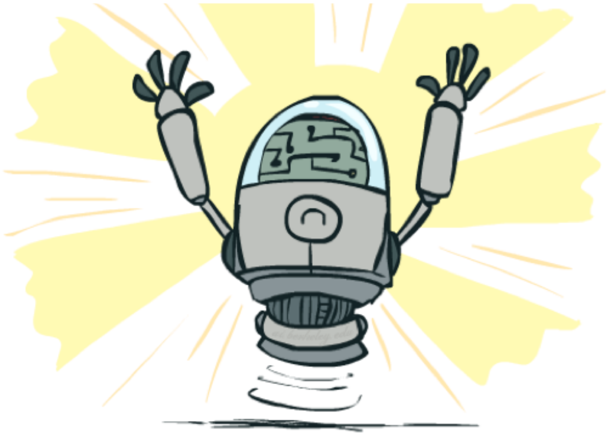
$$s \leftarrow s'$$

How to sample actions?

- Choose random actions?
- Choose action that maximizes $Q_k(s, a)$ (i.e. greedily)?
- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - ... but not decrease it too quickly



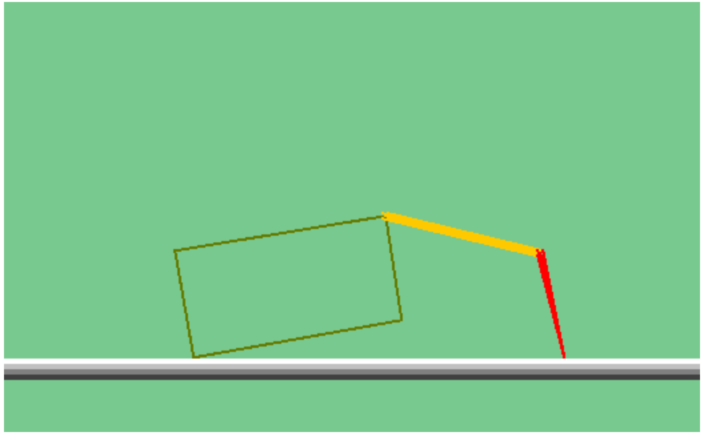
Q-Learning Properties

- Technical requirements.
  - All states and actions are visited infinitely often
    - Basically, in the limit, it doesn't matter how you select actions (!)
  - Learning rate schedule such that for all state and action pairs (s,a):

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty$$

For details, see Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation, 6(6), November 1994.

Q-Learning Demo: Crawler



- States: discretized value of 2d state: (arm angle, hand angle)
- Actions: Cartesian product of {arm up, arm down} and {hand up, hand down}
- Reward: speed in the forward direction

Video of Demo Crawler Bot

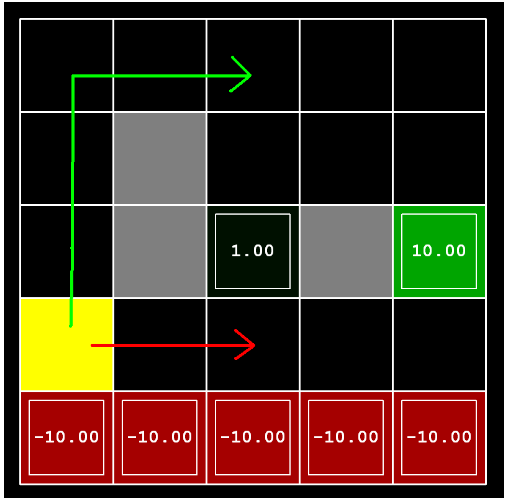
Video of Demo Q-Learning -- Crawler

Outline

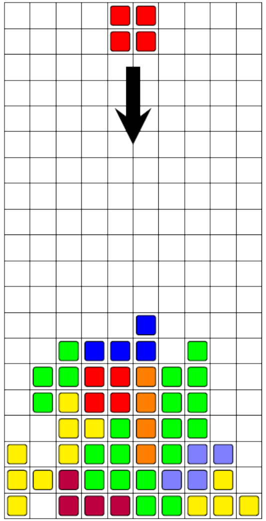
- Motivation
- Q-learning
- ***DQN + variants***
- Q-learning with continuous action spaces (SoftQ)
- Deep Deterministic Policy Gradient (DDPG)
- Soft Actor Critic (SAC)

Can tabular methods scale?

- Discrete environments



Gridworld
10^1

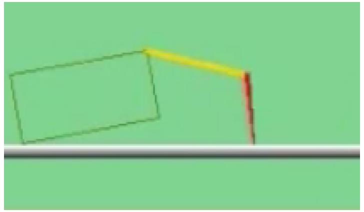


Tetris
10^60

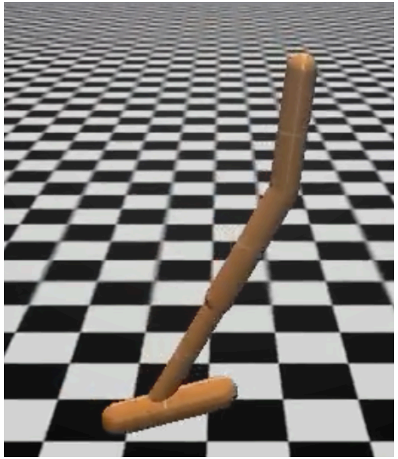
Atari
10^308 (ram) 10^16992 (pixels)

Can tabular methods scale?

- Continuous environments (by crude discretization)



Crawler
10^2



Hopper
10^10

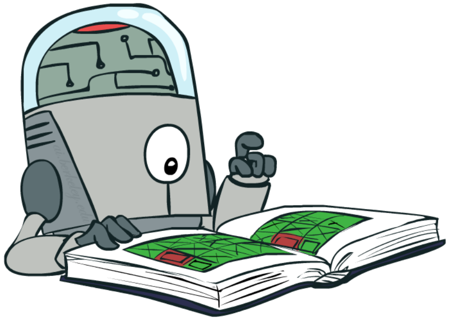
Humanoid
10^100

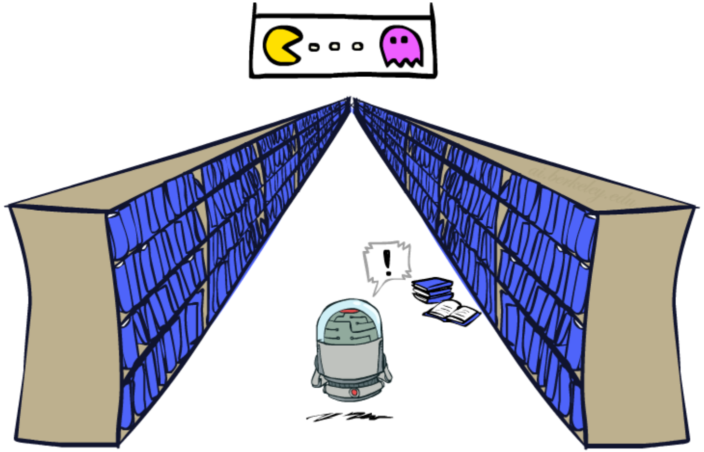
Generalizing Across States

- Basic Q-Learning keeps a table of all q-values

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning





Approximate Q-Learning

- Instead of a table, we have a parametrized Q function: $Q_\theta(s, a)$
  - Can be a linear function in features:

$$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

  - Or a neural net, decision tree, etc.

- Learning rule:
  - Remember: $\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$
  - Update:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right] \Bigg|_{\theta = \theta_k}$$

Recall Approximate Q-Learning

- Instead of a table, we have a parametrized Q function
  - E.g. a neural net $Q_\theta(s, a)$
- Learning rule:
  - Compute target:

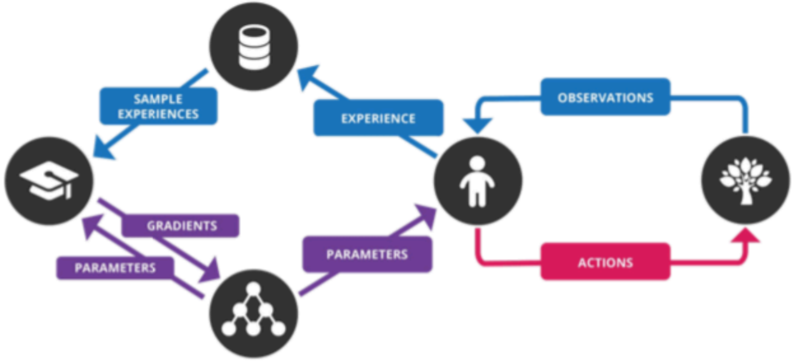
$$\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$$

  - Update Q-network:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta = \theta_k}$$

DQN Training Algorithm

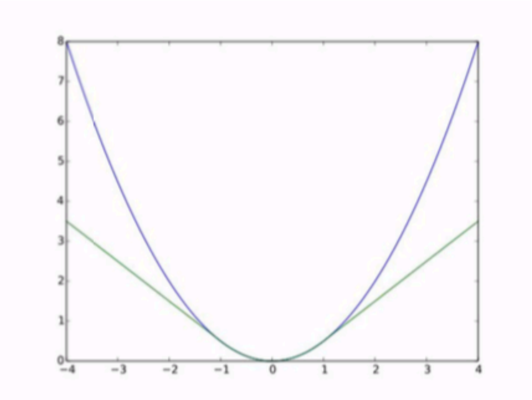
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For



DQN Details

- Uses Huber loss instead of squared loss on Bellman error:

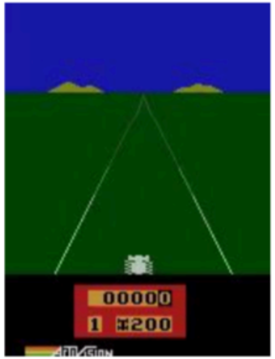
$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$



- Uses RMSProp instead of vanilla SGD.
  - Optimization in RL really matters.
- It helps to anneal the exploration rate.
  - Start ε at 1 and anneal it to 0.1 or 0.05 over the first million frames.

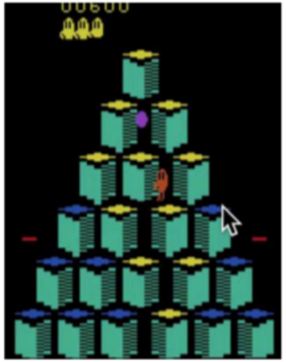
DQN on ATARI

Pong



Enduro

Beamrider

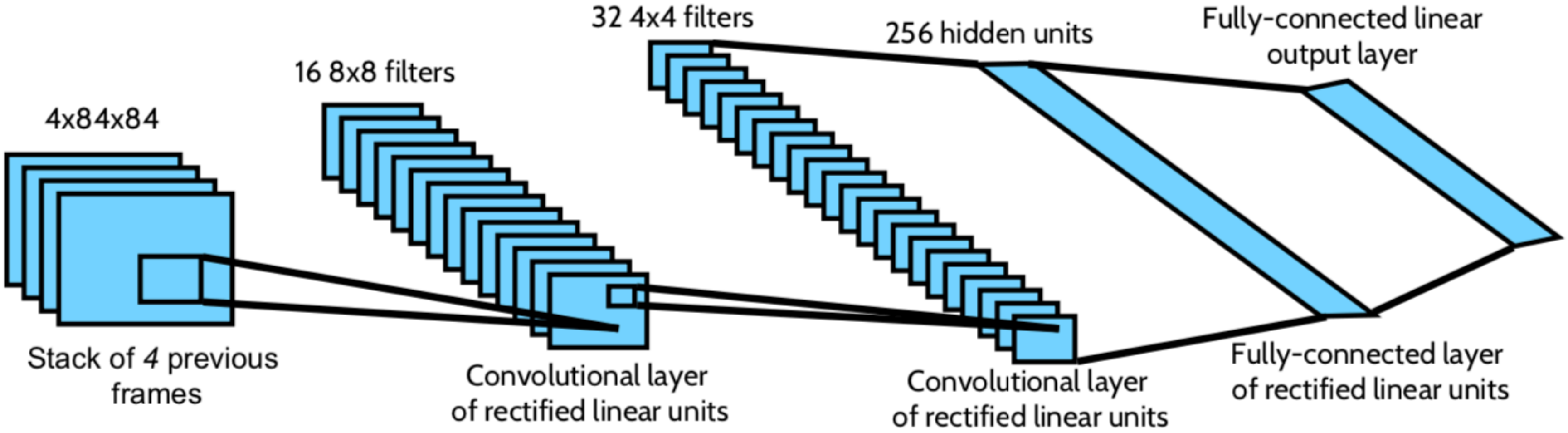


Q*bert

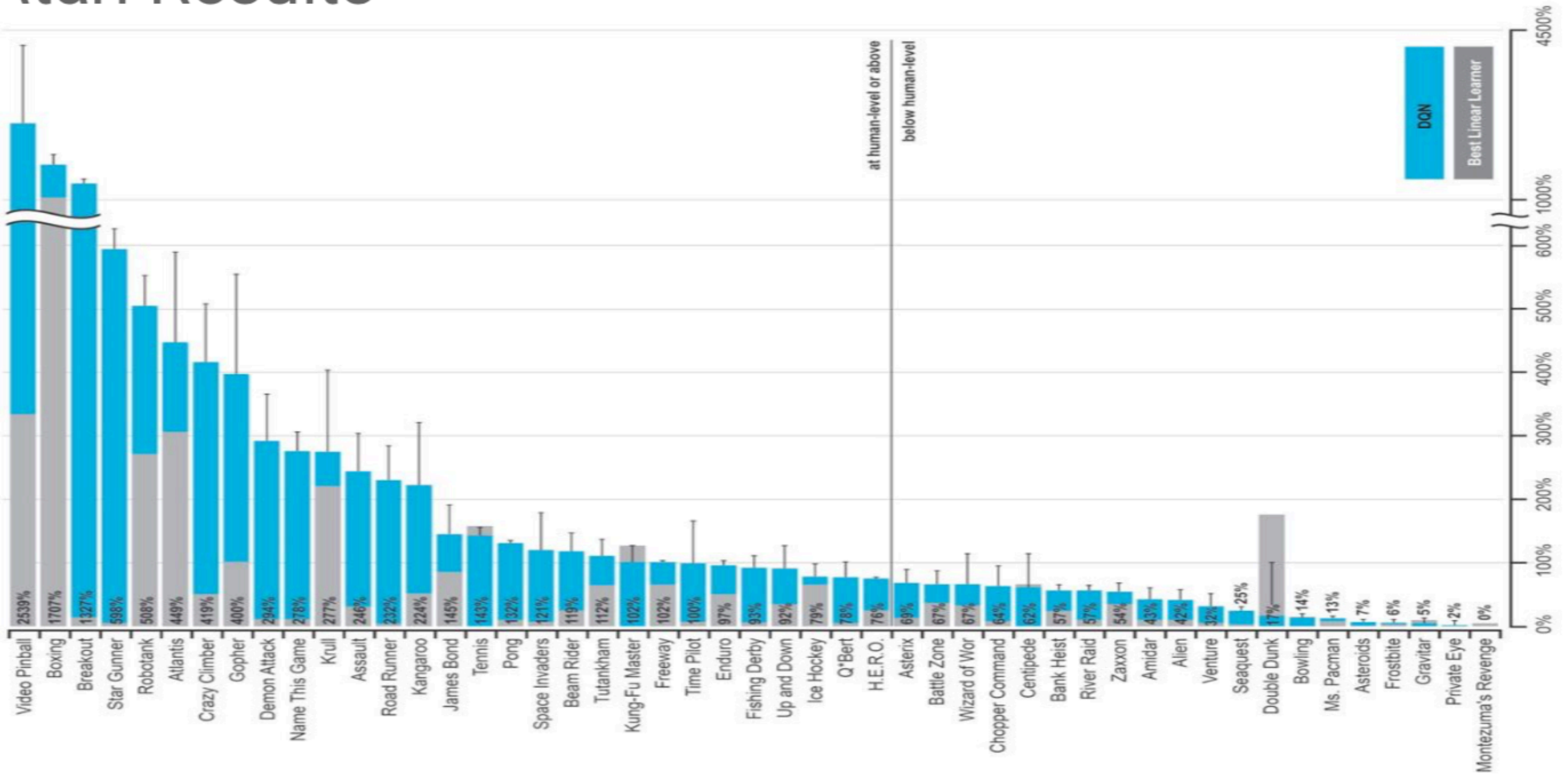
- 49 ATARI 2600 games.
- From pixels to actions.
- The change in score is the reward.
- Same algorithm.
- Same function approximator, w/ 3M free parameters.
- Same hyperparameters.
- Roughly human-level performance on 29 out of 49 games.

ATARI Network Architecture

- Convolutional neural network architecture:
  - History of frames as input.
  - One output per action - expected reward for that action $Q(s, a)$.
  - Final results used a slightly bigger network (3 convolutional + 1 fully-connected hidden layers).



Atari Results



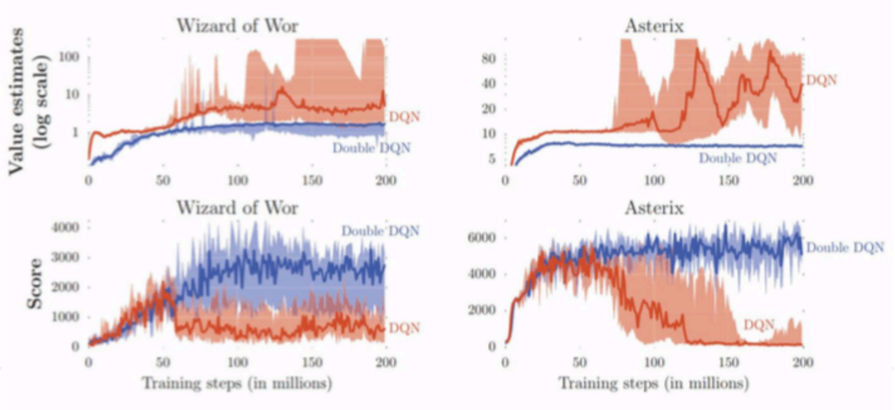
"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

Double DQN

- There is an upward bias in $max_a\ Q(s, a; \theta)$.
- DQN maintains two sets of weight θ and θ^-, so reduce bias by using:
  - θ for selecting the best action.
  - θ^- for evaluating the best action.
- Double DQN loss:

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\ D}\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta_i^-) - Q(s, a; \theta_i)\right)^2$$

	no ops		human starts		
	DQN	DDQN	DQN	DDQN	DDQN (tuned)
Median	93%	**115%**	47%	88%	**117%**
Mean	241%	**330%**	122%	273%	**475%**



“Double Reinforcement Learning with Double Q-Learning”, van Hasselt et al. (2016)

DeepMind

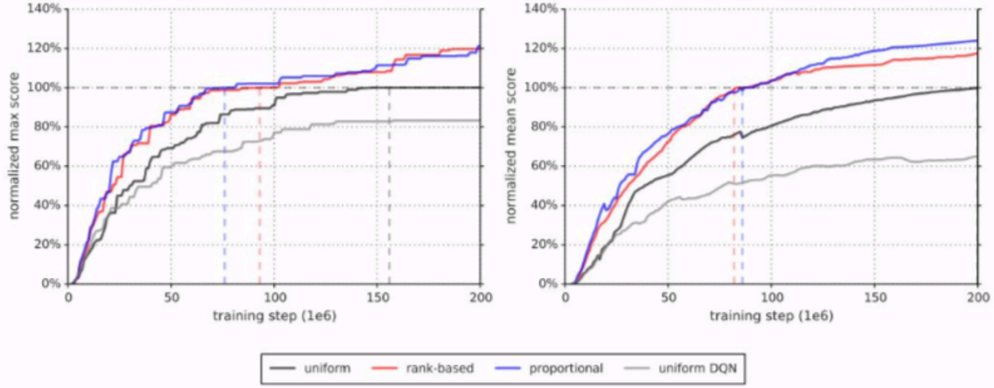
Prioritized Experience Replay

- Replaying all transitions with equal probability is highly suboptimal.
- Replay transitions in proportion to absolute Bellman error:

$$\left| r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right|$$

- Leads to much faster learning.

	DQN		Double DQN (tuned)		
	baseline	rank-based	baseline	rank-based	proportional
Median	48%	106%	111%	113%	128%
Mean	122%	355%	418%	454%	551%
> baseline	–	41	–	38	42
> human	15	25	30	33	33
# games	49	49	57	57	57



DeepMind

"Prioritized Experience Replay", Schaul et al. (2016)

See also

- “Rainbow: Combining Improvements in Deep Reinforcement Learning,” Matteo Hessel et al, 2017
  - Double DQN (DDQN)
  - Prioritized Replay DDQN
  - Dueling DQN
  - Distributional DQN
  - Noisy DQN

Outline

- Motivation
- Q-learning
- DQN + variants
- ***Q-learning with continuous action spaces (SoftQ)***
- Deep Deterministic Policy Gradient (DDPG)
- Soft Actor Critic (SAC)

Soft Q-Learning

$$V_t(\mathbf{s}_t) = \log \int \exp\left(Q_t(\mathbf{s}_t, \mathbf{a}_t)\right) d\mathbf{a}_t$$

→ **Use a sample estimate**

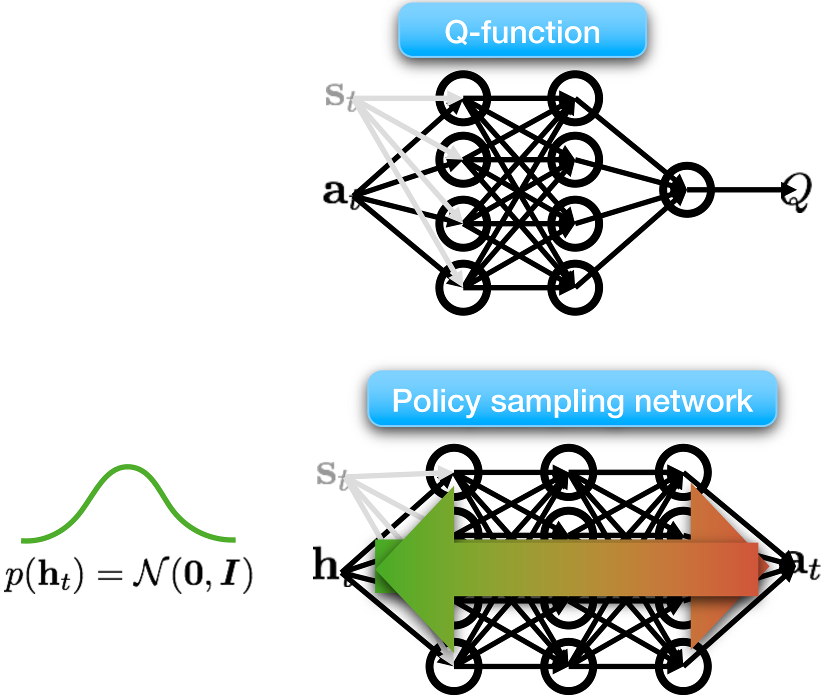
$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\mathbf{s}_{t+1}}\left[V_{t+1}(\mathbf{s}_{t+1})\right]$$

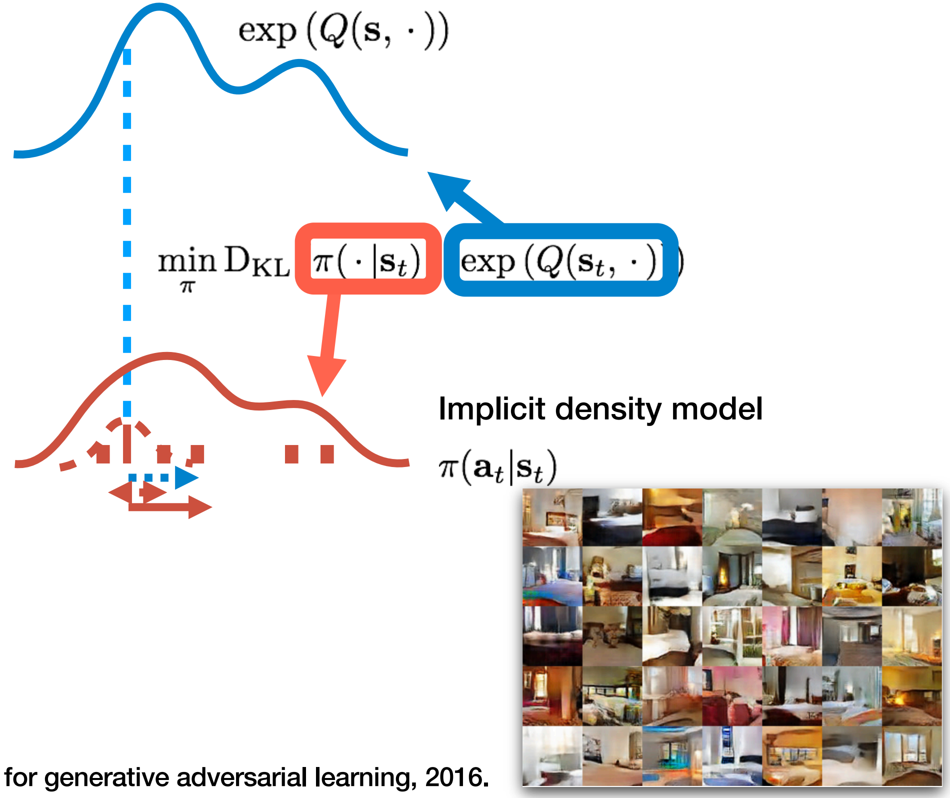
→ **Supervised learning**

$$\pi_t(\mathbf{a}_t|\mathbf{s}_t) \propto \exp\left(Q_t(\mathbf{s}_t, \mathbf{a}_t)\right)$$

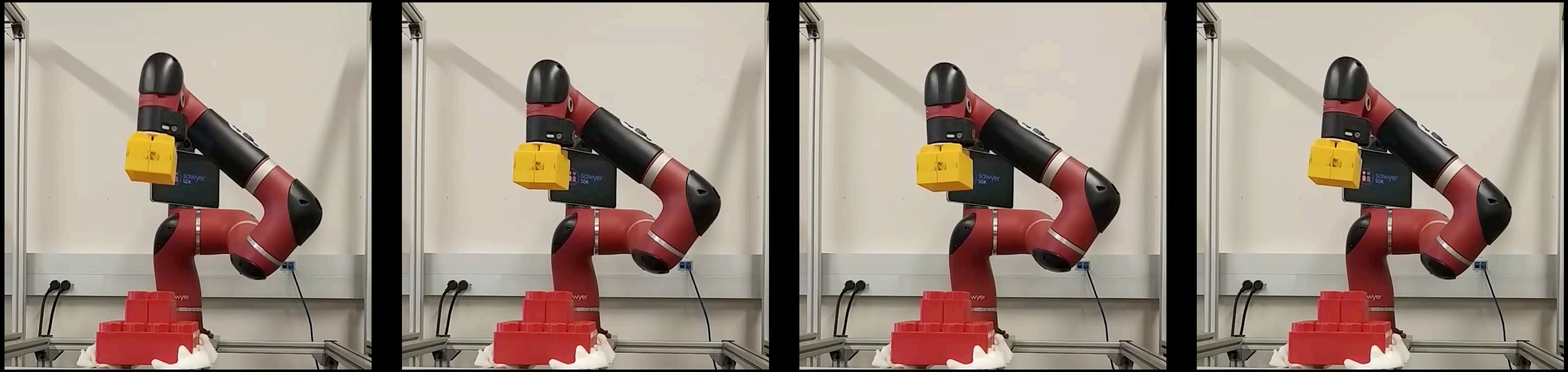
→ **Stein variational gradient descent**

Stein Variational Gradient Descent: Intuition





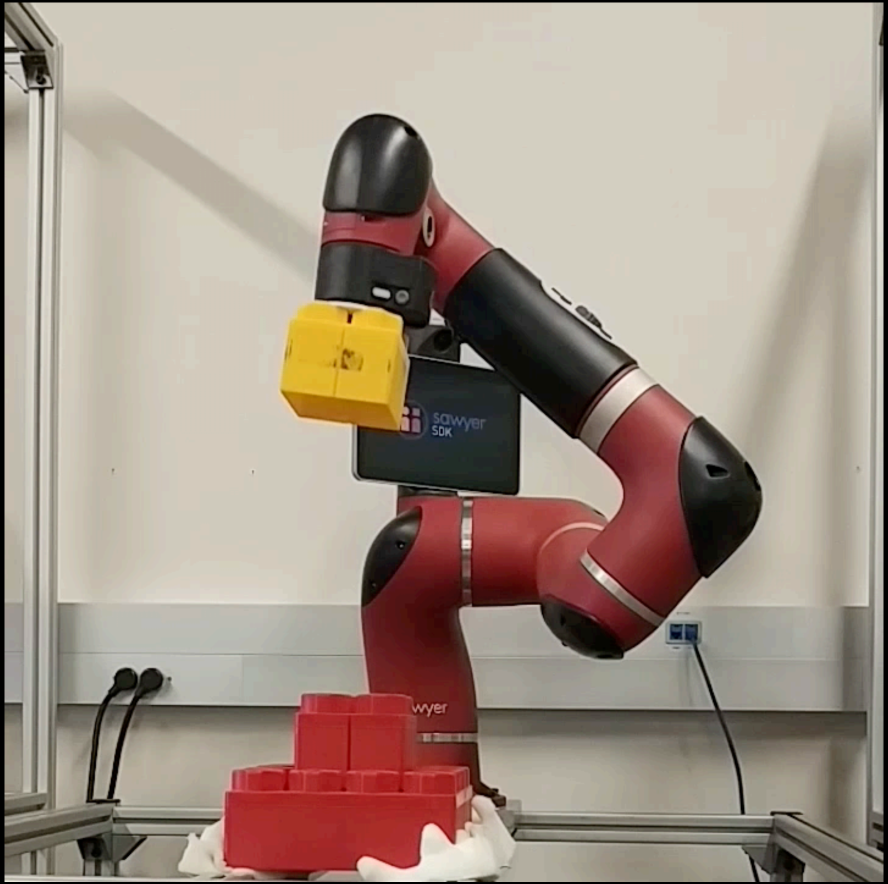
D. Wang et al., Learning to draw samples: With application to amortized MLE for generative adversarial learning, 2016.



0 min | 12 min | 30 min | 2 hours

Training time

sites.google.com/view/composing-real-world-policies/



After 2 hours of training

sites.google.com/view/composing-real-world-policies/

Outline

- Motivation
- Q-learning
- DQN + variants
- Q-learning with continuous action spaces (SoftQ)
- ***Deep Deterministic Policy Gradient (DDPG)***
- Soft Actor Critic (SAC)

Deep Deterministic Policy Gradient (DDPG): Basic (=SVG(0))

- for iter = 1, 2, …

  Roll-outs:
  Execute roll-outs under current policy (+some noise for exploration)

  Q function update:

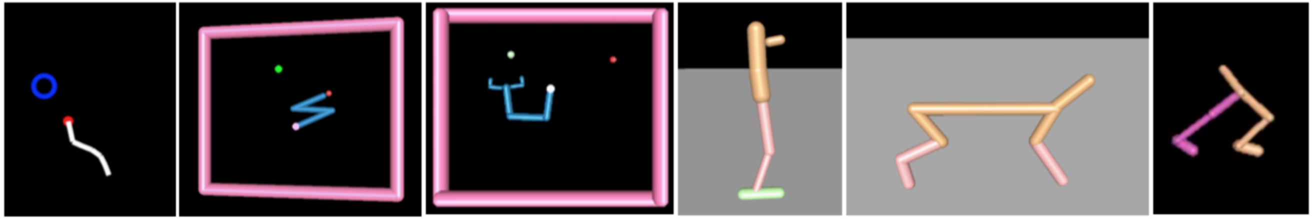
  $$g \propto \nabla_\phi \sum_t (Q_\phi(s_t, u_t) - \hat{Q}(s_t, u_t))^2 \quad \text{with} \quad \hat{Q}(s_t, u_t) = r_t + \gamma Q_\phi(s_{t+1}, u_{t+1})$$

  Policy update:
  Backprop through Q to compute gradient estimates for all t:

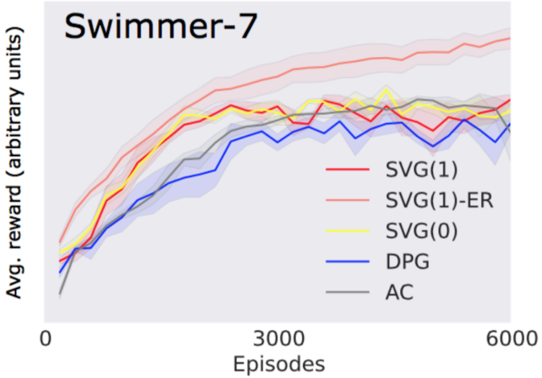
  $$g \propto \sum_t \nabla_\theta Q_\phi(s_t, \pi_\theta(s_t, v_t))$$

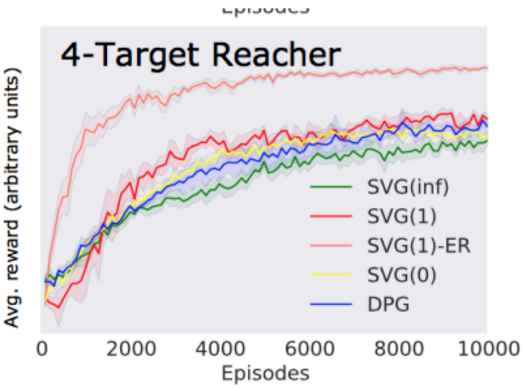
SVG(k)

- Applied to 2-D robotics tasks

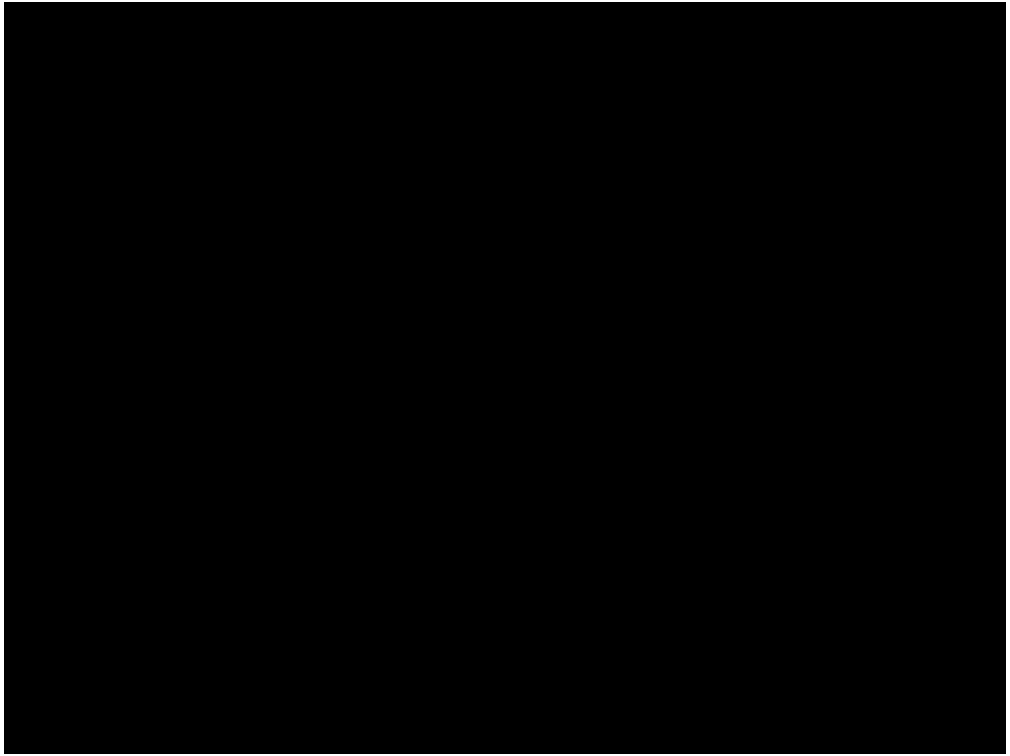


- Different gradient estimators behave similarly





SVG(k)



Deep Deterministic Policy Gradient (DDPG): Complete

- Add noise for exploration
- Incorporate replay buffer for off-policy learning
- For increased stability, use lagged (Polyak-averaging) version of Q_ϕ and π_θ for target values

$$\hat{Q}_t = r_t + \underbrace{\gamma Q_{\phi'}(s_{t+1}, \pi_{\theta'}(s_{t+1}))}_{\text{off-policy!}}$$

DDPG

for iteration$=1,2,\ldots$ **do**

Act for several timesteps, add data to replay buffer

Sample minibatch

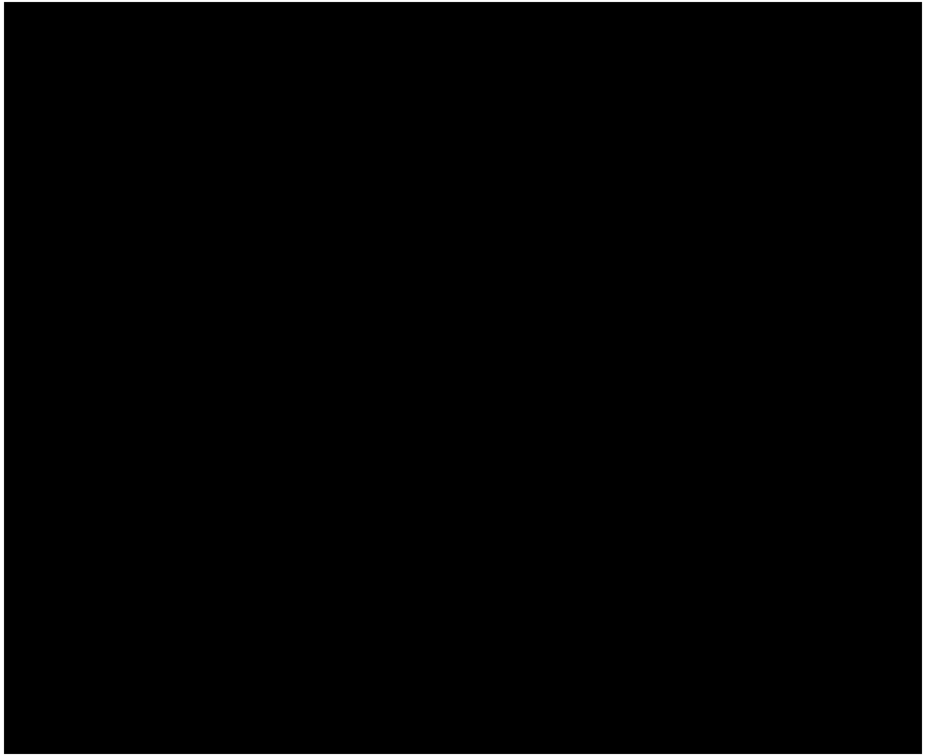
Update π_θ using $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$

Update Q_ϕ using $g \propto \nabla_\phi \sum_{t=1}^{T} (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$,

end for

- Applied to 2D and 3D robotics tasks and driving with pixel input

DDPG



DDPG

\+ very sample efficient thanks to off-policy updates

\- often unstable

→ Soft Actor Critic (SAC), which adds entropy of policy to the objective, ensuring better exploration and less overfitting of the policy to any quirks in the Q-function

Outline

- Motivation
- Q-learning
- DQN + variants
- Q-learning with continuous action spaces (SoftQ)
- Deep Deterministic Policy Gradient (DDPG)
- ***Soft Actor Critic (SAC)***

Soft Policy Iteration

1. Soft policy evaluation:

Fix policy, apply soft Bellman backup until converges:

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \mathbb{E}_{\mathbf{s}'\sim p_{\mathbf{s}},\ \mathbf{a}'\sim\pi}\left[Q(\mathbf{s}',\mathbf{a}') - \log\pi(\mathbf{a}'|\mathbf{s}')\right]$$

This converges to Q^π.

2. Soft policy improvement:

Update the policy through information projection:

$$\pi_{\text{new}} = \arg\min_{\pi'} \mathrm{D}_{\mathrm{KL}}\left(\pi'(\,\cdot\,|\mathbf{s}) \,\middle\|\, \frac{1}{Z}\exp Q^{\pi_{\text{old}}}(\mathbf{s},\,\cdot\,)\right)$$

For the new policy, we have $Q^{\pi^{\text{new}}} \geq Q^{\pi^{\text{old}}}$.

3. Repeat until convergence

Soft Actor-Critic

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* ICML, 2018.

1. Take one stochastic gradient step to minimize soft Bellman residual
2. Take one stochastic gradient step to minimize the KL divergence
3. Execute one action in the environment and repeat

Soft Actor Critic

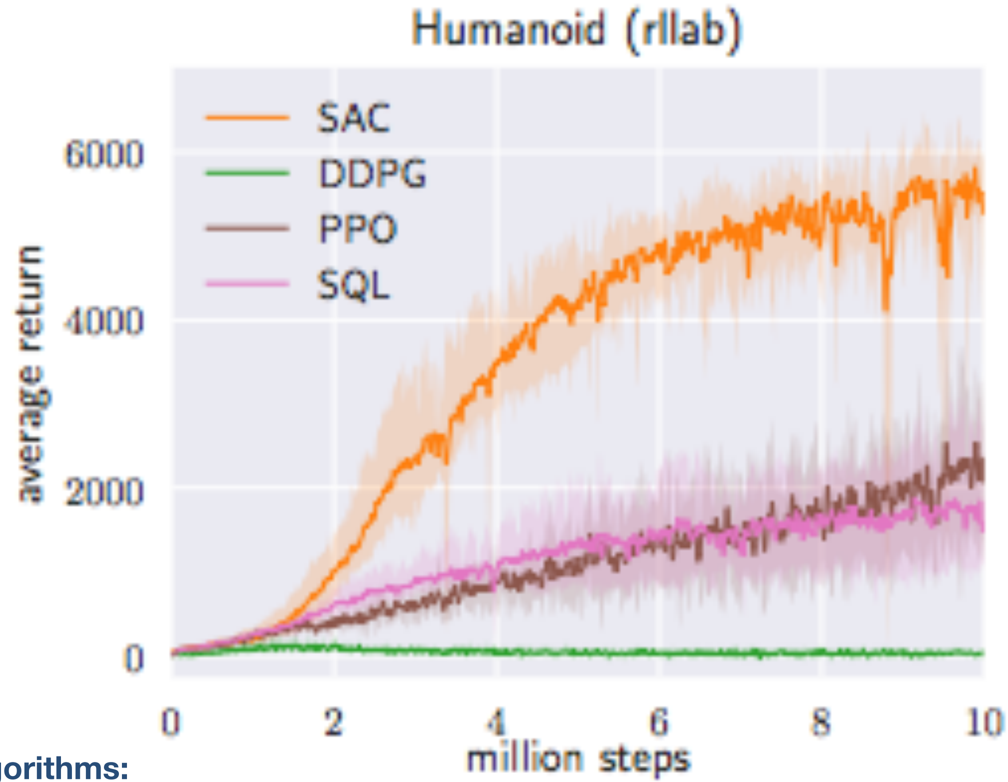
- Objective: $$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi} \left[r(\mathbf{s}_t,\mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right]$$
- Iterate:
  - Perform roll-out from pi, add data in replay buffer
  - Learn V, Q, pi:

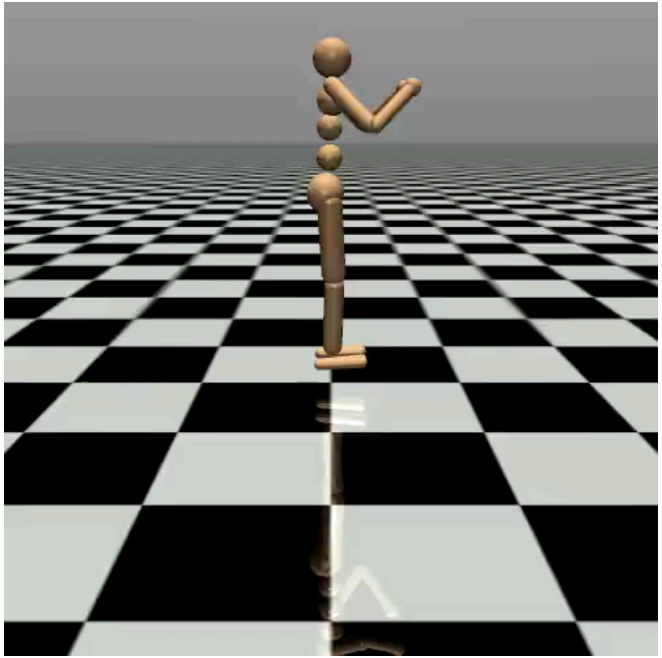
$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}} \left[\frac{1}{2} \left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t\sim\pi_\phi} \left[Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \right] \right)^2 \right]$$

$$\hat{Q}(\mathbf{s}_t,\mathbf{a}_t) = r(\mathbf{s}_t,\mathbf{a}_t) + \gamma \, \mathbb{E}_{\mathbf{s}_{t+1}\sim p} \left[V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right]$$

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}} \left[\mathrm{D}_{\mathrm{KL}} \left(\pi_\phi(\cdot|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q_\theta(\mathbf{s}_t,\cdot)\right)}{Z_\theta(\mathbf{s}_t)} \right) \right]$$

[see also: https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665]

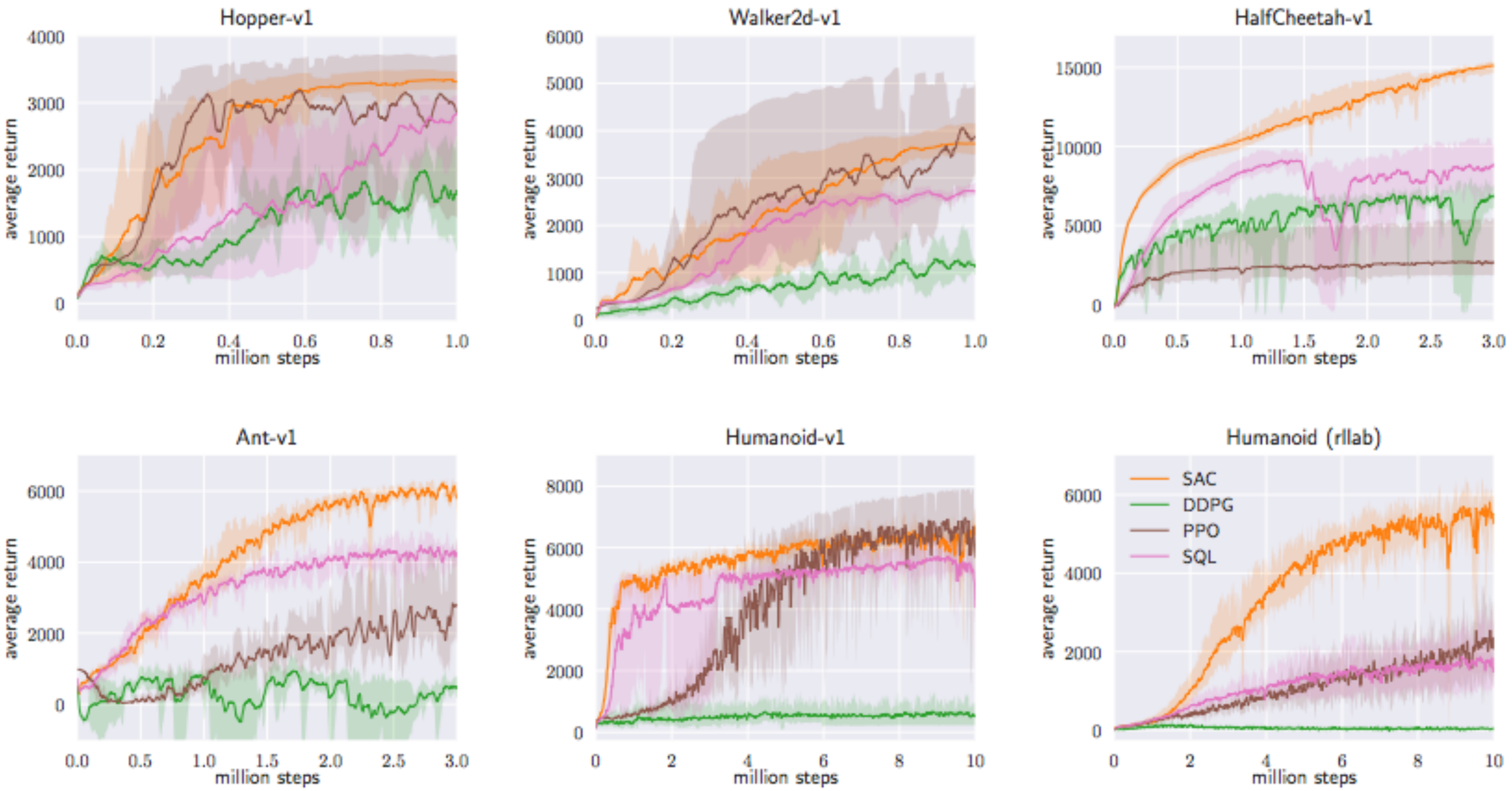




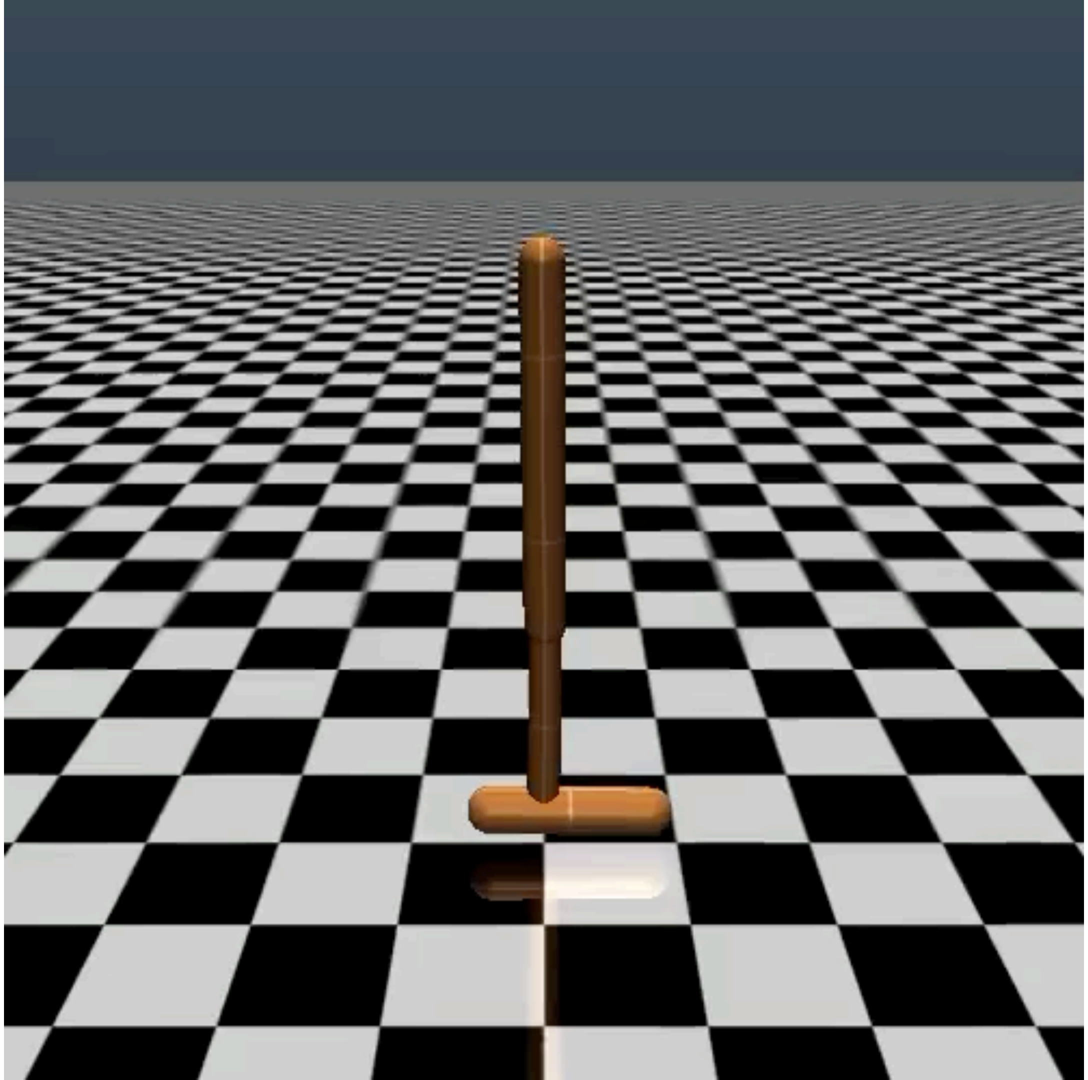
Algorithms:

Soft Actor-Critic (SAC)
Deep Deterministic Policy Gradient (DDPG)
Proximal Policy Optimization (PPO)
Soft Q-Learning (SQL)

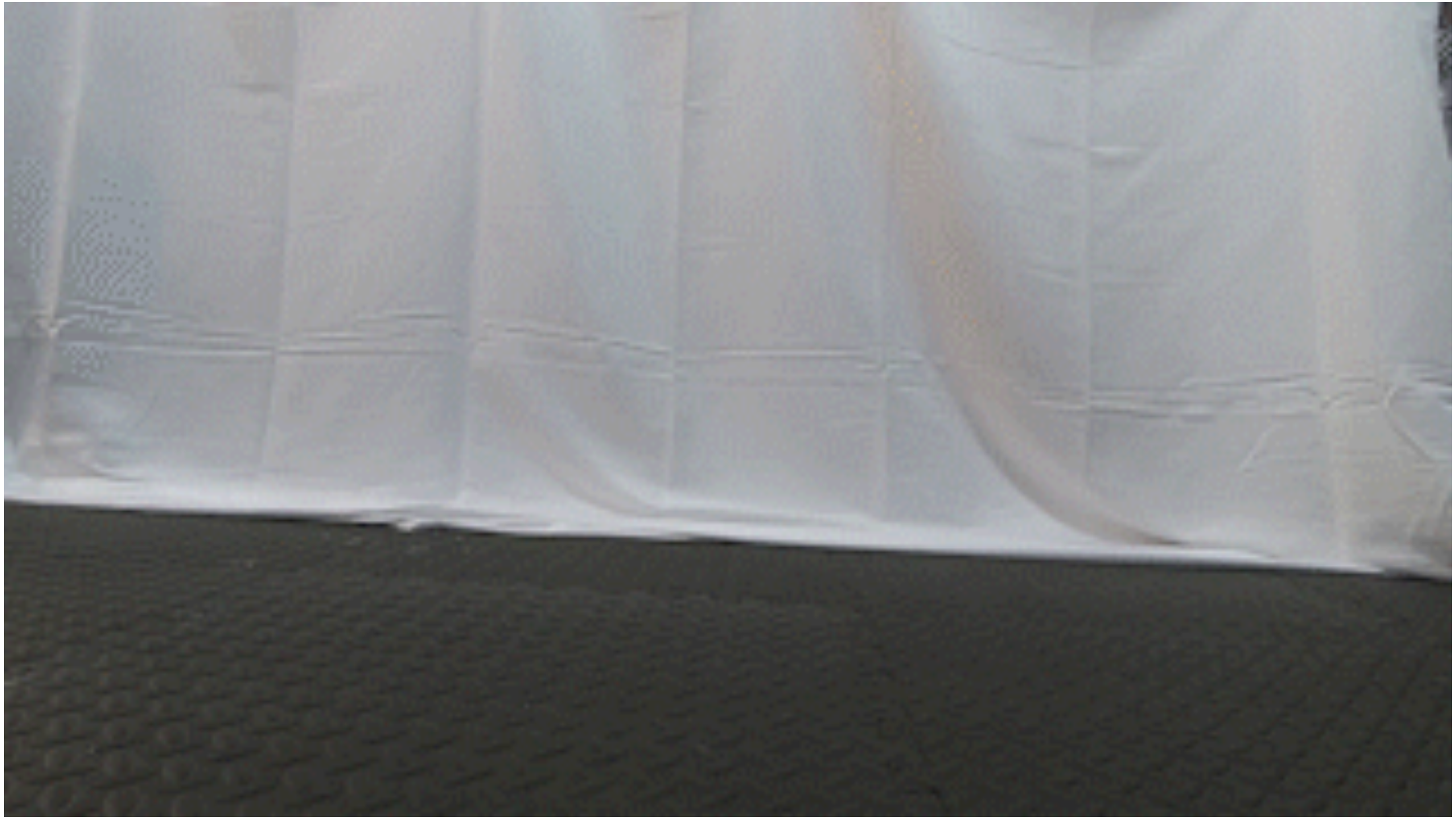
sites.google.com/view/soft-actor-critic



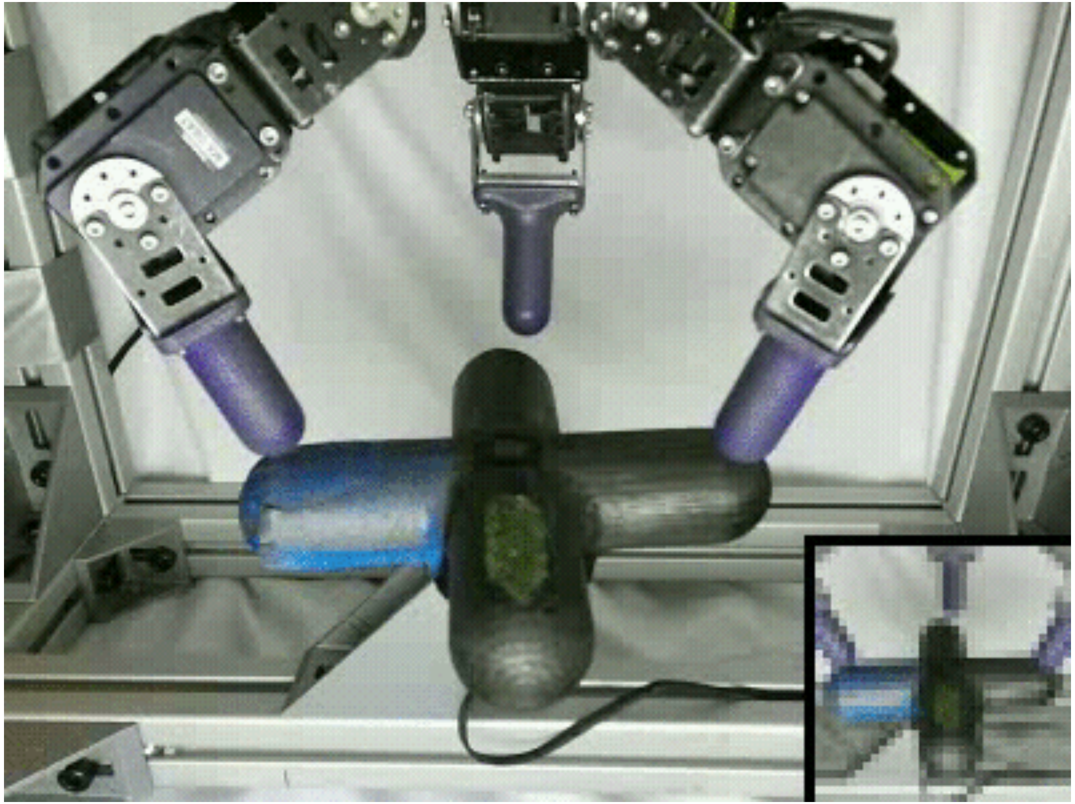
sites.google.com/view/soft-actor-critic



Real Robot Results



Real Robot Results



Real Robot Results