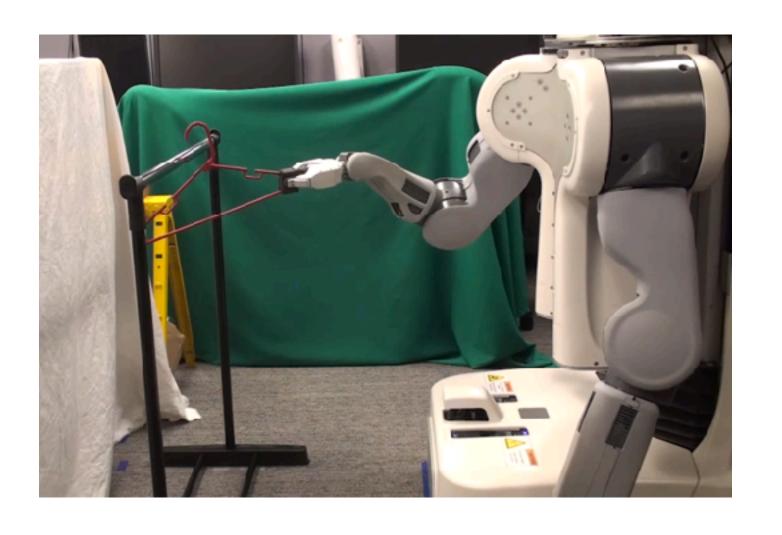
Deep Reinforcement Learning: Foundations and Recent Advances

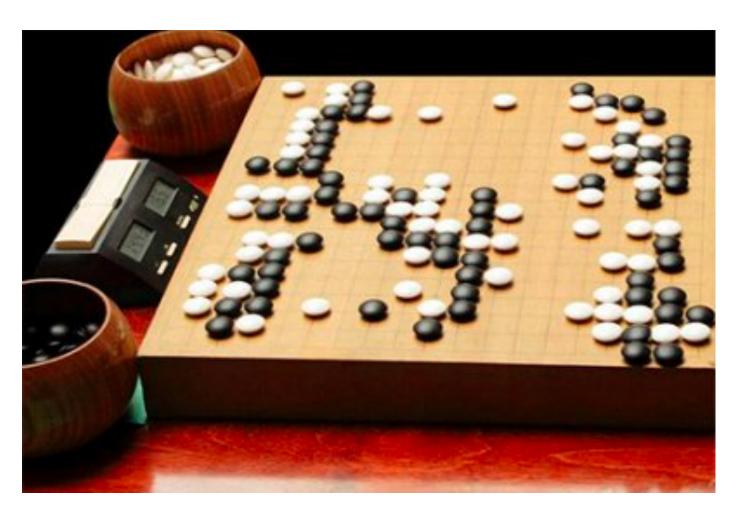
Recent Breakthroughs in Al



[Mnih et al, 2013]



[Levine et al, 2015; Finn et al, 2016]



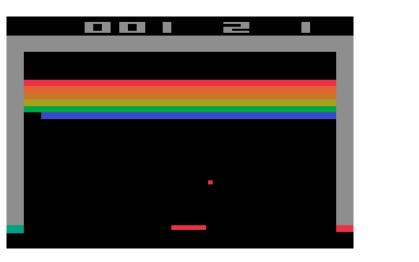
[Silver et al, 2016]



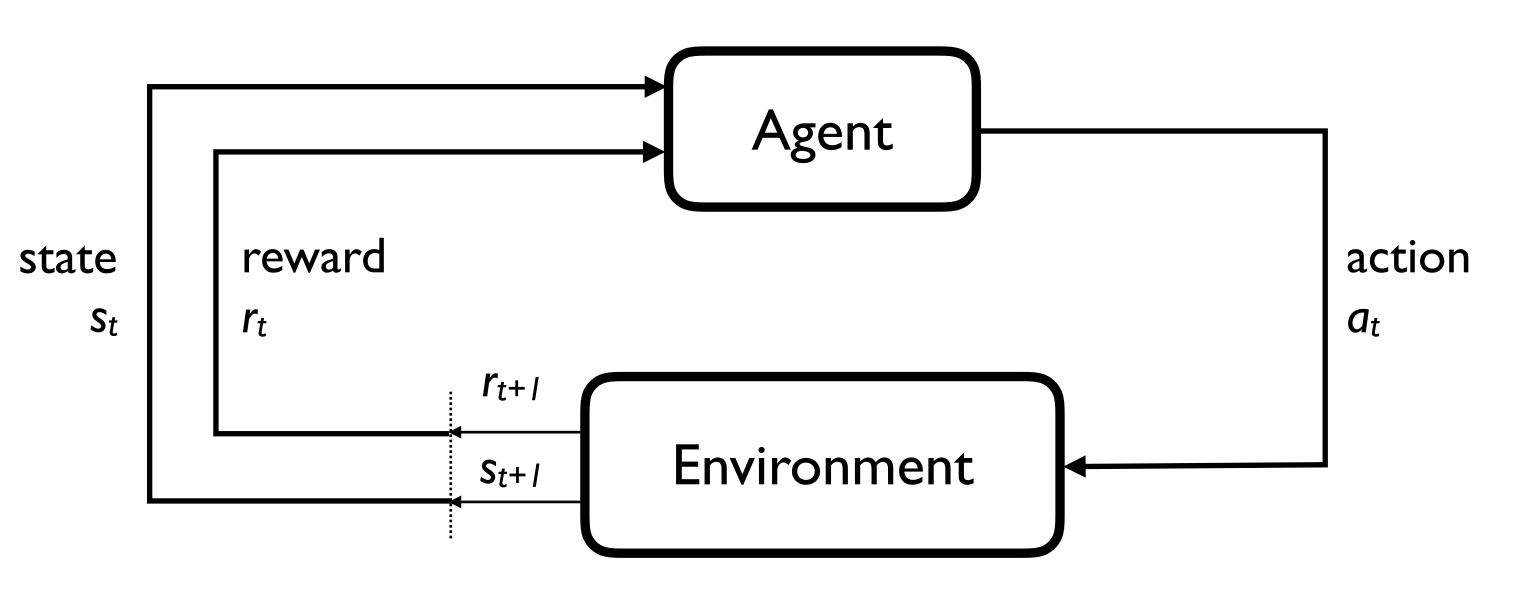
[Google]



Reinforcement Learning







Goal: maximize expected reward



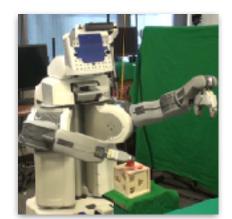


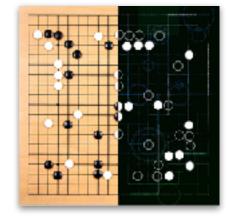


- Basics of Reinforcement Learning
- Model-Free RL
 - Value-Based Methods
 - Policy-Based Methods
- Model-Based RL
 - Guided Policy Search
 - AlphaGo \bullet









Outline

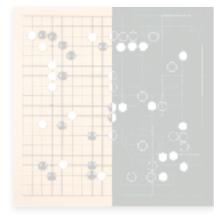


- Basics of Reinforcement Learning
- Model-Free RL
 - Value-Based Methods
 - Policy-Based Methods
- Model-Based RL
 - Guided Policy Search •
 - AlphaGo









Outline



Markov Decision Processes (MDPs)

Markov property: the agent's future is independent of its past history conditioned on the current state.



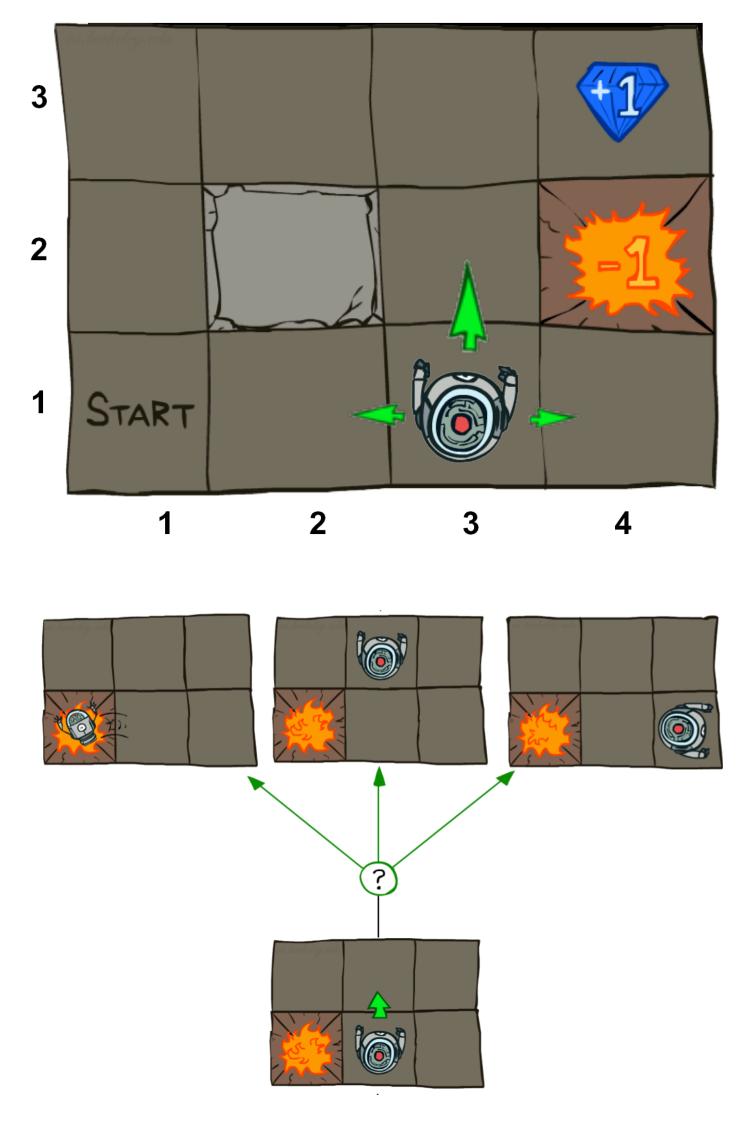
Markov Decision Processes (MDPs)

An MDP is defined by:

- A set of states $s \in \mathcal{S}$
- A set of actions $a \in \mathcal{A}$
- A transition function P(s'|s, a)
- A reward function R(s, a, s')
- A start state s_0
- Maybe a terminal state

[Image credit: CS188 at Berkeley]

,a)

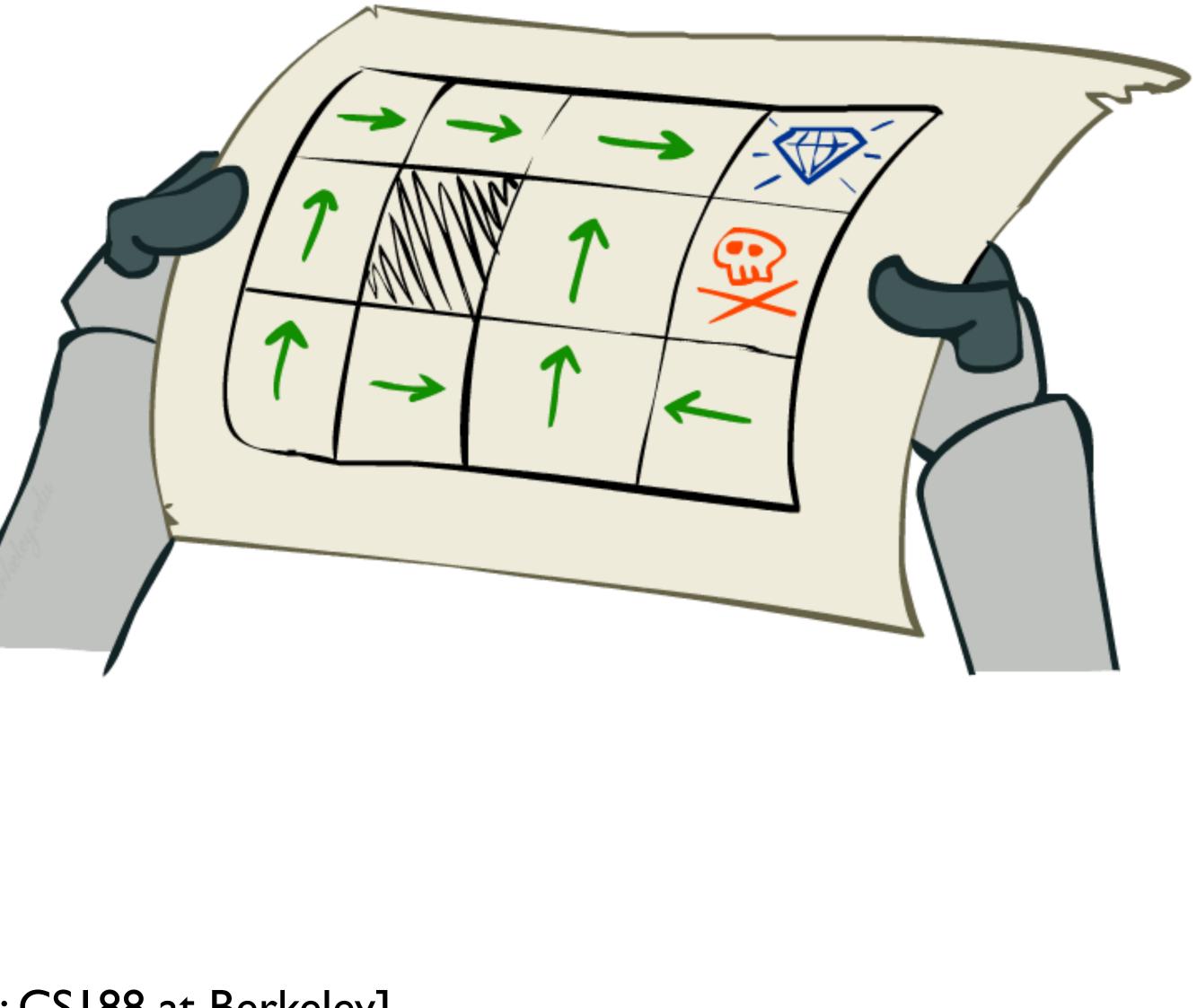




A policy is a mapping $\pi: \mathcal{S} \to \mathcal{A}$ specifying what action to take at each state.

[Image credit: CS188 at Berkeley]

Policies



Utility and Discounting

- Utility of a trajectory $\tau := (s_0, a_0, r_0, s_1, a_1, r_1, ...)$
- It's also reasonable to prefer rewards now to rewards later





Worth Next Step

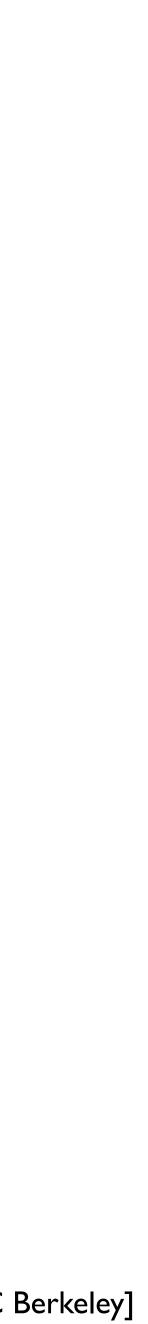
[Image credit: CS188 at Berkeley]

• It's reasonable to maximize the sum of rewards: $U(\tau) = \sum_t r_t$ • One solution: value of rewards decay exponentially $U(\tau) = \sum_{t} \gamma^{t} r_{t}$ $\gamma \in (0,1]$









Values of States

 $V^*(s) = expected$ utility starting in s and acting optimally

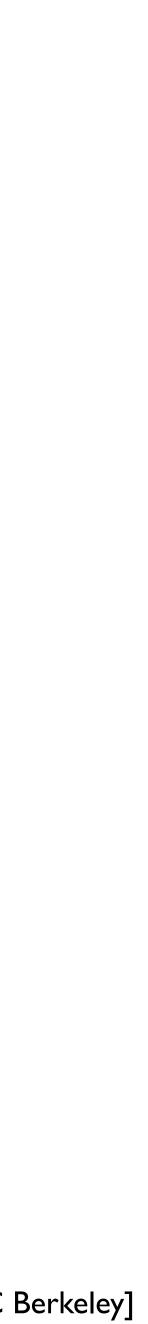
Bellman Equation:

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

Value Iteration:

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} P(s)$$

 $s'|s,a)(R(s,a,s') + \gamma V_k(s'))$



0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00	0.00	0.00
VALUE	ימ אדיידים		

VALUES AFTER 0 ITERATIONS

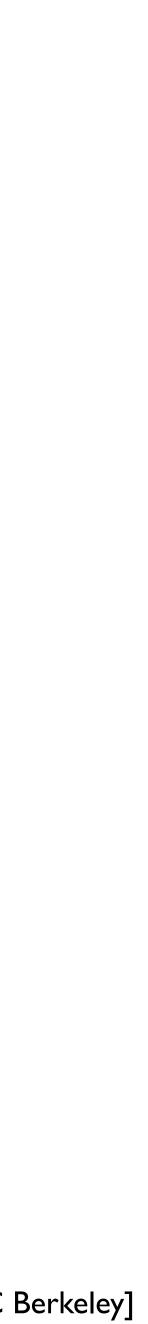
[Image credit: CS188 at Berkeley]

Value Iteration

 $V_0(s) \leftarrow 0$

k = 0

Noise = 0.2Discount = 0.9



 $V_1(s) \leftarrow \max_{a} \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V_0(s'))$

0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00	0.00	0.00
VALUES AFTER O TTERATIONS			

VALUES AFTER U ITERATIONS

[Image credit: CS188 at Berkeley]

Value Iteration

k = 0

Noise = 0.2Discount = 0.9



 $V_1(s) \leftarrow \max_{a} \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V_0(s'))$

0.00	0.00	0.00	1.00	
0.00		0.00	-1.00	
0.00	0.00	0.00	0.00	
VALUE	VALUES AFTER 1 ITERATIONS			

[Image credit: CS188 at Berkeley]

Value Iteration

k = 1

Noise = 0.2Discount = 0.9



 $V_2(s) \leftarrow \max_{a} \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V_1(s'))$

0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00
VALUES AFTER 1 ITERATIONS			

Value Iteration

k = 1

Noise = 0.2Discount = 0.9



 $V_2(s) \leftarrow \max_{a} \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V_1(s'))$

0.00	0.00	0.72	1.00	
0.00		0.00	-1.00	
0.00	0.00	0.00	0.00	
VALUE	VALUES AFTER 2 ITERATIONS			

Value Iteration

k = 2

Noise = 0.2Discount = 0.9



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

0.00	0.52	0.78	1.00
0.00		0.43	-1.00
0.00	0.00	0.00	0.00
VALUE	S AFTER	3 ITERA	FIONS

Value Iteration

k = 3

Noise = 0.2Discount = 0.9



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

0.37	0.66	0.83	1.00	
0.00		0.51	-1.00	
0.00	0.00	0.31	0.00	
VALUE	VALUES AFTER 4 ITERATIONS			

Value Iteration

k = 4

Noise = 0.2Discount = 0.9



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

0.51	0.72	0.84	1.00	
0.27		0.55	-1.00	
0.00	0.22	0.37	0.13	
VALUE	VALUES AFTER 5 ITERATIONS			

Value Iteration

k = 5

Noise = 0.2Discount = 0.9



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

0.59	0.73	0.85	1.00
0.41		0.57	-1.00
0.21	0.31	0.43	0.19
VALUES AFTER 6 ITERATIONS			

Value Iteration

k = 6

Noise = 0.2Discount = 0.9



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

0.62	0.74	0.85	1.00	
0.50		0.57	-1.00	
0.34	0.36	0.45	0.24	
VALUE	VALUES AFTER 7 ITERATIONS			

Value Iteration

k = 7

Noise = 0.2Discount = 0.9



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

0.63	0.74	0.85	1.00
0.53		0.57	-1.00
0.42	0.39	0.46	0.26
VALUE	S AFTER	8 ITERA	FIONS

Value Iteration

k = 8

Noise = 0.2Discount = 0.9



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

0.64	0.74	0.85	1.00	
0.55		0.57	-1.00	
0.46	0.40	0.47	0.27	
VALUE	VALUES AFTER 9 ITERATIONS			

Value Iteration

k = 9

Noise = 0.2Discount = 0.9



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

0.64	0.74	0.85	1.00
0.56		0.57	-1.00
0.48	0.41	0.47	0.27
VALUES AFTER 10 ITERATIONS			

Value Iteration

k = 10

Noise = 0.2Discount = 0.9



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

0.64	0.74	0.85	1.00	
0.56		0.57	-1.00	
0.48	0.42	0.47	0.27	
VALUES AFTER 11 ITERATIONS				

Value Iteration

k = ||

Noise = 0.2Discount = 0.9



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

0.64	0.74	0.85	1.00
0.57		0.57	-1.00
0.49	0.42	0.47	0.28
VALUES AFTER 12 ITERATIONS			

Value Iteration

k = 12

Noise = 0.2Discount = 0.9



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

0.64	0.74	0.85	1.00
0.57		0.57	-1.00
0.49	0.43	0.48	0.28
VALUES AFTER 100 ITERATIONS			

Value Iteration

k = 100

[Image credit: CS188 at Berkeley]

Noise = 0.2Discount = 0.9



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

Bellman Equation:

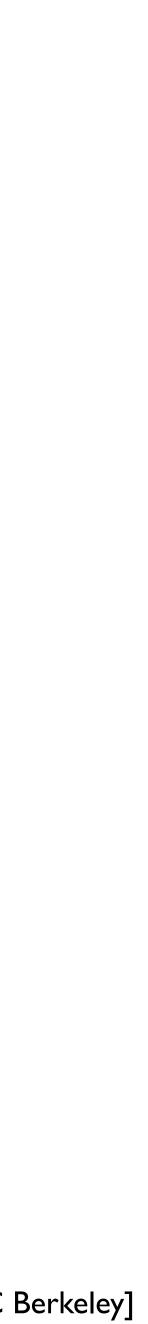
$$Q^*(s,a) = \sum_{s'} P(s'|s,a) (R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

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

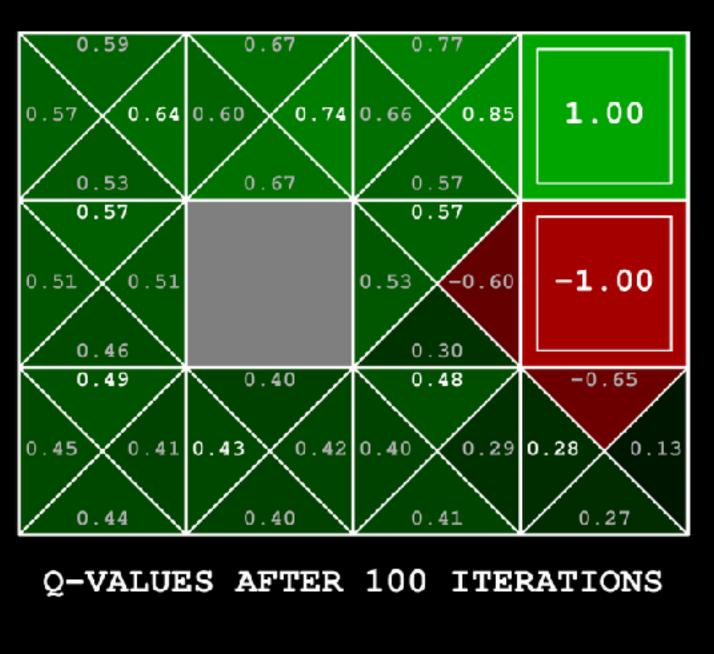
Q-Values

 $|s, a)(R(s, a, s') + \gamma \max_{a'} Q_k(s', a'))$



Q-Value Iteration

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





[Image credit: CS188 at Berkeley]

Noise = 0.2Discount = 0.9

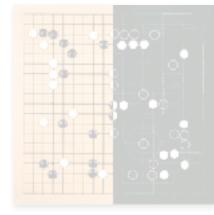


- Basics of Reinforcement Learning
- Model-Free RL
 - Value-Based Methods
 - Policy-Based Methods
- Model-Based RL
 - **Guided Policy Search** •
 - AlphaGo









Outline



Want: $Q_{k+1}(s,a) \leftarrow \sum P(s'|s,$ But no access to P(s'|s,a)

Q-Learning: Collect samples and learn Q(s, a) values as you go.

- Record sample (s, a, s', r)
- Consider your old estimate: Q(
- Consider your new sample estimation
- Incorporate the new estimate into a running average

 $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)[sample]$

Q-Learning

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

$$(s, a)$$

hate: $sample = r + \gamma \max_{a'} Q(s', a')$



each state-action pair infinitely often.

Q-Learning

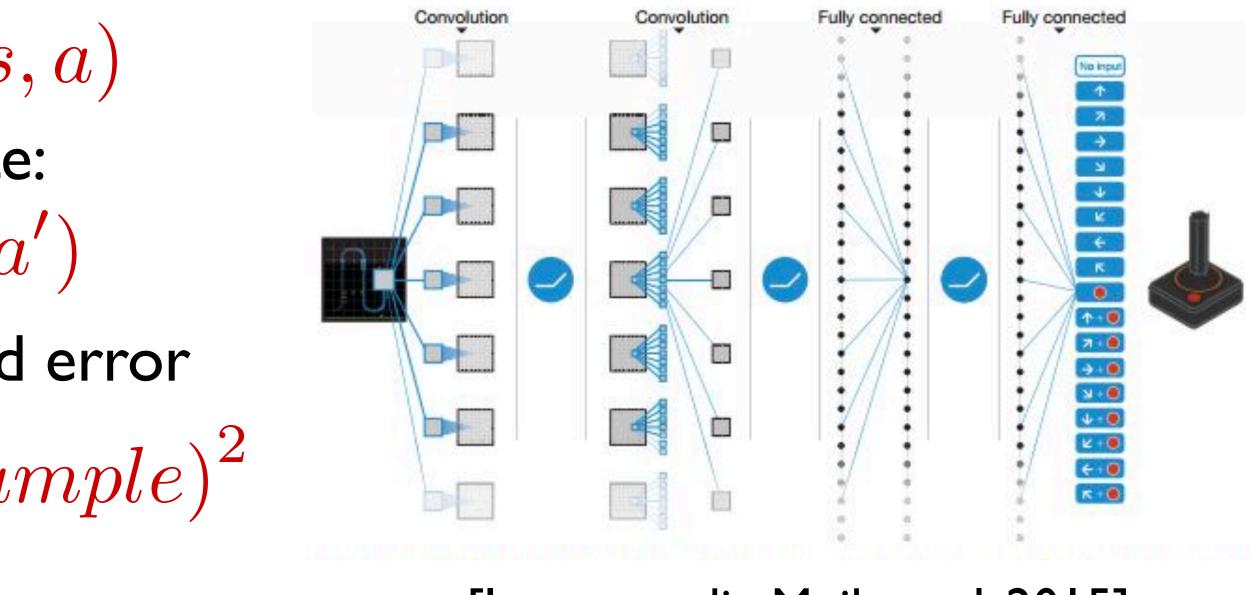
Theorem: Q-Learning converges to the optimal value, as long as you visit



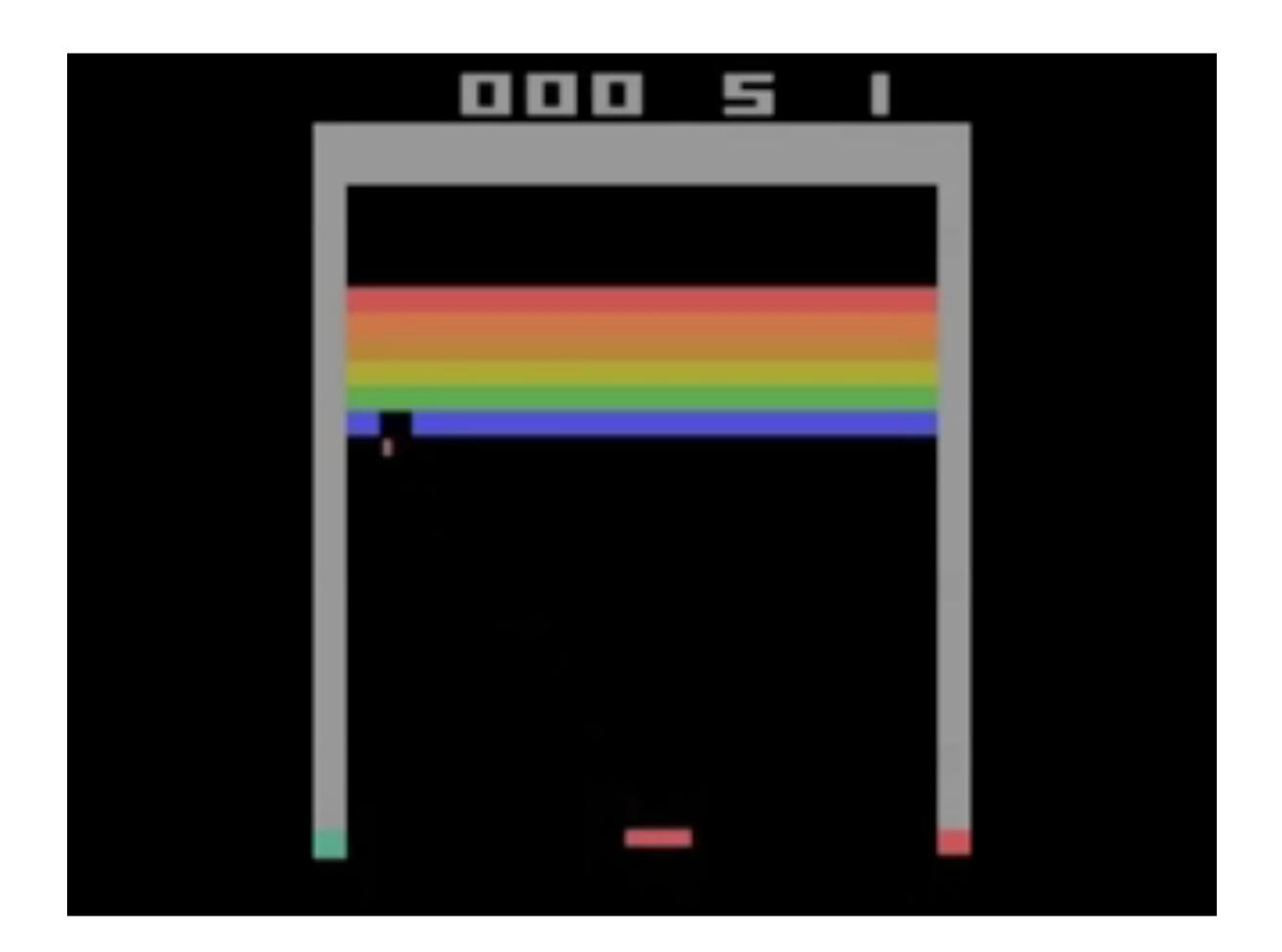
Problem: For very large state spaces, cannot store Q-values explicitly Solution: Use function approximation (e.g. deep neural networks!)

- Receive a sample (s, a, s', r)
- Consider your old estimate: $Q_{\theta}(s, a)$
- Consider your new sample estimate: sample = $r + \gamma \max_{a'} Q_{\theta}(s', a')$
- Perform gradient descent on squared error

 $\theta \leftarrow \theta - \alpha \nabla_{\theta} \left(Q_{\theta}(s, a) - sample \right)^2$



[Image credit: Mnih et al, 2015]

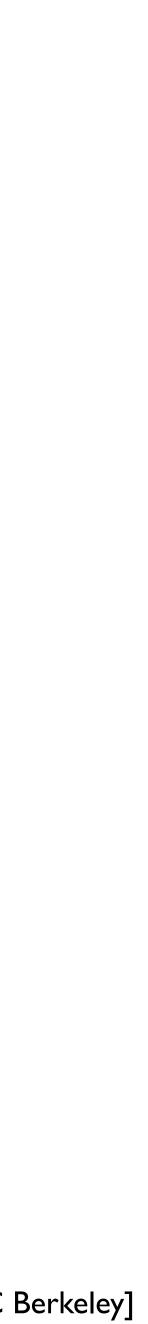


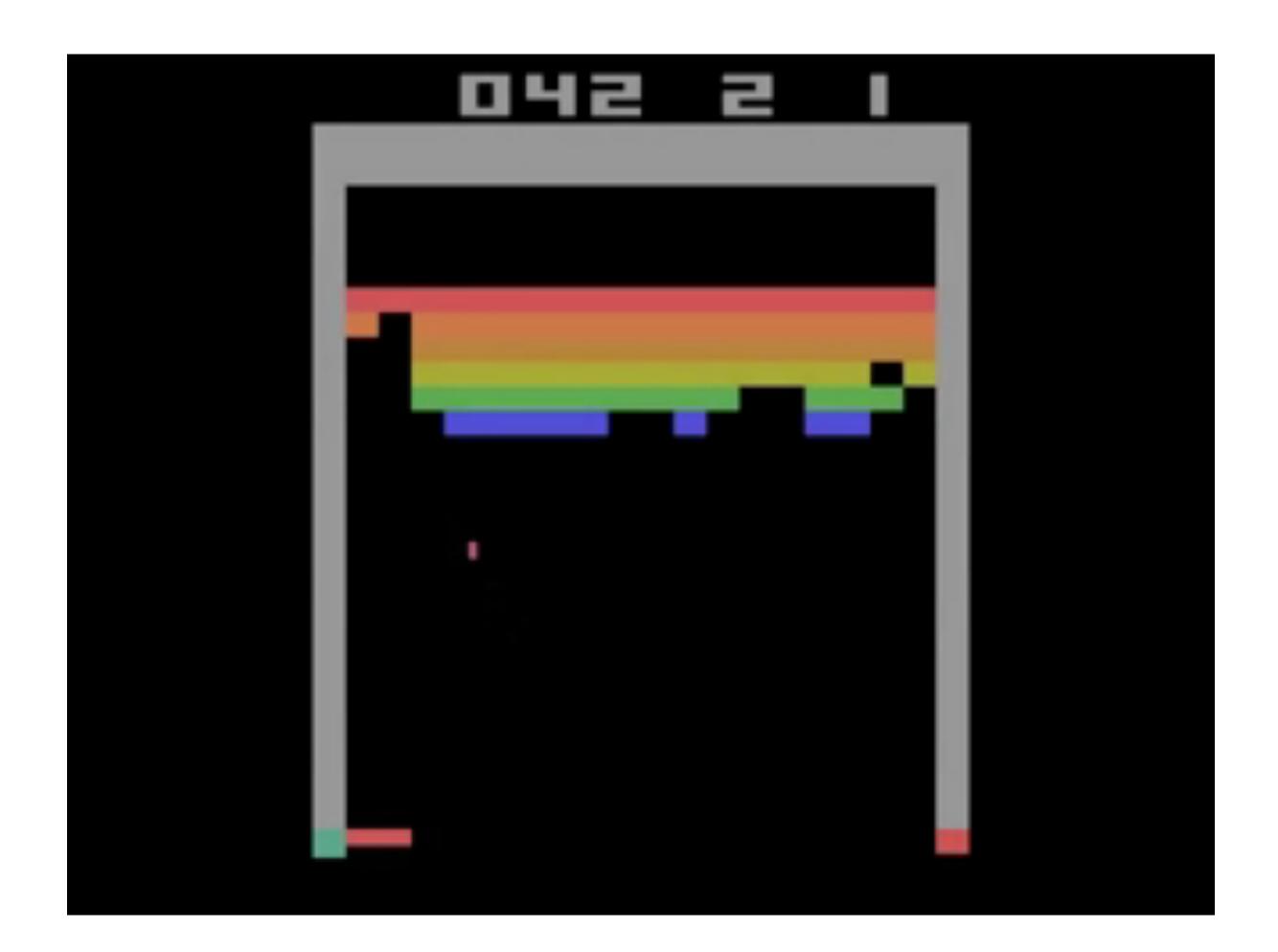
[Mnih et al, NIPS 2013 / Nature 2015]



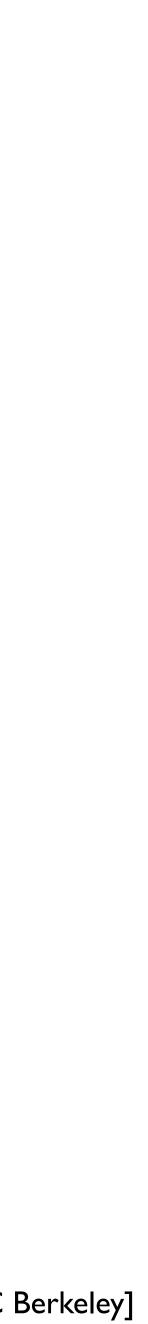


[Mnih et al, NIPS 2013 / Nature 2015]



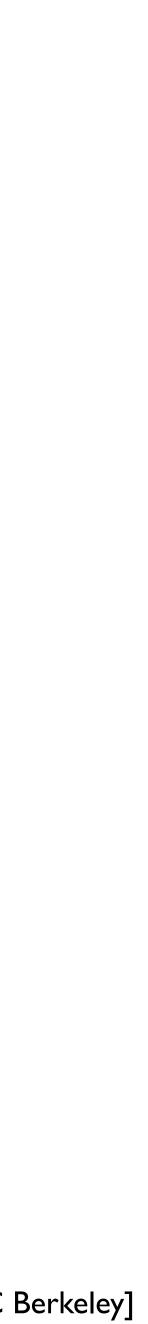


[Mnih et al, NIPS 2013 / Nature 2015]



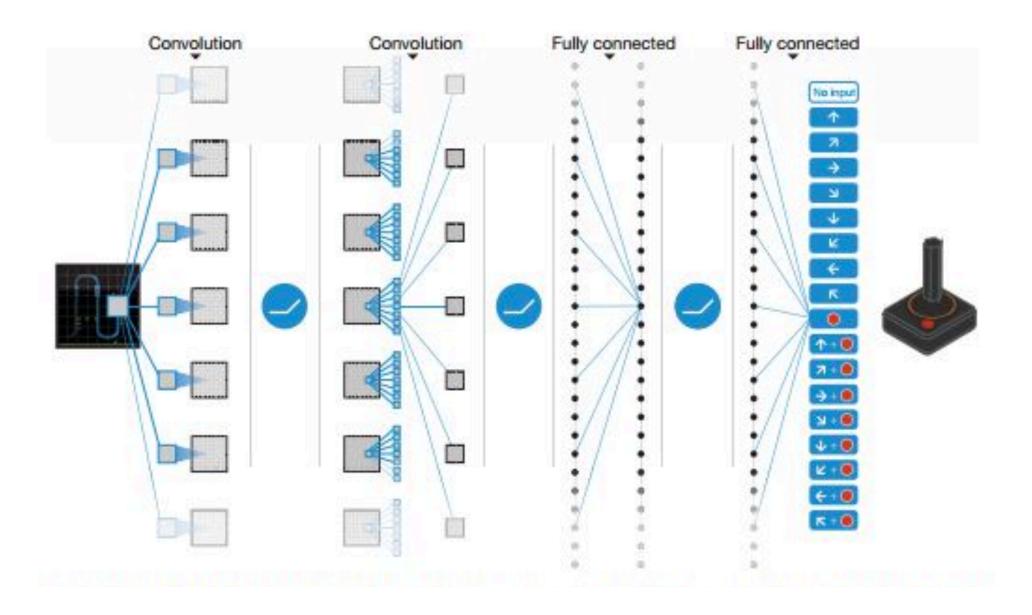


[Mnih et al, NIPS 2013 / Nature 2015]

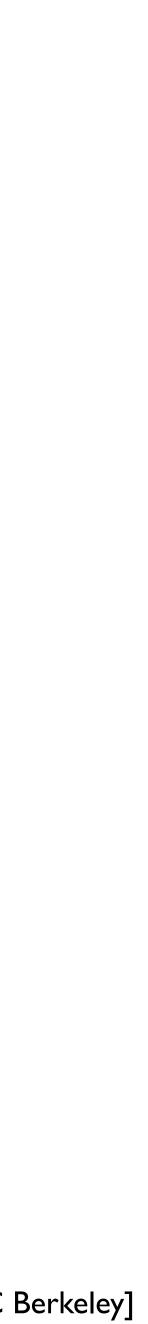


Caveats

- No (known) guarantees of performance improvement when using function approximation
- for every possible action

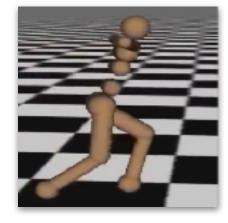


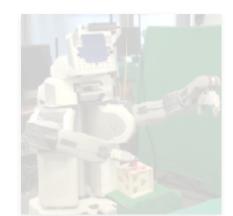
Hard to work with continuous actions since we can't have an output

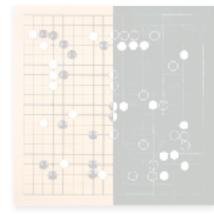


- Basics of Reinforcement Learning
- Model-Free RL
 - Value-Based Methods
 - Policy-Based Methods
- Model-Based RL
 - **Guided Policy Search** •
 - AlphaGo









Outline



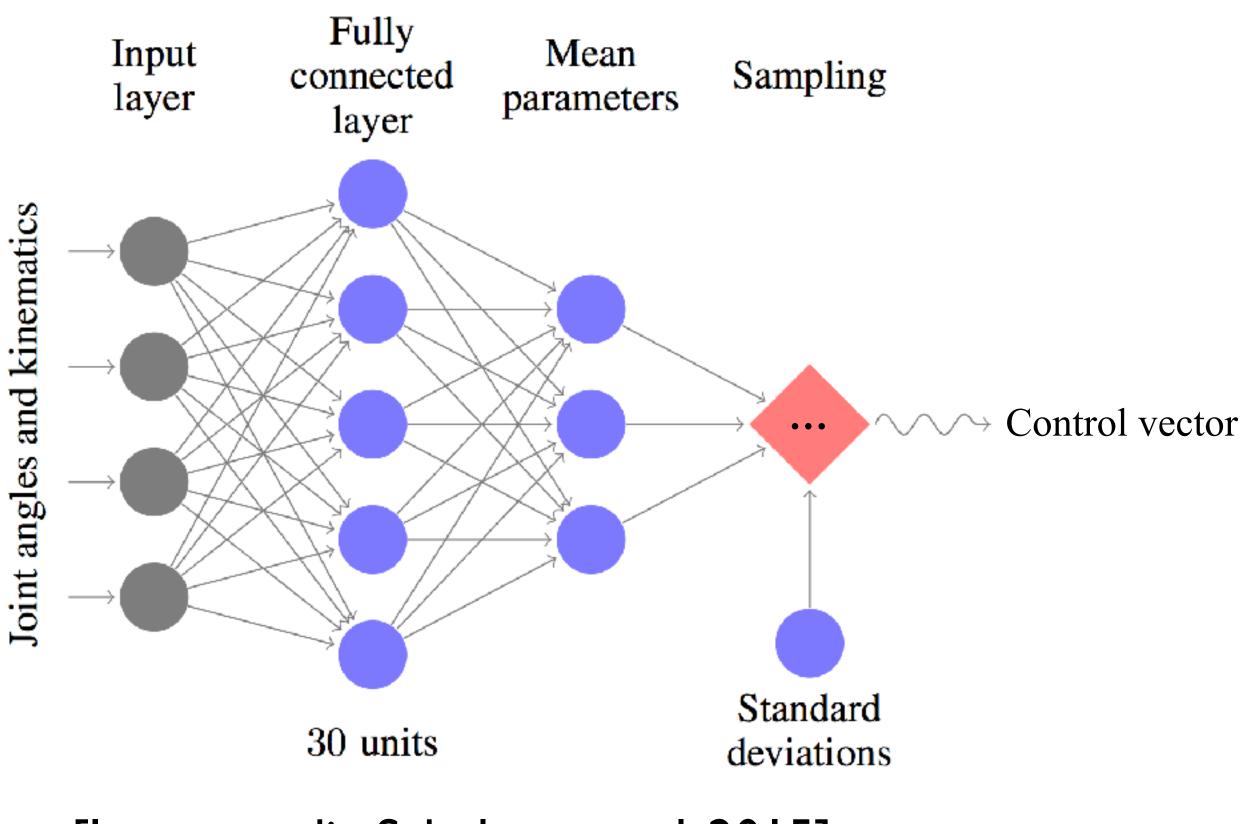
Objective: $\max_{\theta} U(\theta)$ where $U(\theta) := \mathbb{E}_{\tau|\pi_{\theta}}[U(\tau)] = \mathbb{E}_{\pi_{\theta}}[\sum_{t} \gamma^{t} r_{t}]$

Policy Optimization



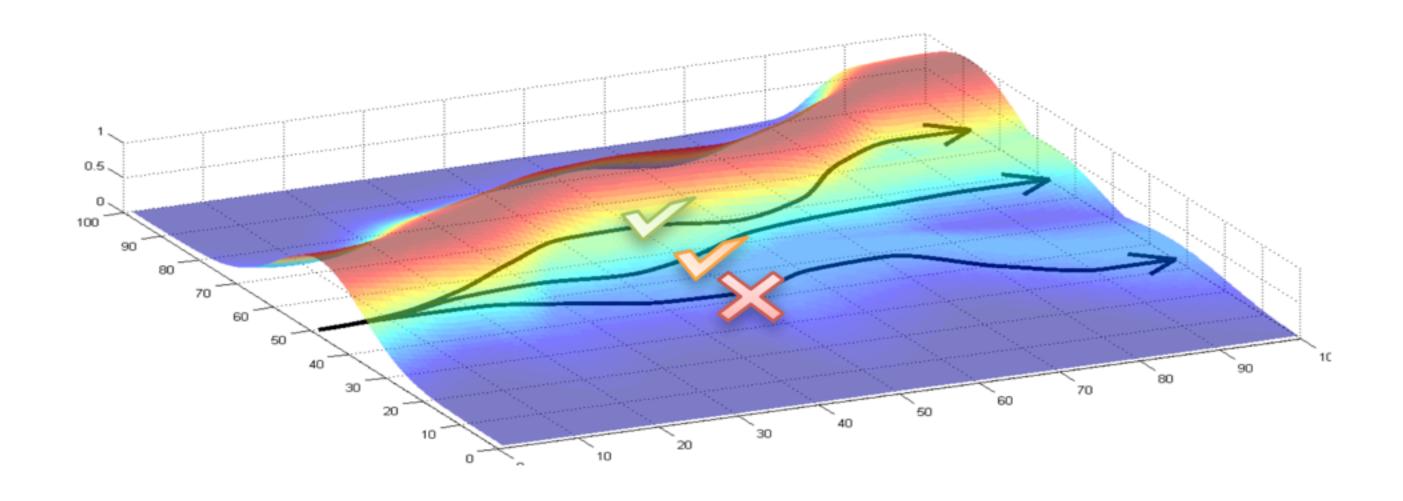
Policy Gradient

Assume a stochastic policy $\pi_{\theta}: S \times \mathcal{A} \to [0, 1]$



[Image credit: Schulman et al, 2015]





Policy Gradient

 $\nabla_{\theta} U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_{\theta} \log \pi_{\theta} (a_t^i | s_t^i) \sum_{t'=t}^{T} \gamma^{t'} r_{t'}^i$



Improving efficiency using baselines:

$$\nabla_{\theta} U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \left(\sum_{t'=t}^{T} \gamma^{t'} r_{t'}^i - V^{\pi_{\theta}}(s_t^i) \right)$$

 $V^{\pi_{\theta}}(s)$: average performance starting in state s and following policy π_{θ}

Policy Gradient



Choosing step size

Given the gradient, how to choose a step size?

- Supervised learning
 - Step too large: next update will correct for it
- Reinforcement Learning
 - Step too large: terrible policy
 - Next mini batch will be collected under this terrible policy!
 - Unclear how to recover



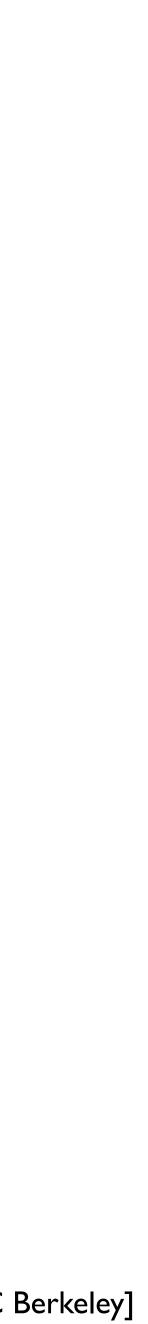


Trust Region Methods

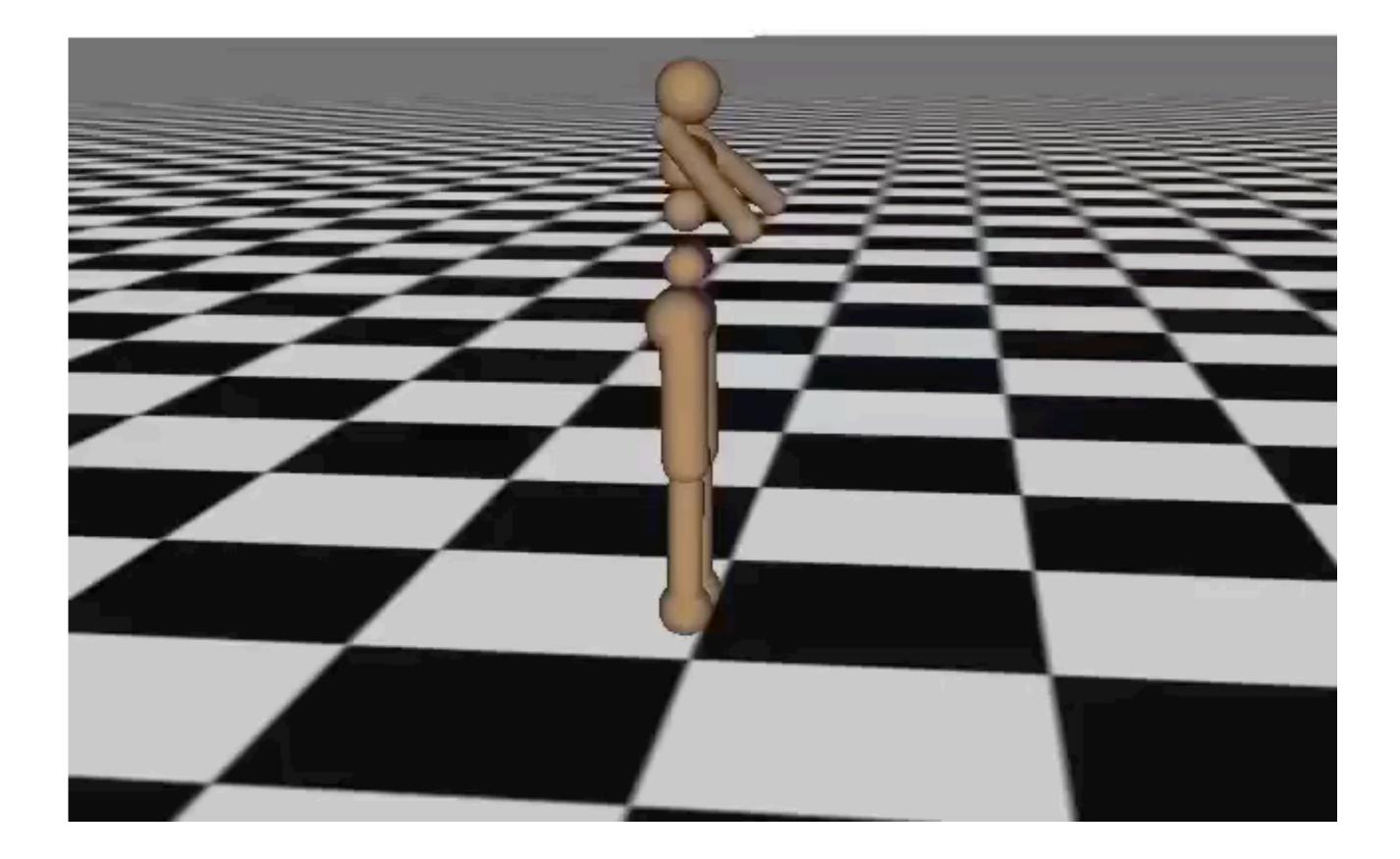
policy can change.

Gives rise to natural policy gradient and TRPO al 2015]

- Formulate as constrained optimization problem, controlling how much the
- [Kakade 2002; Bagnell & Schneider 2003; Peters & Schaal 2003; Schulman et

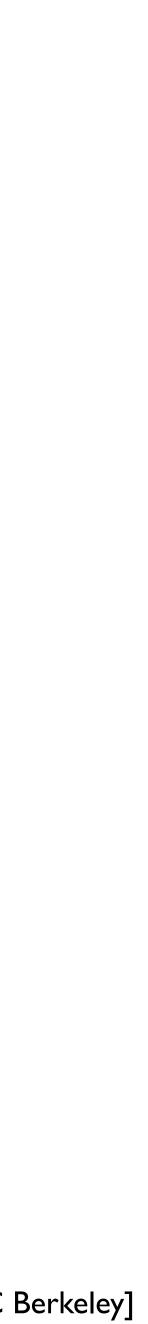


Trust Region Methods: Success Stories Iteration 0

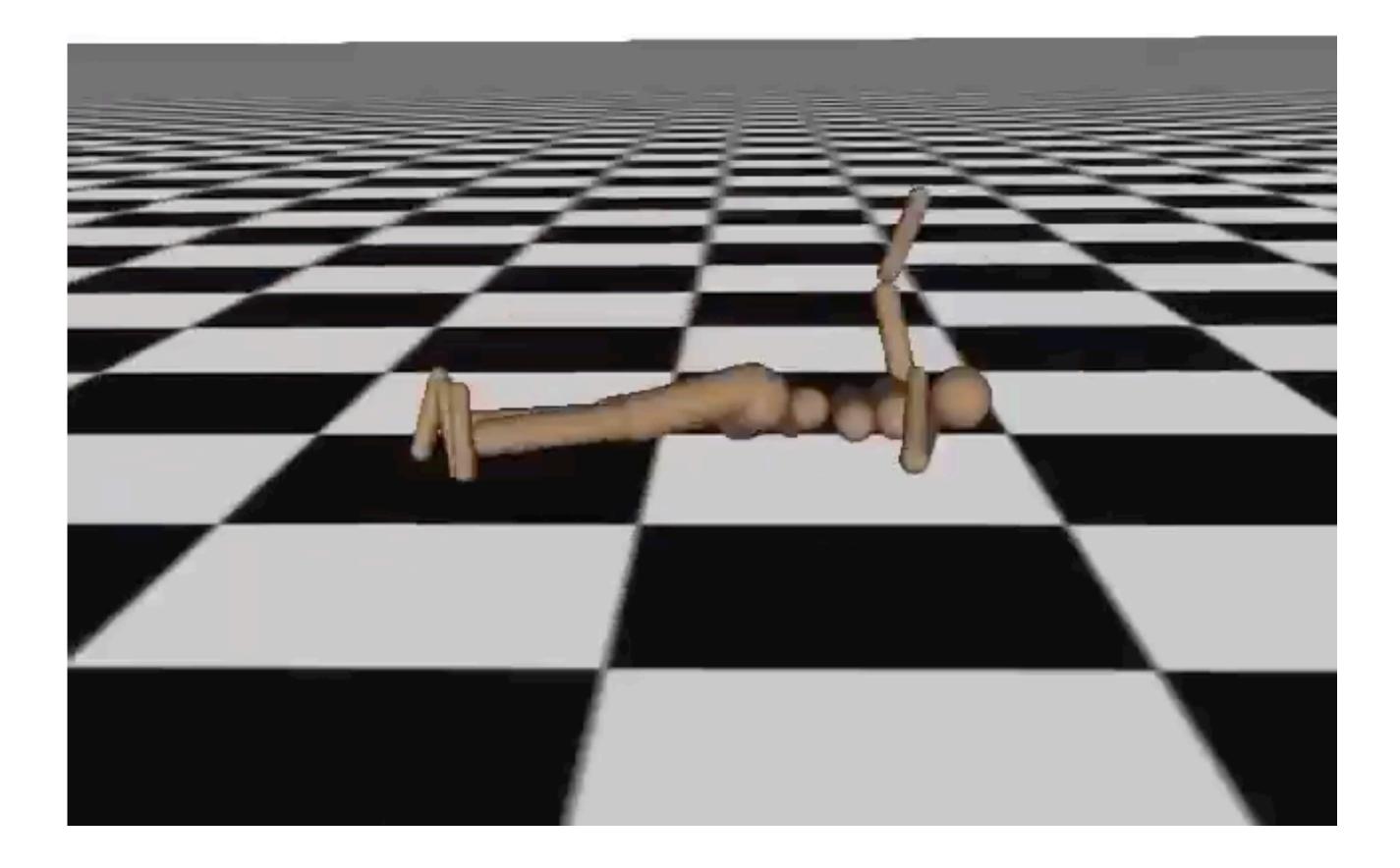




[Schulman et al, 2015]

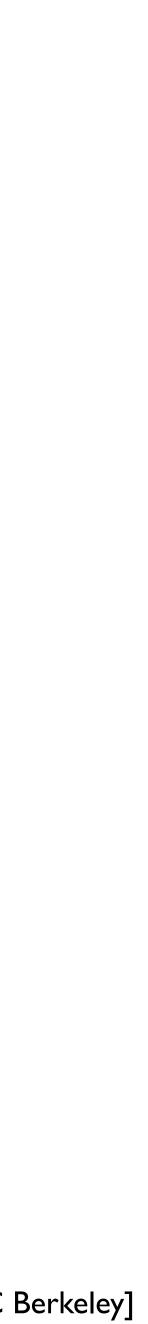


Trust Region Methods: Success Stories Iteration 0

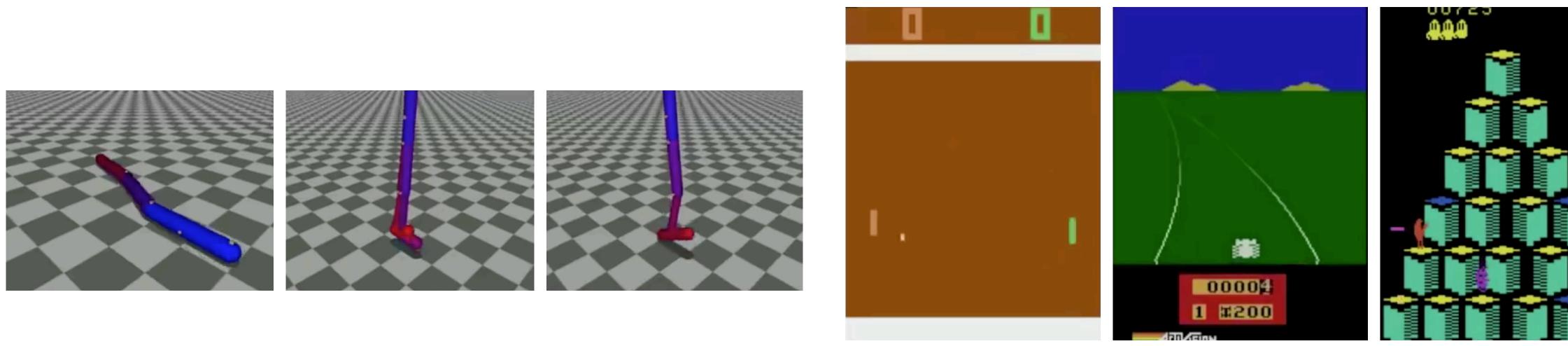


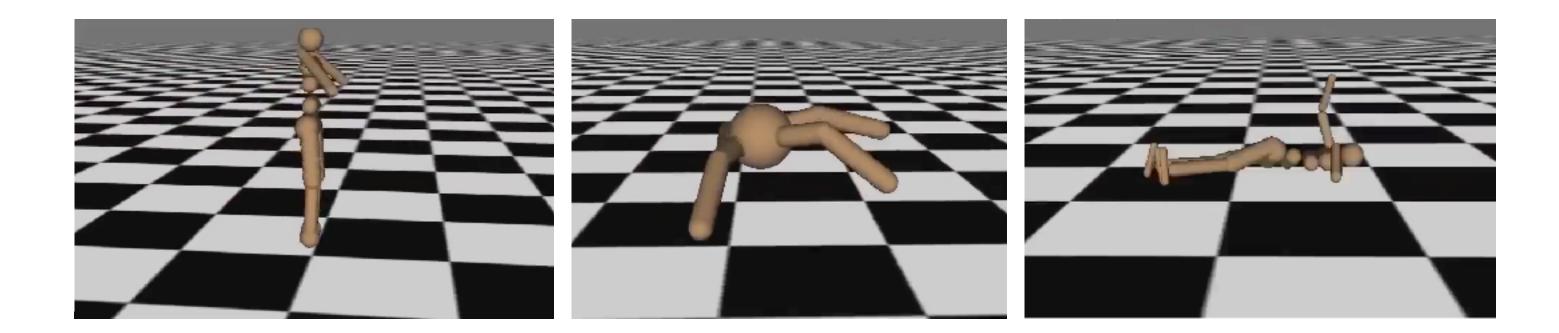


[Schulman et al, 2015]



Trust Region Methods: Success Stories







[Schulman et al, 2015]



Objective: $\max_{\theta} U(\theta)$ where $U(\theta) := \mathbb{E}_{\tau \mid \pi_{\theta}}[U(\tau)] = \mathbb{E}_{\pi_{\theta}}[\sum_{t} \gamma^{t} r_{t}]$

Black-Box Optimization?

Policy Optimization



Cross-Entropy Method Start with initial parameter distribution, $P_{\mu^{(1)}}(\theta)$ say $\mathcal{N}(0,I)$

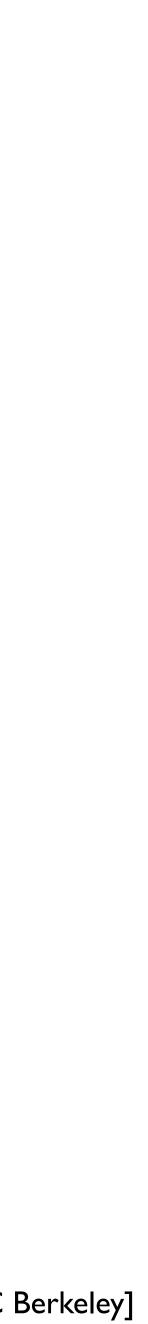
for iter i = 1, 2, ...for population member e = 1, 2, ...Sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$ Collect trajectories under $\pi_{\theta(e)}$: τ_1, \ldots, τ_N

end for $\mu^{(i+1)} \leftarrow \arg \max_{\mu} \sum_{\bar{e}} \log P_{\mu}(\theta^{\bar{e}})$

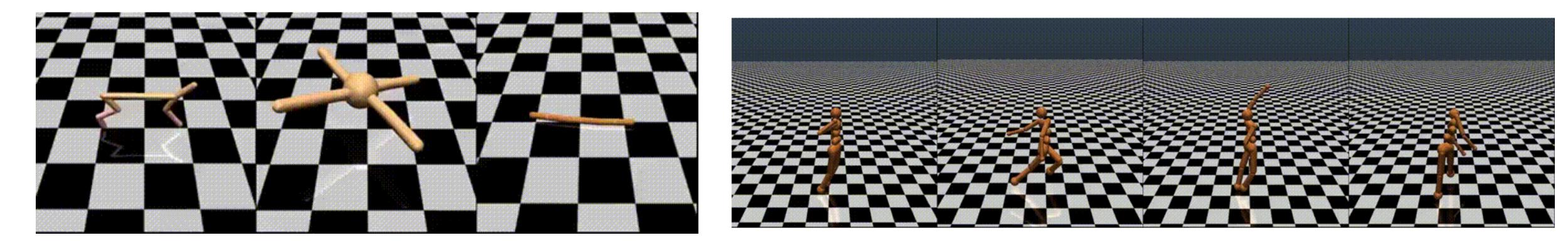
where \bar{e} indexes over top p% performance

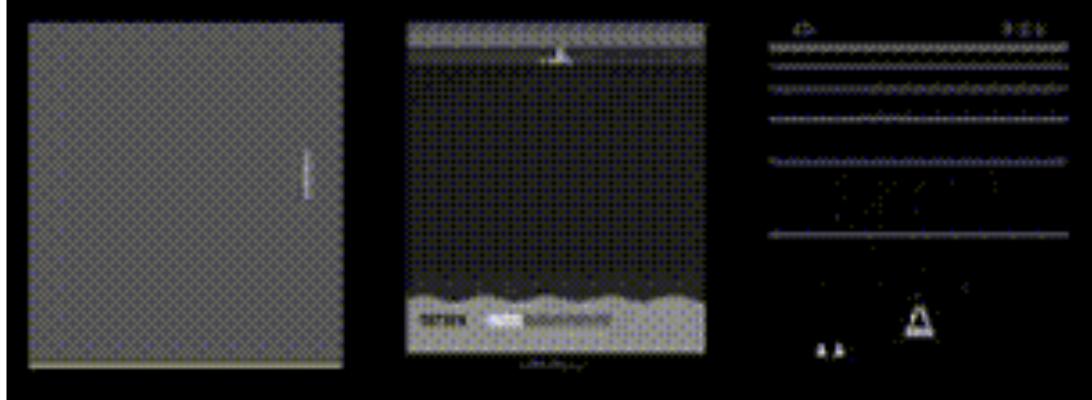
end for

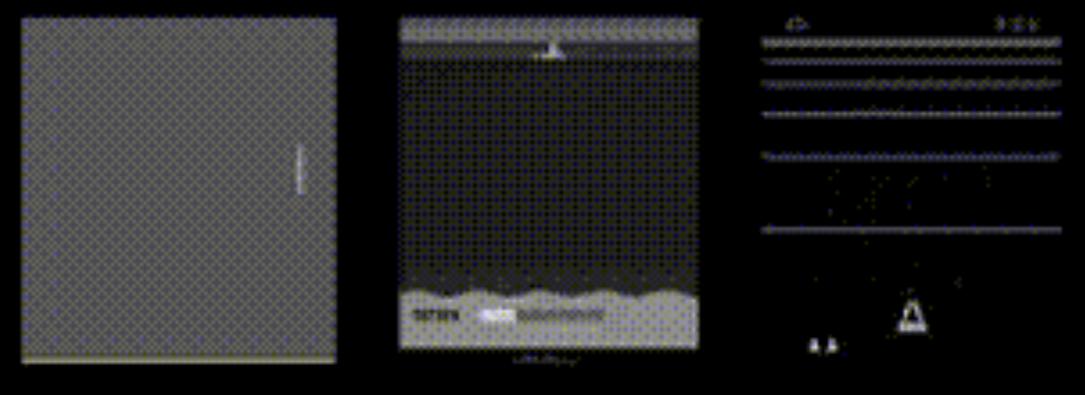
Store $(\theta^{(e)}, \hat{U}(\theta^{(e)}))$ where $\hat{U}(\theta^{(e)}) := \frac{1}{N} \sum_{i=1}^{N} U(\tau_i)$



Evolution Strategies: Success Stories







[Salimans et al, 2017]

Policy Gradient

More sample efficient ③

Trickier to parallelize 🔅

Requires differentiable policies 🔅

vs. Evolution Strategies

Less sample efficient 😔

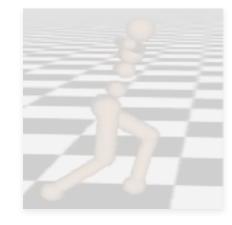
Easier to parallelize ③

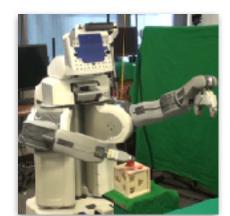
Can work with nondifferentiable policies 😳

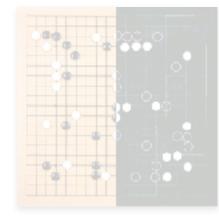


- Basics of Reinforcement Learning
- Model-Free RL
 - Value-Based Methods
 - Policy-Based Methods
- Model-Based RL
 - Guided Policy Search
 - AlphaGo









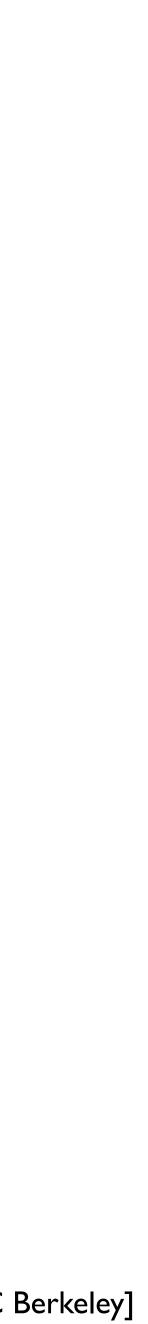
Outline



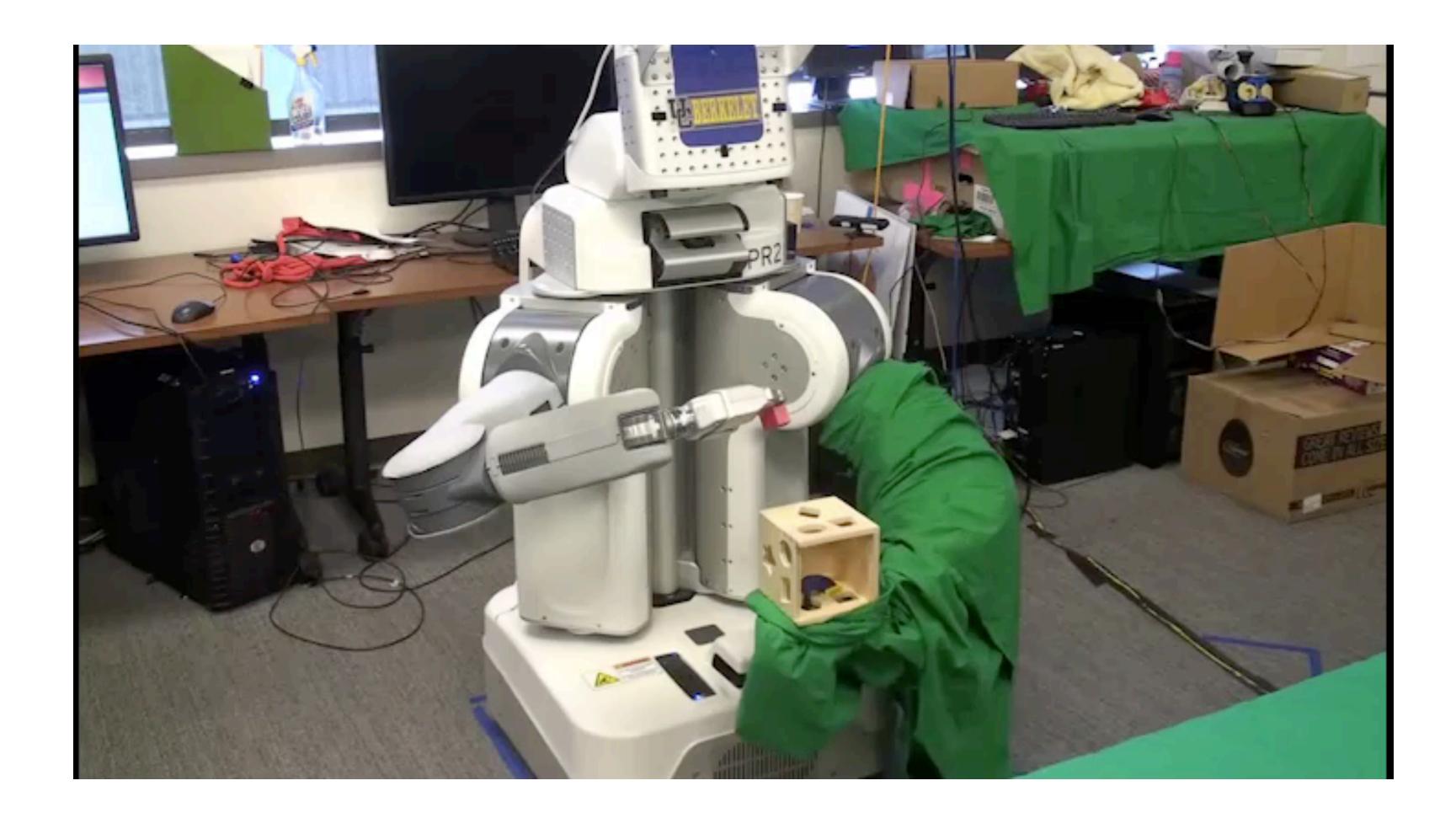
Guided Policy Search

Key idea:

- During training, allow robot to try from the exact same starting state several times
- For such consistent scenarios, iLQR from optimal control theory can be leveraged to help find a solution
- Train a neural network to match the iLQR controllers which generalizes to new situations



Guided Policy Search: Learning on Real Robot

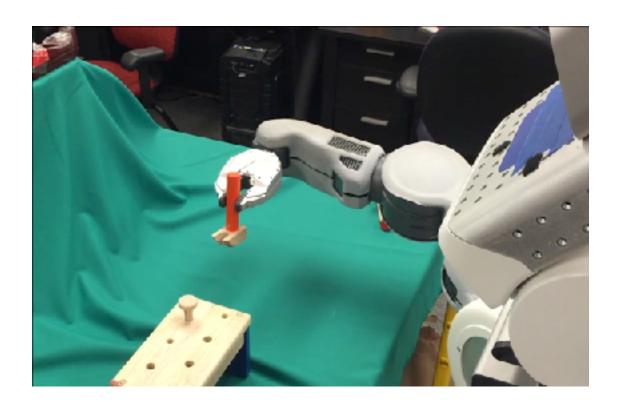




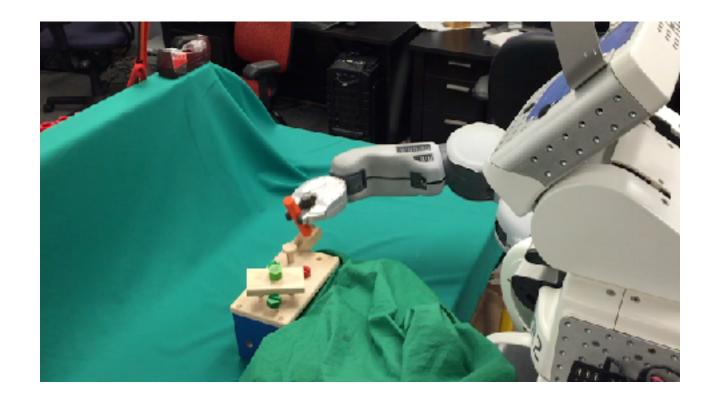
[Levine et al, 2015]



Guided Policy Search: Success Stories

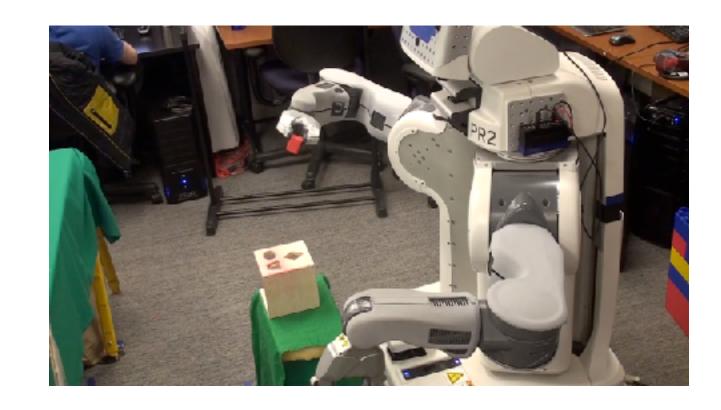








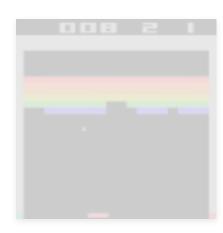




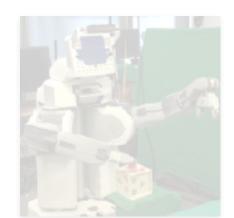
[Levine et al, 2015]

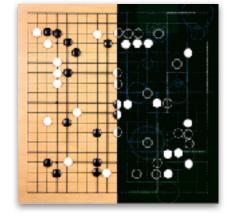


- Basics of Reinforcement Learning
- Model-Free RL
 - Value-Based Methods
 - Policy-Based Methods
- Model-Based RL
 - **Guided Policy Search** •
 - AlphaGo \bullet





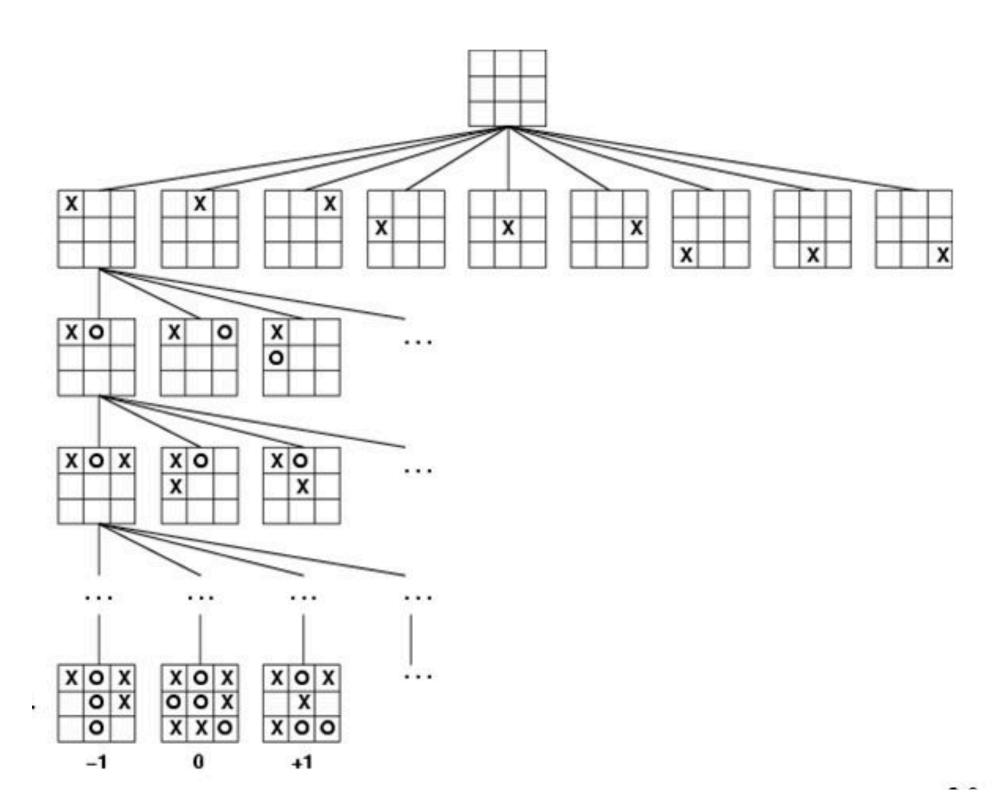




Outline

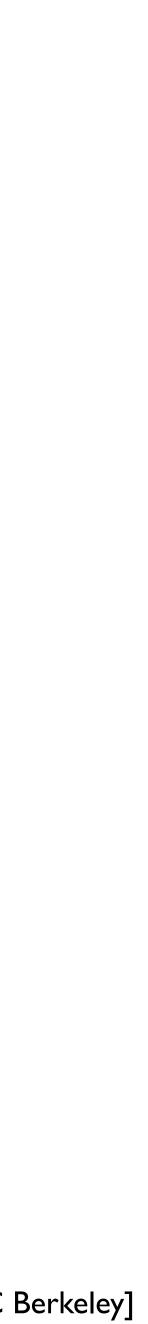


Background: Monte-Carlo Tree Search

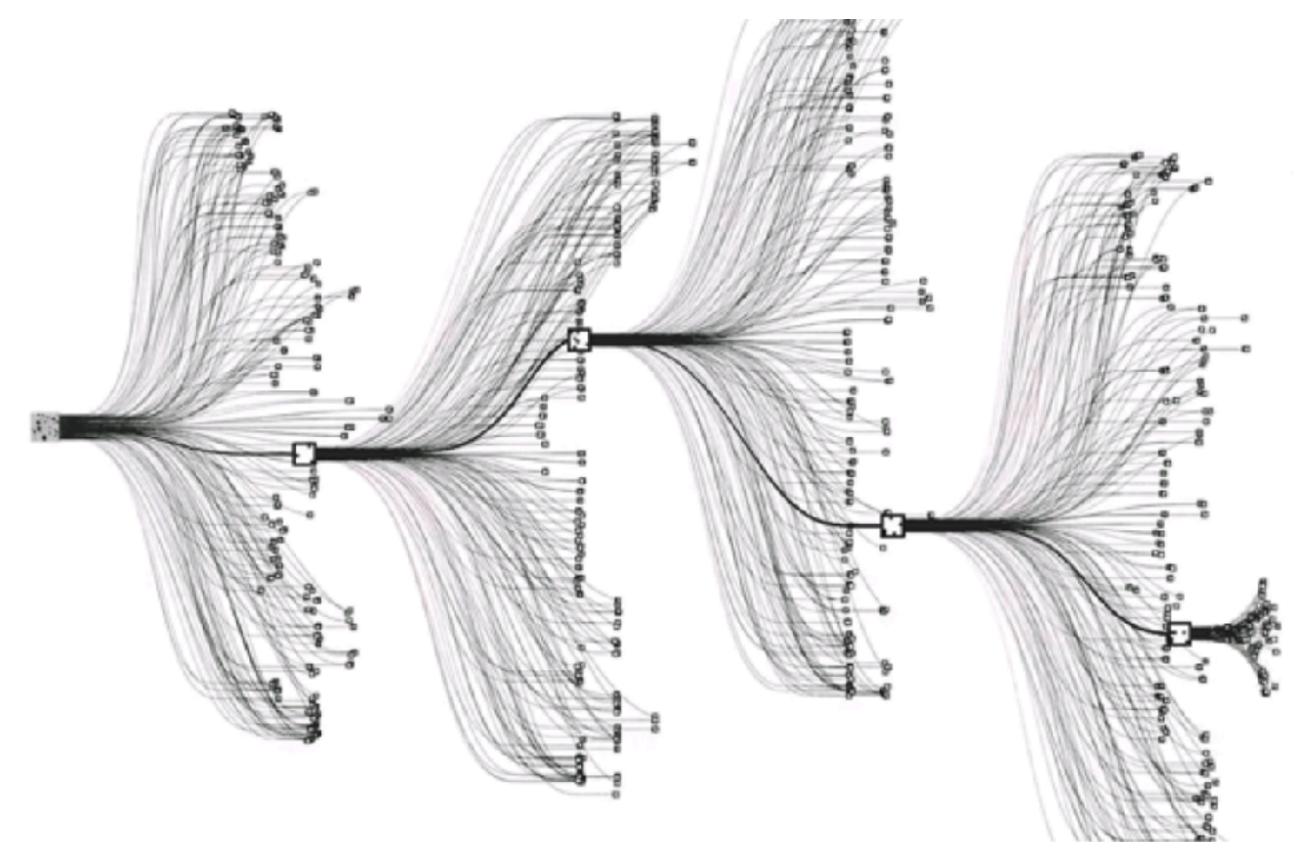


Partial game tree for tic-tac-toe [Image credit:Wikimedia]

Number of states: less than 10⁵



Background: Monte-Carlo Tree Search



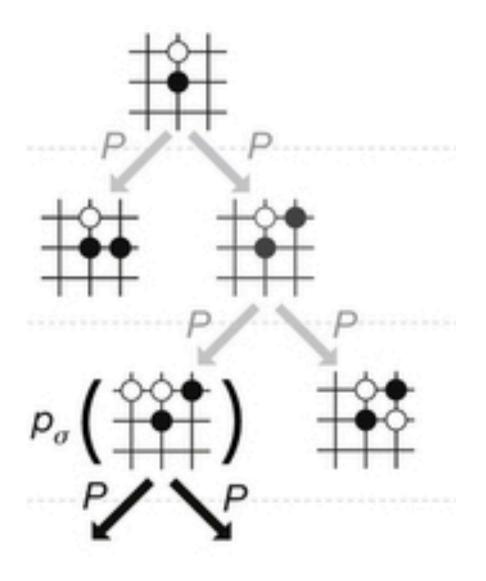
Small snapshot of game tree for Go [Image credit: Deepmind]

Number of states: more than 10¹⁷⁰!



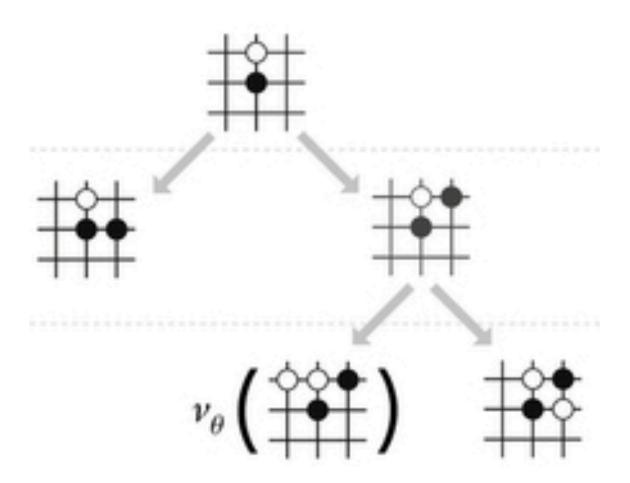
AlphaGo

Reducing breadth



[Image credit: Silver et al, 2016]

Reducing depth

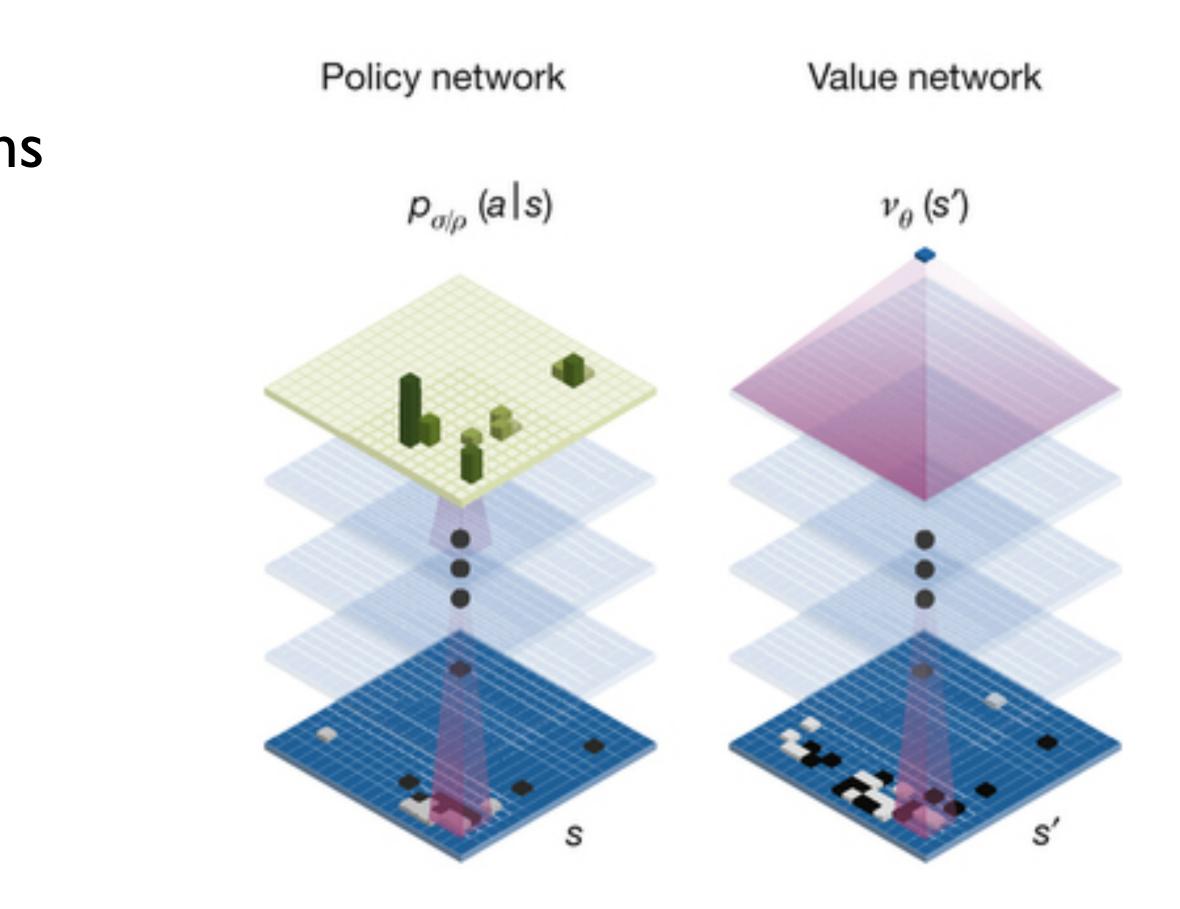




AlphaGo

Learning the policy & value functions

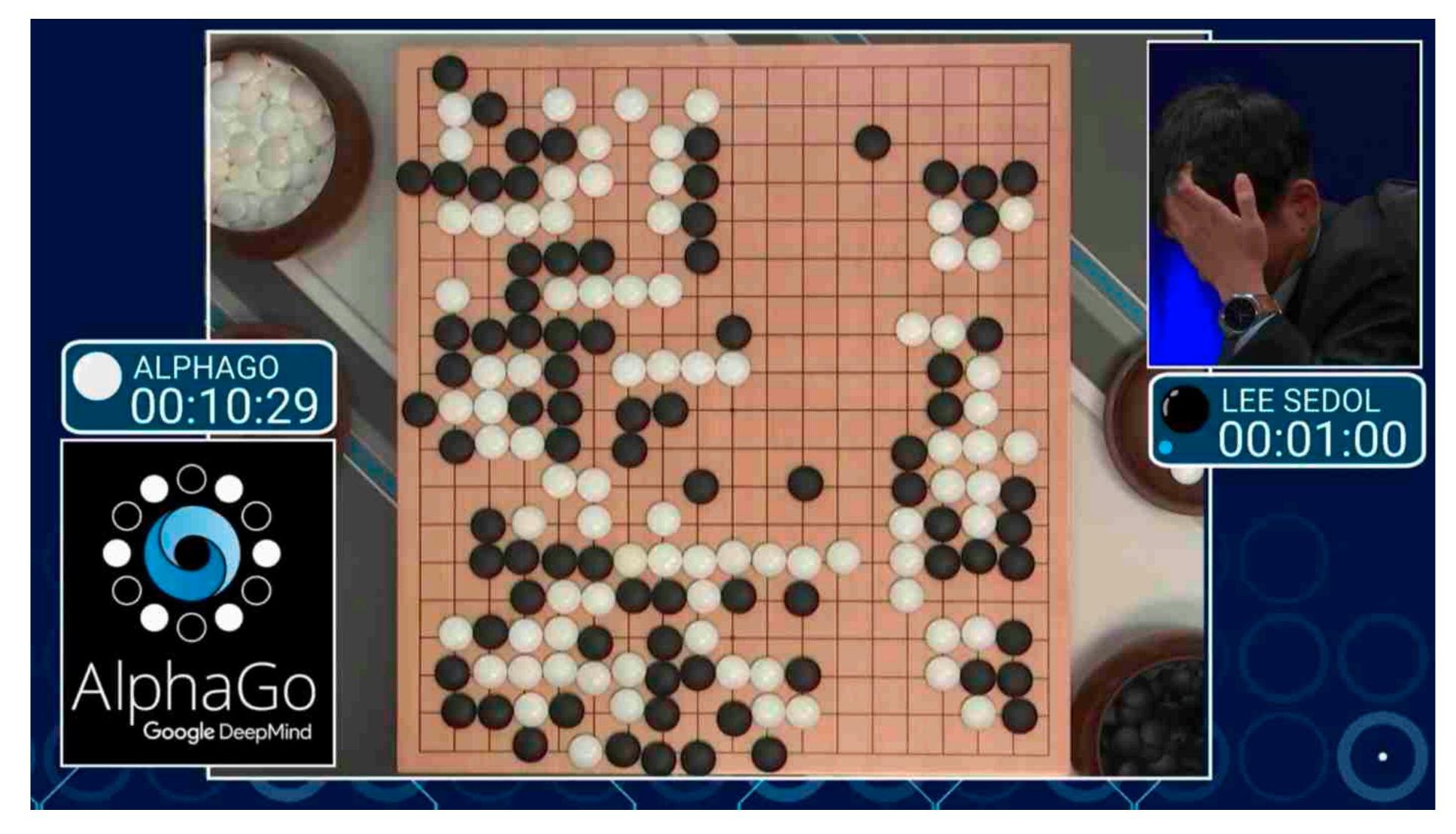
- Supervised pre-training
- Self-play



[Image credit: Silver et al, 2016]



AlphaGo: Success Stories

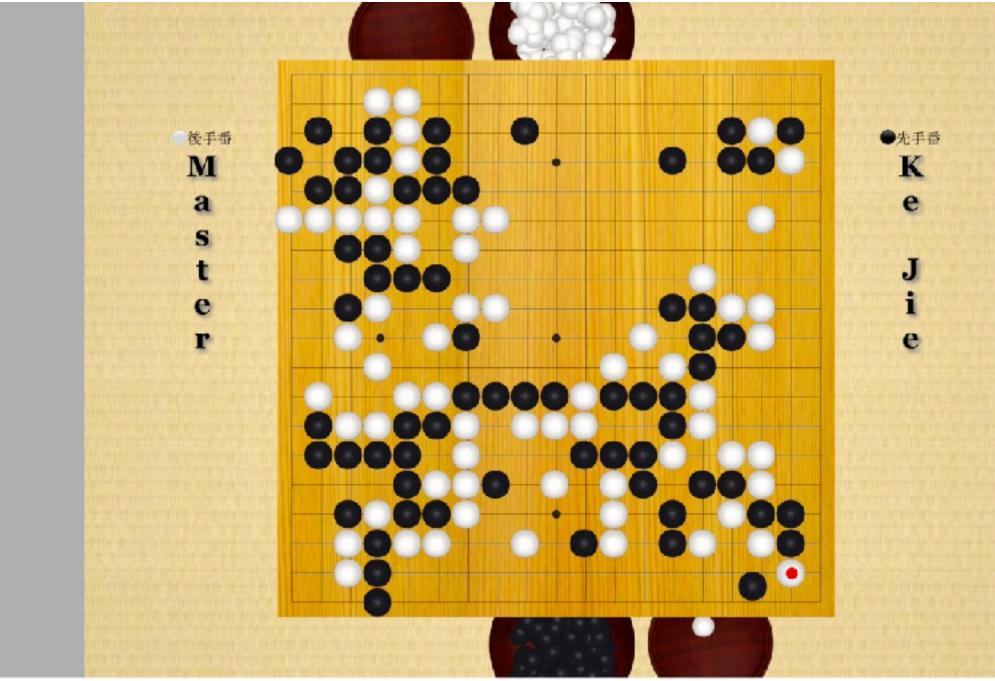




[Silver et al, 2016]



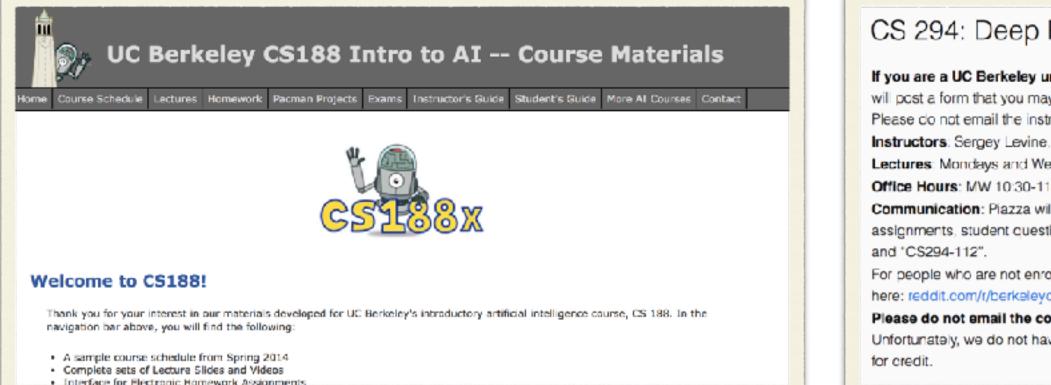
AlphaGo: Success Stories







[Silver et al, 2016]



CS 294: Deep Reinforcement Learning, Spring 2017

If you are a UC Berkeley undergraduate student looking to enroll in the fall 2017 offering of this course: We will post a form that you may fill out to provide us with some information about your background during the summer. Please do not email the instructors about enrollment: the form will be used to collect all information we need. Instructors: Sergey Levine, John Schulman, Chelsea Finn Lectures: Mondays and Wednesdays, 9:00am-10:30am in 306 Soda Hall. Office Hours: MW 10:30-11:30, by appointment (see signup sheet on Piazza) Communication: Piazza will be used for announcements, general questions and discussions, clarifications about assignments, student questions to each other, and so on. To sign up, go to Plazza and sign up with "UC Berkeley"

For people who are not enrolled, but interested in following and discussing the course, there is a subreddit forum here: reddit.com/r/berkeleydeepricourse/

Unfortunately, we do not have any license that we can provide to students who are not officially enrolled in the course

CS188 Artificial Intelligence bit.ly/fnal-cs188

CS294-112 **Deep Reinforcement Learning** bit.ly/fnal-cs294

Want to Learn More?

Please do not email the course instructors about MuJoCo licenses if you are not enrolled in the course.

Docs > Welcome to rilab

Welcome to rllab

rllab is a framework for developing and evaluating reinforcement learning algorithms.

rllab is a work in progress, input is welcome. The available documentation is limited for now.

User Guide

The rllab user guide explains how to install rllab, how to run experiments, and how to implement new MDPs and new algorithms.

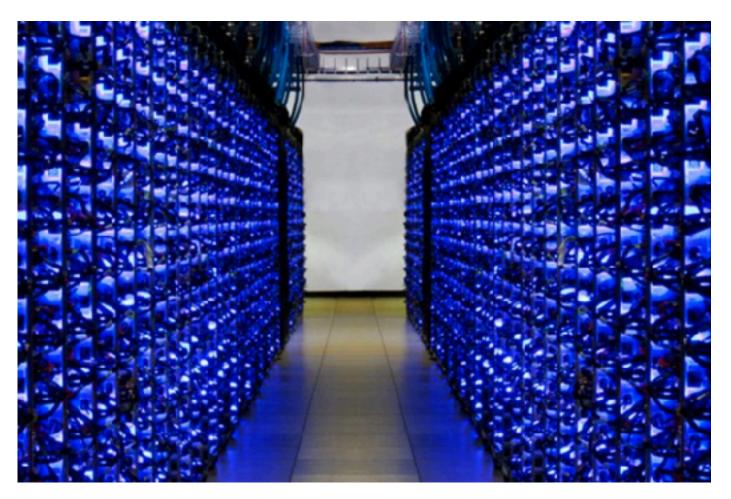
rllab Reinforcement Learning Toolkit bit.ly/fnal-rllab





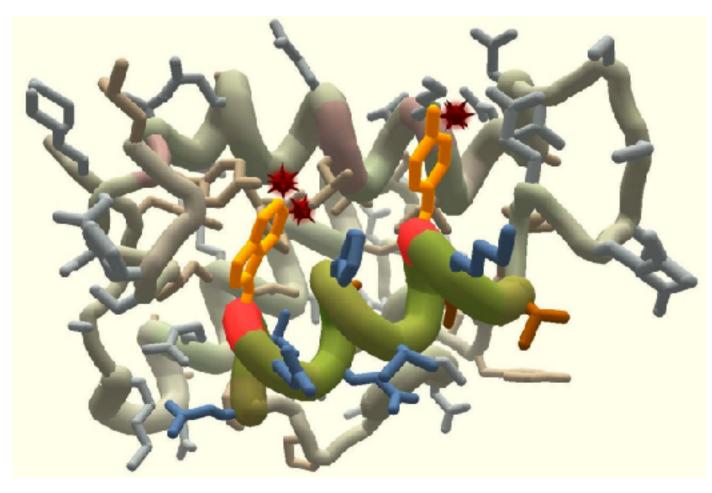


Autonomous Flight





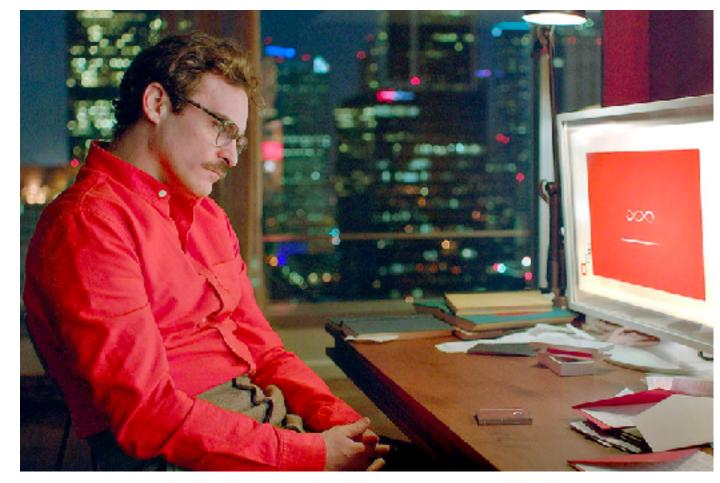




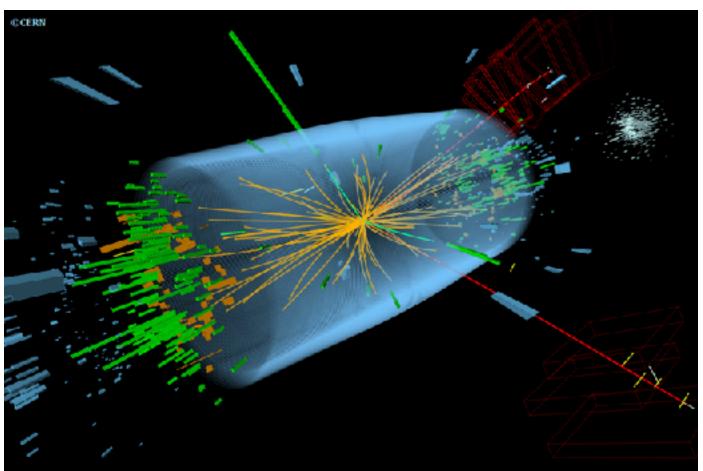
Protein Folding

Potential Applications

Algorithmic Trading



Virtual Assistant



Experiment Design



Thank you!



[Back-up Slides]



Target Network

$sample = r + \gamma$ $\theta \leftarrow \theta - \alpha \nabla_{\theta} (\zeta)$

Problem: constantly regressing against moving target since θ is used in computing sample estimates (and error accumulates). Solution: use a snapshot of the parameter value to compute sample sample = $r + \gamma$

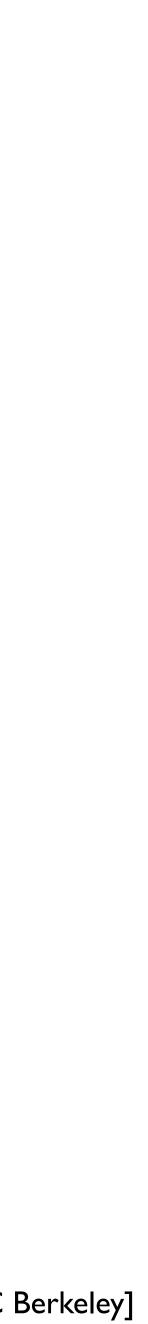
 $\theta \leftarrow \theta - \alpha \nabla_{\theta} (\theta)$

$$\gamma \max_{a'} Q_{\theta}(s', a')$$

 $Q_{\theta}(s, a) - sample)^{2}$

- estimates, θ_{target} which is updated occasionally (once per ~10⁴ updates)

$$\gamma \max_{a'} Q_{ heta_{ ext{target}}}(s',a') \ Q_{ heta}(s,a) - sample)^2$$



Trust Region Methods

Trust Region Policy Optimization [Schulman 2015]

- Efficient scheme through conjugate gradient;
- Replace objective with surrogate loss, which is a better approximation yet equally efficient to evaluate.

- **Problem:** For high-dimensional θ building F_{θ} is impractical!



Policy Gradient

Assumes a stochastic policy $\pi_{\theta}: S \times \mathcal{A} \to [0, 1]$

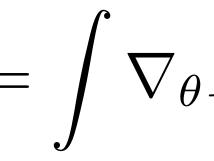
where

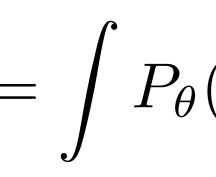
 $P_{\theta}(\tau) = P(s_0) \quad \pi_{\theta}(a_t|s_t) P(s_{t+1}|s_t, a_t)$

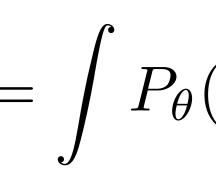
 $U(\theta) = \mathbb{E}_{\tau|\pi_{\theta}}[U(\tau)]$ $= \int P_{\theta}(\tau) U(\tau) \mathrm{d}\tau$

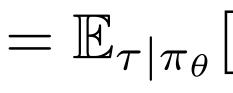


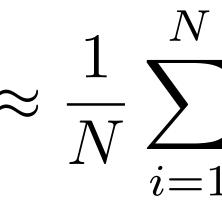
$\nabla_{\theta} U(\theta) = \nabla_{\theta} \int P_{\theta}(\tau) U(\tau) \mathrm{d}\tau$











Policy Gradient

 $= \int \nabla_{\theta} P_{\theta}(\tau) U(\tau) \mathrm{d}\tau$

 $= \int P_{\theta}(\tau) \frac{\nabla_{\theta} P_{\theta}(\tau)}{P_{\theta}(\tau)} U(\tau) d\tau$

 $= \int P_{\theta}(\tau) \nabla_{\theta} \log P_{\theta}(\tau) U(\tau) \mathrm{d}\tau$

 $= \mathbb{E}_{\tau|\pi_{\theta}} |\nabla_{\theta} \log P_{\theta}(\tau) U(\tau)|$

 $\approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log P_{\theta}(\tau_i) U(\tau_i)$



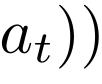
Recall

$P_{\theta}(\tau) = P(s_0) \prod \pi_{\theta}(a_t | s_t) P(s_{t+1} | s_t, a_t)$

Hence $\nabla_{\theta} \log P_{\theta}(\tau) = \nabla_{\theta} \log P(s_0) + \sum_{t} \left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) + \nabla_{\theta} \log P(s_{t+1} | s_t, a_t) \right)$ $=\sum_{t} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

No need to differentiate through dynamics!

Policy Gradient



- Faster learning
 - Exploration [Stadie et al, 2015; Houthooft et al, \bullet 2016; Tang et al, 2016]
 - Meta-learning: RL² [Duan et al, 2016]; One-shot \bullet Imitation Learning [Duan et al, 2017]; MAML [Finn et al, 2017]
- Transfer learning
 - Modular networks [Devin et al, 2017]; Invariant • feature spaces [Gupta et al, 2017]
 - Domain randomization [Tobin et al, 2017] ullet
- Safe learning •
 - [Kahn et al, 2017; Held et al, 2016]

Our Current / Future Directions

- Unsupervised / Semi-supervised learning •
 - InfoGAN [Chen et al, 2016];VLAE [Chen et al, 2017]; Temporal segment models [Mishra et al]
- Grounded language / Multi-agent •
 - "Inventing" language [Mordatch & Abbeel, 2017]
- Imitation
 - Generative Adversarial Imitation Learning [Ho et al, 2016]; Guided Cost Learning [Finn et al, 2016]; Third-person [Stadie et al, 2017]
- Value alignment / Al safety
 - CIRL [Hadfield-Menell et al, 2016]; Off-switch \bullet [Hadfield-Menell et al, 2016]
 - Communication [Huang et al, 2017]

