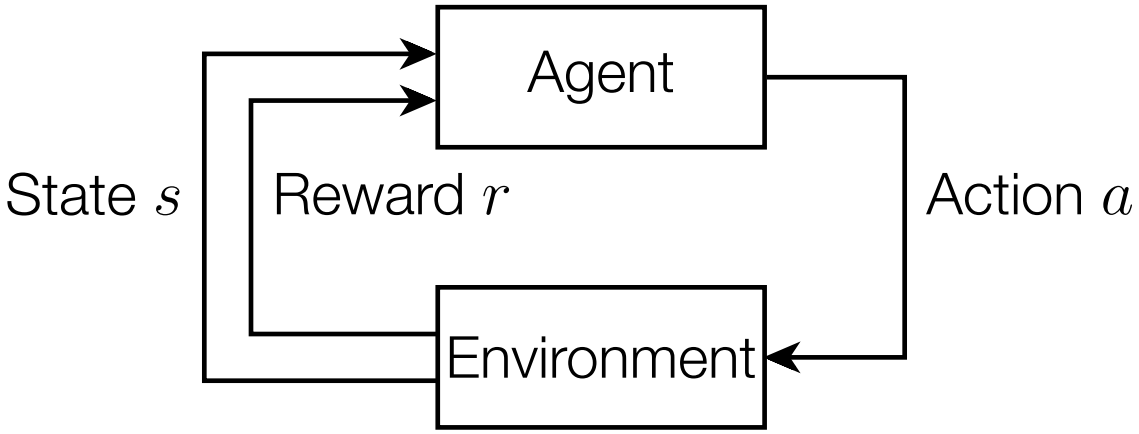
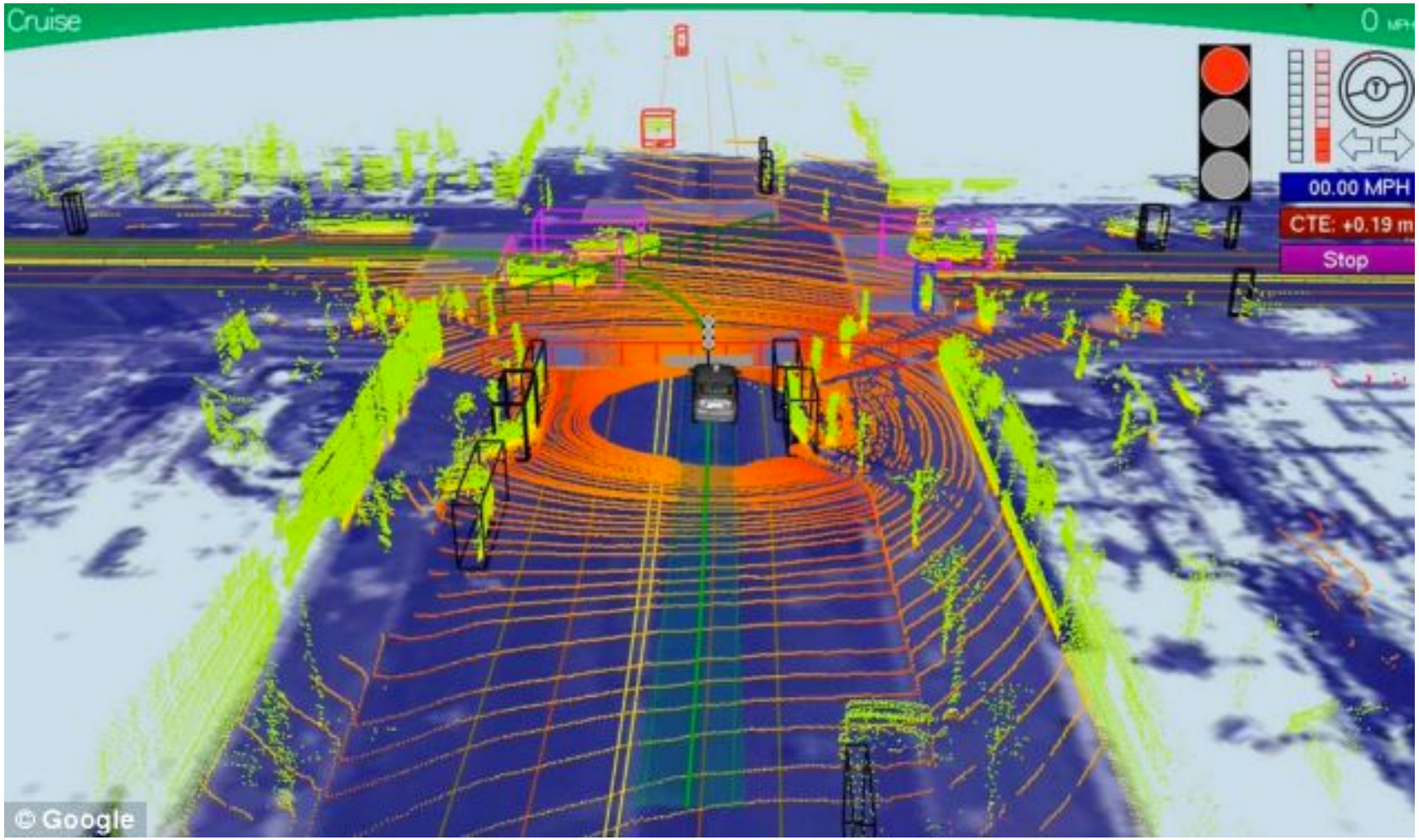
Introduction to Reinforcement Learning

J. Zico Kolter
Carnegie Mellon University

Agent interaction with environment



Of course, an oversimplification



Review: Markov decision process

Recall a (discounted) Markov decision process

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$: transition probability distribution $P(s'|s,a)$
- $R : \mathcal{S} \to \mathbb{R}$: rewards function, $R(s)$ is reward for state s
- γ: discount factor

The RL twist: we don't know P or R, or they are too big to enumerate (only have the ability to act in the MDP, observe states and actions)

Some important quantities in MDPs

(Deterministic) policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$: mapping from states to actions

(Stochastic) policy $\pi: \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$: distribution over actions for each state

Value of a policy π, expected discounted reward if starting in some state and following policy π (expressed via Bellman equation)

$$V^{\pi}(s) = R(s) + \gamma \sum_{s \in \mathcal{S}} P(s'|s, \pi(s)) V^{\pi}(s')$$

Optimal value function (value function of optimal policy $\pi^{\star}$, policy with highest value)

$$V^{\star}(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in S} P(s'|s, a) V^{\star}(s')$$

"Solving" an MDP

Policy evaluation: To find the value of a policy π, start with any $\hat{V}^{\pi}(s)$ and repeat:

$$\forall s \in \mathcal{S}\colon\ \hat{V}^{\pi}(s) := R(s) + \gamma \sum_{s \in \mathcal{S}} P(s'|s, \pi(s)) \hat{V}^{\pi}(s')$$

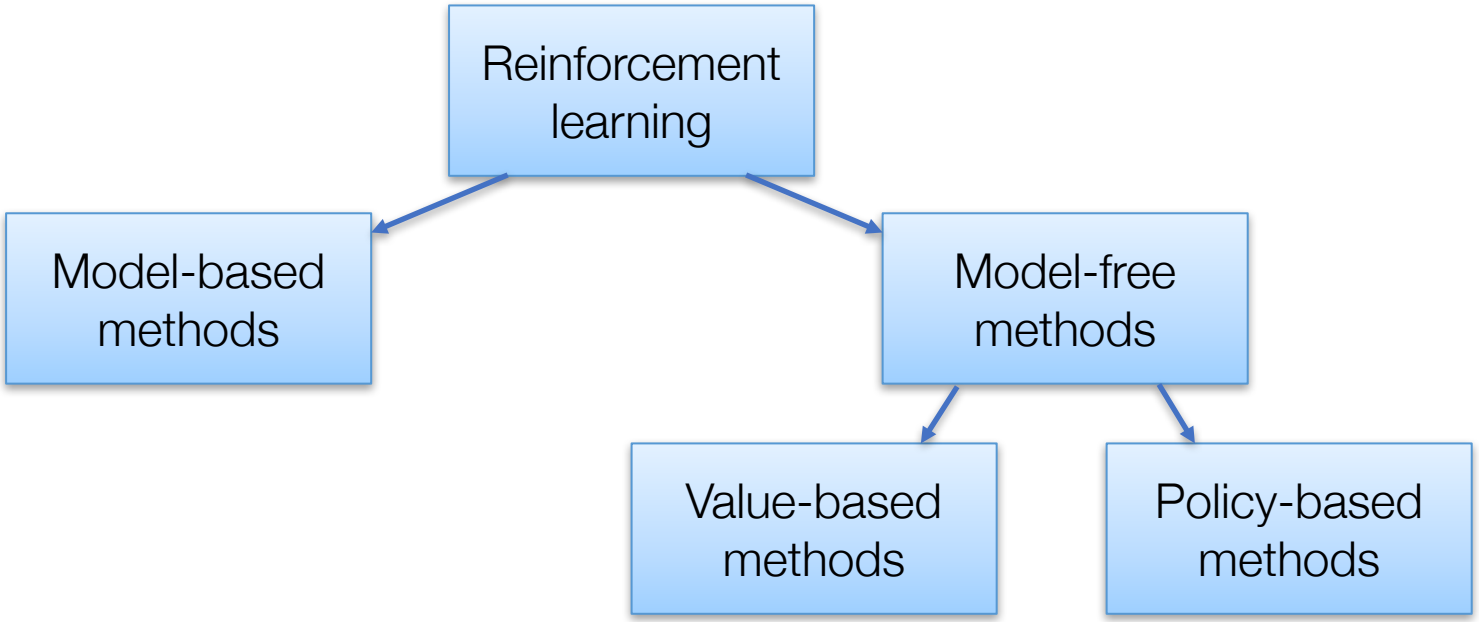
(alternatively, can solve above linear equation to determine V^{π} directly)

Value iteration: To find optimal value function, start with any $\hat{V}^{\star}(s)$ and repeat:

$$\forall s \in \mathcal{S}\colon\ \hat{V}^{\star}(s) := R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} P(s'|s, a) \hat{V}^{\star}(s')$$

But, how do we compute these quantities when P and R are unknown?

Overview of RL



Important note: the term "reinforcement learning" has also been co-opted to mean essentially "any kind of sequential decision-making problem involving some element of machine learning", including many domains different from above (imitation learning, learning control, inverse RL, etc), but we're going to focus on the above outline

Important note regarding domain size

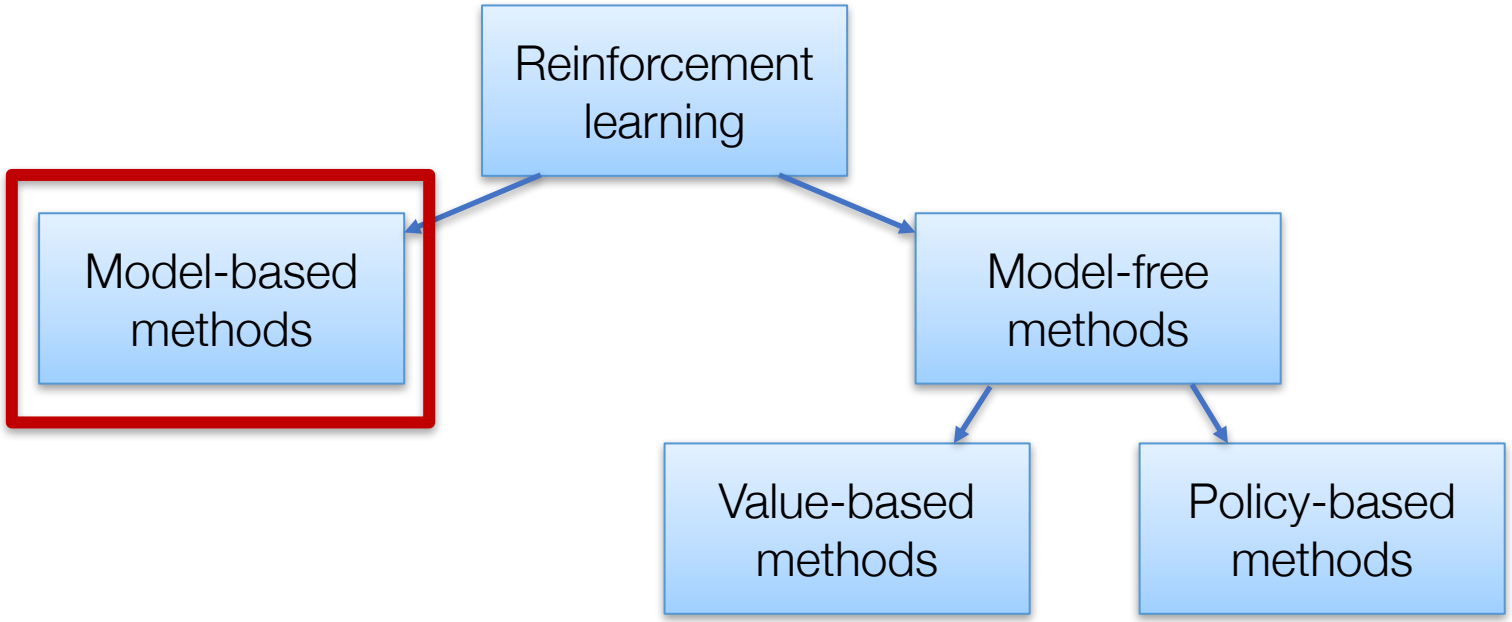
For the purposes of this lecture (except for the last section), we're going to assume a discrete state / discrete action setting where we can enumerate all states

Last part of lecture we will talk about the case of large/continuous state and action spaces

Think: grid-world, not Atari (yet)

0	0	0	1
0		0	-100
0	0	0	0

Overview of RL



Model-based RL

A simple approach: if we don't know the MDP, just estimate it from data

Agent acts in the world (according to some policy), and observes state, actions, reward sequence

$$s_1, r_1, a_1, s_2, r_2, a_2, \ldots, s_m, r_m, a_m$$

Form the empirical estimate of the MDP via the counts

$$\hat{P}(s'|s,a) = \frac{\sum_{i=1}^{m-1} 1\{s_i = s, a_i = a, s_{i+1} = s'\}}{\sum_{i=1}^{m-1} 1\{s_i = s, a_i = a\}}$$

$$\widehat{R}(s) = \frac{\sum_{i=1}^{m-1} 1\{s_i = s\} r_i}{\sum_{i=1}^{m-1} 1\{s_i = s\}}$$

Then solve MDP $\widehat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \hat{P}, \widehat{R}, \gamma)$ via e.g., value iteration

Model-based RL

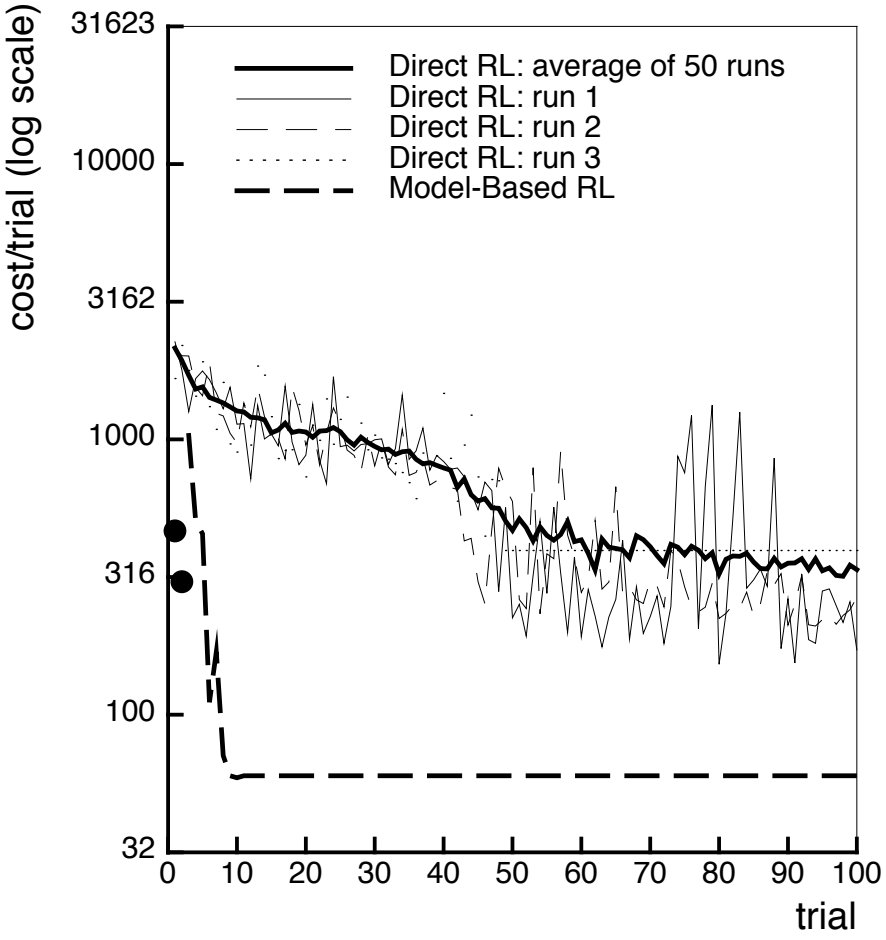
Will converge to the correct MDP (and hence correct optimal value function / optimal policy) given enough samples of each state

How do we ensure that we get the “right” samples? (a challenging problem for all the methods we present here, stay tuned)

Advantages (informally) of model-based RL: makes “efficient” use of data

Disadvantages: requires we build the actual MDP models, solve for optimal policy (not much help if state space is too large)

Performance of model-based RL



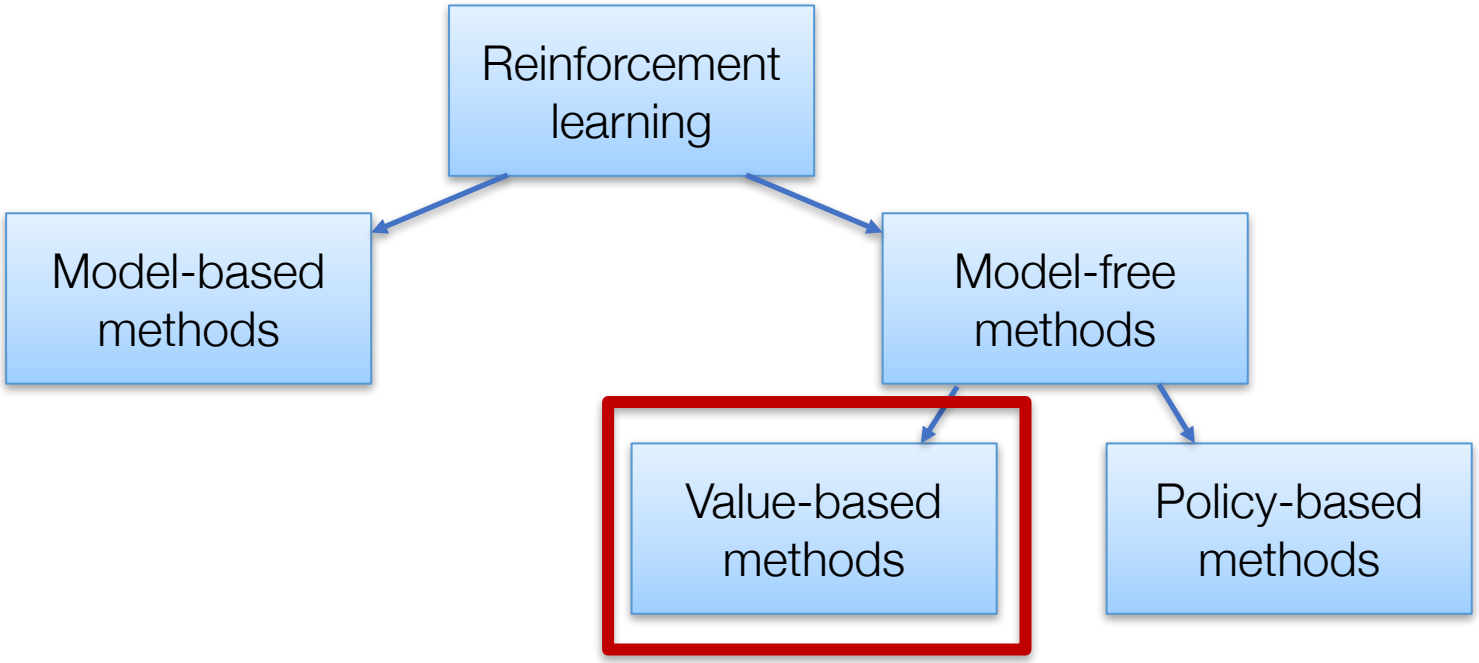
(Atkeson and Santamaría, 96)

Model-free methods

Value-based methods: based upon *temporal difference learning* (TD, SARSA, Q-learning), learn value function V^{π} or $V^{\star}$ (or a slight generalization called the Q-function, that we will discuss shortly)

Policy-based method: directly learn optimal policy $\pi^{\star}$ (or try to approximate optimal policy, if true optimal policy is not attainable)

Overview of RL



Temporal difference (TD) methods

Let's consider the task of evaluating the value of a policy, e.g. finding V^π, which we can do by repeating the iteration

$$\forall s \in \mathcal{S}\colon\ \hat{V}^\pi(s) := R(s) + \gamma \sum_{s \in \mathcal{S}} P(s'|s, \pi(s)) \hat{V}^\pi(s')$$

Now suppose we are in some state s_t, receive reward r_t, take action $a_t = \pi(s_t)$ and end up in state s_{t+1}

We can't perform the update above for *all* $s \in \mathcal{S}$, but can we update just for s_t?

$$\hat{V}(s_t) := r_t + \gamma \sum_{s' \in \mathcal{S}} P(s'|s_t, a_t)\, \hat{V}^\pi(s')$$

Temporal difference (TD) methods

…No, because we still don't know $P(s'|s_t, a_t)$ for all $s' \in \mathcal{S}$

But, s_{t+1} is a *sample* from the distribution $P(s'|s_t, a_t)$, so we could perform the update

$$\hat{V}^\pi(s_t) := r_t + \gamma \hat{V}^\pi(s_{t+1})$$

However, this is too "harsh" an assignment, assumes that s_{t+1} is the only possible next state; instead "smooth" the update using some $\alpha < 1$

$$\hat{V}^\pi(s_t) := (1-\alpha)\hat{V}^\pi(s_t) + \alpha\left(r_t + \gamma \hat{V}^\pi(s_{t+1})\right)$$

This is the *temporal difference* (TD) algorithm

The TD algorithm, more fully

The TD algorithm is essentially a stochastic version of policy evaluation iteration

Algorithm $\hat{V}^\pi = \mathrm{TD}(\pi, \alpha, \gamma)$

// Estimate value function V^π

Initialize $\hat{V}^\pi(s) := 0, \ \forall s \in \mathcal{S}$

Repeat

Observe state s and reward r

Take action $a = \pi(s)$ and observe next state s'

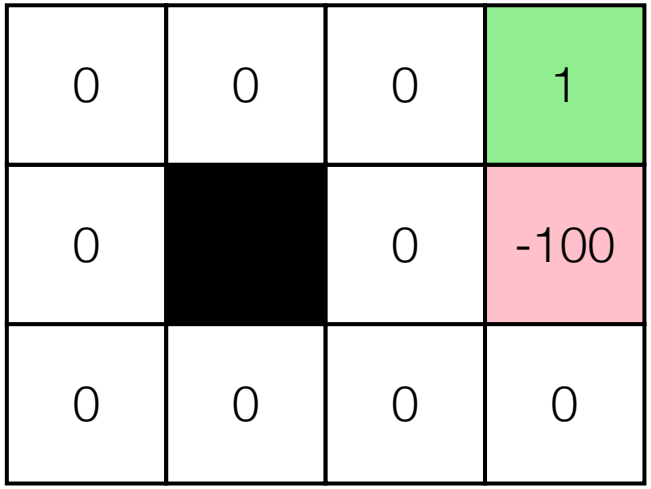
Update: $\hat{V}^\pi(s) := (1-\alpha)\hat{V}^\pi(s) + \alpha\left(r + \gamma\hat{V}^\pi(s')\right)$

Return $\hat{V}^\pi$

Will converge to $\hat{V}^\pi(s) \to V^\pi(s)$ (for all s visited frequently enough)

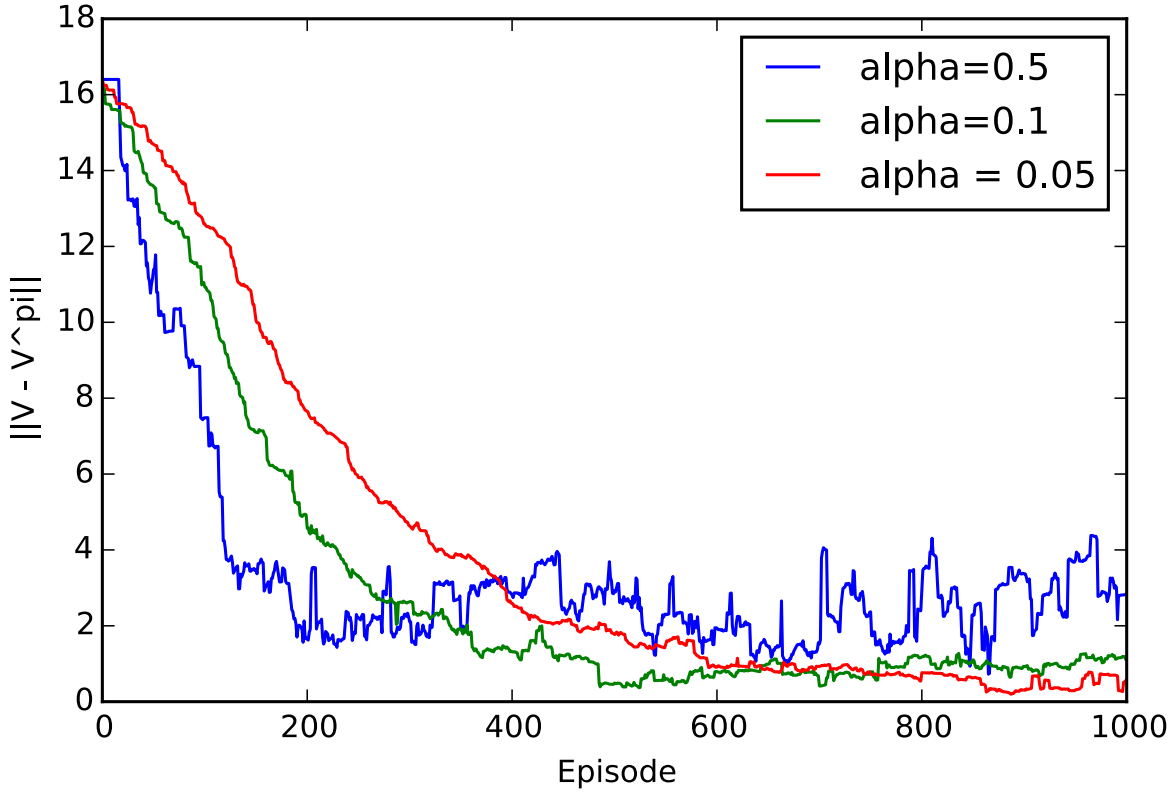
TD experiments

"Grid world" domain:



Run TD on grid world for 1000 episode (each episode consists of 10 steps, sampled according to a policy, starting at a random state), initialized with $\hat{V} = R$

TD Progress



Value estimation error of TD for different α

Issues with traditional TD algorithms

TD lets us learn a value function of a policy π directly, without ever constructing the MDP

But is this really helpful?

Consider trying to execute the greedy policy w.r.t. estimated $\hat{V}^{\pi}$

$$\pi'(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\ \hat{V}^{\pi}(s')$$

We need a model anyway…

Enter the Q function

Q functions (for MDPs in general) are like value functions but defined over state-action pairs

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) Q^{\pi}(s', \pi(s'))$$

$$Q^{\star}(s,a) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} Q^{\star}(s', a')$$

$$= R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\star}(s')$$

I.e., Q function is the value of starting at state s, taking action a, and then acting according to π (or optimally, for $Q^{\star}$)

We can easily construct analogues of value iteration or policy evaluation to construct Q functions directly, given an MDP

SARSA and Q-learning

Q function formulation leads to new TD-like methods

As with TD, observe state s, reward r, take action a (not necessarily $a = \pi(s)$, more on this shortly), observe next state s'

SARSA: estimate $Q^\pi(s, a)$

$$\hat{Q}^\pi(s,a) := (1-\alpha)\hat{Q}^\pi(s,a) + \alpha\left(r + \gamma\hat{Q}^\pi\left(s', \pi(s')\right)\right)$$

Q-learning: estimate $Q^\star(s, a)$

$$\hat{Q}^\star(s,a) := (1-\alpha)\ \hat{Q}^\star(s,a) + \alpha\left(r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}^\star(s', a')\right)$$

Will converge to Q^π, $Q^\star$ respectively given enough samples of each state-action pair

Benefits of Q function learning

The advantage of learning the Q function is that we can now select actions *without* a model of the MDP

E.g., in Q learning, the optimal policy is given by

$$\pi^\star(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}^\star(s, a)$$

so we can learn the *full* optimal policy without a model of the MDP

Aside: on-policy vs. off-policy learning

There is an important distinction in RL methods about on-policy vs. off-policy learning algorithm

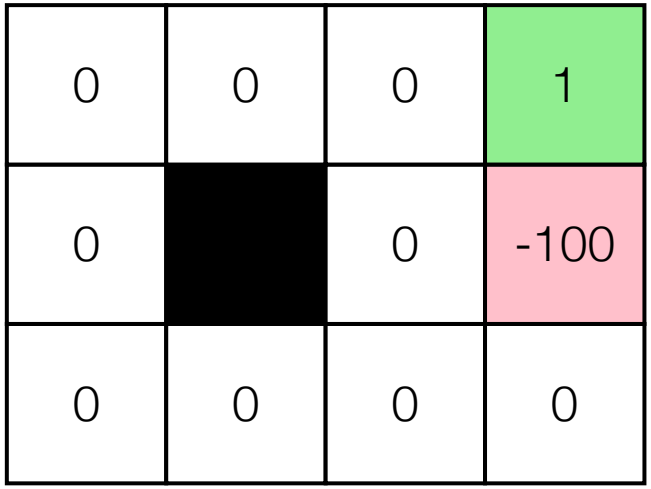
In the *on-policy* setting (e.g. the traditional TD and SARSA algorithms), you are learning the value of the policy π that you are following

In the *off-policy* setting (e.g. Q-learning algorithm) you are learning the value of a *different* policy than the one you are following

Virtually all learning guarantees apply to the on-policy case; there exist some convergence proofs for off-policy case, but only for very limited cases

- My take: Q-learning probably shouldn't work, amazing that it does

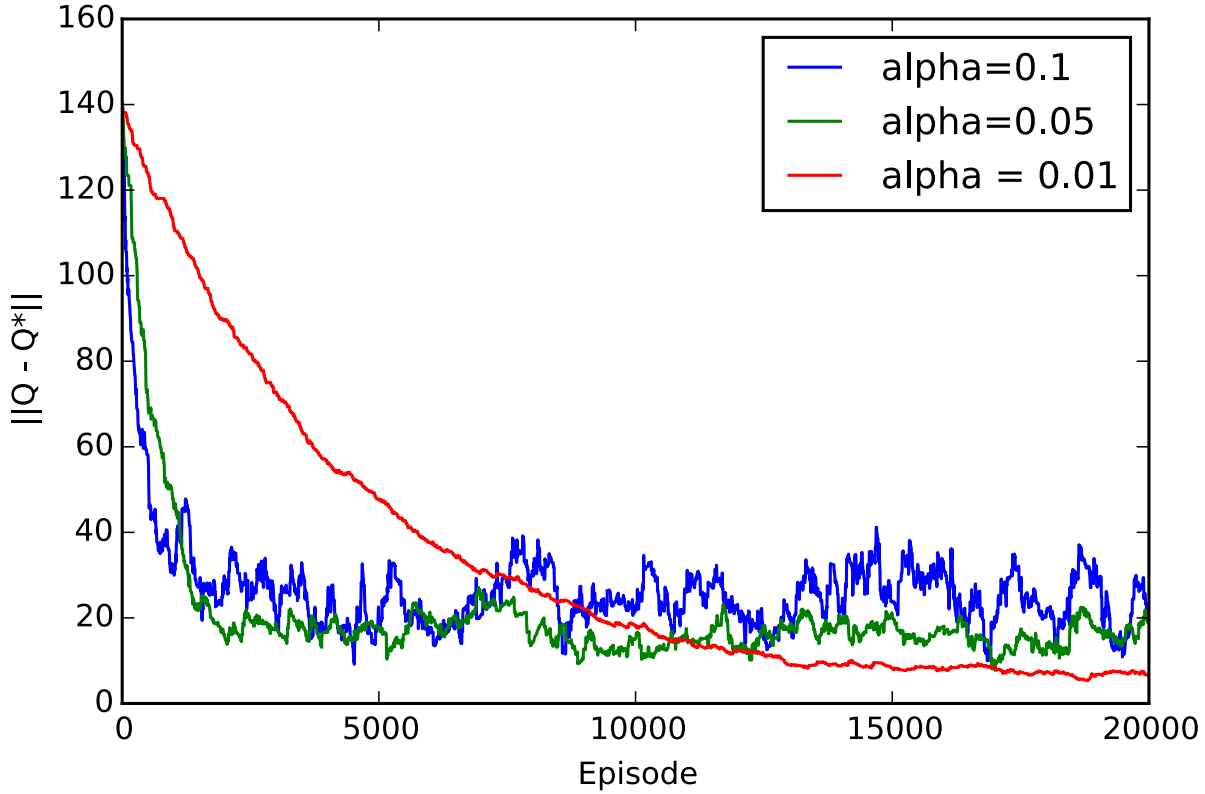
Q-learning experiment



Run Q-learning on grid world domain for 20000 epsides (each episode 10 steps), initializing with $\hat{Q}(s,a) = R(s)$

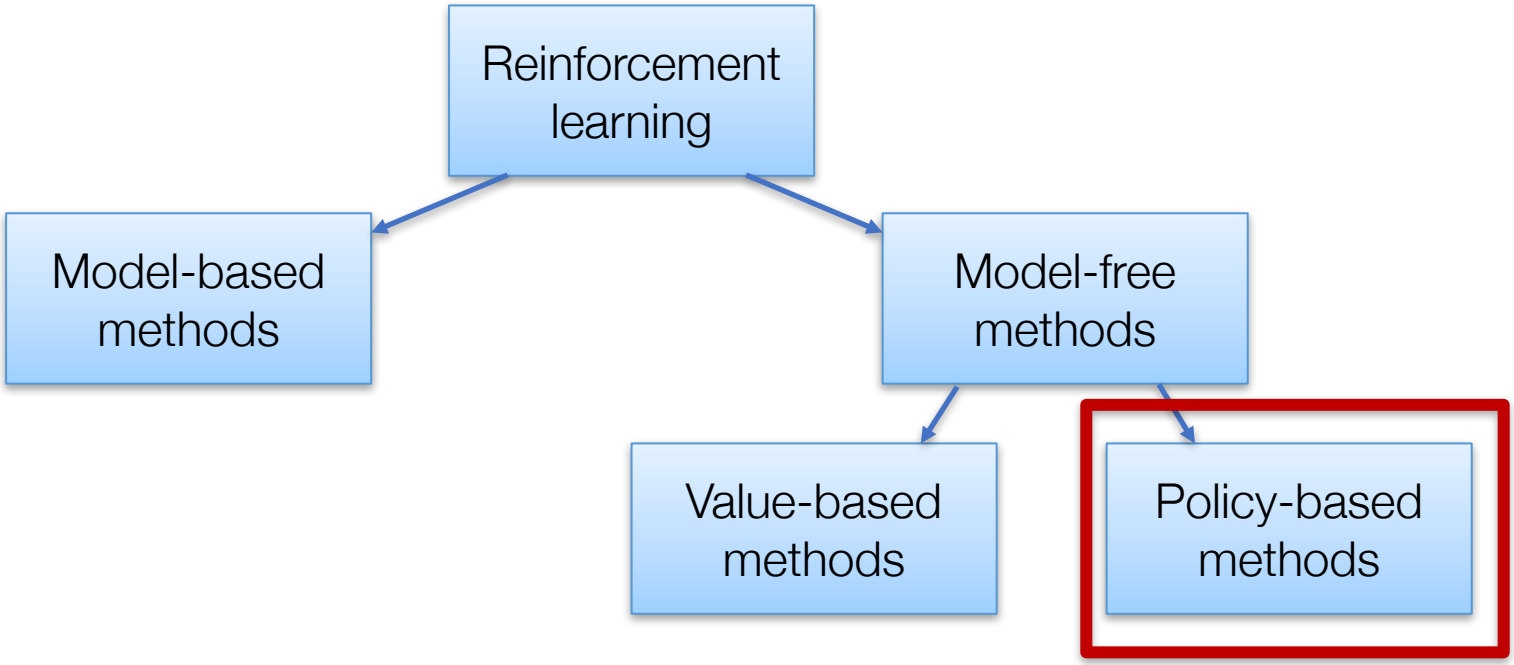
Policy: with probability 0.9, act according to current optimal policy $\pi^\star(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}^\star(s,a)$, act randomly with probability 0.1

Q-learning progress



Optimal Q function estimation error of Q-learning for different α

Overview of RL



Policy-based methods (more briefly)

A final strategy for reinforcement learning is to avoid forming both the MDP and the value/Q function, and instead try to directly learn the (optimal) policy

An example in MDPs:

$$\pi_\theta(s) = \max_{a \in \mathcal{A}} \theta(s, a)$$

or possibly the stochastic policy

$$\pi_\theta(a|s) = \frac{\exp \theta(s, a)}{\sum_{a' \in \mathcal{A}} \exp \theta(s, a')}$$

(looks similar to the Q function policy, but θ here are just arbitrary parameters, not necessarily a value function)

Evaluating policy value

Recall that the value of a policy is just the expected sum of discounted rewards (assume we can always start in the same initial state)

The optimization problem: find policy parameters θ that maximize the the sum of rewards

$$V_\theta(s) = E\left[\sum_{t=1}^{\infty} \gamma^t r_t \;\middle|\; s_{t+1} \sim P(s'|s_t, \pi_\theta(s_t)), s_1 = s\right]$$

(which we can approximate by simply running the policy for a sufficiently long horizon)

However, we can't compute the e.g. gradient $\nabla_\theta V_\theta(s)$ analytically, so we need ways to approximate it

A simple policy search procedure

Repeat:

1. Run M trials with perturbed parameters $\theta^{(1)}, \ldots, \theta^{(M)}$ (e.g., by adding random noise to parameters), and observe empirical sum of rewards $V^{(i)} = \sum_{t=1}^{T} \gamma^t r_t$
2. Fit some machine learning model $V^{(i)} \approx f(\theta^{(i)})$
3. Update parameters $\theta := \theta + \alpha \nabla_\theta f(\theta)$

This and more involved variants (similar to *evolutionary methods*) are surprisingly effective in practice

Policy gradient methods

Another policy search method that comes up often in practice: *policy gradient* methods and the REINFORCE algorithm

Requires modifications to work well in practice, but forms the basis for a large number of policy-based RL approaches

Basic setup: let $\tau = (s_1, a_1, s_2, a_2, \ldots, s_T, a_T)$ denote a *trajectory* of state/action pairs

Then we can write the value function of the policy as

$$V_\theta(s) = \mathbf{E}[R(\tau); \theta] \equiv \int p(\tau; \theta) R(\tau) d\tau$$

The REINFORCE algorithm – Part 1

The first "trick" of the REINFORCE algorithm is to derive a particular form for the gradient for the value function by:

$$\begin{aligned}\nabla_\theta V_\theta(s) &= \nabla_\theta \int p(\tau;\theta) R(\tau) d\tau \\ &= \int \nabla_\theta p(\tau;\theta) R(\tau) d\tau \\ &= \int \frac{p(\tau;\theta)}{p(\tau;\theta)} \nabla_\theta p(\tau;\theta) R(\tau) d\tau \\ &= \int p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)\, R(\tau) d\tau \\ &= \mathbf{E}[\nabla_\theta \log p(\tau;\theta)\, R(\tau)]\end{aligned}$$

I.e., the gradient of the value function can be computed as an *expectation* of the gradient of the (log of) the policy gradient times the reward

Because this is an expectation, we can approximate it using samples

The REINFORCE algorithm – Part 2

The second "trick" of the REINFORCE algorithm is to note that we can further simplify the gradient of the log trajectory probability

$$\begin{aligned}\nabla_\theta \log p(\tau;\theta) &\equiv \nabla_\theta \log \left(\prod_{t=1}^{T} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \right) \\ &= \nabla_\theta \sum_{t=1}^{T} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)) \\ &= \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\end{aligned}$$

(because the dynamics don't depend on the parameters θ)

The final REINFORCE algorithm

Repeat

1. Execute M trajectories each starting in state s and executing (stochastic) policy π_θ
2. Approximate gradient of value function as

$$g_\theta := \frac{1}{M}\left(\sum_{i=1}^{M}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(a_t^{(i)}\middle|s_t^{(i)}\right)\right) R(\tau^{(i)})\right)$$

3. Update policy to maximize value

$$\theta := \theta + \alpha g_\theta$$

An important note: don't let the name "policy gradient" fool you, this is a sampling-based method to approximate gradients (just like the function fitting method above), *not* an actual gradient-based method

Exploration/exploitation problem

All the methods we discussed so far had some condition like “assuming we visit each state enough”, or “taking actions according to some policy”

A fundamental question: if we don’t know the system dynamics, should we take exploratory actions that will give us more information, or exploit current knowledge to perform as best we can?

Example: a model-based procedure that does *not* work

1. Use all past experience to build models $\hat{P}$ and $\widehat{R}$
2. Find optimal policy for MDP $\widehat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \hat{P}, \widehat{R}, \gamma)$ using e.g. value iteration and act according to this policy

Initial bad estimates may lead policy into sub-optimal region, and never explores further

Exploration in RL

Key idea: instead of acting according to the "best" policy based upon the current MDP estimate, act according to a policy that will *explore* less-visted state-action pairs until we get a "good estimate"

- Epsilon-greedy policy

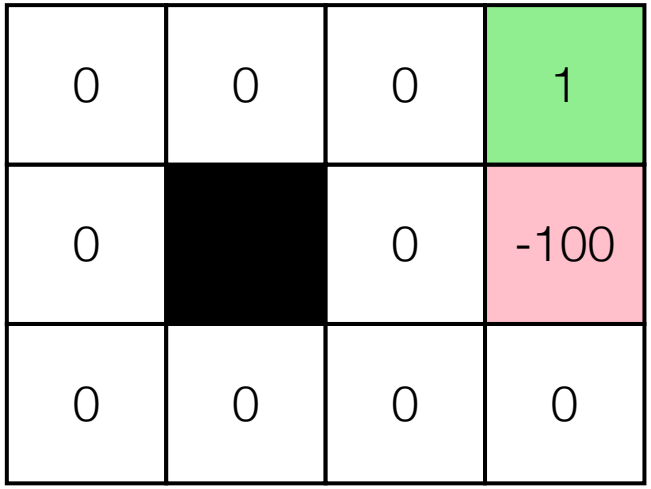
$$\pi(s) = \begin{cases} \max_{a \in \mathcal{A}} \hat{Q}(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

- Boltzman policy

$$\pi(a|s) = \frac{\exp\left(\tau\ \hat{Q}(s,a)\right)}{\sum_{a' \in \mathcal{A}} \exp\left(\tau\ \hat{Q}(s,a')\right)}$$

Want to decrease ϵ, increase τ as we see more examples

Q-learning exploration illustration

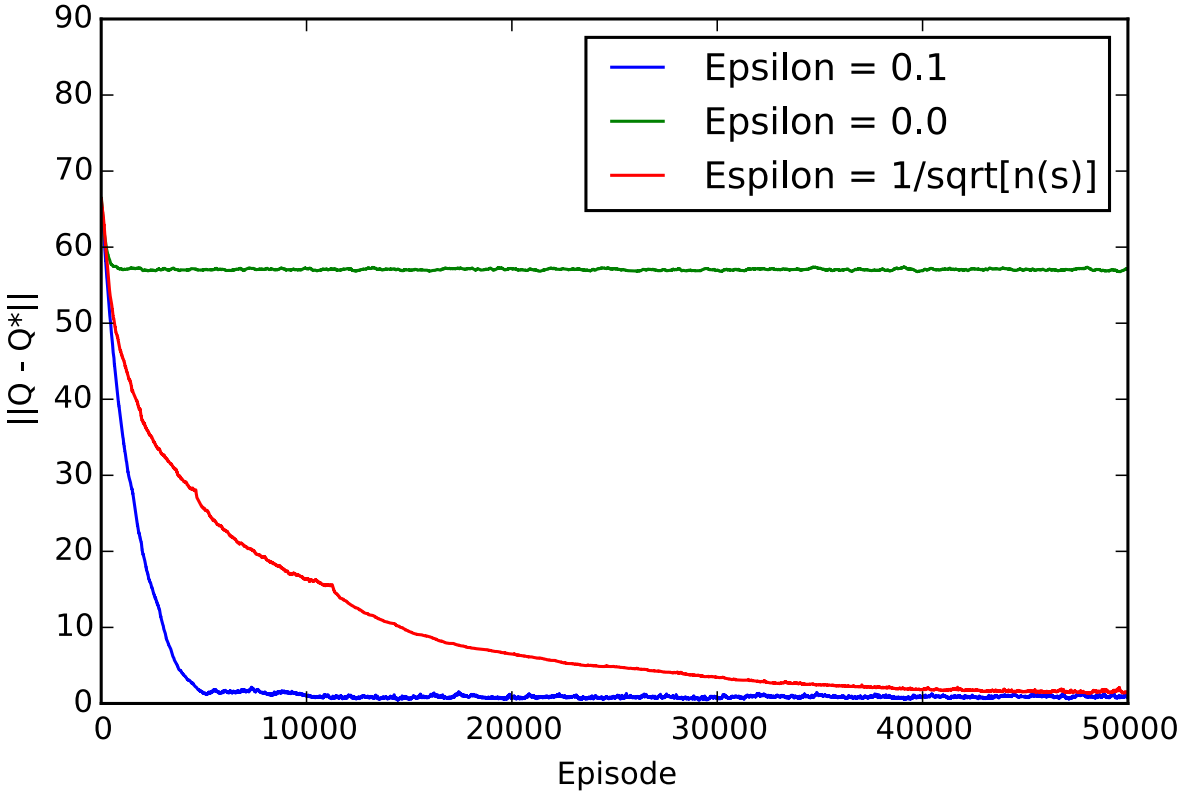


Grid world but with random $[0,1]$ rewards instead of rewards above

Initialize Q function with $\hat{Q}(s,a) := 0$

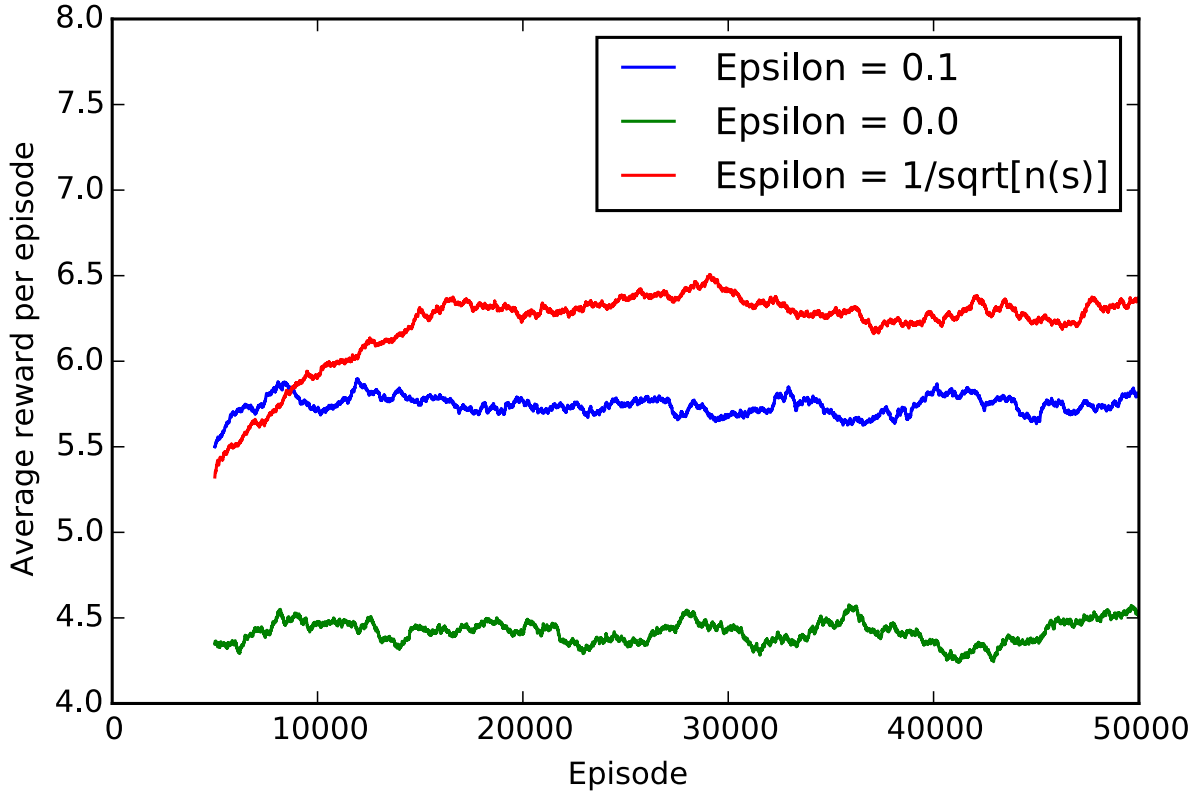
Run with $\alpha = 0.05$, $\epsilon = 0.1$, $\epsilon = 0$ (greedy), $\epsilon = 1/\big(n(s)\big)^{1/2}$, where $n(s)$ is the number of times we have visited state s

Accuracy of Q function approximation



Error in Q function estimation for different exploration strategies

Average attained reward



Average per-episode reward attained by different exploration strategies

Recent trends in RL

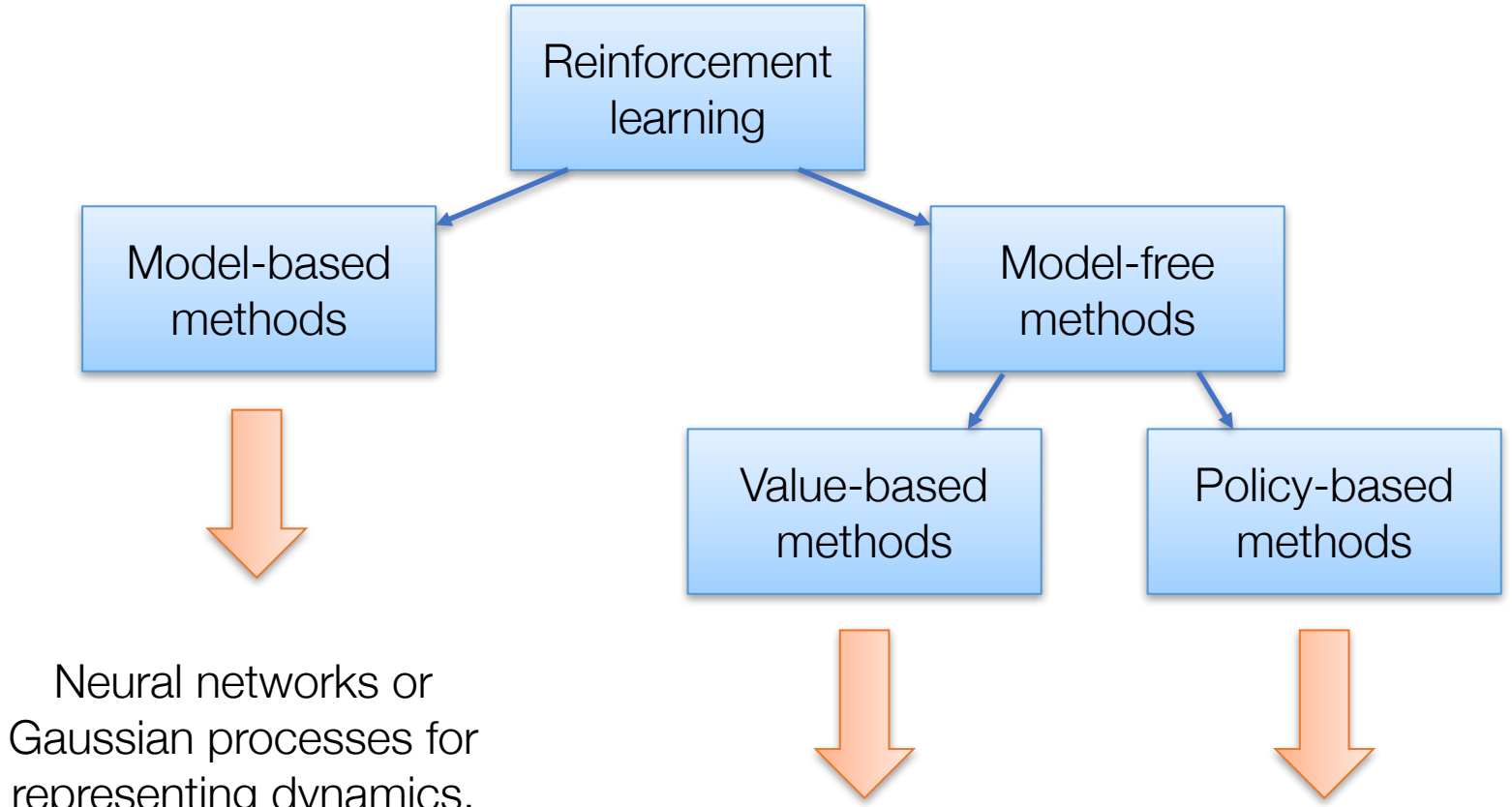
Grid worlds are great, but this isn't really what's driving all the excitement in RL, is it?

Actual interest in RL comes from extensions to large state / observation spaces, large or continuous action spaces

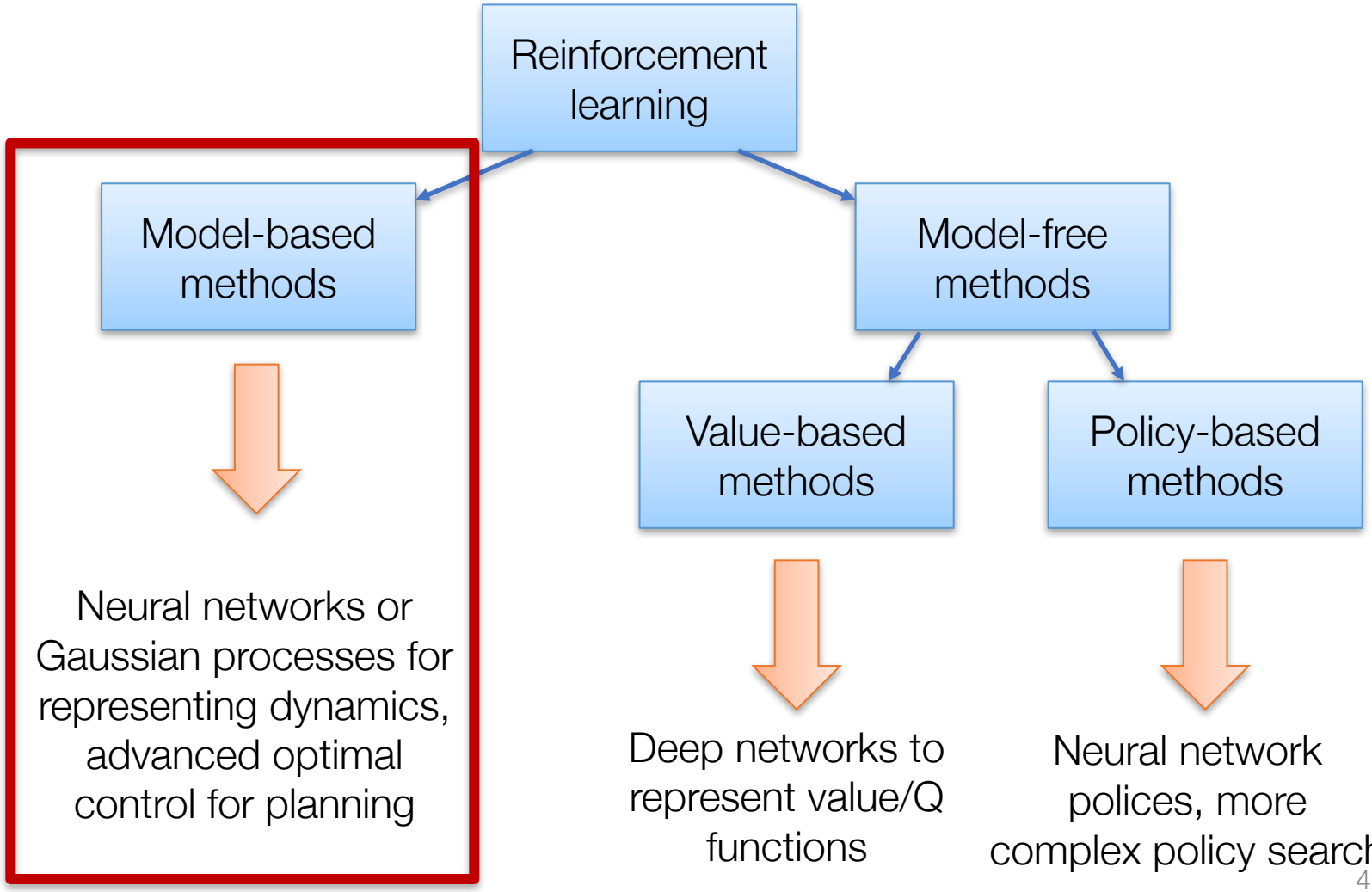
Often accomplished with the aid of deep learning methods to learn unknown quantities in RL setting

Many notable success stories: Atari game playing, AlphaGo, several robotics tasks

Modern advances / “Deep RL”

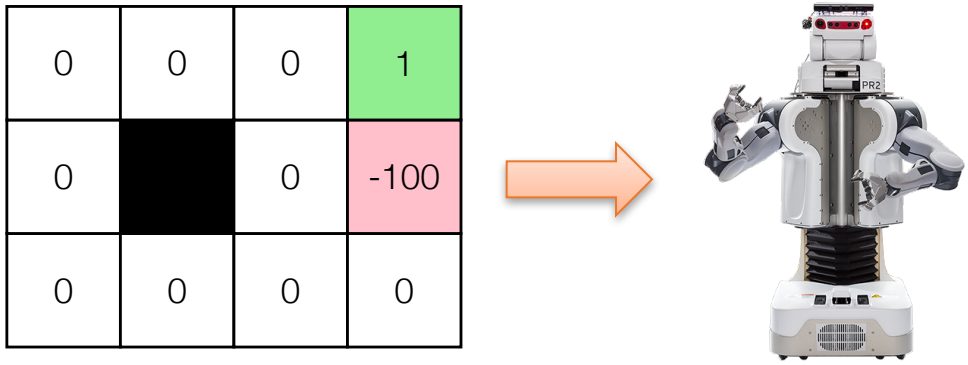


Modern advances / “Deep RL”



Model-based RL control for robotics

From discrete settings to continuous (state and action) settings



Conceptual, process is the same as for MDP model-based RL: learn a dynamics model, solve task in the model, then apply to real system

Key challenges: need better models (can't use simple discrete MDPs), and better control algorithms for "solving" tasks

Example application: Learning manipulation skills

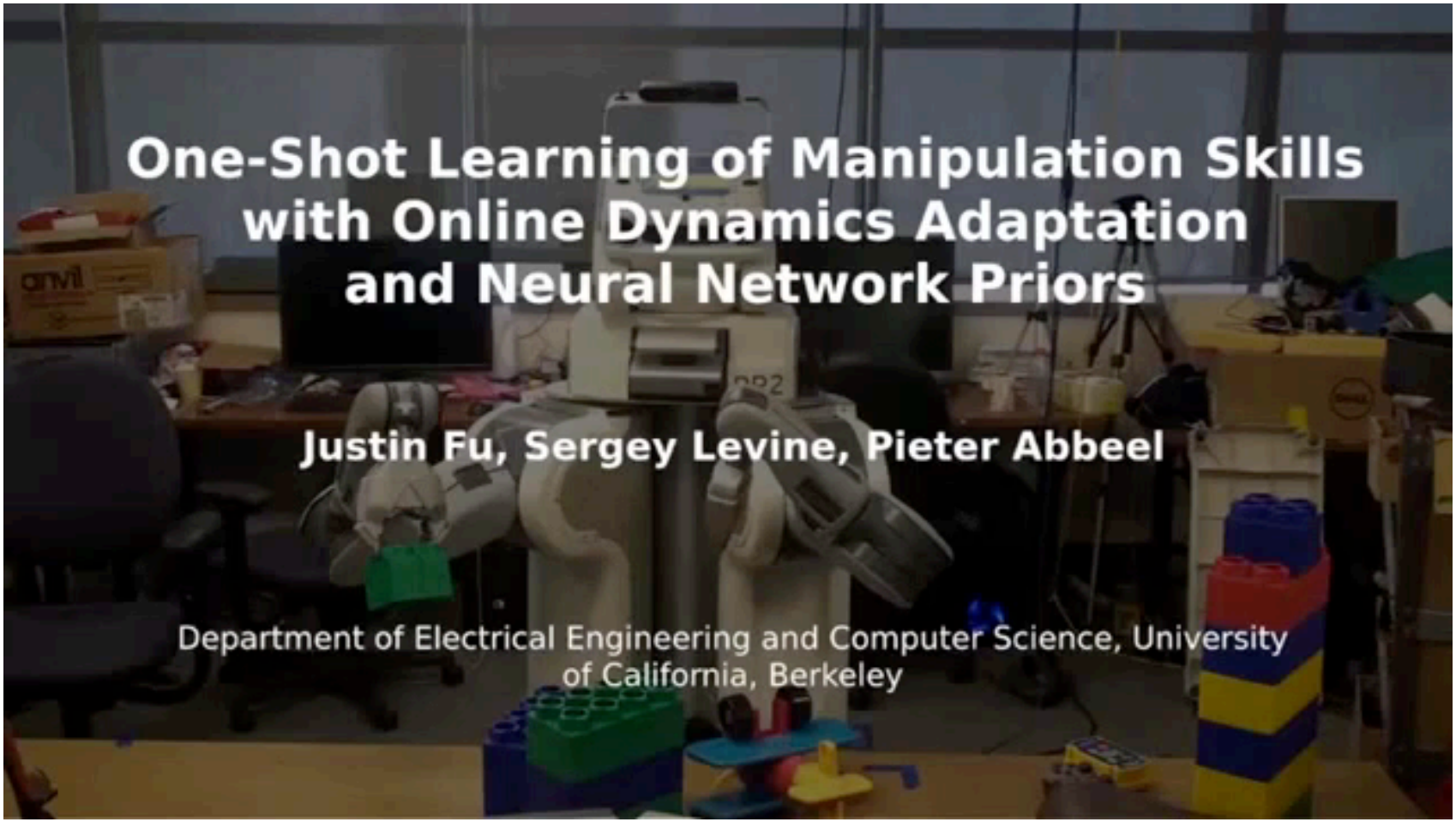
From: Fu, Levine and Abbeel, “One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors”, 2016.

Goal: “Quickly” learn manipulation skills of new objects with for a robotic system (object manipulation with contacts made for challenging problem)

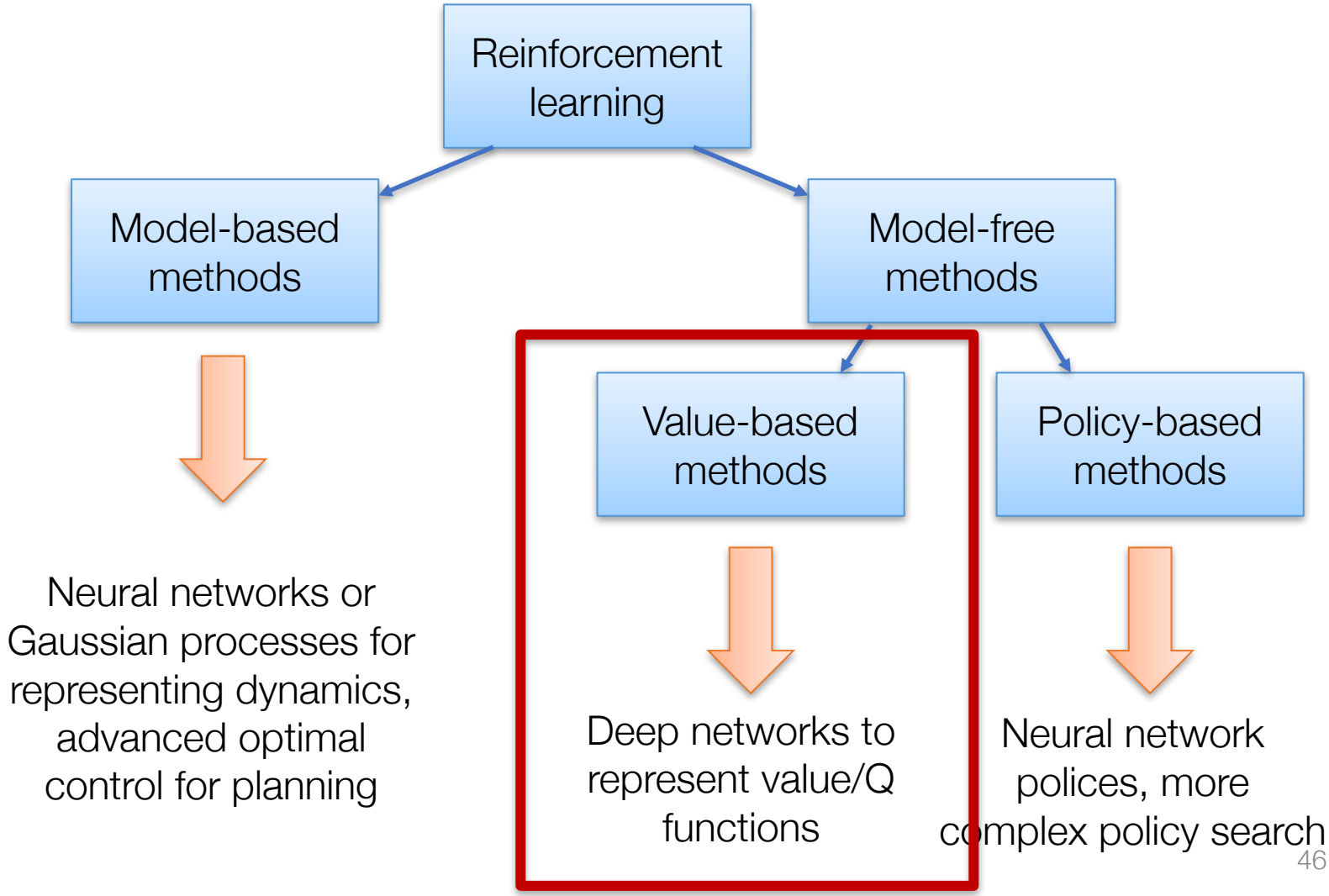
Two high level ideas:

1. Use a neural network (with some tricks to quickly adapt to new data inputs) to model dynamics
$$s_{t+1} \approx f(s_t, a_t, s_{t-1}, a_{t-1})$$
where f is a neural network model (also add noise)
2. Use approximate optimal control algorithm (iLQR) to control systems given the model.

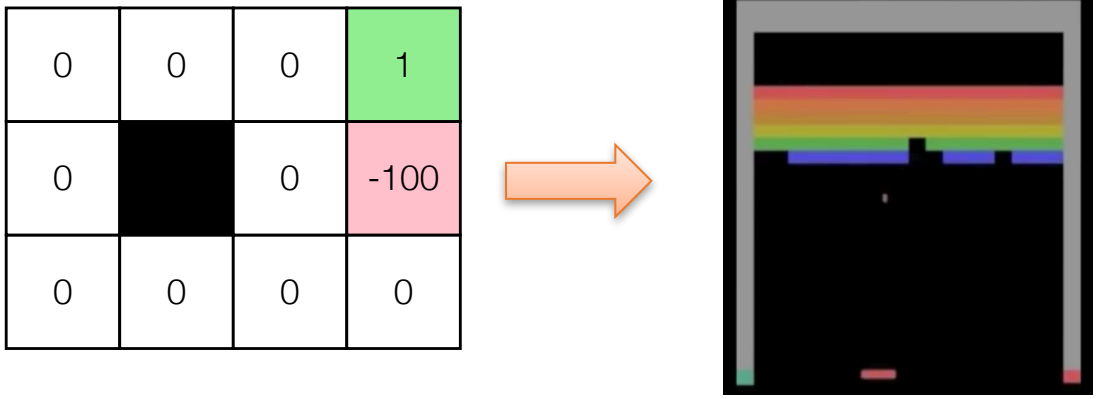
Video illustration: learning manipulation



Modern advances / “Deep RL”



Value-based methods with large state spaces



Goal: learn optimal policies in domains where the state space is too large to represent explicitly (and dynamics may not be easily expressible over this state space)

Key challenge: need a method for representing e.g., value function over extremely large state space

TD methods with value function approximation

A may advantage of TD methods in general (not a new idea, this goes back to some of the original work in TD), is the ability to use a *function approximator* to represent the value function compactly

Consider some parameterized value function approximator

$$\hat{V}^{\pi}(s) = f_{\theta}(s)$$

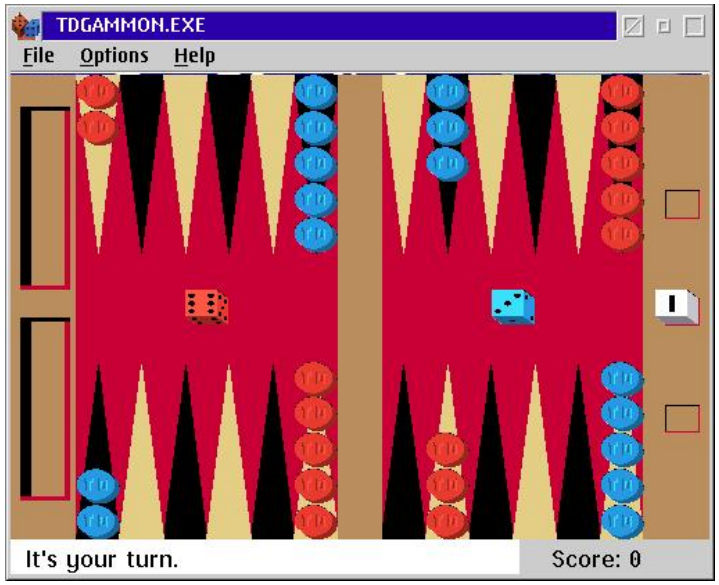
where f is e.g. a neural network, and θ denote the network weights

Then TD update is given by

$$\theta := \theta + \alpha\big(r + \gamma f_{\theta}(s') - f_{\theta}(s)\big)\nabla_{\theta} f_{\theta}(s)$$

which is a generalization of traditional TD update

TD Gammon



Developed by Gerald Tesauro at IBM Watson in 1992

Used TD w/ neural network as function approximator (known model, but much too large to solve as MDP)

Achieved expert-level play, many world experts changed strategies based upon what the strategies that the system used

Deep Q Network (DQN)

DQN method (Q-learning with deep network as Q function approximator) became famous in 2013 for learning to play a wide variety of Atari games better than humans

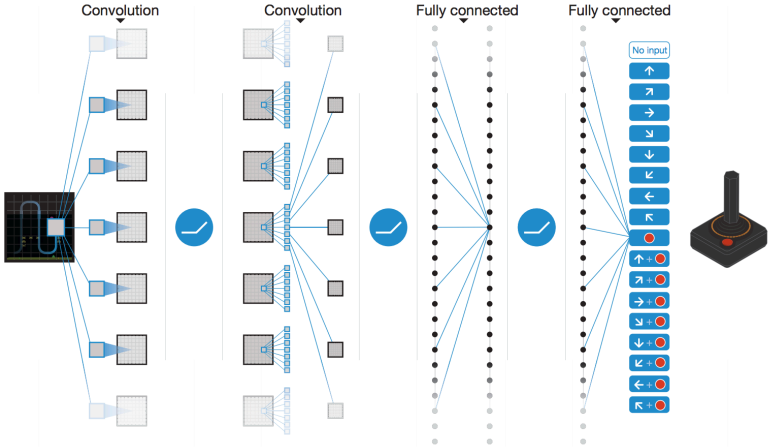
Mnih et al., “Human-level control through deep reinforcement learning”, 2015

DQN Updates: state s, reward r, take action a (ϵ-greedy), next state s'

$$\theta := \theta + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a) \right) \nabla_\theta \, \hat{Q}_\theta(s, a)$$

A few additional tricks: keep first Q function above fixed and only update every C iterations, keep replay buffer of past actions

Application to Atari Games

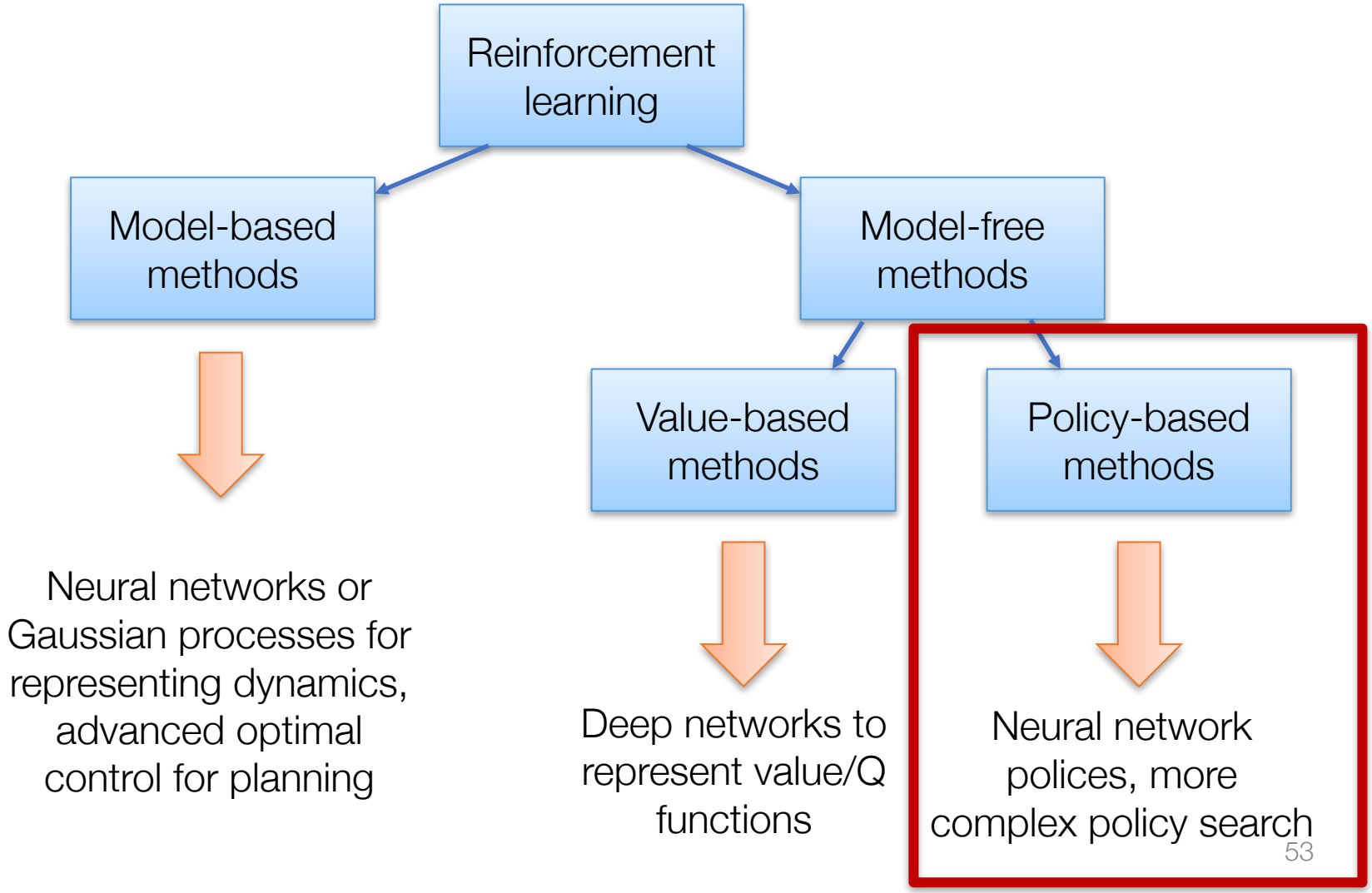


Convolutional network that takes pixel images as input (actual pixel images from past four frames), outputs Q function

Same exact method applied to 50 Atari games, exceeds human performance in most cases

Video of Breakout playing

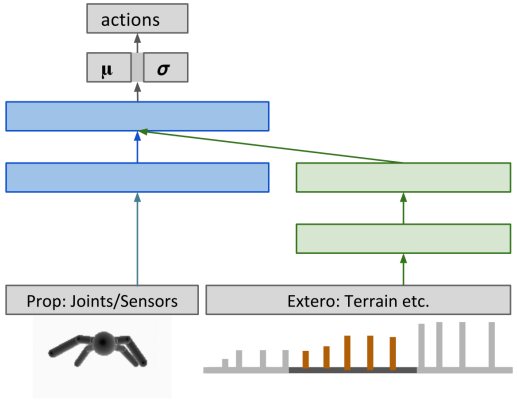
Modern advances / “Deep RL”



Learning locomotion policies

From Heess et al., “Emergence of Locomotion Behaviors of in Rich Environments”, 2017

Goal: learn to control a (simulated) robot to move forward using a deep neural network policy



Policy has input from joints/contacts, as well as surrounding terrain, outputs joint torques

Optimize policy using Proximal Policy Optimization (PPO), a policy gradient method similar to REINFORCE but with some tricks that improve performance substantially (importance sampling, term that penalizes changing policy too much, and estimating value functions for “baseline”)

Demonstration of locomotion behavior

Final thoughts: making sense of the jungle

"Reinforcement learning" (and we only covered a tiny piece), seems like a vast jungle of algorithms, overall methodologies, and applications

…and we may be headed for a replication crisis of sorts (see e.g. Henderson et al., "Deep Reinforcement Learning that Matters", 2017)

Some simple rules on "which approach best suits me"

- Model-based: if you are data-poor, and/or have a real system you can't afford to break 100 times while training
- Value-based: Problem looking a bit more "planning-like" ideally with small numbers of actions
- Policy-based: Relatively simple "reactive" policies could work well, but potentially high dimensional control space

More resources

Sutton and Barto book (updated 2017, though still mainly older material):
http://incompleteideas.net/book/the-book-2nd.html

David Silver's RL Course (pre-AlphaGo):
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Berkeley Deep Reinforcement Learning Course:
http://rail.eecs.berkeley.edu/deeprlcourse/

Deep RL Bootcamp lectures:
https://sites.google.com/view/deep-rl-bootcamp/lectures

Thank you!

Materials will be posted on summer school site