Introduction to Reinforcement Learning

Lecture 3: Policy Gradients & Model-Based RL

Shimon Whiteson
Dept. of Computer Science
University of Oxford

(based on material from
Rich Sutton & Andrew Barto)

November 10, 2020

Policy gradient methods

- Optimise π_θ with gradient ascent on expected return:

$$J_\theta = \mathbb{E}_{s \sim \rho(s), a \sim \pi_\theta(s, \cdot)} \left[Q^\pi(s, a) \right]$$

  where $\rho(s) = p(s_0 = s)$

- Useful when:
  - Greedification is hard, e.g., continuous actions
  - Stochastic policies are preferred, e.g., partial observability
  - Optimal policies are simpler than optimal value functions
  - Prior knowledge is easier to express about policies
- Typically converges to local optimum
- Gradient estimates typically have high variance

Simple case

- One-step MDP with $s \sim \rho(\cdot)$:

$$\begin{aligned} J_\theta &= \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(s,\cdot)}\left[R_s^a\right] \\ &= \sum_s \rho(s) \sum_a \pi_\theta(s,a) R_s^a \end{aligned}$$

- Take the gradient:

$$\begin{aligned} \nabla_\theta J_\theta &= \sum_s \rho(s) \sum_a \nabla_\theta \pi_\theta(s,a) R_s^a \\ &= \sum_s \rho(s) \sum_a \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} R_s^a \\ &= \sum_s \rho(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) R_s^a \\ &= \mathbb{E}_{s\sim\rho, a\sim\pi_\theta(s,\cdot)}\left[\nabla_\theta \log \pi_\theta(s,a) R_s^a\right] \end{aligned}$$

- Sampling yields the *likelihood ratio* or *score function estimator*

Policy gradient theorem & REINFORCE

- The *policy gradient theorem* [Sutton et al. 2000] uses an unrolling argument to extend this to general MDPs:

$$\nabla_\theta J_\theta = \mathbb{E}_{s \sim \rho^\pi(s), a \sim \pi_\theta(s,\cdot)} \left[\nabla_\theta \log \pi_\theta(s,a) Q^\pi(s,a) \right]$$

where $\rho^\pi(s)$ is the *discounted ergodic occupancy measure*:

$$\rho^\pi(s) = \sum_{i=0}^{\infty} \gamma^i p(s_i = s \mid \pi)$$

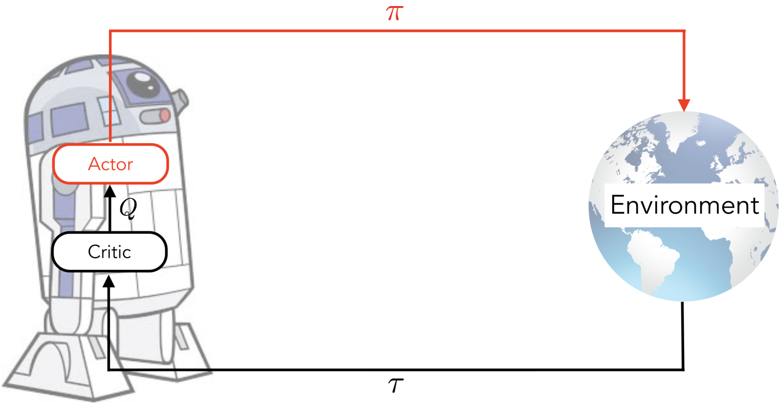
- Using sample returns yields REINFORCE [Williams 1992]:

$$\nabla_\theta J_\theta \approx g(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t) R_t$$

Actor-Critic Methods [Sutton et al. 00]

- Reduce variance in $g(\tau)$ by learning a *critic* $Q(s, a)$:

$$g(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t) Q(s_t, a_t)$$



Control variates

- *Control variates* reduce variance in Monte Carlo sampling
- Let $\hat{x}$ be an unbiased estimator of x: $\mathbb{E}\left[\hat{x}\right] = x$, where x is unknown
- Let $\hat{y}$ be an unbiased estimator of y: $\mathbb{E}\left[\hat{y}\right] = y$, where y is known
- Another unbiased estimator of x is:

$$\hat{x}' = \hat{x} - \lambda(\hat{y} - y),$$

  with variance:

$$\mathrm{Var}(\hat{x}') = \mathrm{Var}(\hat{x}) + \lambda^2 \mathrm{Var}(\hat{y}) - 2\lambda \mathrm{Cov}(\hat{x}, \hat{y})$$

- If $\hat{x}$ and $\hat{y}$ are sufficiently correlated, then $\exists \lambda, \mathrm{Var}(\hat{x}') < \mathrm{Var}(\hat{x})$

Baselines

- Policy gradient methods use a control variate called a *baseline* $b(s)$:

$$g(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)(Q(s_t, a_t) - b(s_t))$$

- Estimator remains unbiased if b does not depend on a:

$$\begin{aligned}
\mathbb{E}_{a\sim\pi_\theta(s,\cdot)}\left[\nabla_\theta \log \pi_\theta(s,a) b(s)\right] &= \mathbb{E}_{a\sim\pi_\theta(s,\cdot)}\left[\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} b(s)\right] \\
&= \sum_a \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} b(s) \\
&= b(s) \sum_a \nabla_\theta \pi_\theta(s,a) \\
&= b(s) \nabla_\theta \sum_a \pi_\theta(s,a) \\
&= b(s) \nabla 1 = 0
\end{aligned}$$

Advantage functions

- Common choice of baseline is the value function: $b(s) = V(s)$:

$$g(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t)$$

where $A(s, a) = Q(s, a) - V(s)$ is the *advantage function*

- $Q(s, a)$ is often harder to learn than $V(s)$
- Replace it with a bootstrap target: $r_t + \gamma V(s_{t+1})$
- TD error $r_t + \gamma V(s_{t+1}) - V(s)$ is an unbiased estimate of $A(s_t, a_t)$:

$$g(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)(r_t + \gamma V(s_{t+1}) - V(s_t))$$

Generalised advantage estimation (1) [Schulman et al. 2015]

- Target used in TD error estimate of advantage could bootstrap later:

$$\hat{A}_t^{(k)} = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t)$$

- Let $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ be the TD error and note that:

$$\begin{aligned}\hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \\ &= r_t + \gamma V(s_{t+1}) - V(s_t) + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - \gamma V(s_{t+1}) \\ &= \delta_t + \gamma \delta_{t+1}\end{aligned}$$

- More generally:

$$\hat{A}_t^{(k)} = \sum_{i=0}^{k-1} \gamma^i \delta_{t+i}$$

Generalised advantage estimation (2) [Schulman et al. 2015]

Now define the generalised advantage estimator:

$$
\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)\Big(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \cdots\Big) \\
&= (1-\lambda)\Big(\delta_t + \lambda(\delta_t + \gamma\delta_{t+1}) + \lambda^2(\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2}) + \cdots\Big) \\
&= (1-\lambda)\Big(\delta_t(1 + \lambda + \lambda^2 + \cdots) + \gamma\delta_{t+1}(\lambda + \lambda^2 + \cdots) + \cdots\Big) \\
&= (1-\lambda)\left(\delta_t \frac{1}{1-\lambda} + \gamma\delta_{t+1}\frac{\lambda}{1-\lambda} + \cdots\right) \\
&= \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i}
\end{aligned}
$$

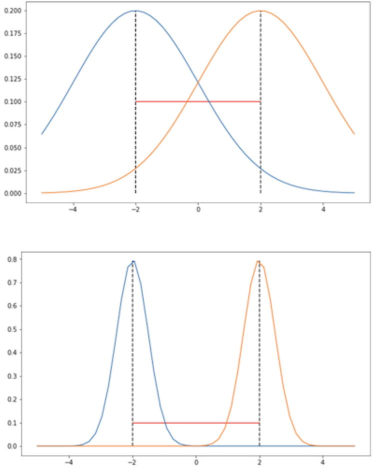
Deep Actor-Critic Methods

- Actor and critic are both deep neural networks
  - Convolutional and recurrent layers
  - Actor and critic share layers
- Both trained with stochastic gradient descent
  - Actor trained on policy gradient
  - Critic trained on TD(λ) or Sarsa(λ)
- Asynchronous advantage actor-critic (A3C) [Mnih et al. 2016]
  - Multiple asynchronous actors
  - Shared convnet, softmax layer for π, linear layer for V
  - Gradient based on k-step TD-error:

$$g(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)(\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t))$$

Performance Collapse

- Steps in parameter space are unbounded in policy space
- Example due to Agustinus Kristiadi[1]:

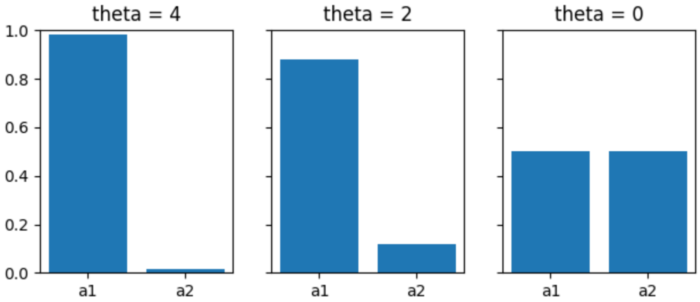


[1] https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/

Performance Collapse

- Another example, due to Joshua Achiam[2]

$$\pi_\theta(a) = \begin{cases} \sigma(\theta) & a = 1 \\ 1 - \sigma(\theta) & a = 2 \end{cases}$$



- Can cause irrevocable *performance collapse*

[2] http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf

Natural policy gradients [Kakade 2001]

- Maximise objective for fixed KL (ignoring s for simplicity):

$$\begin{aligned} &\arg\max_{\Delta\theta} J(\theta + \Delta\theta) \\ &\text{s. t. } \mathrm{KL}(\pi_\theta || \pi_{\theta+\Delta\theta}) = C \end{aligned}$$

- Approximate KL with second-order Taylor expansion:

$$KL(\pi_\theta || \pi_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top \mathbf{F} \Delta\theta,$$

  where **F** is the *Fisher information matrix*:

$$\begin{aligned} \mathbf{F} &= \mathrm{Cov}(\nabla_\theta \log \pi_\theta(a)) = \mathbb{E}_a\left[(\nabla_\theta \log \pi_\theta(a))(\nabla_\theta \log \pi_\theta(a))^\top\right] \\ &= \nabla^2_{\theta'} \mathrm{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} = \nabla^2_{\theta'} \mathrm{KL}(\pi_{\theta'} || \pi_\theta)|_{\theta'=\theta} \end{aligned}$$

- Result is an update based on the *natural gradient*:

$$\nabla_N J(\theta) = \mathbf{F}^{-1} \nabla J(\theta)$$

Trust Region Policy Optimisation [Schulman et al. 2015]

- Computing and inverting $\mathbf{F}$ is intractable for large NNs
- Instead, solve $\mathbf{F}\nabla_N J(\theta) = \nabla J(\theta)$ using *conjugate gradient* method
- Requires only cheaper matrix-vector product function $f(\mathbf{v}) = \mathbf{Fv}$
- Quadratic approx. may violate *trust region*: $\mathrm{KL}(\pi_\theta || \pi_{\theta+\Delta\theta}) \leq C$
- Backtracking line search iterates on j to find update:

$$\begin{aligned} \theta_{i+1} &= \theta_i + \alpha^j \Delta_i \\ \text{s. t. } &\mathcal{L}(\theta_i, \theta_{i+1}) \geq 0, \\ &\mathrm{KL}(\pi_{\theta_i} || \pi_{\theta_{i+1}}) \leq C, \end{aligned}$$

where Δ_i is the CG update and for $\tau \sim \pi_{\theta_i}$:

$$\begin{aligned} \mathcal{L}(\theta_i, \theta_{i+1}) &= \sum_{t=0}^{T} \gamma^t \frac{\pi_{\theta_{i+1}}(s_t, a_t)}{\pi_{\theta_i}(s_t, a_t)} A^{\pi_{\theta_i}}(s_t, a_t) \\ &\approx J(\theta_{i+1}) - J(\theta_i) \end{aligned}$$

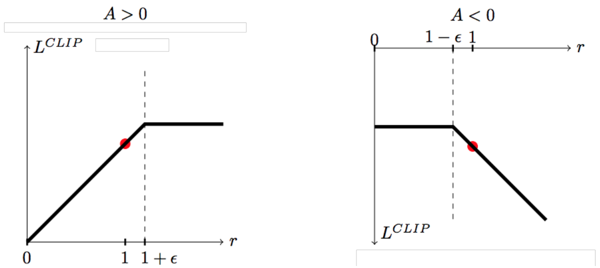
Proximal Policy Optimisation [Schulman et al. 2017]

- TRPO still requires conjugate gradient descent and line search
- Solve unconstrained optimisation problem instead with adaptive λ_i:

$$\theta_{i+1} = \arg\max_{\theta} \mathcal{L}(\theta_i, \theta) + \lambda_i \mathrm{KL}(\pi_{\theta_i} || \pi_\theta),$$

- Or optimise a clipped objective weighted by $r_t^\theta = \frac{\pi_\theta(a_t, s_t)}{\pi_{\theta_{old}}(a_t, s_t)}$:

$$\mathcal{L}_{clip}(\theta_i, \theta) = \sum_{t=0}^{T} \left[\min(r_t^\theta A^{\pi_{\theta_i}}, \mathrm{clip}(r_t^\theta, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_i}}) \right]$$



Deterministic policy gradients [Silver et al. 2014]

- Given continuous actions and a deterministic policy $\pi(s)$, the *deterministic policy gradient theorem* says:

$$\nabla_\theta J_\theta = \mathbb{E}_{s \sim \rho^\pi(s)}\Big[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a = \pi(s))\Big]$$

- Estimated from a τ gathered with a stochastic exploration policy:

$$\nabla_\theta J_\theta \approx g(\tau) = \sum_{t=0}^{T} \nabla_\theta \pi_\theta(s_t) \nabla_a Q(s_t, a = \pi(s_t)),$$

where Q is a critic trained off policy

Expected policy gradients [Ciosek & Whiteson 2018]

- Reexamine the policy gradient theorem:

$$\nabla_\theta J = \mathbb{E}_{s\sim\rho(s)}\left[\int_a \nabla_\theta \pi_\theta(s,a) Q(s,a) da\right] = \mathbb{E}_{s\sim\rho(s)}\left[I(s)\right]$$

- Can often solve $I(s) = \int_a \nabla_\theta \pi_\theta(s,a) Q(s,a) da$ analytically for fixed s
- Theoretical equivalences, e.g., for a Gaussian policy and quadratic critic, mean update equivalent to DPG
- Discrete actions are easy: $I(s) = \sum_a \nabla \pi Q(a,s)$
- In practice: works well for continuous actions; not worth it for discrete actions because Q-function is hard to learn

Model-based reinforcement learning

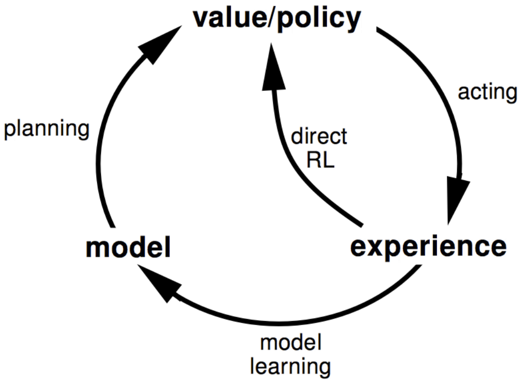
- Planning methods require prior knowledge of the MDP
- Temporal difference methods are *model-free* or *direct* reinforcement learning methods
- *Model-based* or *indirect* reinforcement learning assumes no prior knowledge but learns a model of the MDP and then plans on it
- A *model* is anything the agent can use to predict how the environment will respond to its actions

Types of models

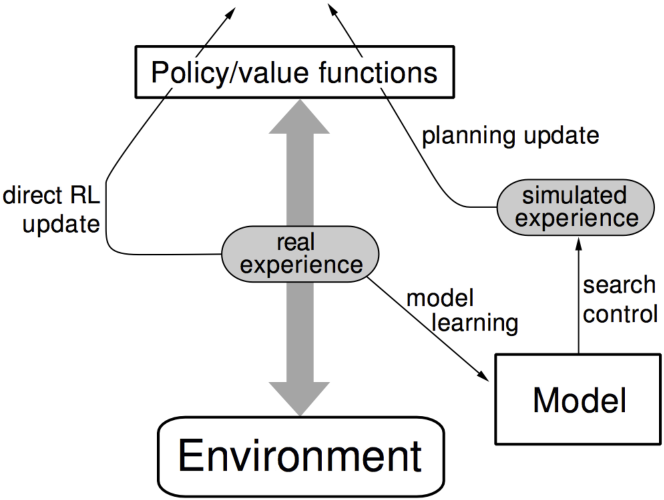
- A *full* or *distribution* model is a complete description of $P^a_{ss'}$ and $R^a_{ss'}$: space complexity is $O(|S|^2|A|)$
- A *sample* or *generative* model can be queried to produce samples r and s' given any s and a
- A *trajectory* or *simulation* model can simulate a complete episode but cannot jump to an arbitrary state

Planning, learning, and acting

- Model-based methods make fuller use of experience: lower *sample complexity*
- Model-free methods are simpler and not affected by modelling errors
- Can also be combined



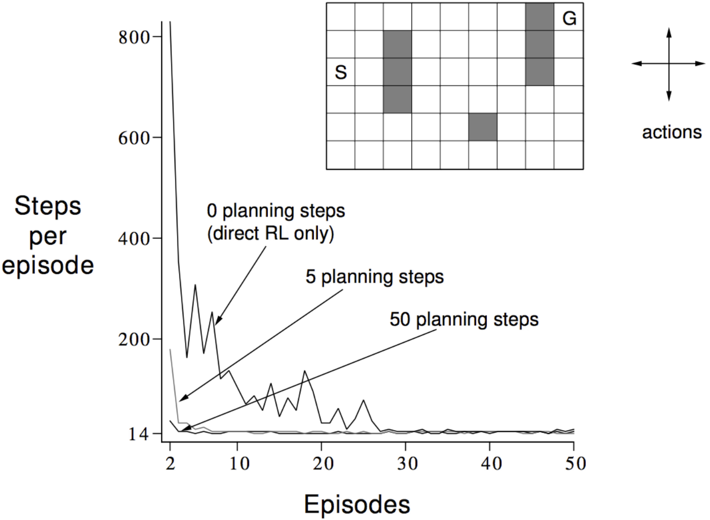
Dyna architecture



Dyna-Q (1)

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
- (a) $s \leftarrow$ current (nonterminal) state
- (b) $a \leftarrow \varepsilon\text{-greedy}(s,Q)$
- (c) Execute action a; observe resultant state, s', and reward, r
- (d) $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- (e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
- (f) Repeat N times:
  - $s \leftarrow$ random previously observed state
  - $a \leftarrow$ random action previously taken in s
  - $s', r \leftarrow Model(s,a)$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Dyna-Q (2)



G
S
actions
Steps per episode
800
600
400
200
14
0 planning steps (direct RL only)
5 planning steps
50 planning steps
2
10
20
30
40
50
Episodes

Vanilla model-based reinforcement learning

- Repeat:
  - Take exploratory action (based on greedy policy)
  - Use resulting immediate reward and state to update a *maximum-likelihood model*:

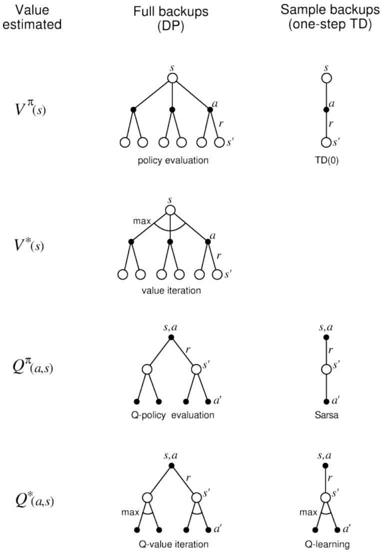
$$\hat{P}^a_{ss'} = \frac{n^a_{ss'}}{n^a_s}, \hat{R}^a_{ss'} = \frac{1}{n^a_{ss'}} \sum_{i=1}^{n^a_{ss'}} r_i$$

  - Solve the model using value iteration
  - Update greedy policy
- Computationally expensive
- But don't have to plan to convergence or plan on every step

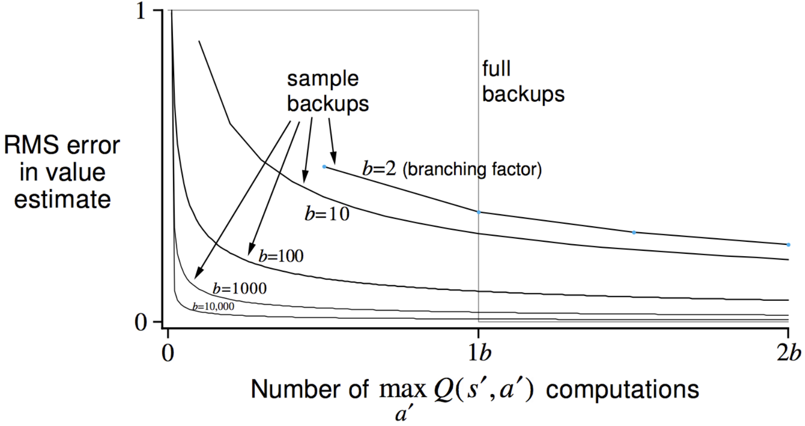
R_{max}

- Use vanilla model-based RL
- However, for all (s, a) for which $n_s^a < m$:
  - Remove all transitions from (s, a) from model
  - Add transition of prob. 1 to artificial, terminal jackpot state
  - Immediate reward on this transition is R_{max}
- Plan on altered model
- Remove artificial transitions once $n_s^a \geq m$
- Agent will plan how to visit insufficiently visited states: efficient exploration

Full versus sample backups (1)



Full versus sample backups (2)



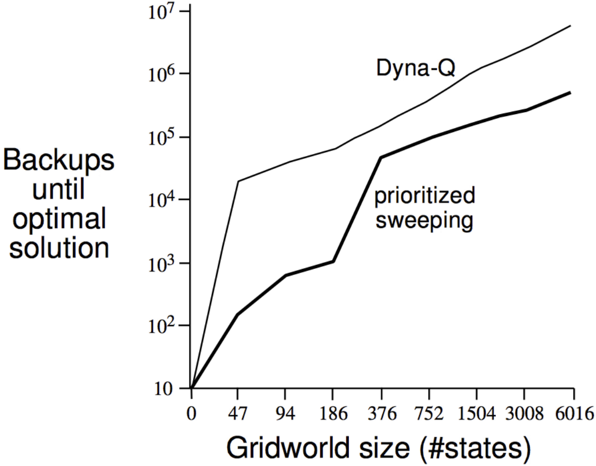
Prioritised sweeping (1)

- Which states or state-action pairs should be generated during planning?
- Work backwards from states whose values have just changed:
- Maintain a queue of state-action pairs whose values would change a lot if backed up, prioritized by the size of the change
- When a new backup occurs, insert predecessors according to their priorities
- Always perform backups from first in queue

Prioritised sweeping (2)

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Do forever:
- (a) $s \leftarrow$ current (nonterminal) state
- (b) $a \leftarrow policy(s,Q)$
- (c) Execute action a; observe resultant state, s', and reward, r
- (d) $Model(s,a) \leftarrow s', r$
- (e) $p \leftarrow |r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$.
- (f) if $p > \theta$, then insert s,a into $PQueue$ with priority p
- (g) Repeat N times, while $PQueue$ is not empty:
  - $s,a \leftarrow first(PQueue)$
  - $s',r \leftarrow Model(s,a)$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  - Repeat, for all $\bar{s},\bar{a}$ predicted to lead to s:
    - $\bar{r} \leftarrow$ predicted reward
    - $p \leftarrow |\bar{r} + \gamma \max_a Q(s,a) - Q(\bar{s},\bar{a})|$.
    - if $p > \theta$ then insert $\bar{s},\bar{a}$ into $PQueue$ with priority p

Prioritised sweeping (3)



Reinforcement learning theory (1)

Model-free temporal difference methods such as Q-learning and Sarsa are guaranteed to converge to the optimal policy in the limit under the following conditions:

1. S and A are finite
2. $\sum_t \alpha_t^{sa} = \infty$ and $\sum_t (\alpha_t^{sa})^2 < \infty$
3. $Var\{R_a^{ss'}\} < \infty$
4. $\gamma < 1$

Reinforcement learning theory (2)

R_{max} is an example of a *PAC-MDP* algorithm, for which the following *probably approximately correct* guarantee holds:

- Let A be a PAC-MDP algorithm and A_t be the policy of A at timestep t
- *Sample complexity* of A is the number of timesteps t such that $V^{A_t}(s_t) < V^*(s_t) - \epsilon$
- With probability at least $1 - \delta$, the sample complexity of A is less than some polynomial in the quantities $(|S|, |A|, R_{\max}, 1/\epsilon, 1/\delta, 1/(1-\gamma))$

Reinforcement learning theory (3)

- PAC guarantees are very general but only apply to states the agent actually visits: do not consider that exploration phase may have doomed the agent to a "hell" region.
- Stronger but less general guarantees are possible by bounding the *regret*: the expected cumulative return of an optimal policy minus the cumulative return of the algorithm
- Bounding regret requires making *reachability* assumptions, e.g., UCRL2 has reget linear in the *diameter*: the maximum average number of steps needed to reach any s' from any s

Reinforcement learning theory (4)

- In principle, we can compute a *Bayes-optimal* policy for balancing exploration and exploitation
- Problem of learning in an MDP is cast as one of planning in a POMDP where the hidden state corresponds to the unknown model parameters: $s_{POMDP} = (s_{MDP}, T, R)$
- We will return to this idea when we have studied POMDPs

Pseudocounts [Bellemare et al. 2016]

- Let $\hat{\mu}(\mathrm{s})$ be a generative model of the on-policy distribution $\mu(s)$
- Let $\hat{\mu}'(\mathrm{s})$ be the updated model after a new visit to s
- Suppose that $\hat{\mu}$ was count-based such that

$$\hat{\mu}(s) = \frac{c(s)}{C} \qquad \hat{\mu}'(s) = \frac{c(s)+1}{C+1}$$

  where $c(s)$ is the number visits to s and C is the total state visits

- Solve this linear system to find $c(s)$ and C
- Give a bonus inversely proportional to pseudocount

TreeQN [Farquhar et al. 2017]