2534 Lecture 7: Reinforcement Learning (Brief Introduction)

- Structured Models of MDPs and MDP approximations
  - (wrap up from last time)
- Introduction to Reinforcement Learning
- Announcements
  - “Preliminary” project proposals due today if you want feedback before the Nov.4 deadline.
  - For those who didn’t meet me last week: slots today
    - 20 minute time slots (come prepared)

Reinforcement Learning: What is it?

- Reinforcement learning: a number of different views
  - Psychology: operant conditioning (Konorkski, Skinner)
  - Neuroscience
  - Statistics: bandit problems
  - OR/control engineering
- From our perspective:
  - Somewhat abstractly: learning how to behave/act based on interactions with environment
  - More precisely: solving a (PO)MDP without model “specified in advance”

Reinforcement Learning: Why?

- Stochastic planning problems where model (transition probabilities, rewards, *even state space*) is unknown, but information can be gleaned while interacting with the environment
  - Robotics, adaptive control (industrial processes, queues, networks), logistics, etc.
- Problems where *simulation models* are available (e.g., return $s^{(t+1)}$, $r^{(t+1)}$ given $s^{(t)}$, $a^{(t)}$), but explicit transition models/rewards not given
- Problems with large state spaces where explicit dynamic programming is intractable/infeasible

Reinforcement Learning: Basic Picture

- Agent acting in a (we'll assume finite) MDP *M=(S, A, P, R)*
  - Assume state, action spaces known
  - Assume transition model *P* and reward function *R* unknown
- Agent can take actions at each time *t*:
  - Collects a series of observations $s^{(t)}, a^{(t)}, r^{(t)}, s^{(t+1)}$
- *Key question:* what is "best" action to take at time *t* given all past observations:

$$\{s^{(k)}, a^{(k)}, r^{(k)}, s^{(k+1)} : k < t\}$$

A First Simple Approach

- One simple approach:
  - Collect samples for "sufficient period of time" (data/learning phase)
  - Build an approximate model of the MDP (say $\widetilde{M}$)
    - Estimate transitions *P(s,a,t)* and *R(s,a)* using empirical observed distributions (or update some prior beliefs using these obsvtn's)
  - Solve $\widetilde{M} = (S, A, \tilde{P}, \tilde{R})$ using standard technique (e.g., policy itert'n)
- Drawbacks (or at least "issues")? There are many:
  - Performance during learning phase?
  - How long should learning phase be to have high-quality est. of $\widetilde{M}$?
  - How do you ensure the data collected is *sufficient* to learn $\widetilde{M}$?
    - Transitions not just handed to you: you actively collect them--- the *exploration problem* (need to cover *(s,a)*-space)
  - Why not use partially learned model during phase 1? Why stop learning during phase 2?

Model-based vs. Model-free RL

- The *model-based approach* above does have some value
  - If learning to act in a simulated model, exploration matters; but performance while learning less critical (apart from efficiency)
  - If learning to act in real-world, one should address *all of the issues above*, but this can still be a valuable approach
- *Model-free methods* are, however, far more common
  - Rough idea: based on interactions with environment, attempt to directly learn the MDP value function (and/or MDP optimal policy)
- We'll focus on model-free methods, but mention issues and usefulness of model-based methods at the end

Learning a Value Function

- Simpler problem: learn V^{π} for fixed (stationary) policy π
  - Policy may be deterministic or stochastic, but regardless, induces Markov chain and *Markov reward process (MRP)*
  - Predict expected sum of discounted rewards from any state s
  - Given data $s^{(0)}, r^{(0)}, s^{(1)}, r^{(1)}, \ldots, s^{(t)}, r^{(t)}, \ldots$
- Sometimes called *value prediction problem*

Aside: Robbins-Munro Stochastic Approximation

- Given samples of random variable *V*: $\{r^{(t)}: t \leq N\}$
- Typically estimate mean of *V* as: $\tilde{V} = \frac{1}{N}\sum_t^N r^{(t)}$
- Can instead estimate online as data collected
  - Estimate at time *t* (*t* th data point):

$$\tilde{V}^{(t)} = \tilde{V}^{(t-1)} + \alpha_t(r^{(t)} - \tilde{V}^{(t-1)}) \quad = (1-\alpha_t)\tilde{V}^{(t-1)} + \alpha_t r^{(t)}$$

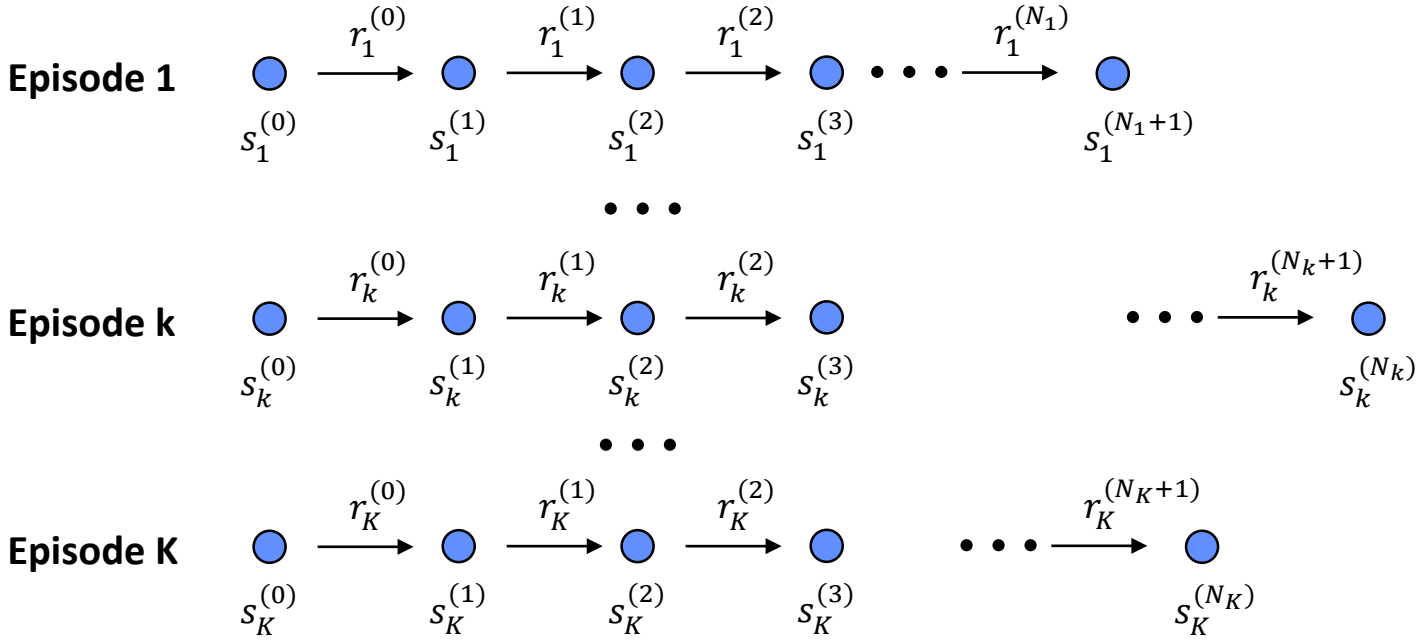
  - Refer to $r^{(t)} - \tilde{V}^{(t-1)}$ as "error", $0 < \alpha_t < 1$ is "step size"
- This converges to true mean of *V* almost surely if

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad and \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

E.g., let $\alpha_t = \frac{1}{t}$

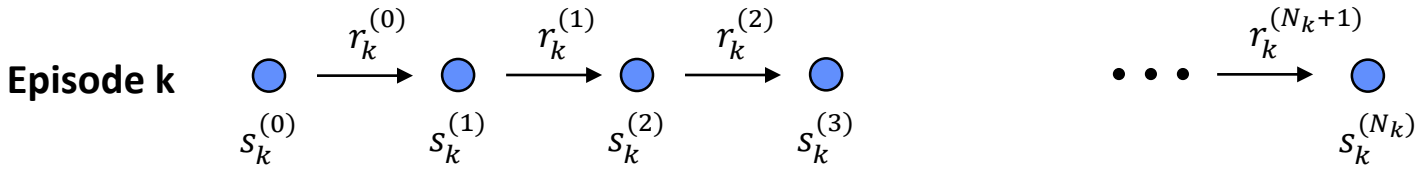
Episodic MDPs

- To start, assume an "episodic" MDP:
  - (any) policy eventually reaches a zero-reward absorbing state, then restart MDP in a new initial state (with some distribution P^0)
  - So we get distinct episodes/trajectories with "restarts"
  - Assume *K* such episodes (of varying, stochastic length)



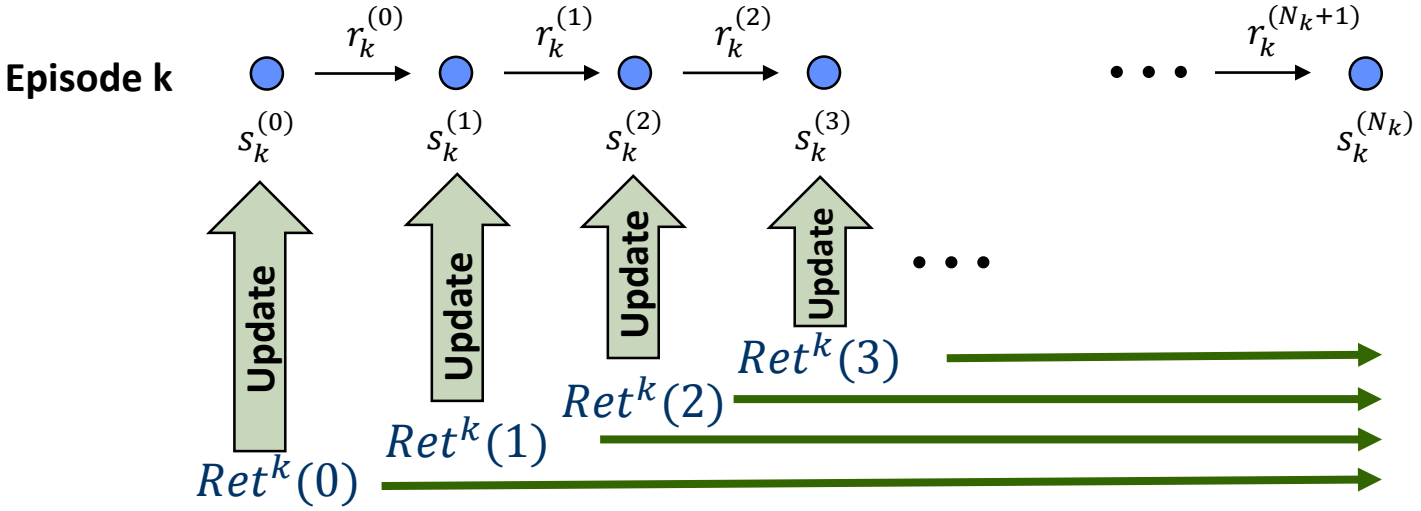
Monte Carlo Value Prediction (Episodic MDPs)

- Simple method: after each episode *k*
  - Let return be discounted sum of rewards: $Ret^k = \sum_{t=0}^{N_k} \gamma^t \, r^{(t)}$
  - Update $\tilde{V}^\pi(s_k^{(0)}) \leftarrow \tilde{V}^\pi(s_k^{(0)}) + \alpha(Ret^k - \tilde{V}^\pi(s_k^{(0)}))$
  - Intuition: Return of episode *k* is a new sample of expected value of policy when starting at state $s_k^{(0)}$
- Some inefficiencies and problems:
  - Not all MDPs are episodic
  - Does not update state values frequently: only at end of episodes
  - Does not update values of "intermediate" states on trajectory



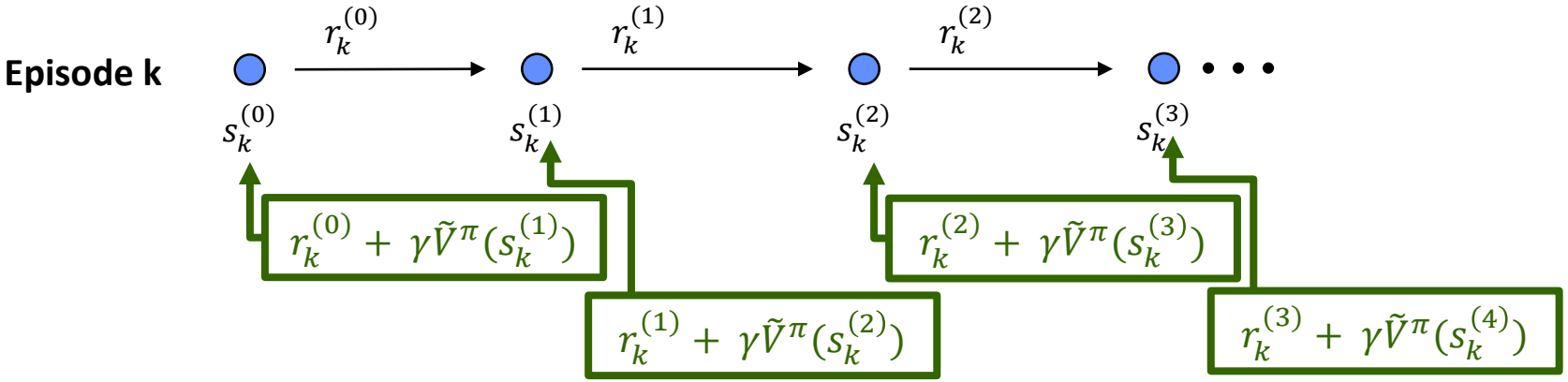
Every Visit Monte Carlo Estimation

- Tackles 3rd problem
- Simple method: after each episode *k*
  - Define return for *each* state in episode: $Ret^k(t) = \sum_{i=t}^{N_k} \gamma^i \, r_k^{(i)}$
  - Update: $\tilde{V}^\pi(s_k^{(t)}) \leftarrow \tilde{V}^\pi(s_k^{(t)}) + \alpha(Ret^k(t) - \tilde{V}^\pi(s_k^{(t)}))$
- Update Intuition: *Each visit* of $s_k^{(t)}$ provides a sample of expected value of policy when starting at state $s_k^{(t)}$



Temporal Difference Learning

- Tackles 2nd problem: don't wait until end of episode
  - *Bootstrap* prediction of current state using predicted val of next state
- After each transition $(s_k^{(t)}, r_k^{(t)}, s_k^{(t+1)})$
  - Define temporal difference: $TD^{k,t} = r_k^{(t)} + \gamma \tilde{V}^{\pi}(s_k^{(t+1)}) - \tilde{V}^{\pi}(s_k^{(t)})$
  - Update $\tilde{V}^{\pi}(s_k^{(t)}) \leftarrow \tilde{V}^{\pi}(s_k^{(t)}) + \alpha \cdot TD^{k,t}$
  - Intuition: use *recursive nature* of value function (Bellman equation)



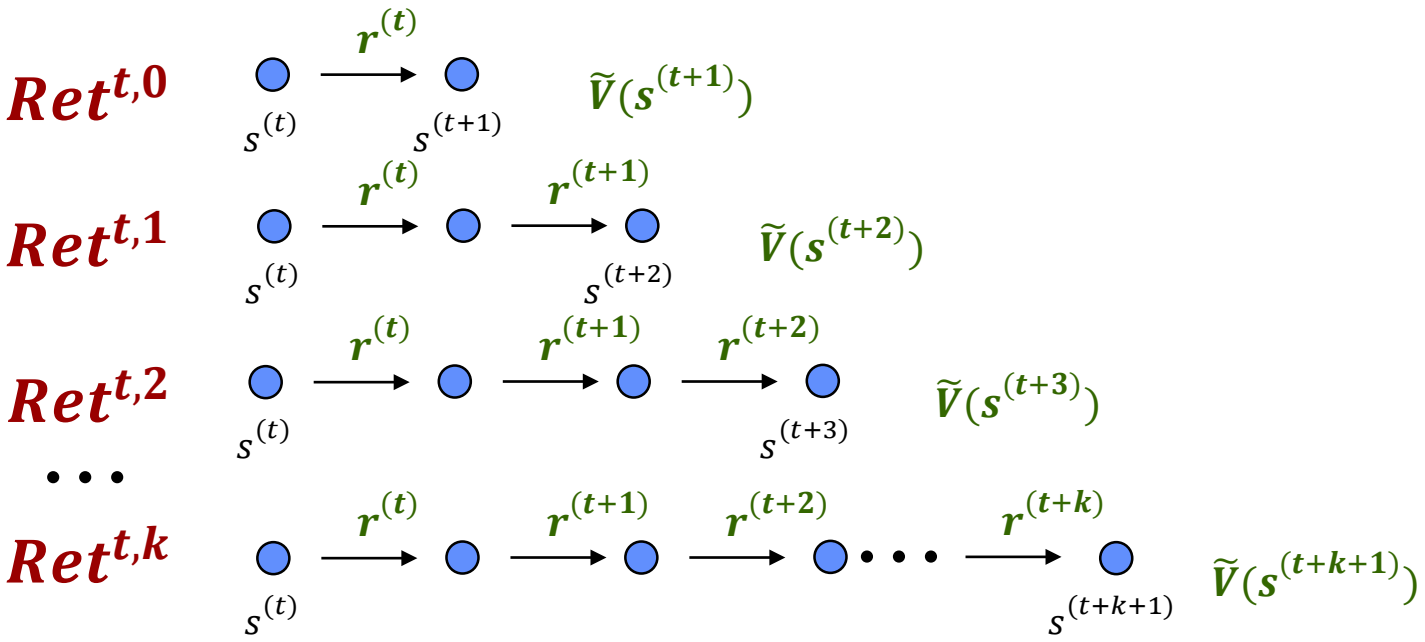
Temporal Difference Learning: TD(0)

- Method above is sometimes known as TD(0)
- Core of most approaches to reinforcement learning
  - Allows full online updates
  - Note: no reliance on episodic MDPs
  - Will converge to true value of stationary policy under mild conditions (e.g., ergodicity)
  - Very simple to implement

Temporal Difference Learning: TD(λ)

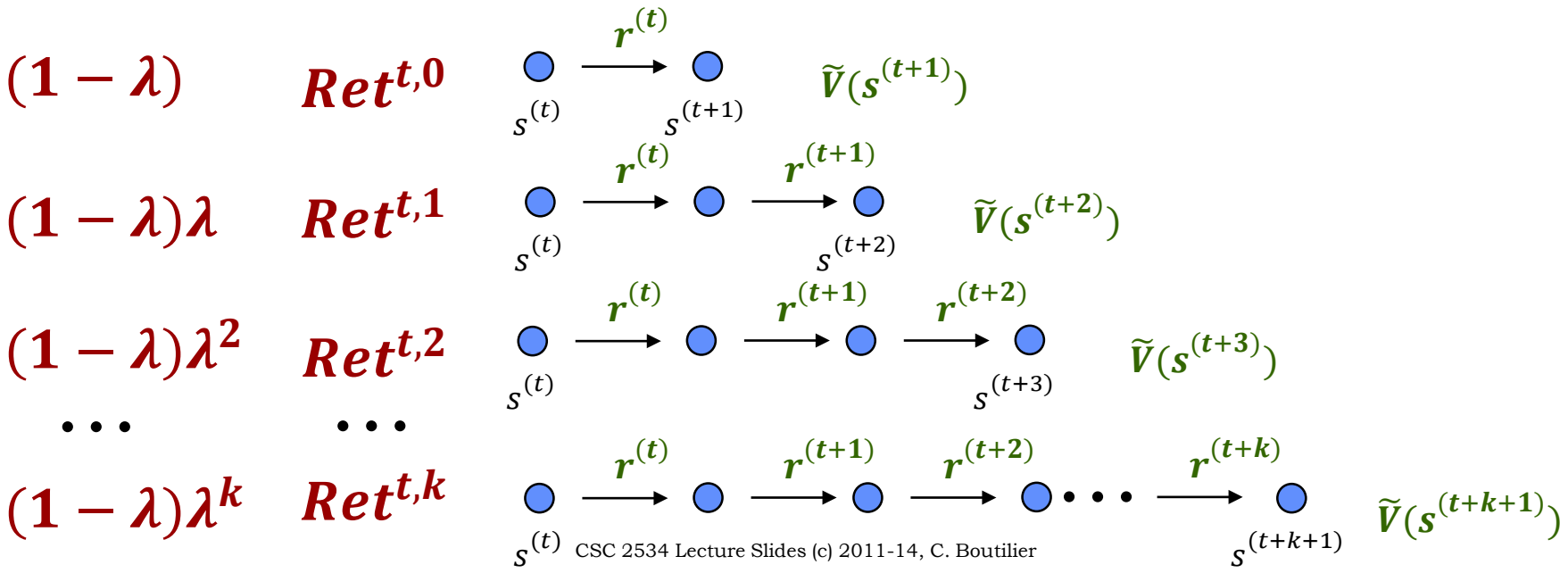
- Monte Carlo and TD(0) are extreme points
  - MC uses no bootstrapping: only *actual* rewards recv'd in trajectory
  - TD(0) uses only immediate reward, future value "bootstrapped"
- TD(λ) uses weighted combination "multi-step returns"

$$Ret^{t,k} = \sum_{i=t}^{t+k} \gamma^{i-t} r^{(i)} + \gamma^{k+1} \tilde{V}(s^{(t+k+1)})$$



Temporal Difference Learning: TD(λ)

- TD(λ) uses weighted combination "multi-step returns"
  - Target Value of $s^{(t)}$: $\mathrm{Tar}(s^{(t)}) = (1-\lambda)\sum_{k=0}^{\infty}\lambda^k Ret^{t,k}$
  - Update $\tilde{V}^{\pi}(s^{(t)}) \leftarrow \tilde{V}^{\pi}(s^{(t)}) + \alpha[\mathrm{Tar}(s^{(t)}) - \tilde{V}^{\pi}(s^{(t)})]$
- TD(0) is what we've seen before, TD(1) is MC (if episodic, def'n above needs a "terminal" value)
  - Can adjust λ over time (from larger to smaller values)



Eligibility Traces

- TD implemented using the notion of *eligibility traces*:
  - Don't wait until you see all returns before updating $\widetilde{V}^{\pi}(s^{(t)})$
  - Update incrementally, by keeping track of "eligibility score": how much should any past-seen state be updated based on most recently observed temporal difference
  - At any point in time, current estimated value is based on current set of truncated eligibility traces
- For details see Csepesvari (or Sutton and Barto)

Control: Deciding How to Act

- Value prediction OK for fixed policy
  - …but we want to *find an optimal policy*
- First idea:
  - Use value prediction to evaluate a policy
  - Use policy improvement step (like policy iteration) to change policy given current estimated values
  - Seems OK but requires some significant assumptions to make work in practice…

Learning Approach to Policy Iteration

- Hypothetically, we could:
  - Execute a policy π for a while, learn an estimated value function $\tilde{V}^{\pi}$ using a TD method
  - Improve policy (find greedy policy w.r.t. $\tilde{V}^{\pi}$):
    - $\pi(s) \leftarrow \text{argmax}_{\,a} \{R(s,a) + \sum_{s'} P(s,a,s')\, \tilde{V}^{\pi}(s')\}$
    - equivalently: $\pi(s) \leftarrow \text{argmax}_{\,a} \{\tilde{Q}^{\pi}(s,a)\}$
  - Repeat until convergence
- Works in principle even if value estimate is approximate
  - E.g., see our discussion of modified policy iteration
- Relies only on being able to estimate greedy policy
  - But this requires *we know a model* to estimate $\tilde{Q}^{\pi}(s,a)$ given $\tilde{V}^{\pi}$

Issues: No Model, Exploration

- With no model, perhaps we could estimate $\tilde{Q}^{\pi}(s,a)$?
  - But no reason policy π would ever do action *a* state *s*
  - We could explicitly explore ("exploring starts") in episodic MDPs
    - At episode start, try arbitrary action with some probability
  - If policy is stochastic and tries each *(s,a)*-pair with positive probability, then we can estimate $\tilde{Q}^{\pi}$ directly
- In either case, we see that we (usually) need to incorporate explicit exploration actions into our policy in order to ensure we "learn enough" about all actions to engage in policy improvement
  - We'll come back to exploration-exploitation tradeoff shortly

Off-policy vs. On-policy RL

- Method above is an *on-policy method*:
  - Learns values (*V,Q*) of specific policy being executed
- *Off-policy method:* learns value (*V,Q*) of policy different from that being executed (ideally, optimal policy)
- Methods above adaptable to be (inefficiently) off-policy

Q-learning

- *Q-learning:* very simple, powerful off-policy method
  - Learn *Q-function* for optimal policy directly
  - Probably most important practical insight for AI-based RL
  - Very simple update rule: sample-based Bellman backup

$$\tilde{Q}\left(s^{(t)}, a^{(t)}\right) \leftarrow \tilde{Q}\left(s^{(t)}, a^{(t)}\right) + \alpha \left[r^{(t)} + \gamma \cdot \max_{a'} \tilde{Q}\left(s^{(t+1)}, a'\right) - \tilde{Q}\left(s^{(t)}, a^{(t)}\right)\right]$$

- Q-learning guaranteed to converge to optimal Q-values if all actions tried infinitely often and learning rate set according to RM conditions
- Can be extended to multi-step returns ($Q(\lambda)$), etc.

Exploration-Exploitation Tradeoff

- To ensure convergence, RL methods need to sample all actions at every state sufficiently often
  - At odds with acting optimally given current estimated $\tilde{V}$
- Exploration-exploitation tradeoff
  - Exploit: execute best estimated action?
  - Explore: try a new action with lower estimated value, but for which I may learn something useful?

Exploration Methods

Assume *n* actions, *a** optimal at current state *s*, estimated *Q*

- *Epsilon-greedy:* choose a* with prob. 1-ε, others with prob. 1/ε(n-1)
- *Boltzmann (softmax):* execute each action with prob $\frac{1}{Z} e^{Q(s,a)\cdot\tau^{-1}}$
  - τ is a temperature parameter: $\tau \to \infty$ means all actions equally likely, $\tau \to 0$ means *a** will be chosen with near certainty
  - 1/Z is normalization term
- *Confidence-based methods:*
  - Roughly: use statistical tests to estimate a confidence bounds on estimated Q-values, then choose actions based on mean values but with an "exploration bonus" if high uncertainty (e.g., large upper confidence bound)
  - Many heuristic and theoretically sound variants: interval-estimation, UCT, "optimism in the face of uncertainty," …

Function Approximation

- As with MDPs, RL methods will be wildly intractable if state (action) space is large
- Problem not just computational: we need a lot of data (i.e., *(s,a,r,s)* samples), not just computation
- The ability to generalize predicted values from one state, or *(s,a)*-pair, to others invaluable for computation and to derive maximum benefit from experience
  - E.g., the *first time* you visit state *s*, it would be nice to estimate its value from other "similar" states

Some approaches

- K-nearest neighbors (memory-based RL)
  - Predict value of a state *s* based on the values of nearby states (even if *s* has been visited in the past)
  - Issues: need similarity metric (sometimes natural), computational search for NNs (esp. in high-dim. state space)
  - Other similarity kernels can be used…
- Gradient descent (or other learning method) on some differentiable approx. function to reduce error in prediction
  - Neural network, linear function approximator, etc.

Linear Function Approximation of V

- For instance, suppose $\tilde{V}(s) = \boldsymbol{w}\boldsymbol{\theta}(s)$
  - Feature vector (basis f'ns) $\boldsymbol{\theta}(s)$, weight vector $\boldsymbol{w}$
  - Train weight vector using RMSE of V, using current target value *tar(s)*, e.g. $\text{tar}(s^{(t)}) = r^{(t)} + \gamma \tilde{V}^{\pi}(s^{(t+1)}; \boldsymbol{w})$

$$w \leftarrow w + \alpha[\text{tar}(s^{(t)}) - \tilde{V}^{\pi}(s^{(t)}; w)]\nabla_w \tilde{V}^{\pi}(s^{(t)}; w)$$

- In linear case:
  - $\nabla_w \tilde{V}^{\pi}(s; w) = \boldsymbol{\theta}(s)$ is simple to work with
  - Under certain conditions, TD($\boldsymbol{\lambda}$), Q, can be proven to converge
  - Of course, it is still an approximation, since true VF may not be representable in liner fashion
- Beyond linear, few theoretical results (but very useful)

Model-based RL

- Use experience tuple to estimate MDP model
- Advantage: you can update state value predictions using dynamic programming with approximate model
  - Often can wring much more value out of existing experience rather than waiting for state visits in real-world (or simulation)
- Uncertainty in model can be exploited to schedule DP backups in an effective fashion:
  - E.g., prioritized sweeping

Bayesian RL

- Bayesian reinforcement learning
  - Have a prior distribution over model parameters
  - Update distribution based on observed transitions
  - Choose actions with greatest expected long-term value
- In principle, solves the exploration-exploitation problem optimally
- In practice, very difficult to work with computationally
  - Large POMDP, with high-dim. continuous state space (model parameters)
  - Approximations and heuristics often show very good performance: tractability is largest bottleneck