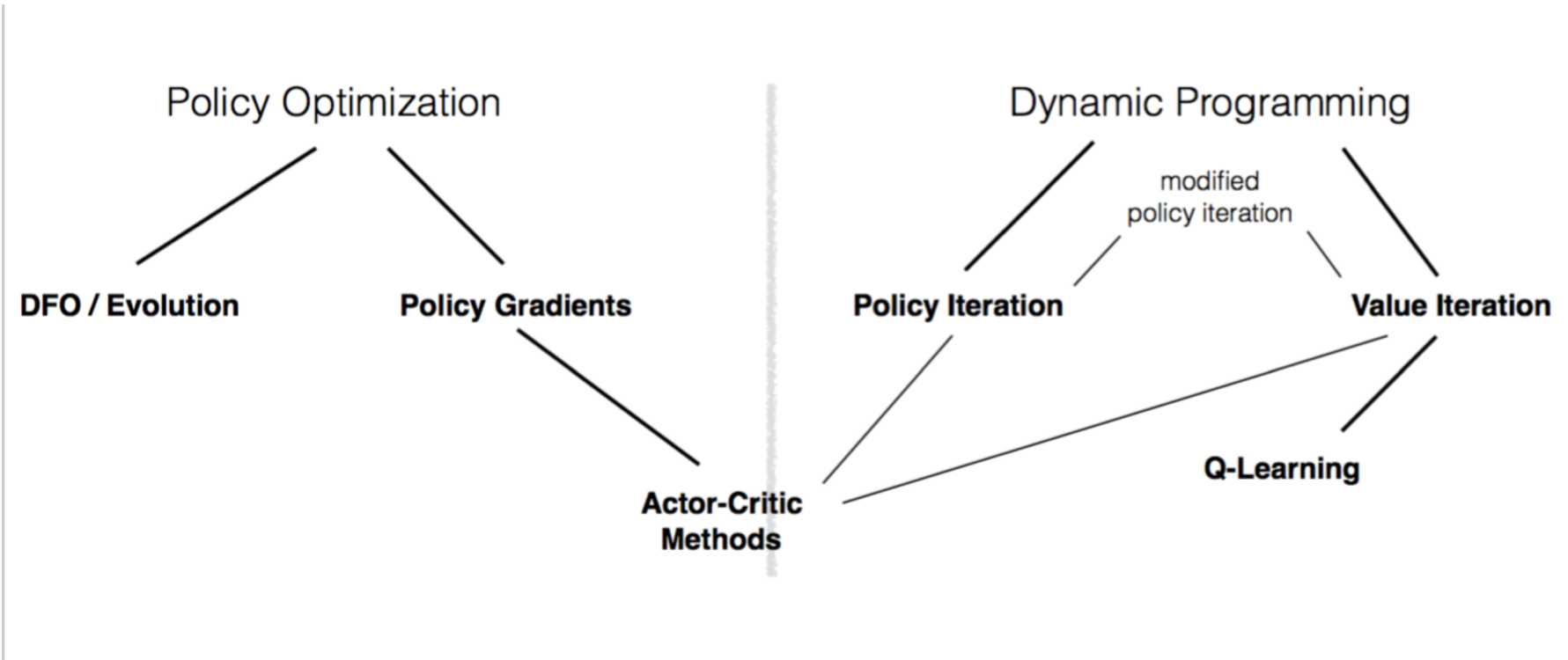
Deep Reinforcement Learning

Ryan Brigden

11-785 Recitation 13

November 30, 2018

Reinforcement Learning Landscape



[1]

[1] Diagram courtesy of Pieter Abbeel

What We Have Considered So Far...

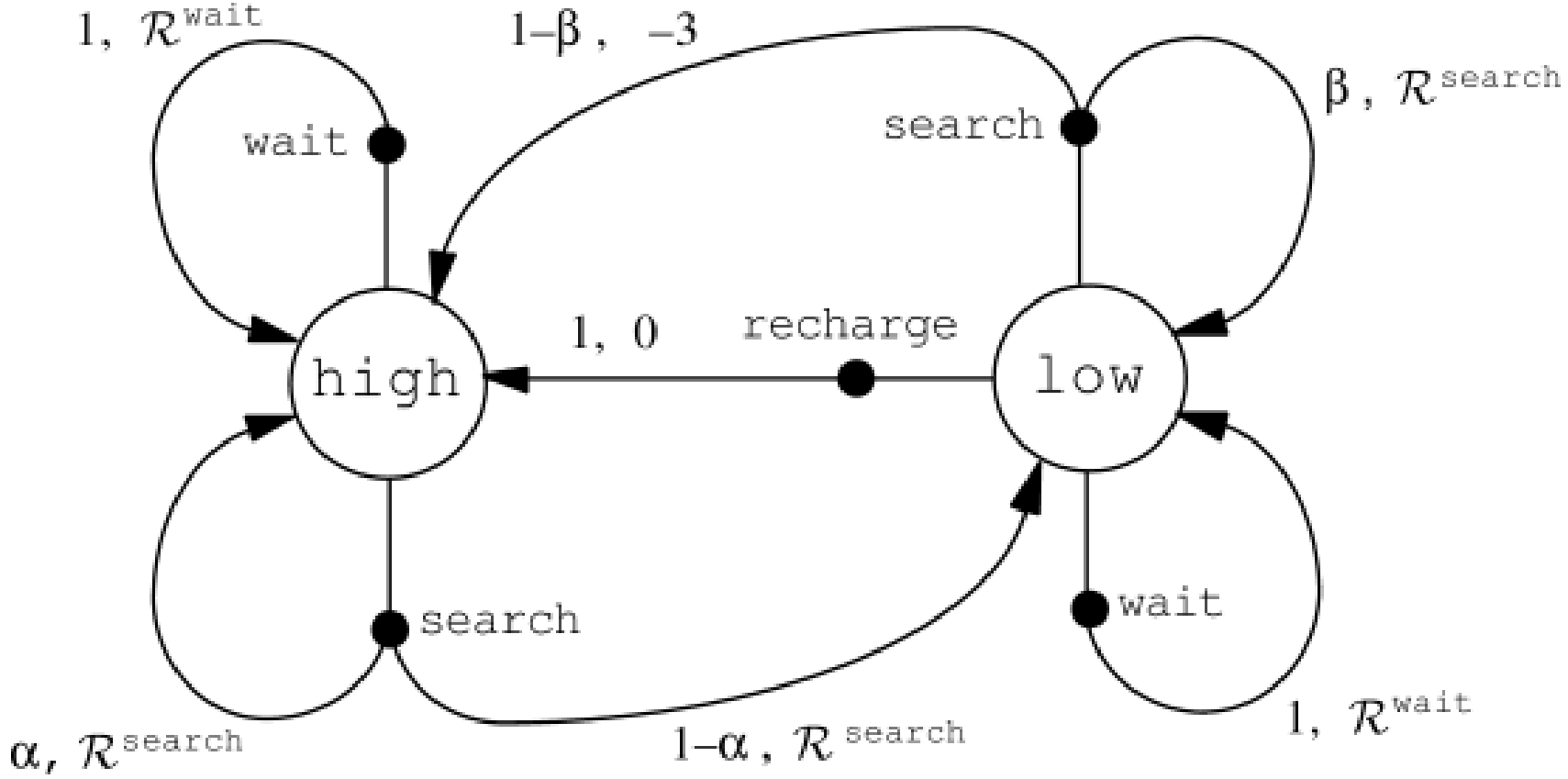
Exact MDP solution methods

- Policy Iteration
  - Initialize policy π_k (randomly) and value table V_0^π
  - Evaluate policy (fill in V_{k+1}^π)
  - Update policy based on V_{k+1}^π
  - Eventually $\pi_k \in \pi^*$
- Value Iteration
  - Why do we need to iteratively refine a policy?
  - Let's just directly compute V^*

Policy Iteration vs. Value Iteration

- Policy Iteration
  - Two steps (evaluate, then up)
  - Computational cheaper (don't consider every action, go with the policy)
  - Policy often converges before the values
- Value Iteration
  - Estimate the value function directly without constructing a separate policy (can have simpler implementation)
  - Extract optimal policy upon convergence
  - Need to consider every action for each transition $\mathcal{O}(|A||S|^2)$

Giving up on the Model



- Models are powerful because we have methods to solve them exactly.
- Error can propagate through the model when our modelled dynamics are a bit off.
- Constructing a model becomes infeasible as $|S|$ and $|A|$ grow.

Focusing on the Task at Hand

- Sometimes optimizing the policy (searching directly in policy space) can be more effective for a given task.
- The state-value function doesn't prescribe actions and a controller operating under an estimate of V would need a complete dynamics model.
- Estimating Q is useful because it can be used for off-policy control, but this can be challenging for continuous and high-dimensional action spaces. It may also be indirect and less sample efficient.

Focusing on the Task at Hand

- Instead, let's just optimize directly on maximizing expected reward over trajectories.

$$J(\theta) = \max_{\theta} \mathsf{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

  Where $\pi_\theta(u|s)$ is a parameterized policy and H is the horizon.
- Note the and change in convention here – u is the action conditioned on the state s. Something to look out for in policy gradient literature.
- We opt for a soft (stochastic) policy because it smooths the problem and allows us to use back-propagation to optimize the policy directly.

The Policy Gradient

Now we can formalize this some more

- Let τ be the state-action sequence $s_0, u_0, ..., s_H, u_H$, also known as the sample trajectory of the agent acting in the environment.
- We will more informally call arbitrary fixed-length sub-sequences of a trajectory rollouts.
- For a given trajectory, we denote the cumulative reward (undiscounted) as

$$R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$$

This is mainly to simplify notation.

The Policy Gradient

We can rewrite our objective more simply. Observe that this expectation is over a distribution dependent on the policy parameters θ.

$$J(\theta) = \max_{\theta} \mathsf{E}_{\tau \ \ \pi_\theta}[R(\tau)]$$

The Policy Gradient

- Ultimately what we want is to maximize the expected return accrued under a policy parameterized by θ. This is also known as the utility of the agent $U(\theta)$

$$U(\theta) = \mathsf{E}[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

- It would be very nice if we could maximize the utility via gradient descent...

The Policy Gradient

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \tag{1}$$

$$= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \tag{2}$$

$$= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \tag{3}$$

$$= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \tag{4}$$

The Policy Gradient

$$\nabla_\theta U(\theta) = \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau)$$

Recall that...

$$\frac{\partial}{\partial x} \log f(x) = \frac{1}{f(x)} \left(\frac{\partial}{\partial x} f(x) \right)$$

$$) \ \nabla_\theta U(\theta) = \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)$$

The Policy Gradient

$$\nabla_\theta U(\theta) = \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)R(\tau)$$

We can estimate this expectation empirically by doing m rollouts of π_θ and observing $\tau^{(i)}$ trajectories

$$\nabla_\theta U(\theta) \approx \sum_{i=1}^{m} P(\tau^{(i)};\theta)\ \nabla_\theta \log P(\tau^{(i)};\theta)R(\tau^{(i)})$$

The Policy Gradient

$$\nabla_\theta U(\theta) \approx \sum_{i=1}^{m} P(\tau^{(i)};\theta)\ \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

- The intuition behind this result (known as the Likelihood Ratio Gradient) is that it is not trying to change the trajectories themselves, but increases the probability of taking a trajectory $\tau^{(i)}$ proportional to R.

The Policy Gradient

This is grand but we are left with a serious problem. $P(\tau^{(i)}; \theta)$ is the probability of following trajectory $\tau^{(i)}$ under the policy π_θ. This quantity is dependent on a dynamics model, which we cannot obtain in an environment with intractable state/action space. Let's keep working on this.

The Policy Gradient

We can start by decomposing the trajectory into transition probabilities from s_0 to s_H for a trajectory $\tau^{(i)}$ under policy π_θ.

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log [\prod_{t=0}^{H} P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) \cdot \pi_\theta(u_t^{(i)} | s_t^{(i)})] \\
&= \nabla_\theta [\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) \cdot \pi_\theta(u_t^{(i)} | s_t^{(i)})] \\
&= \nabla_\theta [\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \log \pi_\theta(u_t^{(i)} | s_t^{(i)})] \\
&= \nabla_\theta [\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)})]
\end{aligned}$$

The Policy Gradient

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta [\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)})]$$

Notice that the sum of log transition function is not dependent on the parameter θ, so when we take the gradient of this term w.r.t θ, this term becomes zero.

$$\begin{aligned} \nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta [\sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)})] \\ &= \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \end{aligned}$$

The Policy Gradient

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$$

We no longer need a dynamics model and can learn from sampled experiences!

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \ \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

where,

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$$

Variance Reduction

- As formulated, this estimate of the gradient is unbiased, that is, but it is high variance.
- This is one of the great problems and limiting factor of policy gradient methods.
- Some ideas
  - Subtract off a baseline to increase the log probability of an action proportionally to how much its returns are better than the expected return under the current policy (also known as the advantage).
  - Modify the temporal structure, including removing terms that don't depend on the current state/action (we are assuming a markov state).

Variance Reduction

- Let's first consider the addition of a baseline

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) \cdot (R(\tau^{(i)}) - b)$$

- We can remove terms that do not depend upon the current action (invoking our assumption of the markov property). Empirically this can lower the variance. Let's write everything out first can then update the formulation to reflect this change...

$$\nabla_\theta U(\theta) \approx \hat{g}$$

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \cdot (\sum_{k=t}^{H} R(s_k^{(i)}, u_k^{(i)}) - b(s_t^{(i)}))$$

Variance Reduction

Now what should we choose for our baseline? One intuitive choice is the state-dependency expected return for the state at t, $V^{\pi}(s_t)$.

$$b(s_t) = \mathsf{E}[r_t + r_{t+1} + r_{t+2} + ... + r_H] = V^{\pi}(s_t)$$

- The quantity $\sum_{k=t}^{H} R(s_k^{(i)}, u_k^{(i)}) - V^{\pi}(s_k^{(i)})$ can be thought of as how much better our actual returns from the trajectory were from time t than our estimate of $V^{\pi}(s_t)$.
- As a convention, we refer to this use of the state-value function as the critic of our agent's (ie actor) policy, which gives rise to the name of this family of algorithms, Actor-Critic methods. It is also common to refer to this quantity as the advantage A, a term that follows its intuitive meaning.

Adding the Critic In

By adding the state-value function (our critic) as a baseline, we now need to estimate it. Similar to the policy, we can parameterize it with parameters ϕ. Now we need an update rule that makes sense. A simple regression will often do the trick.

$$\phi_{i+1} \leftarrow \min_{\phi} \sum_{s,u,s',r} \|r + V^{\pi}_{\phi_i}(s^0)\| - V_{\phi}(s)\|_2^2 + \lambda\|\phi - \phi_i\|_2^2$$

Vanilla Policy Gradient Algorithm

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b

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

Collect a set of trajectories by executing the current policy

At each timestep in each trajectory, compute

the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,

summed over all trajectories and timesteps.

Update the policy, using a policy gradient estimate $\hat{g}$,

which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta) \hat{A}_t$

end for

Function Approximation

- Parametric policies and value function approximators have been studied since the inception of RL.
- Updates in parameter space have no guarentee about the change in the function space.
- (Stochastic) gradient descent assumes stationary data distribution and i.i.d data (decorrelated updates).
- With the success of AlexNet and new computing platforms (GPUs) in the later 2000s early 2010s, deep networks were applied to RL and found success.

The Deep RL 'Hack'

Premise: Take a set of optimization algorithms (stochastic gradient descent) with well established guarantees and apply them to function approximation in an online learning task with non-stationary targets and non-iid data...

Much of the research in deep reinforcement learning concerns addressing learning problems induced by the violation of supervised learning.

The Case for End-to-End Control

Asynchronous Actor Critic (A2C)

Popular Actor-Critic algorithm that combines computational efficiency with improved decorrelation of updates.

1. Maintain π_θ and V^ϕ as two separate networks or a single network with two heads.
2. Run N environment simulators in parallel executing π_θ for T_r timesteps (i.e. one rollout r)
3. Compute advantage estimate with $\hat{A}_t = R_t - V_\phi$, where R is computed using the sample return bootstrapped with V_ϕ at T_r (taking into account termination before T_r in episodic environments).
4. Update policy according to policy gradient using $\hat{A}$. Update value function as usual.