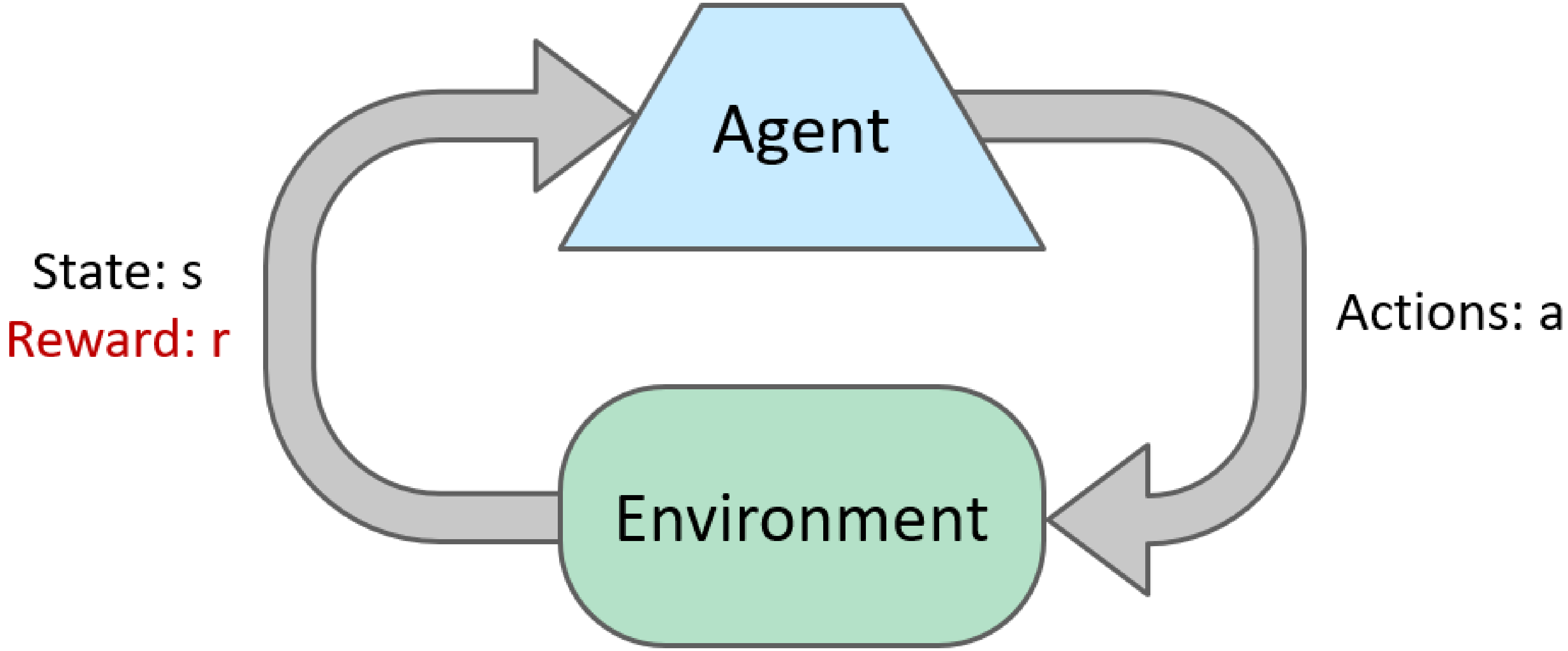
Policy gradient reinforcement learning

Some material on these slides are derived from the slides created by David Silver for University College London, available at https://davidstarsilver.wordpress.com/ .


Agent
State: s
Reward: r
Actions: a
Environment

RL model

- We assume a world governed by an MDP
  - States $s \in S$
  - Actions A
  - System dynamics $T(s, a, s')$
  - Reward function $R(s, a, s')$
- We are looking for the policy $\pi(s)$
- But we don't know T and/or R as functions
- But if we take an action a, in state s, we can observe that we landed in s' and received reward r.

What we have done until now is primarily value-based RL

- What we have done until now is primarily value-based RL
  - We estimated values: $V(s)$, $Q(s,a)$
- Let us introduce a new value, the **advantage**
  - It shows how much a certain action is worse than the optimal action

$$A(s,a) = Q(s,a) - V(s)$$

- The whole point of Q-learning was to calculate the Q-value without having to estimate the whole MDP (the transition function T, the reward function R)
  - We estimated the Q by getting sample rewards as we were acting in the world
- Then, we used the Q value to infer the policy π

Policy gradient RL

- Can we estimate directly the policy π without going through the Q value first?
- The idea is that we have a parameterized policy $\pi_\theta(s, a)$
    - Find the best θ
    - Based on the rewards sampled from the environment.
- How do we measure the quality of the policy π_θ?
- We will define a function $J(\theta)$ that describes the quality of the policy

Describing the quality of a policy $\pi(\theta)$

- In discrete environments we can use the **start value**

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments we can use the **average value**

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V_{\pi_\theta}(s)$$

- Or the **average reward per time-step**

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R_s^a$$

- Where $d^{\pi_\theta}(s)$ is the **stationary distribution of the Markov chain** for π_θ
  - Basically: how much time are you going to spend in state s if you follow in π_θ?

Policy optimization

- Policy-based reinforcement learning is an optimization problem
  - Find θ that optimizes $J(\theta)$
- There are many approaches one can use, and not all of them use the gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
- Greater efficiency often possible using gradient descent
  - We will focus on that.

Policy gradient

- Let $J(\theta)$ be any policy objective function.
- **Policy gradient algorithms** search for a local maximum of $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

where $\Delta_\theta J(\theta)$ is the **policy gradient** and α is a step-size parameter

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

Computing gradients by finite differences

- To evaluate the policy gradient of $\pi_\theta(s, a)$
- For each dimension $k \in [1, n]$
  - Estimate the k-th partial derivative of the objective function wrt to θ
  - By perturbing θ with a small amount ϵ in the k-th dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- where u_k is the unit vector in the k-th component, 0 elsewhere
- We need to evaluate J n times for one step.
- Simple, noisy, inefficient - but sometimes effective.
- Works for arbitrary policies, even if the policy is not differentiable.

Computing the gradient analytically

- Directly calculating the objective function J is a very big pain in the neck
  - Even for the definition with the value of the start state... that is basically solving the MDP! Which we don't have!
  - And with the density function: that assumes a level of knowledge about the environment that we don't have
- So we want some way to evaluate the policy gradient without actually calculating J
- Let us assume that the policy π_θ is differentiable whenever it is non-zero

Policy gradient theorem

- Applies to start state objective, average reward and average value objective.

Theorem

For any differentiable policy $\pi_\theta(s,a)$, for any of the policy objective functions $J = J_1$, J_{avR} or $\frac{1}{1-\gamma} J_{avV}$, the policy gradient is:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \; log(\pi_\theta(s,a)) \; Q^{\pi_\theta}(s,a)\right]$$

Why is the policy gradient a big deal

- We don't have to take the gradient of the J function, we don't even have to calculate it.
- We need to calculate the gradient of the log policy - and the policy function is something that we know, because we created it.
  - Eg. it is the neural network implementing the policy.
  - And we can just use the automatic differentiator
- It works both for discrete and continuous states and actions.
- We still need a way to estimate $Q^{\pi_\theta}(s, a)$
  - Note that it is the Q of the policy, not the perfect Q^*
  - Still, how do we estimate it?

REINFORCE

- The grandfather of all policy gradient approaches (Ronald Williams, 1992)
- Idea: estimate $Q^{\pi_\theta}(s_t, a_t)$ with the **return** v_t
    - That is: consider a run that terminates in state s_T. Add up all the future rewards between current timestep t to the end: this is v_t.
    - This is a single, unbiased sample of the Q-value at that point.
- Also called
    - "vanilla policy gradient" - no other tricks in this algorithm
    - "Monte Carlo policy gradient" - because it waits to the end of the run to calculate the total reward, and uses that as a learning signal.

REINFORCE algorithm

```
function REINFORCE
   initialize theta arbitrarily
   for each episode (s1,a1, r2, ...sT−1, aT−1, rT) do
      for t=1 to T−1 do
         theta = theta + alpha ∗ nabla log pi_theta (st,at) ∗ vt
      end for
    end for
end function
```

Why is REINFORCE not the perfect solution

- We need to wait to the end of the run
  - So we need to wait to see how the run turns out before we can update the policy
  - In Q-learning, we were learning at every step
- **High variance**
  - The return of a particular run can be very different from the Q value
  - Maybe you were just unlucky... but now you changed the policy away from the correct one...
- **On-policy algorithm**
  - We need to run the current policy to gather data.
  - Compare this to Q-learning which is off-policy: you can use samples from any other agent running any policy, and learning will still work.

Reducing variance using a critic

- We can use a **critic** to estimate the action-value function of the current policy:

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor critic algorithms maintain *two* sets of parameters:
  - **Critic** updates action-value function parameters w
  - **Actor** updates policy parameters θ, in the direction suggested by the critic
- Actor-critic algorithms follow an *approximate* policy gradient:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta\, log\left(\pi_\theta(s, a)\right) Q_w(s, a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta\, log\left(\pi_\theta(s, a)\right) Q_w(s, a)$$

Reducing variance using a baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance without changing expectation:

$$\mathbb{E}_{\pi_\theta}[\nabla_t log\ \pi_\theta(s,a)B(s)] = \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \nabla_\theta \pi_\theta(s,a)B(s)$$

We can take out the B(s) from the ∇_θ because it does not depend on θ

$$= \sum_{s\in S} d^{\pi_\theta}(s)\ B(s)\nabla_\theta \sum_{a\in A} \pi_\theta(s,a)$$

the sum for all a is 1, because one of the actions will be taken

$$= \sum_{s\in S} d^{\pi_\theta}(s)\ B(s)\nabla_\theta 1 = 0$$

Reducing variance using a baseline

- This means that we can subtract any function from the function we are taking the gradient on, as long as it depends only on the state, not on the action.
- A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the **advantage function**

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \, log \, \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right]$$

- This creates the family of algorithms called **advantage actor critic** (A2C/A3C). Can significantly reduce the variance of policy gradient: faster and more stable convergence.