

# Reinforcement Learning

## Policy Search: Actor-Critic and Gradient Policy search

Mario Martin

CS-UPC

May 28, 2018

# Goal of this lecture

- So far we approximated the value or action-value function using parameters  $\theta$  (e.g. neural networks)

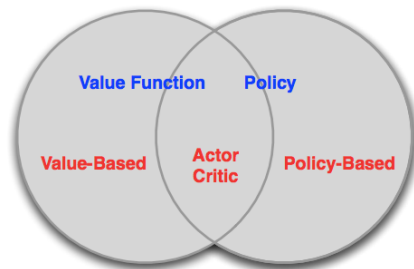
$$\begin{aligned} V_{\theta} &\approx V^{\pi} \\ Q_{\theta}(s, a) &\approx V^{\pi}(s) \end{aligned}$$

- A policy was generated directly from the value function e.g. using  $\epsilon$ -greedy
- In this lecture we will directly parameterize the policy in a stochastic setting

$$\pi_{\theta}(a|s) = P_{\theta}(a|s)$$

- and do a direct **Policy search**
- Again on model-free setting

# Three approaches to RL



**Value based** learning: **Implicit policy**

- Learn value function  $Q_{\theta}(s, a)$  and from there infer policy  $\pi(s) = \arg \max_a Q(s, a)$

**Policy based** learning: **No value function**

- Explicitly learn policy  $\pi_{\theta}(a|s)$  that implicitly maximize reward over all policies

**Actor-Critic** learning: **Learn both Value Function and Policy**

# Advantages of Policy over Value approach

- Advantages:

- ▶ In some cases, computing Q-values is harder than picking optimal actions
- ▶ **Better convergence properties**
- ▶ **Effective in high dimensional or continuous action spaces**
- ▶ Can benefit from demonstrations
- ▶ Policy subspace can be chosen according to the task
- ▶ Exploration can be directly controlled
- ▶ **Can learn stochastic policies**

- Disadvantages:

- ▶ Typically converge to a **local optimum** rather than a global optimum
- ▶ Evaluating a policy is typically **data inefficient and high variance**

# Stochastic Policies

- In general, two kinds of policies:

- ▶ Deterministic policy

$$a = \pi_{\theta}(s)$$

- ▶ Stochastic policy

$$P(a|s) = \pi_{\theta}(a|s)$$

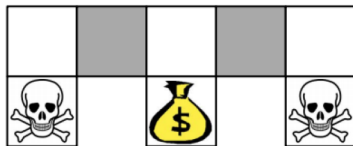
- Not new,  $\epsilon$ -greedy is stochastic, but different idea. Stochastic policy is good on its own, not because it is an approx. of a greedy policy
- Any example where stochastic could be better than deterministic?

# Stochastic Policies: Rock-Paper-Scissors



- Two-player game of rock–paper–scissors:
  - ▶ Scissors beats paper
  - ▶ Rock beats scissors
  - ▶ Paper beats rock
- Consider policies for iterated rock–paper–scissors
  - ▶ A deterministic policy is easily exploited
  - ▶ A uniform random policy is optimal (i.e., Nash equilibrium)

# Stochastic Policies: Aliased World (POMDSPs)



- The agent *cannot differentiate* the grey states
- Consider features of the following form:

$$\phi_d(s) = 1(\text{wall to } d) \quad \forall d \in \{N, E, S, W\}$$

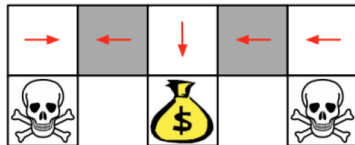
- Compare value-based RL, using an approximate value function

$$Q_\theta(s, a) = f_\theta(\phi(s, a))$$

- To policy-based RL, using a parametrized policy

$$\pi_\theta(a|s) = g_\theta(\phi(s, a))$$

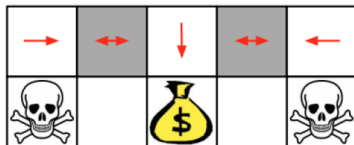
# Stochastic Policies: Aliased World (POMDSPs)



- Under aliasing, an optimal deterministic policy will either
  - ▶ move W in both gray states
  - ▶ move E in both gray states
- Either way, it can get stuck and never reach the money
- Value-based RL learns a near-deterministic policy
- So it will traverse the corridor for a long time



# Stochastic Policies: Aliased World (POMDSPs)



- An optimal stochastic policy will randomly move E or W in gray states
  - ▶  $\pi_{\theta}(\text{move E} \mid \text{wall to N and S}) = 0.5$
  - ▶  $\pi_{\theta}(\text{move W} \mid \text{wall to N and S}) = 0.5$
- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

# Policy Objective Functions

- Goal: given policy  $\pi_\theta(a|s)$  with parameters  $\theta$ , find best  $\theta$
- ... but how do we measure the quality of a policy  $\pi_\theta$ ?
- In episodic environments we can use the **start value**

$$J_1(\theta) = V^{\pi_\theta}(s_1)$$

# Policy Objective Functions

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

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

where  $d^{\pi_\theta}(s)$  is stationary distribution of Markov chain for  $\pi_\theta$

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

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a|s) r(s, a)$$

where  $d^{\pi_\theta}(s)$  is the expected number of time steps on  $s$  in a randomly generated episode following  $\pi_\theta$  divided by time steps of trial

- For simplicity, we will mostly discuss the episodic case, but can easily extend to the continuing / infinite horizon case

# Policy optimization

# Policy optimization

- Goal: given policy  $\pi_{\theta}(a|s)$  with parameters  $\theta$ , find best  $\theta$
- Policy based reinforcement learning is an **optimization** problem
- Find policy parameters  $\theta$  that maximize  $J(\theta)$
- Two approaches for solving the optimization problem
  - ▶ Gradient-free
  - ▶ Policy-gradient

## Subsection 1

# Gradient Free Policy Optimization

# Gradient Free Policy Optimization

- Goal: given parametrized method (with parameters  $\theta$ ) to approximate policy  $\pi_\theta(a|s)$ , find best values for  $\theta$
- Policy based reinforcement learning is an **optimization** problem
- Find policy parameters  $\theta$  that maximize  $J(\theta)$
- Some approaches do not use gradient
  - ▶ Hill climbing
  - ▶ Simplex / amoeba / Nelder Mead
  - ▶ Genetic algorithms
  - ▶ Cross-Entropy method (CEM)
  - ▶ Covariance Matrix Adaptation (CMA)

# Gradient Free Policy Optimization

- Goal: given parametrized method (with parameters  $\theta$ ) to approximate policy  $\pi_\theta(a|s)$ , find best values for  $\theta$
- Policy based reinforcement learning is an **optimization** problem
- Find policy parameters  $\theta$  that maximize  $J(\theta)$
- Some approaches do not use gradient
  - ▶ Hill climbing
  - ▶ Simplex / amoeba / Nelder Mead
  - ▶ Genetic algorithms
  - ▶ Cross-Entropy method (CEM)
  - ▶ Covariance Matrix Adaptation (CMA)



# Cross-Entropy Method (CEM)

- A simplified version of Evolutionary algorithm
- Works *embarrassingly* well in some problems, f.i.
  - ▶ Playing Tetris ([Szita et al., 2006](#)), ([Gabillon et al., 2013](#))
  - ▶ A variant of CEM called Covariance Matrix Adaptation has become standard in graphics ([Wampler et al., 2009](#))
- Very simple idea:
  - 1 From current policy, sample  $N$  trials (large)
  - 2 Take the  $M$  trials with larger Long.return (we call the **elite**)
  - 3 Fit new policy to behave as in  $M$  best sessions
  - 4 Repeat until satisfied
- Policy improves gradually

# Tabular Cross-Entropy

## Tabular Cross-Entropy Algorithm

Given  $M$  (f.i. 20),  $N$  (f.i. 200)

Initialize matrix policy  $\pi(a|s) = A_{s,a}$  randomly

**repeat**

    Sample  $N$  roll-outs of the policy and collect for each  $R_t$

    elite =  $M$  best samples

$$\pi(a|s) = \frac{[\text{times in } M \text{ samples took } a \text{ in } s] + \lambda}{[\text{times in } M \text{ samples was at } s] + \lambda|A|}$$

**until** convergence

**return**  $\pi$

Notice! No value functions!

# Tabular Cross-Entropy

- If you were in some state only *once*, you only take this action now
- Solution: Introduction of  $\lambda$ , a parameter to smooth probabilities
- Due to randomness, algorithm will prefer “lucky” sessions (training on lucky sessions is no good)
- Solution: run several simulations with these state-action pairs and average the results.

# Approximated Cross-Entropy Method (CEM)

## Approximated Cross-Entropy Method

Given  $M$  (f.i. 20),  $N$  (f.i. 200) and function approximation (f.i. NN) depending on  $\theta$

Initialize  $\theta$  randomly

**repeat**

    Sample  $N$  roll-outs of the policy and collect for each  $R_t$

    elite =  $M$  best samples

$$\theta = \theta + \alpha \nabla \left[ \sum_{s, a \in \text{elite}} \log \pi_{\theta}(a|s) \right]$$

**until** convergence

**return**  $\pi_{\theta}$

# Approximated Cross-Entropy Method (CEM)

- No Value function involved
- Notice that best policy is:

$$\arg \max_{\pi_{\theta}} \sum_{s, a \in elite} \log \pi_{\theta}(a|s) = \arg \max_{\pi_{\theta}} \prod_{s, a \in elite} \pi_{\theta}(a|s)$$

so gradient goes in that direction (some theory about Entropy behind)

- Intuitively, is the policy that maximizes similarity with behavior of successful samples
- Tabular case is a particular case of this algorithm
- I promised no gradient, but notice that gradient is for the approximation, not for the policy
- Can easily be extended to continuous action spaces (f.i. robotics)

# Gradient Free methods

- Often a great simple baseline to try
- Benefits
  - ▶ Can work with any policy parameterizations, including non-differentiable
  - ▶ Frequently very easy to parallelize (faster training *time*)
- Limitations
  - ▶ Typically not very *sample* efficient because it ignores temporal structure

## Subsection 2

### **Policy gradient**

# Policy gradient methods

- Policy based reinforcement learning is an **optimization** problem
- Find policy parameters  $\theta$  that maximize  $V^{\pi_\theta}$
- We have seen gradient-free methods, but greater efficiency often possible using gradient in the optimization
- Pletora of methods:
  - ▶ Gradient descent
  - ▶ Conjugate gradient
  - ▶ Quasi-newton
- We focus on *gradient ascent*, many extensions possible
- And on methods that exploit sequential structure



# Policy gradient differences wrt Value methods

- With Value functions we use Greedy updates:

$$\theta_{\pi'} = \arg \max_{\theta} \mathbb{E}_{\pi_{\theta}} [Q^{\pi}(s, a)]$$

- $V^{\pi_0} \xrightarrow{\text{small change}} \pi_1 \xrightarrow{\text{large change}} V^{\pi_1} \xrightarrow{\text{small change}} \pi_2 \xrightarrow{\text{large change}} V^{\pi_2}$
- Potentially unstable learning process with large policy jumps because  $\arg \max$  is not differentiable
- On the other hand, Policy Gradient updates are:

$$\theta_{\pi'} = \theta_{\pi} + \alpha \frac{\partial J(\theta)}{\partial \theta}$$

- Stable learning process with smooth policy improvement

# Policy gradient method

- Define  $J(\theta) = J^{\pi_\theta}$  to make explicit the dependence of the evaluation policy on the policy parameters
- Assume episodic MDPs
- Policy gradient algorithms search for a local **maximum** in  $J(\theta)$  by **ascending** the gradient of the policy, w.r.t parameters  $\theta$

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

- Where  $\nabla_\theta J(\theta)$  is the *policy gradient* and  $\alpha$  is a step-size parameter

# Computing the gradient analytically

- We now compute the policy gradient analytically
- **Assume policy is differentiable whenever it is non-zero**

# Computing the gradient analytically

- We now compute the policy gradient analytically
- **Assume policy is differentiable whenever it is non-zero**
- and that we know the gradient  $\nabla_{\theta} \pi_{\theta}(a|s)$
- Denote a state-action **trajectory** (or trial)  $\tau$  as

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T, s_T)$$

- Define long-term-reward to be the sum of rewards for the trajectory ( $R(\tau)$ )

$$R(\tau) = \sum_{t=1}^T r(s_t)$$

- It can be discounted or not. Now not important because we will not use Bellman equations.

# Computing the gradient analytically

- The value of the policy  $J(\theta)$  is:

$$J(\theta) = \mathbb{E}_{\pi_{\theta}} [R(\tau)] = \sum_{\tau} P(\tau|\theta)R(\tau)$$

where  $P(\tau|\theta)$  denotes the probability of trajectory  $\tau$  when following policy  $\pi_{\theta}$

- Notice that sum is for all possible trajectories
- In this new notation, our goal is to find the policy parameters *theta*) that:

$$\arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau|\theta)R(\tau)$$

## [Log-trick: a convenient equality]

- In general, assume we want to compute  $\nabla \log f(x)$  :

$$\begin{aligned}\nabla \log f(x) &= \frac{1}{f(x)} \nabla f(x) \\ f(x) \nabla \log f(x) &= \nabla f(x)\end{aligned}$$

- It can be applied to any function and we can use the equality in any direction
- The term  $\frac{\nabla f(x)}{f(x)}$  is called *likelihood ratio* and is used to analytically compute the gradients
- Btw. Notice the caveat... *Assume policy is differentiable whenever it is non-zero.*

# Computing the gradient analytically

- In this new notation, our goal is to find the policy parameters  $\theta$  that:

$$\arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau|\theta) R(\tau)$$

- So, taken the gradient wrt  $\theta$

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \sum_{\tau} P(\tau|\theta) R(\tau) \\&= \sum_{\tau} \nabla_{\theta} P(\tau|\theta) R(\tau) \\&= \sum_{\tau} \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_{\theta} P(\tau|\theta) R(\tau) \\&= \sum_{\tau} P(\tau|\theta) R(\tau) \frac{\nabla_{\theta} P(\tau|\theta)}{P(\tau|\theta)} \\&= \sum_{\tau} P(\tau|\theta) R(\tau) \nabla_{\theta} \log P(\tau|\theta)\end{aligned}$$

# Computing the gradient analytically

- Goal is to find the policy parameters  $\theta$  that:

$$\arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau|\theta) R(\tau)$$

- So, taken the gradient wrt  $\theta$

$$\nabla_{\theta} J(\theta) = \sum_{\tau} P(\tau|\theta) R(\tau) \nabla_{\theta} \log P(\tau|\theta)$$

- Of course we cannot compute all trajectories...



# Computing the gradient analytically

- Goal is to find the policy parameters  $\theta$  that:

$$\arg \max_{\theta} J(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau|\theta) R(\tau)$$

- So, taken the gradient wrt  $\theta$

$$\nabla_{\theta} J(\theta) = \sum_{\tau} P(\tau|\theta) R(\tau) \nabla_{\theta} \log P(\tau|\theta)$$

- Of course we cannot compute all trajectories...but we can sample  $m$  trajectories because of the form of the equation

$$\nabla_{\theta} J(\theta) \approx (1/m) \sum_{i=1}^m R(\tau_i) \nabla_{\theta} \log P(\tau_i|\theta)$$

# Computing the gradient analytically: at last!

- Sample  $m$  trajectories:

$$\nabla_{\theta} J(\theta) \approx (1/m) \sum_{i=1}^m R(\tau_i) \nabla_{\theta} \log P(\tau_i | \theta)$$

- However, we still have a problem, we don't know how to compute  $\nabla_{\theta} \log P(\tau | \theta)$
- Fortunately, we can derive it from the stochastic policy

$$\begin{aligned} \nabla_{\theta} \log P(\tau | \theta) &= \nabla_{\theta} \log \left[ \mu(s_0) \prod_{i=0}^{T-1} \pi_{\theta}(a_i | s_i) P(s_{i+1} | s_i, a_i) \right] \\ &= \nabla_{\theta} \left[ \log \mu(s_0) + \sum_{i=0}^{T-1} \log \pi_{\theta}(a_i | s_i) + \log P(s_{i+1} | s_i, a_i) \right] \\ &= \sum_{i=0}^{T-1} \underbrace{\nabla_{\theta} \log \pi_{\theta}(a_i | s_i)}_{\text{No dynamics model required!}} \end{aligned}$$

# Computing the gradient analytically

- We assumed at the beginning that policy is differentiable and that we now the derivative wrt parameters  $\theta$
- So, we have the desired solution:

$$\nabla_{\theta} J(\theta) \approx (1/m) \sum_{i=1}^m R(\tau_i) \sum_{i=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_i | s_i)$$

# Differentiable policies? Soft-max

- One popular way to do action selection instead of using  $\epsilon$ -greedy is to assign probabilities to actions according to values:

$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}} \propto e^{Q(s,a)/\tau}$$

where  $\tau$  is parameter that controls exploration. Let's assume  $\tau = 1$

- Let's consider the case where  $Q(s, a) = \phi^T(s, a)\theta$  is approximated by a linear function

$$\begin{aligned}\nabla_{\theta} \log \pi_{\theta}(a_i|s_i) &= \nabla_{\theta} \log \frac{e^{\phi^T(s,a)\theta}}{\sum_{a'} e^{\phi^T(s,a)\theta}} \\ &= \nabla_{\theta} \phi^T(s, a)\theta - \nabla_{\theta} \sum_{a'} \phi^T(s, a)\theta \\ &= \phi(s, a) - \mathbb{E}_{\pi_{\theta}} [\phi(s, \cdot)]\end{aligned}$$

# Differentiable policies? Gaussian Policy

- **In continuous spaces of actions**, action is generated by a random distribution with parameters. The most popular is Gaussian distribution
- Parameter of the Gaussian is a linear combination of feature ( $\mu = \phi^T(s, a) \theta$ ). Variance  $\sigma^2$  can be fixed or also approximated.
- Policy select actions following Gaussian distribution:

$$a \sim \mathcal{N}(\mu_\theta(s), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(a - \phi^T(s, a) \theta)^2}{2\sigma^2}}$$

- In this case,

$$\begin{aligned} \nabla_\theta \log \pi_\theta(a|s) &= \nabla_\theta - \frac{(a - \phi^T(s, a) \theta)^2 \phi(s)}{2\sigma^2} - \nabla_\theta \log \sigma^2 \\ &= \frac{(a - \phi^T(s, a) \theta) \phi(s)}{\sigma^2} \end{aligned}$$

# Differentiable policies? Deep Neural Network

- A very popular way to approximate the policy is to use a Deep NN with soft-max last layer with so many neurons as actions.
- In this case, use *autodiff* of the neural network package you use! In tensorflow:

```
loss = - tf.reduce_mean(tf.log(prob_outputs) * reward)
```

where `prob_outputs` is the output layer of the DNN

- Backpropagation implemented will do the work for you.

# Vanilla Policy Gradient

## Vanilla Policy Gradient

Given architecture with parameters  $\theta$  to implement  $\pi_\theta$

Initialize  $\theta$  randomly

**repeat**

    Generate episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T, s_T\} \sim \pi_\theta$

    Get  $R \leftarrow$  long-term return for episode

**for** all time steps  $t = 1$  to  $T - 1$  **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t | s_t) R$

**end for**

**until** convergence

Substitute  $\nabla_\theta \log \pi_\theta(a_t | s_t)$  with appropriate equation

# Vanilla Policy Gradient

- Remember:

$$\nabla_{\theta} J(\theta) \approx (1/m) \sum_{i=1}^m R(\tau_i) \sum_{i=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_i | s_i)$$

- Unbiased but very noisy
- Fixes that can make it practical
  - ▶ Temporal structure
  - ▶ Baseline



## Subsection 3

**Reduce variance using temporal structure: Reinforce and Actor-Critic architectures**

# Policy Gradient using Temporal structure

- Instead on focusing on reward of trajectories,

$$J(\theta) = \mathbb{E}_{\pi_{\theta}} [R(\tau)] = \sum_{\tau} P(\tau|\theta) R(\tau)$$

- We want to optimize the expected return

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

where  $d^{\pi_{\theta}}(s)$  is the expected number of time steps on  $s$  in a randomly generated episode following  $\pi_{\theta}$  divided by time steps of trial

- Let's start with an MDP with one single step.

$$J_{avR}(\theta) = \sum_s d^{\pi_{\theta}}(s) V(s) = \sum_s d^{\pi_{\theta}} \sum_a \pi_{\theta}(a|s) r(s, a)$$

# Policy Gradient using Temporal structure

$$\begin{aligned}J_{avR}(\theta) &= \sum_s d^{\pi_\theta} \sum_a \pi_\theta(a|s) r(s, a) \\ \nabla_\theta J_{avR}(\theta) &= \nabla_\theta \sum_s d^{\pi_\theta} \sum_a \pi_\theta(a|s) r(s, a) \\ &= \sum_s d^{\pi_\theta} \sum_a \nabla_\theta (\pi_\theta(a|s) r(s, a)) \\ &= \sum_s d^{\pi_\theta} \sum_a \nabla_\theta \pi_\theta(a|s) \log \pi_\theta(a|s) r(s, a) \\ &= \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) r(s, a)]\end{aligned}$$

- And this expectation can be sampled

# Policy Gradient theorem!

- The policy gradient theorem generalize the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward  $r$  with long-term value  $Q(s, a)$
- Policy gradient theorem applies to all objective functions we have seen

## Policy gradient theorem

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

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

---

Simple proof in pag. 269 of [\(Sutton 2016\)](#)

# REINFORCE algorithm

- REINFORCE algorithm (also called Monte–Carlo Policy Gradient) use policy gradient theorem and long-term reward  $R$  as unbiased sample of  $Q^{\pi_\theta}(s, a)$

## REINFORCE algorithm

Given architecture with parameters  $\theta$  to implement  $\pi_\theta$

Initialize  $\theta$  randomly

**repeat**

Generate episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T, s_T\} \sim \pi_\theta$

**for** all time steps  $t = 1$  to  $T - 1$  **do**

Get  $R_t \leftarrow$  long-term return from step  $t$  to  $T$

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t | s_t) R_t$

**end for**

**until** convergence

# REINFORCE algorithm

- Let's analyze the update:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R_t$$

- Let's us rewrite is as follows

$$\theta \leftarrow \theta + \alpha \frac{\nabla_{\theta} \pi_{\theta}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} R_t$$

- Update is proportional to:
  - ▶ the product of a return  $R_t$  and
  - ▶ the gradient of the probability of taking the action actually taken,
  - ▶ divided by the probability of taking that action.

# REINFORCE algorithm

- Update:

$$\theta \leftarrow \theta + \alpha \frac{\nabla_{\theta} \pi_{\theta}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} R_t$$

- ▶ ...move most in the directions that favor actions that yield the highest return
- ▶ ...is inversely proportional to the action probability (actions that are selected frequently are at an advantage (the updates will be more often in their direction))
- Is it necessary to change something in the algorithm for continuous actions?

# REINFORCE algorithm

- Update:

$$\theta \leftarrow \theta + \alpha \frac{\nabla_{\theta} \pi_{\theta}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} R_t$$

- ▶ ...move most in the directions that favor actions that yield the highest return
- ▶ ...is inversely proportional to the action probability (actions that are selected frequently are at an advantage (the updates will be more often in their direction))
- Is it necessary to change something in the algorithm for continuous actions?
- No! Just uses a continuous action policy mechanism and everything is the same!



# REINFORCE algorithm with baseline

- Monte-Carlo policy gradient still has high variance because  $R_t$  has a lot of variance
- We can **reduce variance subtracting a baseline** to the estimator

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (R_t - b(s_t))$$

- without introducing any bias *when baseline does not depend on actions* taken
- A good baseline is  $b(s_t) = V^{\pi_{\theta}}(s_t)$  so we will use that

# REINFORCE algorithm with baseline

- Monte-Carlo policy gradient still has high variance because  $R_t$  has a lot of variance
- We can **reduce variance subtracting a baseline** to the estimator

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (R_t - b(s_t))$$

- without introducing any bias *when baseline does not depend on actions* taken
- A good baseline is  $b(s_t) = V^{\pi_{\theta}}(s_t)$  so we will use that
- **How to estimate  $V^{\pi_{\theta}}$ ?**
- **We'll use another set of parameters  $w$  to approximate**

# REINFORCE algorithm with baseline

## REINFORCE algorithm with baseline (aka MC Actor Critic)

Given architecture with parameters  $\theta$  to implement  $\pi_\theta$  and parameters  $w$  to approximate  $V$

Initialize  $\theta$  randomly

**repeat**

Generate episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T, s_T\} \sim \pi_\theta$

**for** all time steps  $t = 1$  to  $T - 1$  **do**

Get  $R_t \leftarrow$  long-term return from step  $t$  to  $T$

$\delta \leftarrow R_t - V_w(s_t)$

$w \leftarrow w + \beta \delta \nabla_w V_w(s_t)$

$\theta \leftarrow \theta + \alpha \delta \nabla_\theta \log \pi_\theta(a_t | s_t)$

**end for**

**until** convergence

# Actor-Critic Architectures

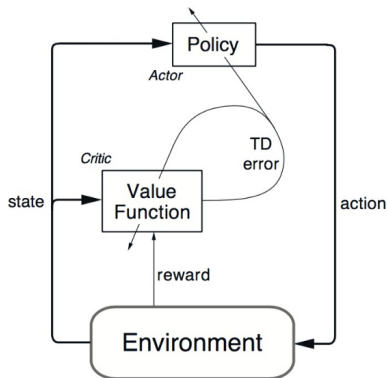
- Monte-Carlo policy gradient has high variance
- So we used a baseline to reduce the variance  $R_t - V(s_t)$
- Can we do something to speed up learning like we did with MC using TD?

# Actor-Critic Architectures

- Monte-Carlo policy gradient has high variance
- So we used a baseline to reduce the variance  $R_t - V(s_t)$
- Can we do something to speed up learning like we did with MC using TD?
- Yes, **use different estimators of  $R_t$  that do bootstrapping** f.i. TD(0), n-steps, etc.
- These algorithms are called **Actor Critic**

# Actor-Critic Architectures

- The **Critic**, *evaluates the current policy* and the result is used in the policy training
- The **Actor** *implements the policy* and is trained using Policy Gradient with estimations from the critic



# Actor-Critic Architectures

- Actor-critic algorithms maintain two sets of parameters (like in REINFORCE with baseline):
  - Critic** parameters: approximation parameters  $w$  for action-value function under current policy
  - Actor** parameters: policy parameters  $\theta$
- Actor-critic algorithms follow an approximate policy gradient:
  - Critic:** Updates action-value function parameters  $w$  like in *policy evaluation* updates (you can apply everything we saw in FA for prediction)
  - Actor:** Updates policy gradient  $\theta$ , in direction suggested by critic

# Actor-Critic Architectures

- Actor updates are always in the same way:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G_t$$

where  $G_t$  is the evaluation of long-term returned by the critic for  $s_t$

- Critic updates are done to evaluate the current policy

$$w \leftarrow w + \alpha \delta \nabla_w V_w(a_t | s_t)$$

where  $\delta$  is the estimated error in evaluating the  $s$  state and that implements the kind of bootstrapping done.



# One step Actor Critic (QAC)

One step actor-critic:  $\delta \leftarrow r + Q_w(s', a') - Q_w(s, a)$

## One step Actor Critic

Given architecture with parameters  $\theta$  to implement  $\pi_\theta$  and parameters  $w$  to approximate  $Q$

Initialize  $\theta$  randomly

**repeat**

Set  $s$  to initial state

Get  $a$  from  $\pi_\theta$

**repeat**

Take action  $a$  and observe reward  $r$  and new state  $s'$

Get  $a'$  from  $\pi_\theta$

$\delta \leftarrow r + Q_w(s', a') - Q_w(s, a)$

$w \leftarrow w + \beta \delta \nabla_w Q_w(s, a)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) Q_w(s, a)$

$s \leftarrow s'$

**until**  $s$  is terminal

**until** convergence

# Advantage Actor Critic (AAC or A2C)

- In this critic Advantage value function is used:

$$A^{\pi_{\theta}}(s, a) = Q^{\pi_{\theta}}(s, a) - V^{\pi_{\theta}}(s)$$

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function, for instance, estimating **both**  $V(s)$  and  $Q$  using two function approximators and two parameter vectors:

$$V^{\pi_{\theta}}(s) \approx V_v(s) \tag{1}$$

$$Q^{\pi_{\theta}}(s, a) \approx Q_w(s, a) \tag{2}$$

$$A(s, a) = Q_w(s, a) - V_v(s) \tag{3}$$

- And updating both value functions by e.g. TD learning

# From A2C to REINFORCE with baseline

- One way to implement A2C method without two different networks to estimate  $Q_w(s, a)$  and  $V_v(s)$  is the following.
- For the true value function  $V^{\pi_\theta}(s)$ , the TD error  $\delta^{\pi_\theta}(s)$

$$\delta^{\pi_\theta}(s) = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

- is an unbiased estimate of the advantage function:

$$\begin{aligned}\mathbb{E}_{\pi_\theta} [\delta_\theta^\pi | s, a] &= \mathbb{E}_{\pi_\theta} [r + \gamma V^{\pi_\theta}(s') | s, a] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) = A^{\pi_\theta}(s, a)\end{aligned}$$

- So we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) \delta_\theta^\pi]$$

- In practice this approach only requires one set of critic parameters  $v$  to approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- Notice this algorithm is *exactly* REINFORCE with baseline

# Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - ▶ use  $N$  copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - ▶ After some time, pass gradients to a main network that updates actor and critic using the gradients of all
  - ▶ After some time the worker copy the weights of the global network
- This parallelism decorrelates the agents' data, so **no Experience Replay Buffer needed**
- Even one can explicitly use different exploration policies in each actor-learner to maximize diversity
- Asynchronism can be extended to other update mechanisms (Sarsa, Q-learning...) but it works better in Advantage Actor critic setting

# Generalized Advanced Estimator (GAE)

- Generalized Advanced Estimator (Schulman et al. 2016)
- Use a version of Advantage that consider weighted average of n-steps estimators of advantage like in TD( $\lambda$ ):

$$A_{GAE}^{\pi} = \sum_{t'=t}^{\infty} (\lambda\gamma)^{t'-t} \underbrace{[r_{t'+1} + \gamma V_{\theta}^{\pi}(s_{t'+1}) - V_{\theta}^{\pi}(s_{t'})]}_{t'\text{-step advantage}}$$

- Used in continuous setting for locomotion tasks

## Subsection 4

### **Conclusions and other approaches**

# Summary

- The policy gradient has many equivalent forms

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) R_t] && \text{REINFORCE} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q_w(s, a)] && \text{Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) A_w(s, a)] && \text{Advantage Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) \delta] && \text{TD Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) \delta e] && \text{TD}(\lambda) \text{ Actor-Critic}\end{aligned}$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses **policy evaluation** (e.g. MC or TD learning) to estimate  $Q^{\pi}(s, a)$ ,  $A^{\pi}(s, a)$  or  $V^{\pi}(s)$

# Compatible Function Approximation: Bias in AC

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
- Luckily, if we choose value function approximation carefully, then we can avoid bias
- If the following two conditions are satisfied:

- 1 Value function approximator is compatible to the policy

$$\nabla_w Q_w(s, a) = \nabla_{\theta} \log \pi_{\theta}(a|s)$$

- 2 Value function parameters  $w$  minimize the mean-squared error

$$\nabla_w \mathbb{E}_{\pi_{\theta}} [(Q^{\pi_{\theta}}(s, a) - Q_w(s, a))^2] = 0$$

- Then the policy gradient is **without bias**



# Problems with Policy Gradient Directions

- Goal: Each step of policy gradient yields an updated policy  $\pi'$  whose value is greater than or equal to the prior policy  $\pi$ :  $V^{\pi'} \geq V^{\pi}$
- Several inefficiencies:
  - ▶ Gradient ascent approaches update the weights a **small step** in direction of gradient
  - ▶ Gradient ascent algorithms can follow *any* ascent direction (a good ascent direction can significantly **speed** convergence)
  - ▶ Gradient is First order / linear approximation of the value function's dependence on the **policy parameterization** instead of **actual policy**<sup>1</sup>

---

<sup>1</sup>A policy can often be re-parameterized without changing action probabilities (f.i., increasing score of all actions in a softmax policy). Vanilla gradient is sensitive to these re-parameterizations.

# About step size

- Step size is important in any problem involving finding the optima of a function
- Supervised learning: Step too far  $\rightarrow$  next updates will fix it
- But in Reinforcement learning
  - ▶ Step too far  $\rightarrow$  bad policy
  - ▶ Next batch: collected under bad policy
  - ▶ **Policy is determining data collect!** Essentially controlling exploration and exploitation trade off due to particular policy parameters and the stochasticity of the policy
  - ▶ May not be able to recover from a bad choice, collapse in performance!

# Better Policy Gradient Directions: Natural Gradient

- A more efficient gradient in learning problems is the **natural gradient**
- It corresponds to **steepest ascent in policy space and not in the parameter space** with **right step size**
- Also, the natural policy gradient is *parametrization independent*
- Convergence to a local minimum is guaranteed
- It finds ascent direction that is closest to vanilla gradient, when changing policy by a small, fixed amount

$$\nabla_{\theta}^{nat} \pi_{\theta}(a|s) = G_{\theta}^{-1} \nabla_{\theta} \pi_{\theta}(a|s)$$

- Where  $G_{\theta}$  is the Fisher information matrix

$$G_{\theta} = \mathbb{E}_{\pi_{\theta}} \left[ \nabla_{\theta} \log \pi_{\theta}(a|s) \nabla_{\theta} \log \pi_{\theta}(a|s)^T \right]$$

# Natural Actor Critic (Peters et al 2005)

- Under linear model modelization of critic:

$$A^{\pi_{\theta}}(s, a) = \phi(s, a)^T w$$

- Using compatible function approximation,

$$\nabla_w A_w(s, a) = \nabla_{\theta} \log \pi_{\theta}(a|s)$$

- The natural policy gradient nicely simplifies,

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) A^{\pi_{\theta}}(s, a)] \\ &= \mathbb{E}_{\pi_{\theta}} \left[ \nabla_{\theta} \log \pi_{\theta}(a|s) \nabla_{\theta} \log \pi_{\theta}(a|s)^T w \right] \\ &= G_{\theta} w \\ \nabla_{\theta}^{\text{nat}} J(\theta) &= w\end{aligned}$$

- i.e. update actor parameters in direction of critic parameters

# TRPO (Schulman et al 2017)

- *Trust Region Policy Optimization* (TRPO) maximize parameters that change the policy increasing advantage in action over wrt. old policy in proximal spaces to avoid too large step size.

$$\arg \max_{\theta} L_{\theta_{old}}(\theta) = \arg \max_{\theta} \mathbb{E}_{s_0:\infty} \left[ \sum_{t=0}^{T-1} \mathbb{E}_{a \sim \theta} \left[ \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} A_{\theta}(s_t, a_t) \right] \right]$$

- Under penalizing constraint (using KL divergence of  $\theta$  and  $\theta_{old}$ ) that ensures improvement of the policy in the proximity (small step size)
- Solves using Natural Gradient

# TRPO (Schulman et al 2017)

- *Trust Region Policy Optimization* (TRPO) maximize parameters that change the policy increasing advantage in action over wrt. old policy in proximal spaces to avoid too large step size.

$$\arg \max_{\theta} L_{\theta_{old}}(\theta) = \arg \max_{\theta} \mathbb{E}_{s_0:\infty} \left[ \sum_{t=0}^{T-1} \mathbb{E}_{a \sim \theta} \left[ \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} A_{\theta}(s_t, a_t) \right] \right]$$

- Under penalizing constraint (using KL divergence of  $\theta$  and  $\theta_{old}$ ) that ensures improvement of the policy in the proximity (small step size)
- Solves using Natural Gradient
- Some TRPO videos [here](#).
- Proximal Policy Optimization **PPO** inspired in TRPO simplifies computation

# Deterministic Policy Gradient (DPGA)

- A different idea: Reparametrization
- Suppose problem has continuous action space,  $a \in \mathbf{R}^d$
- Then  $\frac{\partial Q(s,a)}{\partial a}$  tells use how to improve our action
- We can use reparameterization trick, so  $a$  is now a deterministic function  $a = f(s, z)$ , where  $z$  is noise. Then,

$$\nabla_{\theta} \mathbb{E}_{\tau}[R] = \nabla_{\theta} Q^{\pi}(s_0, a_0) + \nabla_{\theta} Q^{\pi}(s_1, a_1) + \dots$$

- This method is called the deterministic policy gradient (Silver et al. 2014)
- A generalized version, which also uses a dynamics model, is described as the stochastic value gradient