

DS-GA 3001 005 | Lecture 7

Reinforcement Learning

Jeremy Curuksu, PhD

NYU Center for Data Science

jeremy.cur@nyu.edu

April 3, 2024

DS-GA 3001 RL Curriculum

Reinforcement Learning:

- ▶ Introduction to Reinforcement Learning
- ▶ Multi-armed Bandit
- ▶ Dynamic Programming on Markov Decision Process
- ▶ Model-free Reinforcement Learning
- ▶ Value Function Approximation (Deep RL)
- ▶ Examples of Industrial Applications
- ▶ **Policy Function Approximation (Actor-Critic)**
- ▶ Planning from a Model of the Environment (AlphaZero)
- ▶ Reinforcement Learning from Human Preferences

Reinforcement Learning

Lecture 5: Value Function Approximation

- ▶ Categories of Functions in Reinforcement Learning
- ▶ Approximation of State-Update and Value Functions
- ▶ Deep Reinforcement Learning

Today: Policy Function Approximation

- ▶ Policy Gradient Reinforcement Learning
- ▶ Advanced Sampling of Policy Gradient
- ▶ Reinforcement Learning in Continuous Action Space

RL Functions and Approximations

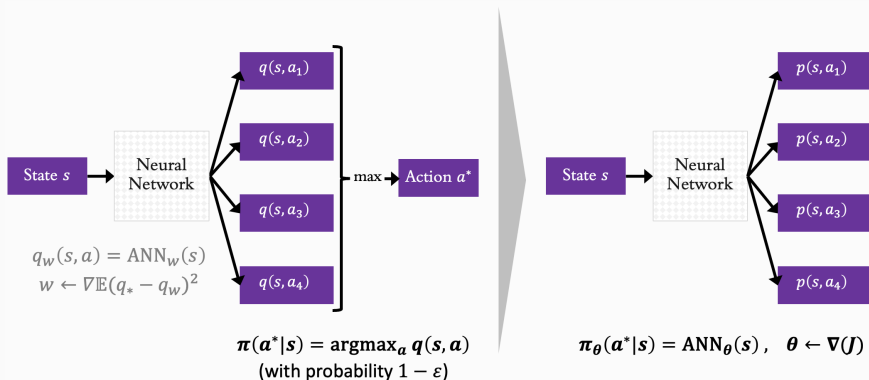
- ▶ **All key components of an RL agent are functions**
 - ▶ **State update functions** map observations to states
 - ▶ **Value functions** map states to values
 - ▶ **Policy functions** map states to actions
 - ▶ **Models** map states and actions to next states and rewards
- ▶ **Functions can be parameterized and approximated** by linear or non-linear gradient methods e.g., Deep Learning
- ▶ **A parameterized policy function is called an "actor" function.** Policies are action-selection functions which can be derived deterministically from a value function (example: ϵ -greedy soft updates), but more generally they can be *parameterized*
- ▶ **Challenge:** Supervised learning assumptions are often violated

Policy Gradient Reinforcement Learning

Policy-Gradient Optimization

Predict optimal action using a Deep Neural Network

- Replace q-value and ϵ -greedy functions by one ANN function



Policy-Gradient Optimization

Define a parametric function to approximate policies

$$\pi_{\theta}(a|s) \approx p(a|s, \theta)$$

- ▶ The function can be a linear combination of features, often with a soft-max to define action-probabilities normalizing to 1:

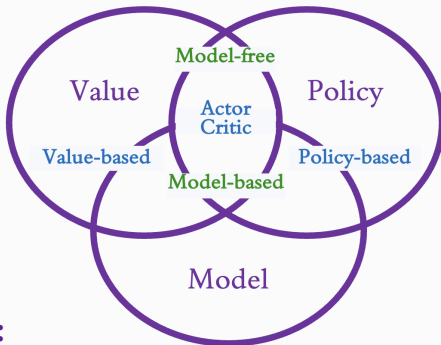
$$\pi_{\theta}(a|s) = \frac{e^{\theta^T \phi(s,a)}}{\sum_k e^{\theta^T \phi(s,a_k)}}$$

- ▶ Parameters θ can be updated incrementally by optimizing any objective measure of performance $J(\theta)$:

$$\theta \sim \arg \max_{\theta} J(\theta)$$

- ▶ Gradient ascent converges to global optimum if $J(\theta)$ is known

Policy-Gradient Optimization



Categories of RL agents:

- ▶ **Model-based:** Use a model to learn policy or value function (lectures 3, 8, 9)
- ▶ **Model-free:** Learn policy or value function without model (lectures 4, 5)
- ▶ **Value-based:** Learn value function, not policy function (lectures 4, 5)
- ▶ **Policy-based:** Learn policy function, not value function (today)
- ▶ **Actor-Critic:** Learn policy and value functions (today)

Model-, Value-, or Policy-based RL ?

► Model-based RL:

- ✓ Learns 'all there is to know' from the data
- ✓ Very well understood method (supervised learning)
- × Objective captures irrelevant information
- × May focus compute/capacity on irrelevant details
- × Deriving a policy from a model (planning) is non-trivial

► Value-based RL:

- ✓ Focus compute/capacity on how actions affect reward
- ✓ Relatively well-understood (similar to regression)
- × May still focus compute/capacity on irrelevant details

► Policy-based RL:

- ✓ Direct search of optimal policy = true objective of RL
- × Ignores all other learnable knowledge

Pros & Cons of Policy-based RL

► Strengths:

- ✓ Focus on true objective of RL. Sometimes policies are simple while values and models are complex
- ✓ Can learn high-dimensional or continuous action policies
- ✓ Can learn stochastic (= probabilistic) policies
- ✓ Can learn appropriate levels of exploration autonomously (no need to manually define an ϵ -exploration schedule)

► Limitations:

- ✗ Ignores all other learnable knowledge so may not efficiently use the available data
- ✗ Easily trapped in local optima (actor is what creates data)
- ✗ May not generalize well when environment changes

Policy Gradient Optimization

Define Policy's Objective Measure of Quality

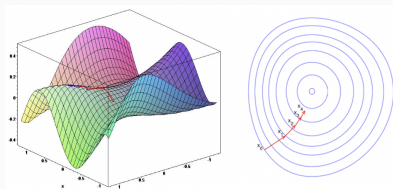
- **Ex 1 (Episodic Case):** Maximize expected return in s_0 :

$$J_o(\theta) = \mathbb{E}_{\pi_\theta}(G_t | s_0) = v_{\pi_\theta}(s_0)$$

- **Ex 2 (Continuous Case):** Maximize average return across states:

$$J_{avg}(\theta) = \mathbb{E}_{\pi_\theta}(G_t | s) = \sum_s d_{\pi_\theta}(s) v_{\pi_\theta}(s)$$

Update policy parameters by stochastic gradient ascent



Policy Gradient Optimization

Update policy parameters by stochastic gradient ascent

- To find a maximum of $J(\theta)$, define the gradient of $J(\theta)$ and move θ in the direction of positive gradient at every step:

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

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta)$$

- **Some methods do not use gradient (out of scope):**
 - Hill climbing
 - Genetic algorithms

Policy Gradient Optimization

How can the agent learn

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \mathbb{E}_{\pi_{\theta}}(G_t | s)$$

?

Sample what? Compute what?

Policy Gradient Theorem

Theorem's Proof

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{\pi_{\theta}}(G_t | s) \\&= \nabla_{\theta} \sum_s \mu(s) \sum_a \pi_{\theta}(a|s) G_t \\&= \sum_s \mu(s) \sum_a \nabla_{\theta} \pi_{\theta}(a|s) G_t \\&= \sum_s \mu(s) \sum_a \pi_{\theta}(a|s) \frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta}(a|s)} G_t \\&= \mathbb{E}_{\pi_{\theta}} \left(G_t \frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta}(a|s)} \right) = \mathbb{E}_{\pi_{\theta}} (G_t \nabla_{\theta} \log \pi_{\theta}(a|s))\end{aligned}$$

- **In plain english:** Gradient of the expected value of G_t under π_{θ} equals the expected value of G_t weighted by gradient of $\log \pi_{\theta}$

Policy Gradient Theorem

∀ differentiable policy π_θ , the gradient of $J = \mathbb{E}_{\pi_\theta}(G_t|s)$ is:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}(G_t|s) = \mathbb{E}_{\pi_\theta} \left(G_t \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \right) = \mathbb{E}_{\pi_\theta} (G_t \nabla_\theta \log \pi_\theta(a|s))$$

Theorem's Implications

- ▶ The gradient of $J(\theta)$ can be sampled because it is equal to the expected value of some known quantities (G_t , π_θ , and $\nabla_\theta \pi_\theta$)
- ▶ Gradient updates of π_θ do not involve derivatives of the state distribution thus are agnostic to detailed dynamics of the MDP

Practice: Policy Gradient MC (REINFORCE)

REINFORCE updates $\pi_\theta(a|s)$ using the policy gradient computed over complete episodes, with no model of the environment

Initialize θ arbitrarily

Loop forever:

 Initialize s_0

 Experience an episode $(s_0, a_0, r_1, s_1, a_1, r_2, \dots, r_T)$ following π_θ :

 Loop for each step t of episode:

$$G = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^T r_{t+1+T}$$

$$\theta = \theta + \alpha G \nabla_\theta \log(\pi_\theta(a|s))$$

Advanced Sampling of Policy Gradient

Advanced Sampling of the Gradient $\nabla_{\theta} J(\theta)$

- Variance can be reduced by adding a baseline to the MC target:

$$\theta_{t+1} = \theta_t + \alpha (G_t - v_{\pi}(s_t)) \nabla_{\theta} \log \pi_{\theta}(a|s_t)$$

- It is called the *advantage* of following π_{θ} relative to baseline:

$$\theta_{t+1} = \theta_t + \alpha (q_{\pi}(s_t, a) - v_{\pi}(s_t)) \nabla_{\theta} \log \pi_{\theta}(a|s_t)$$

- A TD target can be used to learn at every step:

$$\theta_{t+1} = \theta_t + \alpha (r_{t+1} + \gamma v_{\pi}(s_{t+1}) - v_{\pi}(s_t)) \nabla_{\theta} \log \pi_{\theta}(a|s_t)$$

- Value function approximation refines (*critics*) the gradient:

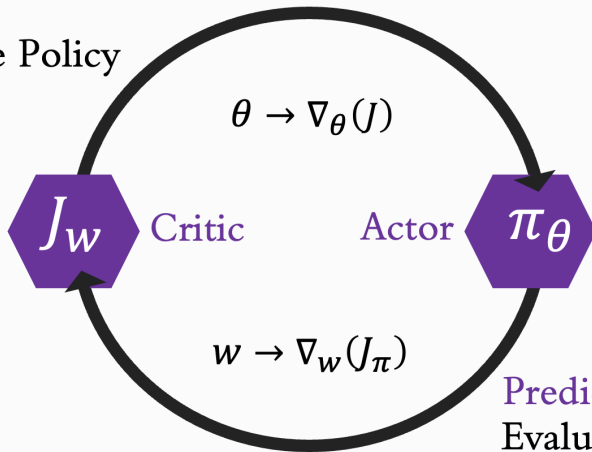
$$\theta_{t+1} = \theta_t + \alpha (r_{t+1} + \gamma v_w(s_{t+1}) - v_w(s_t)) \nabla_{\theta} \log \pi_{\theta}(a|s_t)$$

Generalized Policy Iteration

All RL methods are Generalized Policy Iteration methods

Control

Improve Policy



Prediction
Evaluate Policy

Practice: A3C Actor-Critic

Asynchronous Advantage Actor-Critic (A3C) updates π_θ using the policy gradient computed from v_w at every step, in any state s

Initialize w , θ , and s , arbitrarily

Loop forever:

Take a from s following $\pi_\theta(a|s)$, observe r_{t+1} and s'

$$\delta = r_{t+1} + \gamma v_w(s_{t+1}) - v_w(s_t) \quad [\text{TD-error} = \text{Advantage}]$$

$$w_{t+1} = w_t - \alpha_1 \delta \nabla_w v_w(s_t) \quad [\text{Update of Critic}]$$

$$\theta = \theta + \alpha_2 \delta \nabla_\theta \log \pi_\theta(a|s_t) \quad [\text{Update of Actor}]$$

$$s = s'$$

Bias-variance tradeoff in A3C

Update policy gradient from multi-step TD error

- ▶ The full return $G_t = (r_{t+1} + \gamma r_{t+2} + \dots)$ has high variance
- ▶ The TD target $r_{t+1} + \gamma v_w(s_{t+1})$ has high bias
- ▶ A useful middle ground is to define TD error with n -step return:

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{n-1} r_{t+n} + \gamma^n v_w(s_{t+n})$$

- ▶ Or to define TD error with λ -returns as in TD(λ):

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$

- ▶ **BUT both require more compute/memory (can be prohibitively too slow), and must wait n steps before learning update occurs**

Regularisation of Policy Function

A biased policy samples biased data

- ▶ The policy gradient objective only considers improvement under observed/sampled data
- ▶ But the policy is what drives the sampling of data, so the agent gets easily trapped in local optima, which perpetuate biases
- ▶ Entropy needs be increased (encourage exploration) without causing breakage due to widely different policies (instability)
- ▶ Many variants from original A3C method have been developed to address the bias-variance tradeoff of actor-critic RL
- ▶ For example to prevent instability and perpetuating biases, the policy can be regularised to not change too much

Regularisation of Policy Function

Trust Region Policy Optimization (TRPO)

- ▶ Prevent instability by regularizing $J(\theta)$ to limit the difference between subsequent policies (= limit speed of exploration)
- ▶ The difference between two probability distributions/densities is the Kullback-Leibler divergence D_{KL} :

$$D_{KL}(\pi_{old}||\pi_{\theta}) = \sum_a \pi_{old}(a|s) \log \frac{\pi_{\theta}(a|s)}{\pi_{old}(a|s)}$$

- ▶ TRPO defines a "trust-region" by adding a restraint on the KL divergence, which is equivalent to maximizing this objective:

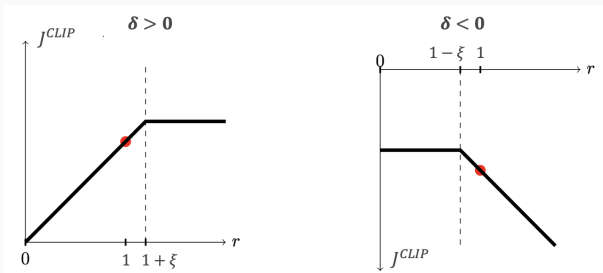
$$\nabla_{\theta} \left[\mathbb{E}_{\pi_{old}} \left(\delta \frac{\pi_{\theta}(a|s)}{\pi_{old}(a|s)} \right) - \eta D_{KL}(\pi_{\theta}||\pi_{old}) \right]$$

Regularisation of Policy Function

Proximal Policy Optimization (PPO)

- PPO defines a trust region by regularizing the objective with a clipped probability ratio (similar to TRPO but simpler)

$$\nabla_{\theta} \left[\mathbb{E}_{\pi_{\text{old}}} \left(\delta \text{ clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\text{old}}(a|s)}, 1 - \xi, 1 + \xi \right) \right) \right]$$



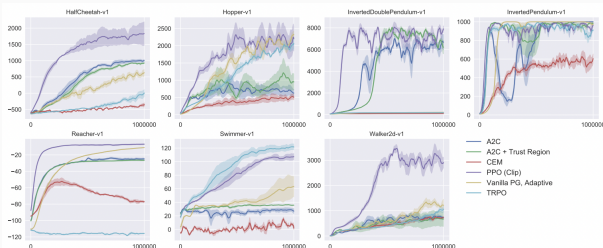
Regularisation of Policy Function

Proximal Policy Optimization (PPO)

- PPO defines a trust region by regularizing the objective with a clipped probability ratio (similar to TRPO but simpler)

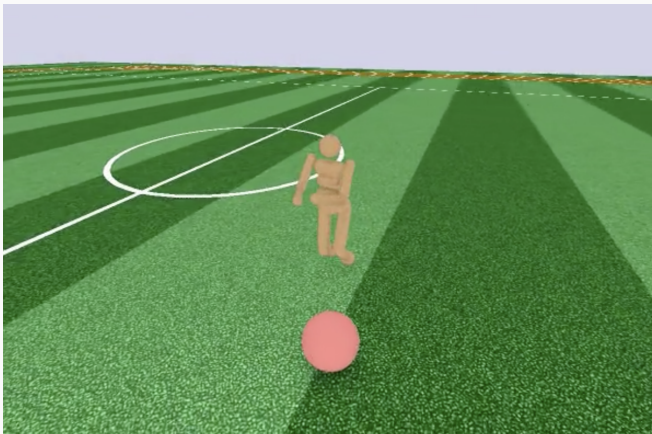
$$\nabla_{\theta} \left[\mathbb{E} \left(\delta \text{ clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\text{old}}(a|s)}, 1 - \xi, 1 + \xi \right) \right) \right]$$

- Benchmark of PPO performance on MuJoCo Gym environments



Example: Moving a Robot with PPO

Proximal Policy Optimization (PPO)



For more details: <https://openai.com/blog/openai-baselines-ppo>

Example: Moving a Robot with PPO

Proximal Policy Optimization (PPO)



(Source: DeepMind (2017))

Reinforcement Learning in Continuous Action Space

Policy Gradient for Continuous Action Space

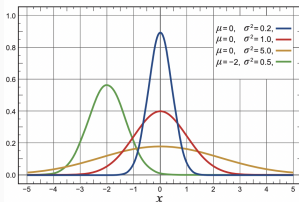
Policy Gradient easily extends to continuous actions

- Instead of learning probabilities for discrete actions, RL can learn a statistics of continuous action-probability densities
- For example, an action can be chosen from a Gaussian density:

$$\pi_{\theta}(a|s) \sim \mathcal{N}(\mu_{\theta}(s), \sigma_{\theta}^2)$$

$$\forall \sigma^{\dagger} : \nabla_{\theta} J(\theta) = \mathbb{E}(\delta \nabla_{\theta} \log \pi_{\theta}(a|s_t)) \approx \delta \frac{a - \mu_{\theta}(s)}{\sigma^2} \nabla \mu_{\theta}(s)$$

- Sampling actions from a probability density guarantees exploration



Practice: CACLA algorithms

Continuous Actor-Critic Learning Automaton (CACLA) samples actions from a parameterized probability density $\pi_\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$, updated from the gradient of v_w at every step, in any state s

Initialize w , θ , and s , arbitrarily

Loop forever:

Take a from s following $\mathcal{N}(\mu_\theta(s), \sigma_\theta^2)$, observe r_{t+1} and s'

$$\delta = r_{t+1} + \gamma v_w(s_{t+1}) - v_w(s_t)$$

$$w_{t+1} = w_t - \alpha_1 \delta \nabla_w v_w(s_t)$$

$$\text{If } \delta > 0: \theta = \theta + \alpha_2 \delta \nabla_\theta \log \pi_\theta(a|s_t)$$

$$s = s'$$

* The original CACLA algorithm (2007) updated actor iff sampled a_t increases value

Practice: DDPG algorithms

Deep Deterministic Policy Gradient (DDPG) generalizes CACLA and DQN by sampling actions from a probability density $\pi_{\theta} \sim \mathcal{N}(\mu_{\theta}, \sigma^2)$ updated directly from the gradient of q_w learned by deep learning

Initialize w , θ , and s , arbitrarily

Loop forever:

Take a from s following $\mathcal{N}(\mu_{\theta}(s), \sigma^2)$, observe r_{t+1} and s'

$$w_{t+1} = w_t - \alpha_1 (r_{t+1} + \gamma q_w(s_{t+1}, \mu_{\theta}(s_{t+1})) - q_w(s_t, a)) \nabla_w q_w(s_t, a)$$

$$\theta = \theta + \alpha_2 \nabla_{\theta} q_t(s_t, \mu_{\theta}(s_t))$$

Store past transitions in a Replay Buffer

$s = s'$

Periodically:

Sample mini-batches from the Experience Replay Buffer

Loop through these mini-batches to further update w and θ

Today's Takeaways

Actor-critic algorithms parameterize the policy function, and search for optimal parameters based on the gradient of some value function.

- ▶ **Directly parameterizing the policy helps in many ways:**
 - ▶ Agent can learn a probabilistic action space
 - ▶ Agent can learn a continuous action space
 - ▶ Agent can learn an optimal exploration schedule
- ▶ **With some regularization like in PPO, or with very good estimates of q -values like in DDPG, Actor-Critic RL can stabilize and generalize through large and complex state spaces.**

Thank you!