Actors from the Comédie Française, by Antoine Watteau, 1720. Public Domain, https://commons.wikimedia.org/w/index.php?curid=15418670

# Policy Learning
# CS440/ECE448

Mark Hasegawa-Johnson


The Critic, by Lajos Tihanyi. Oil on canvas, 1916. Public Domain, https://commons.wikimedia.org/w/index.php?curid=17837438

# Outline

- The prerequisite for policy learning: Stochastic policy
- Imitation learning
- Actor-critic
- REINFORCE

# What should RL learn?

So far, we've been discussing $P(s'|s,a)$ and $q(s,a)$, because Bellman proved that the optimum policy can be computed by solving

$$u(s) = r(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s'} P(s'|s,a)u(s')$$

…but the solution to a Markov decision process is not $P(s'|s,a)$ or $q(s,a)$. The solution is a policy, $a = \pi(s)$, that tells us what action to perform in state $s$.

Why not just learn $\pi(s)$ directly?

# Stochastic policy

We can't just learn $a = \pi(s)$ directly, because it's not a differentiable function, so gradient descent and most other methods don't work.

To make it learnable, let's define a stochastic policy:

$$\pi_a(s) = P(a \text{ is the action the agent will perform} | \text{state } s)$$

…now, instead of always choosing the same action in state $s$, the agent will, instead, choose action $a$ with probability $\pi_a(s)$.

# Examples of stochastic policies

- Neural network, with hidden features $\boldsymbol{h}$:

$$\pi_a(s) = \frac{\exp(\boldsymbol{w}_a^T \boldsymbol{h})}{\sum_{k \in \mathcal{A}} \exp(\boldsymbol{w}_k^T \boldsymbol{h})}$$

- Epsilon-greedy exploration policy:

$$\pi_a(s) = \begin{cases} (1-\epsilon) + \dfrac{\epsilon}{|\mathcal{A}|} & a = \operatorname{argmax} q(s,a) \\ \dfrac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

# Outline

- The prerequisite for policy learning: Stochastic policy
- Imitation learning
- Actor-critic
- REINFORCE

# How do we learn a stochastic policy?

Suppose we compute the probabilities $\pi_a(s)$ using a neural network with hidden features $\boldsymbol{h}$:

$$\pi_a(s) = \frac{\exp(\boldsymbol{w}_a^T \boldsymbol{h})}{\sum_{k \in \mathcal{A}} \exp(\boldsymbol{w}_k^T \boldsymbol{h})}$$

How should we learn the vectors $\boldsymbol{w}_k$?

# Imitation learning

- Notice: This is like a classification problem!
  - Input: state $s$
  - Output: action $a$
- Solution: Learn to imitate a training database
  - Database consists of paired examples produced by a teacher doing the same task, e.g., a human:

$$\mathcal{D} = \{(s_1, a_1), \cdots, (s_n, a_n)\}$$

  - Gradient descent is used to minimize

$$\mathcal{L} = -\log P(\mathcal{D}) = -\sum_{t=1}^{n} \log P(a_t|s_t) = -\sum_{t=1}^{n} \log \pi_{a_t}(s_t)$$

# Imitation learning

- $\pi_a(s)$ is learned in order to imitate a training database:

$$\mathcal{L} = -\log P(\mathcal{D}) = -\sum_{t=1}^{n} \log P(a_t|s_t) = -\sum_{t=1}^{n} \log \pi_{a_t}(s_t)$$

- Advantages: Usually converges quickly
- Disadvantages: Only learns what the teacher knows

# Outline

- The prerequisite for policy learning: Stochastic policy
- Imitation learning
- Actor-critic
- REINFORCE

# Review: Q-learning

$q(s, a)$ is the expected sum of all future rewards if we perform action $a$ in state $s$ :

$$q(s, a) = r(s) + \gamma \sum_{s'} P(s'|s, a) u(s')$$

$u(s)$ is the expected sum of all future rewards if we are in state $s$ :

$$u(s) = \max_{a \in \mathcal{A}} q(s, a)$$

# The Actor-Critic Algorithm

- Deep Q-learning gives us a network $q(s, a)$ which is very noisy, so we don't really want to trust it
- A policy network can directly estimate $\pi_a(s)$. But how do we train it, unless we imitate human behavior?

Actors from the Comédie Française, by Antoine Watteau, 1720. Public Domain, https://commons.wikimedia.org/w/index.php?curid=15418670

# Actor-critic algorithm

Answer: train two neural nets!

- $q(s, a)$ is the **critic**, and is trained according to the deep Q-learning algorithm.
- $\pi_a(s)$ is the **actor**, and is trained to satisfy the critic

The Critic, by Lajos Tihanyi. Oil on canvas, 1916. Public Domain, https://commons.wikimedia.org/w/index.php?curid=17837438

# The Actor-Critic Algorithm

Main idea:

- The **actor** decides what action to perform:

  $\pi_a(s) =$ Probability of doing action $a$ in state $s$

- The **critic** is a deep Q-learning network that estimates the quality of that action:

  $q(s, a) =$ Expected sum of future rewards if $(s, a)$

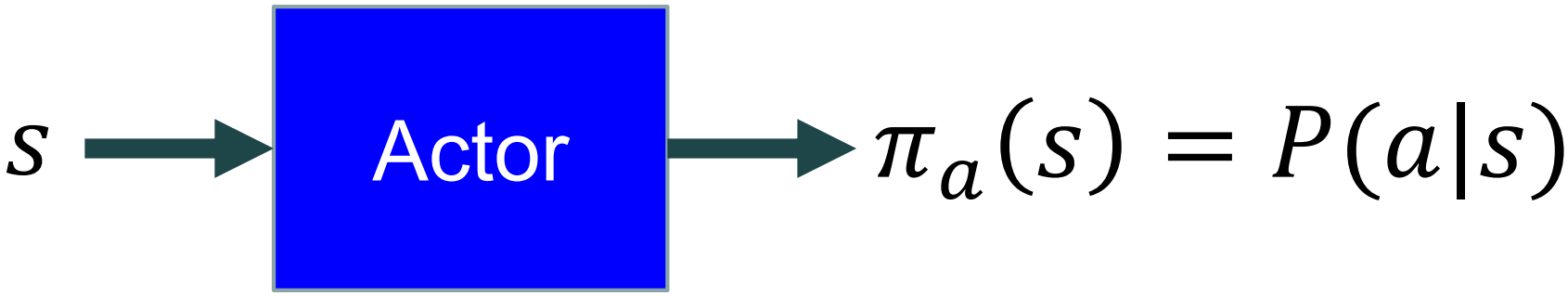
# How to train the actor

The actor, $\pi_a(s)$, is trained to maximize the sum of all future rewards:

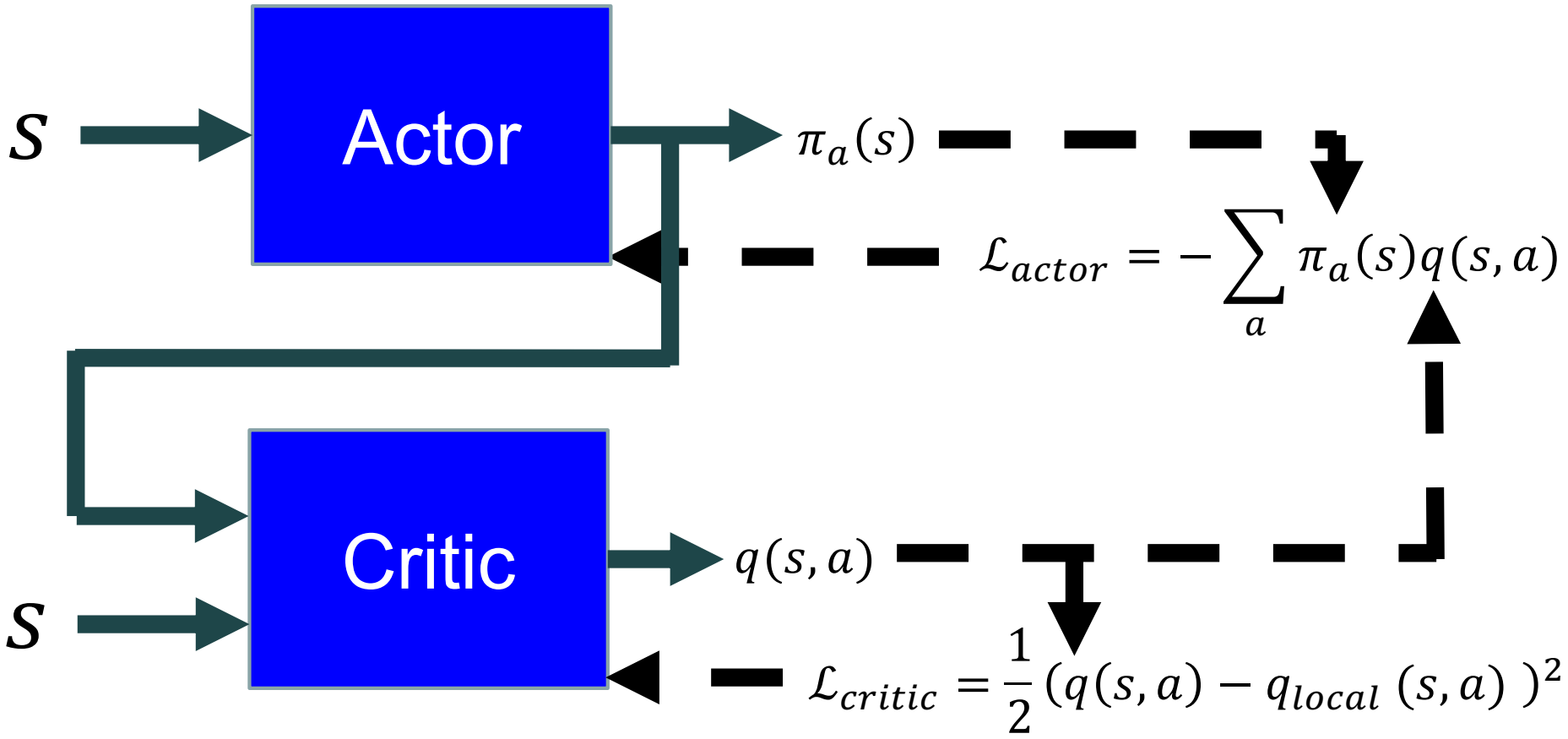
$$\mathcal{L} = -E\left[\begin{matrix}\text{sum of}\\ \text{all future}\\ \text{rewards}\end{matrix}\middle|\ \begin{matrix}\text{state}\\ s\end{matrix}\right] = -\sum_a P\left(\begin{matrix}\text{action}\\ a\end{matrix}\middle|\ \begin{matrix}\text{state}\\ s\end{matrix}\right) E\left[\begin{matrix}\text{sum of}\\ \text{all future}\\ \text{rewards}\end{matrix}\middle|\ \begin{matrix}\text{state } s,\\ \text{action } a\end{matrix}\right]$$

$$= -\sum_a \pi_a(s) q(s,a)$$

# The Actor-Critic Algorithm: Forward-Prop



# The Actor-Critic Algorithm: Back-Prop



# Quiz

Try the quiz!

# Outline

- The prerequisite for policy learning: Stochastic policy
- Imitation learning
- Actor-critic
- REINFORCE

# Why don’t we do the obvious thing?

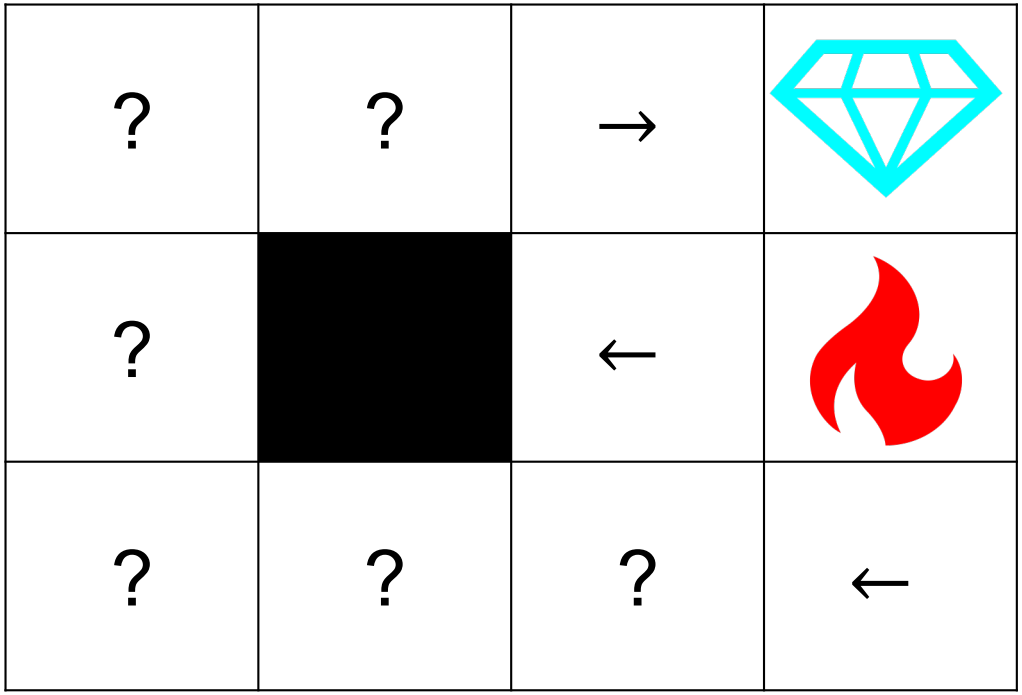
Why don’t we just use the following algorithm?

- Choose action $a$ with probability $\pi_a(s)$
- If we get a positive reward, then increase $\pi_a(s)$

Let’s call this “learning a policy to maximize instant gratification”

# The problem with instant gratification…

…is that it doesn't learn about rewards that take time. For example, in gridworld, it learns the following policy:



# Episodes: A cure for instant gratification

- Start the agent in state $s_1$ at time $1$, and for $t = 1$ to $T$:
  - Compute the probabilities $\pi_a(s_t)$
  - Choose action $a_t$ with probability $\pi_{a_t}(s_t)$
  - Observe the resulting state $s_{t+1}$
- Store the episode

$$\{(s_1, a_1), \cdots, (s_T, a_T), r\}$$

…where $r$ is the total of all rewards accumulated.

# Episodes: A cure for instant gratification

Now we have lots of episodes, $i$, each of the form

$$\{(s_{i,1}, a_{i,1}), \cdots, (s_{i,T}, a_{i,T}), r_i\}$$

We can calculate an expected reward for all episodes, as

$$\mathbb{E}[r] = \frac{1}{n}\sum_{i=1}^{n} r_i \, P(\text{episode}) = \frac{1}{n}\sum_{i=1}^{n} r_i \prod_{t} \pi_{a_{i,t}}(s_{i,t})$$

# REINFORCE: A cure for instant gratification

Suppose that the neural net is parameterized by a matrix $\boldsymbol{W}$ whose elements are $w_{ij}$. We want to adjust $\boldsymbol{W}$ to maximize $\mathbb{E}[r]$:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \eta \frac{\partial}{\partial \boldsymbol{W}} \sum_{i=1}^{n} r_i \prod_t \pi_{a_{i,t}}(s_{i,t})$$

Differentiating a product is ugly. If we can find the average reward $\mu = \frac{1}{n}\sum_{i=1}^{n} r_i$, however, the following update will be almost in the right direction:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \eta \frac{\partial}{\partial \boldsymbol{W}} \sum_{i=1}^{n} (r_i - \mu) \sum_t \frac{\partial}{\partial \boldsymbol{W}} \log \pi_{a_{i,t}}(s_{i,t})$$

# REINFORCE: A cure for instant gratification

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta\boldsymbol{W}, \qquad \Delta\boldsymbol{W} = \eta(r-\mu)\sum_{t=1}^{T}\frac{\partial \log \pi_{a_t}(s_t)}{\partial \boldsymbol{W}}$$

REward Increment=Nonnegative Factor×Offset Reinforcement×Characteristic Eligibility

REINFORCE

That's a complicated formula. But unlike $\frac{\partial r}{\partial \boldsymbol{W}}$, it consists entirely of things we know. Given the record of one episode, $\{(s_1, a_1), \cdots, (s_T, a_T), r\}$, all the terms in $\Delta\boldsymbol{W}$ can be computed.

# What's the constant for?

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta\boldsymbol{W}, \qquad \Delta w_{ij} = \eta(r - \mu) \sum_{t=1}^{T} \frac{\partial \log \pi_{a_t}(s_t)}{\partial w_{ij}}$$

Usually, REINFORCE is implemented in a minibatch of four or five episodes. The constant $\mu$ is set equal to the minibatch-average of $r$, so that:

- If an episode's $r$ is better than average, $\boldsymbol{W}$ is adjusted to increase $\pi_{a_t}(s_t)$ for its actions
- If an episode's $r$ is worse than average, $\boldsymbol{W}$ is adjusted to decrease $\pi_{a_t}(s_t)$ for its actions

# Modern Policy Learning Algorithms

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta\boldsymbol{W}, \qquad \Delta\boldsymbol{W} = \eta(r - \mu)\sum_{t=1}^{T}\frac{\partial \log \pi_{a_t}(s_t)}{\partial \boldsymbol{W}}$$

- The general idea of REINFORCE is that we rollout a complete episode, find its total reward, then adjust $\boldsymbol{W}$ according to (reward $\times$ derivative of the log probability of the episode's actions).
- This general framework is used by all modern RL, including:
  - PPO: Proximal Policy Optimization
  - DPO: Direct Policy Optimization

# Summary

- Policy learning

$$\pi_a(s) = P(a \text{ is the action the agent will perform} | \text{state } s)$$

- Imitation learning: Given a database of teacher actions,

$$\mathcal{L} = -\log P(\mathcal{D}) = -\sum_{t=1}^{n} \log \pi_{a_t}(s_t)$$

- Actor-Critic:

$$\mathcal{L}_{critic} = \frac{1}{2}\left(q(s,a) - q_{local}\,(s,a)\,\right)^2, \qquad \mathcal{L}_{actor} = -\sum_{a} \pi_a(s) q(s,a)$$

- REINFORCE: Given a stored episode $\{(s_1, a_1), \cdots, (s_T, a_T), r\}$,

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}, \qquad \Delta \boldsymbol{W} = \eta(r - \mu) \sum_{t=1}^{T} \frac{\partial \log \pi_{a_t}(s_t)}{\partial \boldsymbol{W}}$$