Introduction to Agentic AI

-- DRL for Agentic AI

Instructor: Guangjing Wang

guangjingwang@usf.edu

Last Lecture

- Reinforcement Learning
- Policy-based and Value-based Methods
- Monte Carlo and Temporal Difference Learning
- Q-learning

This Lecture

- Deep Q-learning
- Policy Gradient
- Actor-Critic Method
- Proximal Policy Optimization

Recap: Value-based and Policy-based methods

- Value-based method:
  - The idea is that an optimal value function leads to an optimal policy π^*.
  - The objective is to **minimize the loss between the predicted and target value** to approximate the true action-value function.
  - The policy is implicit. For instance, in Q-Learning, we used an epsilon-greedy policy.
- Policy-based method:
  - Directly learn to approximate π^* without having to learn a value function.
  - The idea is **to parameterize the policy**. For instance, using a neural network π_θ, this policy will output a probability distribution over actions (stochastic policy).

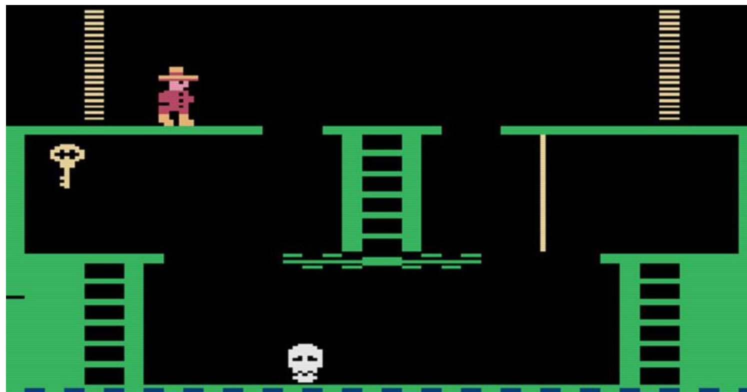
Recap: Q-Learning Algorithm

Algorithm 14: Sarsamax (Q-Learning)

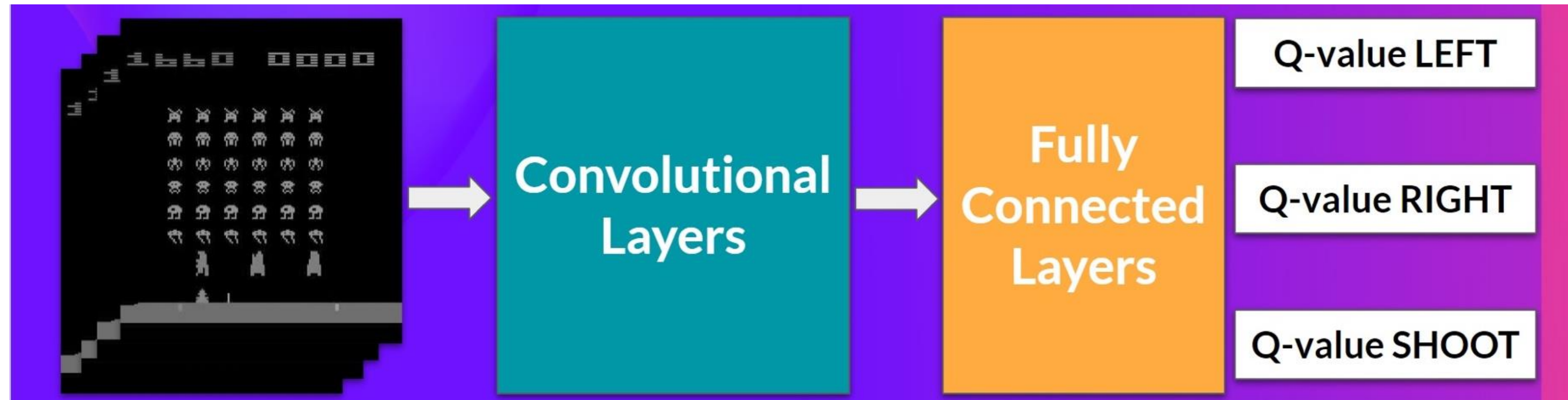
Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s, a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(terminal\text{-}state, \cdot) = 0$) ↖ Step 1
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy) Step 2
    Take action A_t and observe R_{t+1}, S_{t+1} Step 3
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$ Step 4
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Limitations of Q-learning

- Q-learning works for small state and action spaces that can **be represented efficiently by arrays and tables**
  - **Not Scalable**
- **Atari game environments** have an observation space with an image shape of (210, 160, 3), containing values ranging from 0 to 255, so that gives us $256^{210\times160\times3} = 256^{100800}$ possible observations



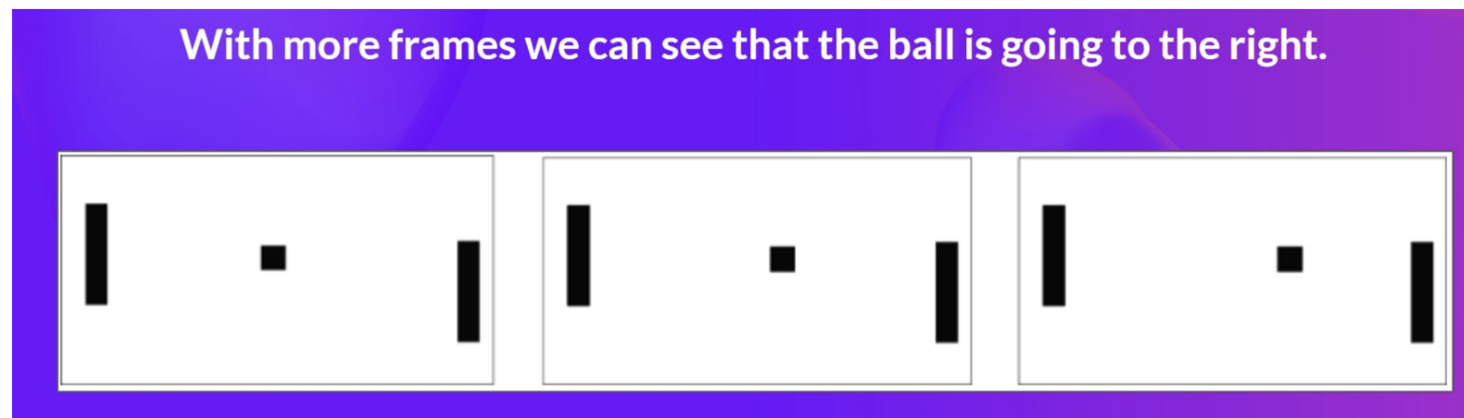
Deep Q-learning



- As input, we take a **stack of 4 frames** passed through the neural network as a state and output a **vector of Q-values for each possible action at that state**.
- Then, like with Q-Learning, we just need to use our epsilon-greedy policy to select which action to take.
- During training, our Deep Q-Network agent will associate a situation with the appropriate action and **learn to play the game well**.

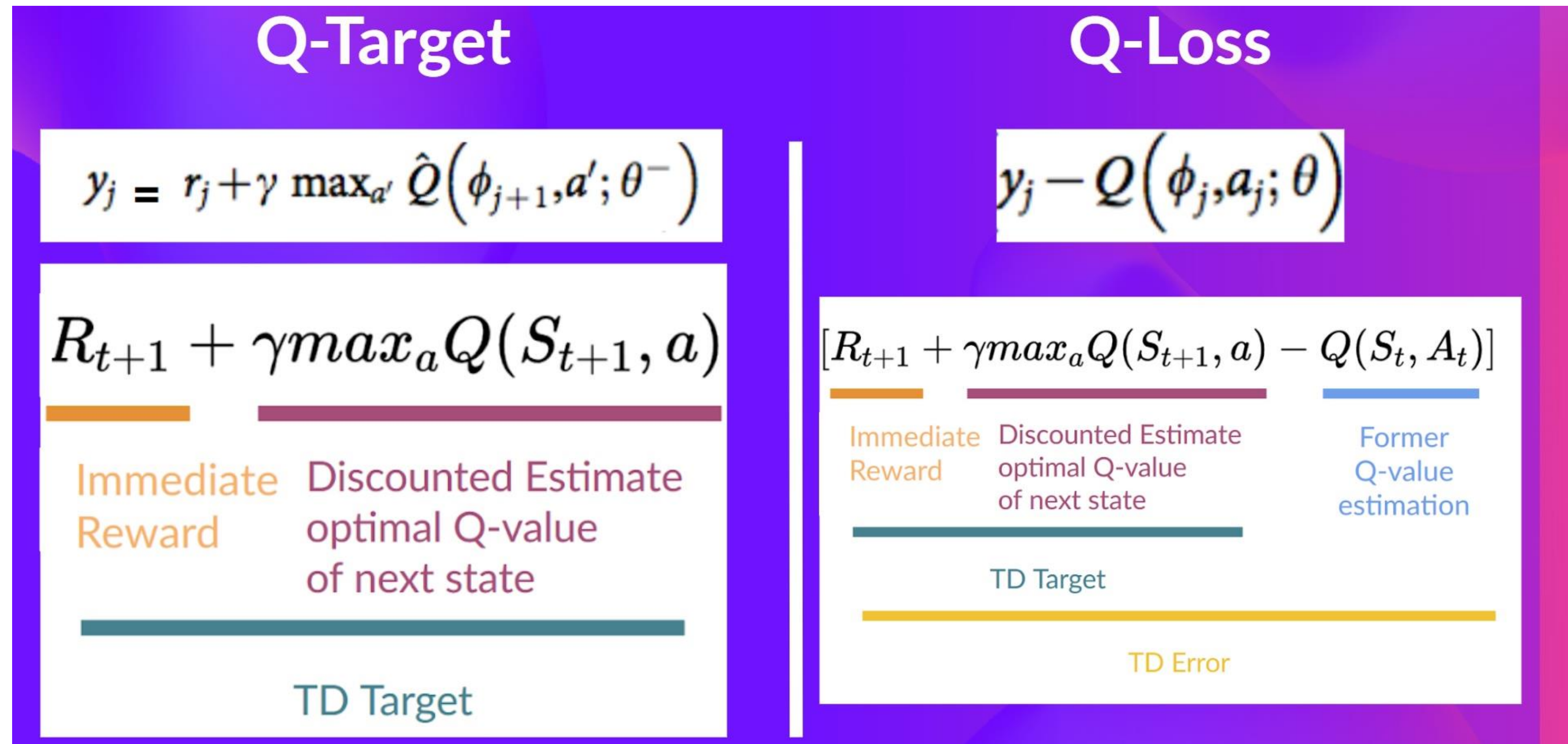
Input Pre-processing in Deep Q-Learning

- Reducing the complexity of state to reduce the computation time.
  - Reducing the state space **from 210x160 to 84x84** and **grayscale** it
  - Why? Colors in Atari environments don’t add important information.
- Stack four frames together
  - Helping us handle the problem of **temporal limitation**.



Deep Q-learning Algorithm

- A loss function that compares Q-value prediction and the Q-target and uses gradient descent to update the weights of the Deep Q-Network to approximate the Q-values better.



Deep Q-learning Training

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Sampling

Training

Policy Gradient

- Deep Q-Learning: value-based deep reinforcement learning algorithm, we **used a deep neural network to approximate the different Q-values for each possible action at a state.**

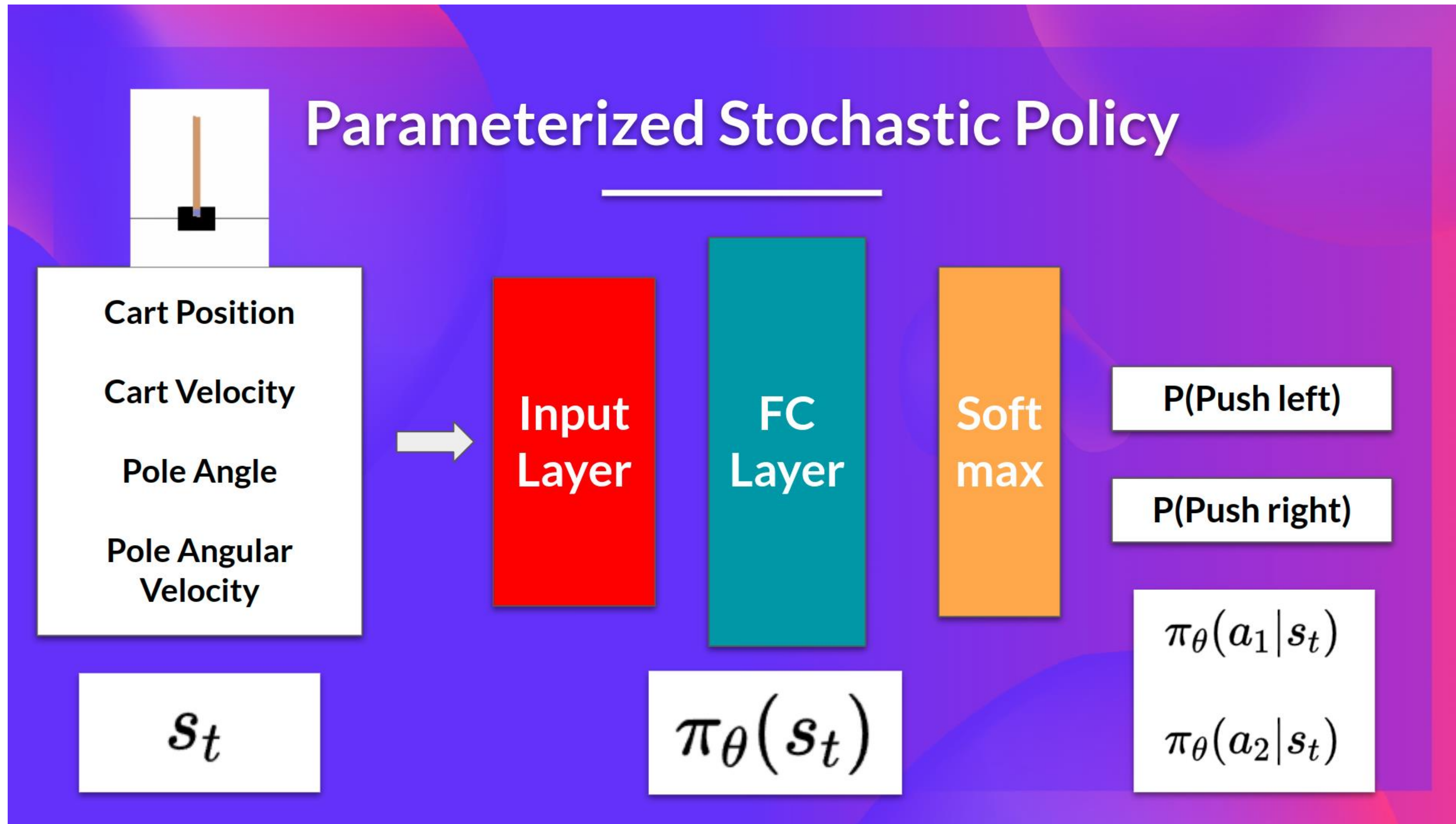
- Policy-based methods optimize the policy directly **without having an intermediate step of learning a value function.**
  - A subset of these methods called **policy gradient**;
  - **Policy gradient** uses a neural network π_θ, this policy will output a probability distribution over actions (stochastic policy), without having to learn a value function

Policy and Policy-gradient Methods

- In *policy-based methods*, we search directly for the optimal policy.
  - We can optimize the parameter θ **indirectly** by maximizing the local approximation of the objective function with techniques like **hill climbing, simulated annealing, or evolution strategies**.

- In *policy-gradient methods*, we also search directly for the optimal policy.
  - We optimize the parameter θ **directly** by performing the **gradient ascent** on the performance of the objective function $J(\theta)$.

Parameterized Stochastic Policy



Policy Gradient

- If we win the episode, we consider that each action taken was good and must be more sampled in the future since they lead to win. Or decrease if we lost.

Training Loop:

Collect an **episode with the π** (policy).

Calculate the return (sum of rewards).

Update the weights of the π:

If positive return → **increase** the probability of each (state, action) pairs taken during the episode.

If negative return → **decrease** the probability of each (state, action) taken during the episode

The Objective Function for Optimization

$$J(\theta) = E_{\tau \sim \pi}[R(\tau)]$$

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots$$

Return: cumulative reward

Gamma: discount rate

Trajectory (read Tau)
Sequence of states and actions

$$J(\theta) = \sum_{\tau} P(\tau;\theta) R(\tau)$$

We calculate the expected return J(θ) by **summing for all trajectories**, the probability of taking that trajectory given θ and the return of this trajectory.

Probability of the trajectory (depends on θ since it **defines the policy that it uses to select the actions of the trajectory which as an impact of the states visited).**

Cumulative return from trajectory

$$P(\tau;\theta) = [\prod_{t=0} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)]$$

Enviroment dynamics (state distribution

Probability of taking that action a_t at state s_t

https://huggingface.co/learn/deep-rl-course/unit4/policy-gradient

The Return in Objective Function

- The return $R(\tau)$ is calculated using a *Monte-Carlo sampling*.
  - We collect a trajectory and calculate the discounted return, and use this score to increase or decrease the probability of every action taken in that trajectory.
  - Unbiased: we are not estimating the return, and we use only the true return.
- Given the stochasticity of the environment (random events during an episode) and stochasticity of the policy, **trajectories can lead to different returns, which can lead to high variance**.
  - The same starting state can lead to very different returns.
  - The solution is to mitigate the variance by **using a large number of trajectories, hoping that the variance introduced in any one trajectory will be reduced in aggregate.**

Reformulate the Objective Function

$$\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) R(\tau^{(i)})$$

Estimation of the gradient (given we use some trajectories to estimate the gradient)

Scaling factor inversely proportional to the number of trajectories (m)

Probability of the agent to select action at from state st given our policy in trajectory (i)

Direction of the steepest increase of the (log) probability of selecting action at from state st

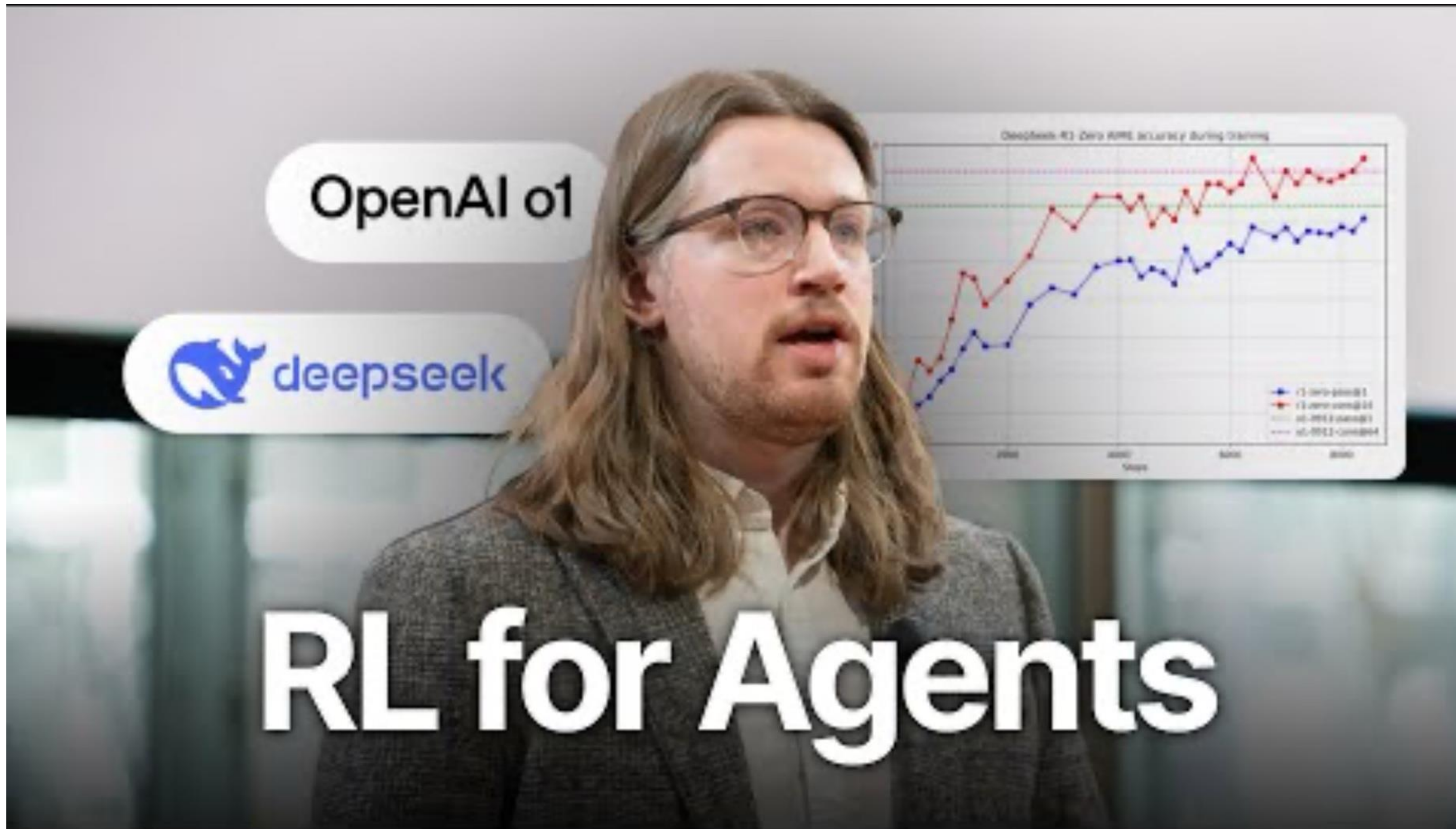
Cumulative return of i-th trajectory

Check the Policy Gradient Theorem: https://huggingface.co/learn/deep-rl-course/unit4/pg-theorem

The Pros and Cons of Policy Gradient Methods

- Pros:
  - Policy-gradient methods can learn a **stochastic** policy.
    - Natural exploration; Break the tie;
  - Policy-gradient methods are more effective in high-dimensional action spaces and continuous actions spaces
- Cons:
  - Policy-gradient methods may converge to **a local maximum** instead of a global optimum.
  - Policy-gradient goes slower: it can take longer to train (inefficient).
  - Policy-gradient can have high variance (noise/inconsistency).

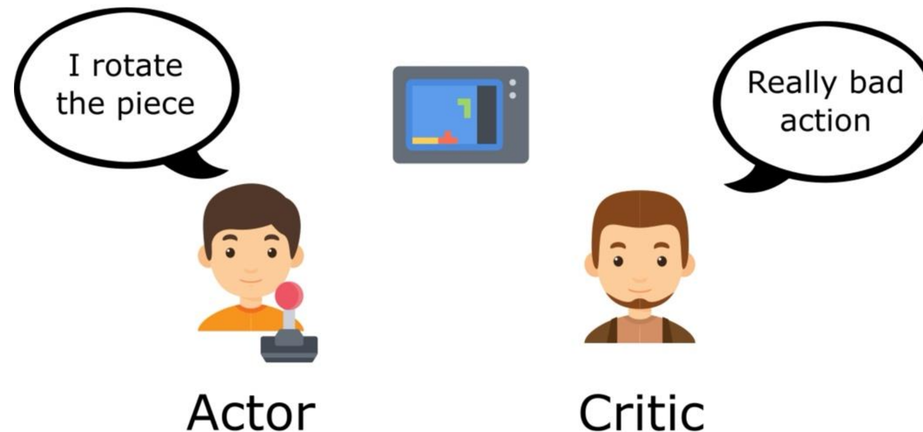
Reinforcement Learning for Agents



https://www.youtube.com/watch?v=JIsgyk0Paic

Actor-Critic Method

- **Actor-Critic methods**, a hybrid architecture combining value-based and Policy-Based methods that helps to stabilize the training by reducing the variance using:
  - *An Actor* that controls **how our agent behaves** (Policy-Based method)
  - *A Critic* that measures **how good the taken action is** (Value-Based method)

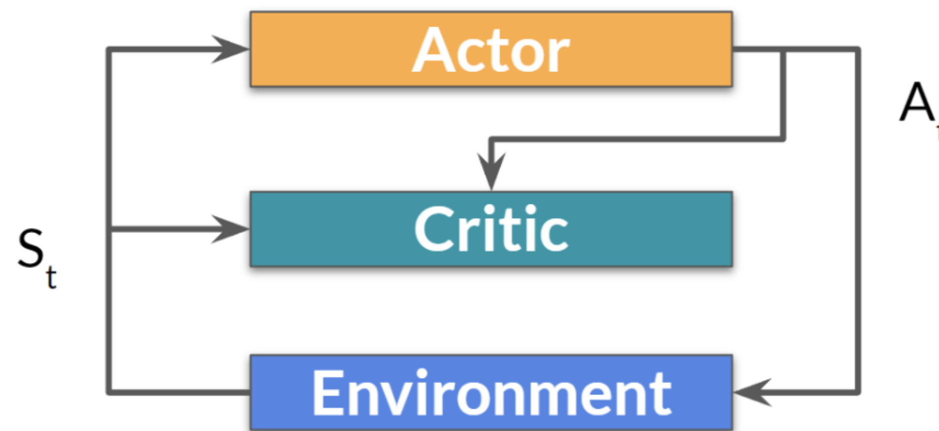


Actor-Critic Method (Cont'd)

With Actor-Critic methods, there are two function approximations (two neural networks):

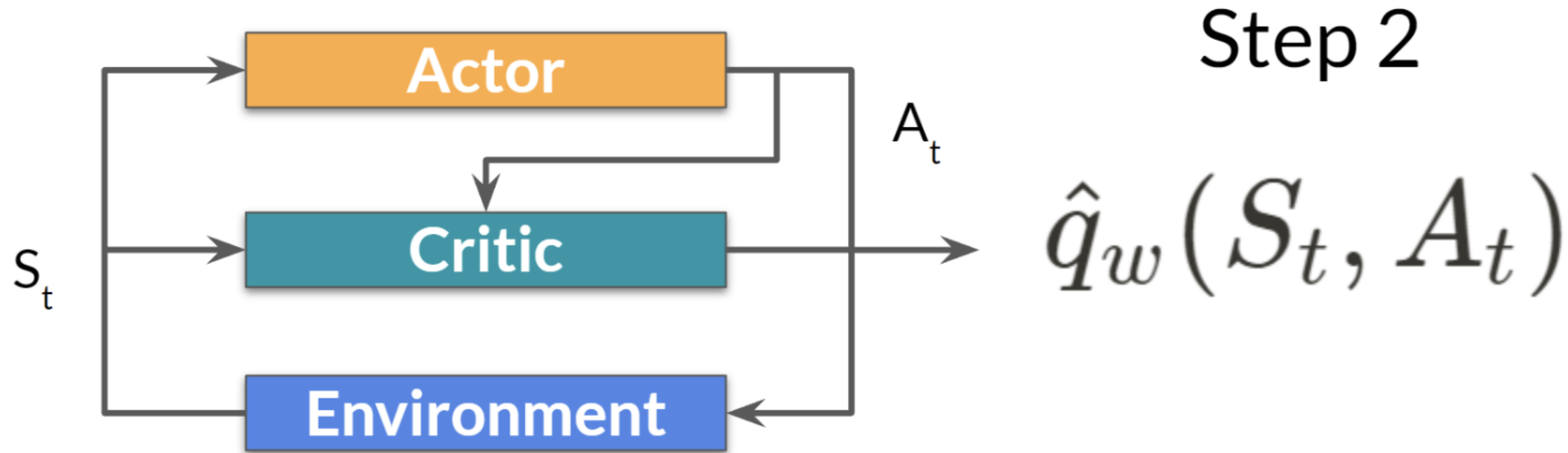
- *Actor*, a **policy function** parameterized by theta: $\pi_\theta(s)$
- *Critic*, a **value function** parameterized by w: $\hat{q}_w(s, a)$

- Pass state S_t into Actor and Critic
- The Actor (Policy) Outputs an action A_t



Step 1

Actor-Critic Method (Cont'd)



- The Critic (Value Function) takes that action and state as input to compute the value of taking that action at that state: the Q-value.
- The action A_t performed in the environment outputs a new state S_{t+1} and a reward R_{t+1}.

Actor-Critic Method (Cont'd)

- The Actor updates its policy parameters using the value.

$$\Delta\theta = \alpha \nabla_\theta (log \pi_\theta (s, a)) \hat{q}_w (s, a)$$

Change in policy parameters (weights) — Action value estimate

- The Actor produces the next action to take at A_{t+1} given the new state S_{t+1}. The Critic then updates its value parameters.

$$\Delta w = \beta \left(R(s, a) + \gamma \hat{q}_w (s_{t+1}, a_{t+1}) - \hat{q}_w (s_t, a_t) \right) \nabla_w \hat{q}_w (s_t, a_t)$$

Policy and value have different learning rates

TD error

Gradient of our value function

Adding Advantage in Actor-Critic (A2C)

- Using the Advantage function as Critic instead of the Action value function.
- The Advantage function calculates the relative advantage of an action compared to the others possible at a state.
  - How taking that action at a state is better compared to the average value of the state

$$A(s, a) = \underbrace{Q(s, a)}_{\text{q value for action a in state s}} - \underbrace{V(s)}_{\text{average value of that state}}$$

Proximal Policy Optimization (PPO)

- We use a ratio that indicates the difference between our current and old policy and clip this ratio to a specific range $[1-\epsilon, 1+\epsilon]$.
  - Improving our agent's training stability by avoiding policy updates that are too large.

- A too-big step in a policy update can result in falling "off the cliff" (getting a bad policy) and taking a long time or even having no possibility to recover.

PPO's Objective Function and Clipped Surrogate Objective Function

$$L^{PG}(\theta) = E_t[\log \pi_\theta(a_t|s_t) * A_t]$$

log probability of taking that action at that state

Advantage if A>0, this action is better than the other action possible at that state

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

The probability of taking action a_t at state s_t in the current policy, divided by the same for the previous policy.

Other Opinions about Reinforcement Learning

https://www.youtube.com/watch?v=36OBX5lQjGc (Oct. 18, 2025)

References

- https://huggingface.co/learn/deep-rl-course/