(a)
Environment
Critic
TD
TD error δ
Actor
States/Stimuli
(b)
Environment
Reward
Critic
Ventral striatum
VTA SNc
Dopamine
Cortex
Dorsal striatum
Actor
Actions

S&DS 365 / 665

Intermediate Machine Learning

Reinforcement Learning: Actor-Critic Methods

November 6

Yale

Reminders

- Final exam: Sunday, December 15, 2024 at 2pm
- https://registrar.yale.edu/general-information/final-exams
- Assignment 4 due Monday, November 18
- Quiz 4 posted today on Canvas at 2:30pm (48 hours/30 min)

Reminders

- Final exam: Sunday, December 15, 2024 at 2pm
- https://registrar.yale.edu/general-information/final-exams
- Assignment 4 due Monday, November 18
- Quiz 4 posted today on Canvas at 2:30pm (48 hours/30 min)
  - Graphs
  - GNNs
  - Q-learning

Outline

- Policy iteration and gradients (continued)
- Combining policy and value estimation
- Actor-critic methods
- Demo: Cartpole
- Neuroscience connection
- Demo: SET

Main reference: “Reinforcement Learning: An Introduction” (Second Edition), Richard S. Sutton and Andrew G. Barto

Multi-armed bandits

Multi-armed bandits

- The rewards are independent and noisy
- Arm k has expected payoff μ_k with variance σ_k^2 on each pull
- Each time step, pull an arm and observe the resulting reward
- Played often enough, can estimate mean reward of each arm
- What is the best policy?
- Exploration-exploitation tradeoff

Multi-armed bandits

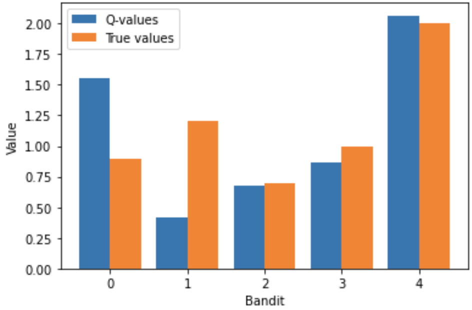
Can treat this as an RL problem and hit it with a big hammer:
Deep Q-learning

Note: *Contextual* bandits is a framework very similar to RL

Multi-armed bandits

```
======episode 10000 ======
Q-values ['1.556', '0.412', '0.675', '0.866', '2.065']
Deviation ['72.8%', '-65.7%', '-3.6%', '-13.4%', '3.3%']

<Figure size 864x504 with 0 Axes>
```

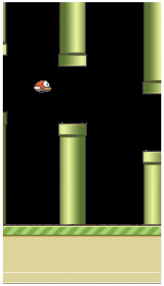


Please review the notebook from last class

Assn 4: Flappy Bird

Problem 3: Deep Q-Learning for Flappy Bird (25 points)

In this problem, we will walk you through the implementation of deep Q learning to learn to play the Flappy Bird game.



Policy iteration: Idea

0. Initialize policy arbitrarily
1. Compute values for current policy (policy evaluation)
2. Update policy to match values (policy improvement)
3. Go to 1.

Policy iteration

- As for vanilla Q-learning, this only works for small state spaces
- A “tabular” method, computes all values $V(s)$ and actions $\pi(s)$
- This will compute an optimal policy—it will satisfy Bellman’s equations. Step 2 can only increase the value of the policy.

Policy gradient methods: Loss function

We start with the loss function: Expected reward $\mathcal{J}(\theta) = \mathbb{E}(R)$

- Parameterize the policy—$\pi(s; \theta)$—and use features of states
- Perform gradient ascent of $\mathcal{J}(\theta)$
- Well-suited to deep learning approaches

Policy gradient methods: Loss function

Policy is probability distribution $\pi_\theta(a \mid s)$ over actions given state s.

The episode unfolds as a random sequence

$$\tau : (s_0, a_0) \to (s_1, r_1, a_1) \to (s_2, r_2, a_2) \to \cdots \to (s_T, r_T, a_T) \to s_{T+1}$$

where s_{T+1} is a terminal state.

Reward $R(\tau)$

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

Policy gradient methods: Loss function

Policy is probability distribution $\pi_\theta(a \,|\, s)$ over actions given state s.

The episode unfolds as a random sequence

$$\tau : (s_0, a_0) \to (s_1, r_1, a_1) \to (s_2, r_2, a_2) \to \cdots \to (s_T, r_T, a_T) \to s_{T+1}$$

where s_{T+1} is a terminal state.

Expected reward

$$\mathcal{J}(\theta) = \mathbb{E}_\theta(R(\tau))$$

Calculating the gradient

Using Markov property, calculate $\mathbb{E}_\theta(R(\tau))$ as

$$\mathbb{E}_\theta(R(\tau)) = \int p(\tau \mid \theta) R(\tau)\, d\tau$$

$$p(\tau \mid \theta) = \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t)\, p(s_{t+1}, r_{t+1} \mid s_t, a_t)$$

If states or rewards are finite the integral becomes a sum, or a mix of sums and integrals.

Calculating the gradient

Using Markov property, calculate $\mathbb{E}_\theta(R(\tau))$ as

$$\mathbb{E}_\theta(R(\tau)) = \int p(\tau \mid \theta) R(\tau)\, d\tau$$

$$p(\tau \mid \theta) = \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t)\, p(s_{t+1}, r_{t+1} \mid s_t, a_t)$$

If states or rewards are finite the integral becomes a sum, or a mix of sums and integrals. It follows that

$$\nabla_\theta \log p(\tau \mid \theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) = \sum_{t=0}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)}$$

Calculating the gradient

Now we use

$$
\begin{aligned}
\nabla_\theta \mathcal{J}(\theta) &= \nabla_\theta \, \mathbb{E}_\theta R(\tau) \\
&= \nabla_\theta \int R(\tau)\, p(\tau \mid \theta)\, d\tau \\
&= \int R(\tau)\, \nabla_\theta p(\tau \mid \theta)\, d\tau \\
&= \int R(\tau)\, \frac{\nabla_\theta p(\tau \mid \theta)}{p(\tau \mid \theta)}\, p(\tau \mid \theta)\, d\tau \\
&= \mathbb{E}_\theta \Big(R(\tau) \nabla_\theta \log p(\tau \mid \theta) \Big) \\
&= \mathbb{E}_\theta \Big(R(\tau) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \Big)
\end{aligned}
$$

Need regularity conditions, such as uniform continuity of the derivatives of the policy.

Why is this important?

- This manipulation is important because it gets the policy explicity into the objective function
- Similar to the “reparameterization trick” for VAEs

Approximating the gradient

We can approximate this by sampling:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N} R(\tau^{(i)}) \nabla_\theta \log p(\tau^{(i)} \mid \theta) \\ &= \frac{1}{N}\sum_{i=1}^{N} R(\tau^{(i)}) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \\ &\equiv \widehat{\nabla_\theta \mathcal{J}(\theta)}\end{aligned}$$

Approximating the gradient

We can approximate this by sampling:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N} R(\tau^{(i)})\nabla_\theta \log p(\tau^{(i)} \mid \theta)\\ &= \frac{1}{N}\sum_{i=1}^{N} R(\tau^{(i)}) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})\\ &\equiv \widehat{\nabla_\theta \mathcal{J}(\theta)}\end{aligned}$$

The policy gradient algorithm is then

$$\theta \longleftarrow \theta + \alpha \widehat{\nabla_\theta \mathcal{J}(\theta)}$$

Approximating the gradient

With discounting this becomes

$$\begin{aligned}\nabla_\theta \mathcal{J}(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} R_{t+1}(\tau^{(i)})\nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \\ &\equiv \widehat{\nabla_\theta \mathcal{J}(\theta)}\end{aligned}$$

where discounted long term reward is

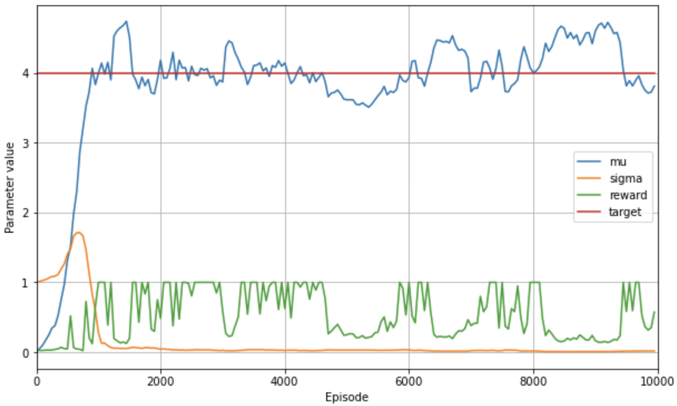
$$\begin{aligned}R_t(\tau) &= r_t + \gamma R_{t+1}(\tau) \\ R_{T+1}(\tau) &= 0\end{aligned}$$

Demo

Demo

- In this demo we try to estimate a fixed target value
- The policy chooses an “action” by sampling according a Gaussian with an estimated mean and variance.
- (Similar to an encoder network for VAEs)
- The reward depends on how close the action is to the target
- The code applies policy gradient descent directly

Demo



Let’s go to the notebook

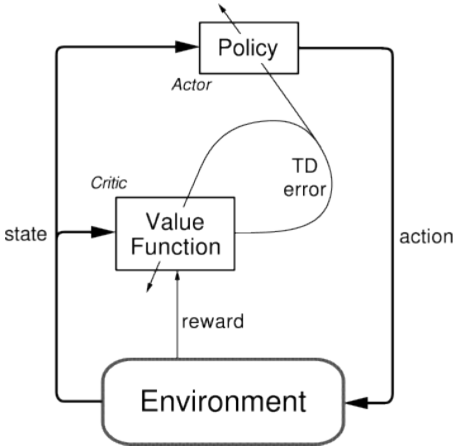
Actor-critic approaches: Idea

- Estimate policy and value function together
- Actor: policy used to select actions
- Critic: value function used to criticize actor
- Error signal from the critic drives all learning
- An on-policy approach

Actor-critic approaches

- After each selected action, critic evaluates new state
- Have things gone better or worse than expected?
- The error signal is used to update actor and value function

Actor-critic approaches



Sutton

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$\delta_t > 0$: action was better than expected

$\delta_t < 0$: action was worse than expected

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Value function is updated as

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t$$

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Used to update parameters of policy.

If δ_t is positive (negative), action a_t should become more (less) probable in state s_t

For example, with

$$\pi_\theta(a \mid s) = \text{Softmax}\Big\{ f_\theta(s, a_1), \ldots, f_\theta(s, a_D) \Big\}$$

parameters θ adjusted so $f_\theta(s_t, a_t)$ increases (decreases)

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

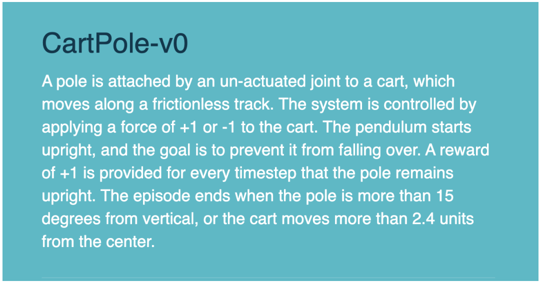
Used to update parameters of policy.

For example, with

$$\pi_\theta(a \,|\, s) = \text{Softmax}\Big\{ f_\theta(s, a_1), \ldots, f_\theta(s, a_D) \Big\}$$

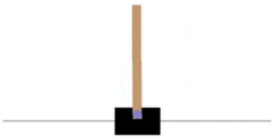
parameters θ adjusted so $f_\theta(s_t, a_t)$ increases (decreases)

$$\theta \leftarrow \theta + \eta \delta_t \nabla_\theta \log \pi_\theta(a_t \,|\, s_t)$$



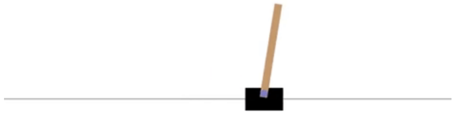
CartPole-v0

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.



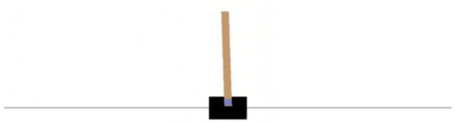
https://gym.openai.com/envs/CartPole-v0/
tensorflow tutorial

Before learning:



https://gym.openai.com/envs/CartPole-v0/
tensorflow tutorial

After learning:



https://gym.openai.com/envs/CartPole-v0/
tensorflow tutorial

Demo

If you study the code, you will see how

1. Actor log-probs, critic values, and rewards are buffered
2. After the episode, TD errors are calculated at each time step
3. Loss for actor is negative log-prob weighted by TD error
4. Loss for critic is absolute value of TD error
5. Gradients of actor/critic networks computed using auto diff
6. Parameters of network are updated

Summary: Actor-critic RL

- Estimate policy and value function together
- Actor: policy used to select actions
- Critic: value function used to criticize actor
- Error signal from the critic drives learning
- Connections to neuroscience of behavior and reward