S&DS 365 / 665

Intermediate Machine Learning

Reinforcement Learning: Actor-Critic Methods

April 13

Yale

Reminders

- Assignment 3 due tonight
- Assignment 4 (last!) out today
- Quiz 4 (last!) next Wednesday

Outline

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

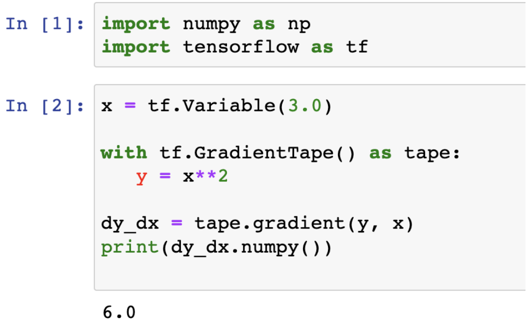
Main reference: “Reinforcement Learning: An Introduction” (Second Edition), Richard S. Sutton and Andrew G. Barto

Recall: Autograd

- For supervised problems, loss function is $F(\widehat{Y}, Y)$
- Can program this directly
- Often RL loss functions are built up dynamically
- Automatic differentiation allows us to handle this

`https://www.tensorflow.org/guide/autodiff`

Automatic differentiation: Hello world!



```
In [1]: import numpy as np
        import tensorflow as tf
```

```
In [2]: x = tf.Variable(3.0)

        with tf.GradientTape() as tape:
           y = x**2

        dy_dx = tape.gradient(y, x)
        print(dy_dx.numpy())
```

```
        6.0
```

https://www.tensorflow.org/guide/autodiff

Automatic differentiation: Hello world!

In [3]:
```
w = tf.Variable(tf.random.normal((3, 2)), name='w')
b = tf.Variable(tf.zeros(2, dtype=tf.float32), name='b')
x = [[1., 2., 3.]]

with tf.GradientTape() as tape:
    y = x @ w + b
    loss = tf.reduce_mean(y**2)

[dloss_dw, dloss_db] = tape.gradient(loss, [w, b])
print(dloss_dw.numpy(), "\n\n", dloss_db.numpy())
```

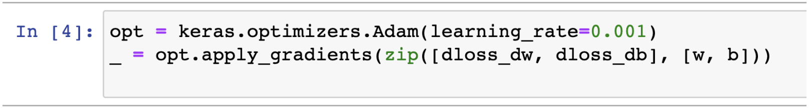
```
[[2.3997068  0.70033383]
 [4.7994137  1.4006677 ]
 [7.1991205  2.1010015 ]]

 [2.3997068  0.70033383]
```

https://www.tensorflow.org/guide/autodiff

Automatic differentiation: Gradient updates

Parameters are then updated as shown here:



```
In [4]: opt = keras.optimizers.Adam(learning_rate=0.001)
        _ = opt.apply_gradients(zip([dloss_dw, dloss_db], [w, b]))
```

We’ll see examples shortly

https://www.tensorflow.org/guide/autodiff

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

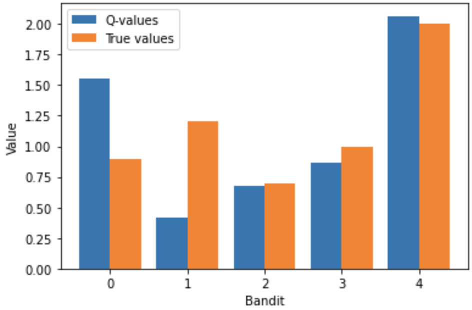
Multi-armed bandits

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

Multi-armed bandits

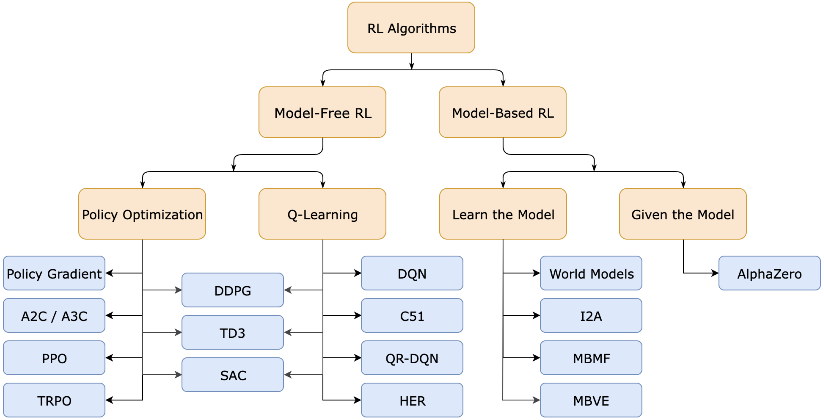
```
======episode 10000 ======
Q-values ['1.556', '0.412', '0.675', '0.866', '2.065']
Deviation ['72.8%', '-65.7%', '-3.6%', '-13.4%', '3.3%']

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



Let's go to the notebook

Landscape of RL algorithms



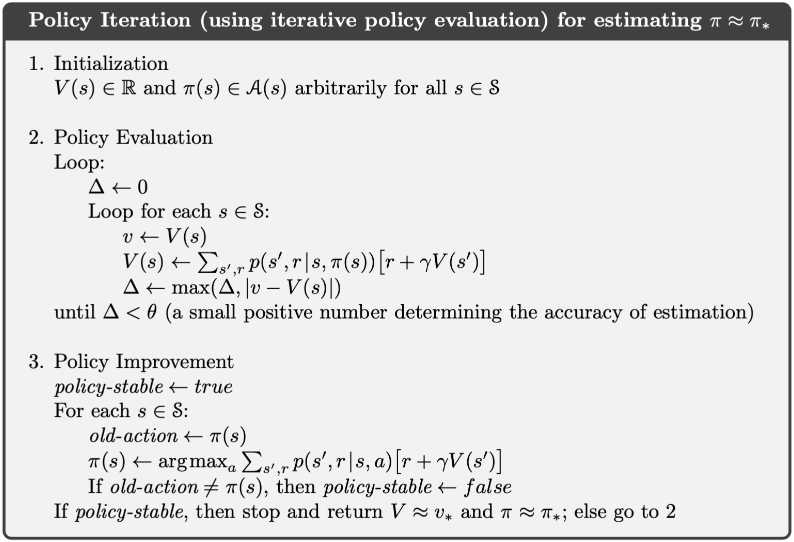
https://spinningup.openai.com/en/latest/index.html

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.

This will compute an optimal policy—it will satisfy Bellman's equations. Step 2. can only increase the value of the policy.

Policy iteration: Algorithm



Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[ r + \gamma V(s') \big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $\textit{policy-stable} \leftarrow \textit{true}$
   For each $s \in \mathcal{S}$:
       $\textit{old-action} \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[ r + \gamma V(s') \big]$
       If $\textit{old-action} \neq \pi(s)$, then $\textit{policy-stable} \leftarrow \textit{false}$
   If $\textit{policy-stable}$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

Policy iteration

- As for vanilla Q-learning, this only works for small state spaces
- A “tabular” method, computes all values $V(s)$ and actions $\pi(s)$

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 $\pi_\theta(a \,|\, s)$ 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 $R(\tau)$, for example

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

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

$$\begin{aligned}
\nabla_\theta \mathcal{J}(\theta) &= \nabla_\theta \, \mathbb{E}_\theta R(\tau) \\
&= \nabla_\theta \int R(\tau)\, p(\tau \,|\, \theta)\, d\tau \\
&= \int R(\tau)\, \nabla_\theta p(\tau \,|\, \theta)\, d\tau \\
&= \int R(\tau)\, \frac{\nabla_\theta p(\tau \,|\, \theta)}{p(\tau \,|\, \theta)}\, p(\tau \,|\, \theta)\, d\tau \\
&= \mathbb{E}_\theta \Big(R(\tau) \nabla_\theta \log p(\tau \,|\, \theta) \Big)
\end{aligned}$$

Approximating the gradient

We can approximate this by sampling:

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

Approximating the gradient

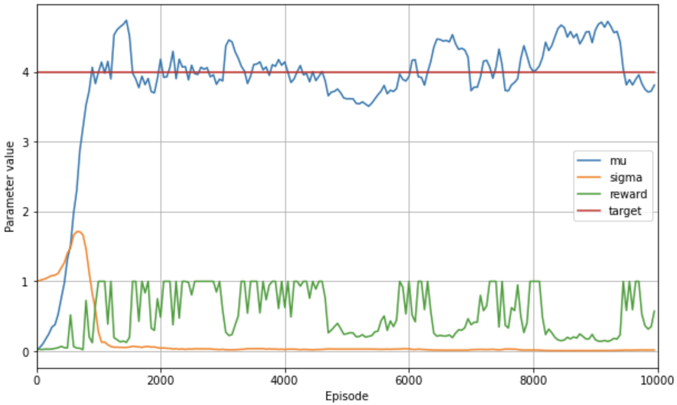
We can approximate this by sampling:

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

The policy gradient algorithm is then

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

Demo



Let’s go to the notebook

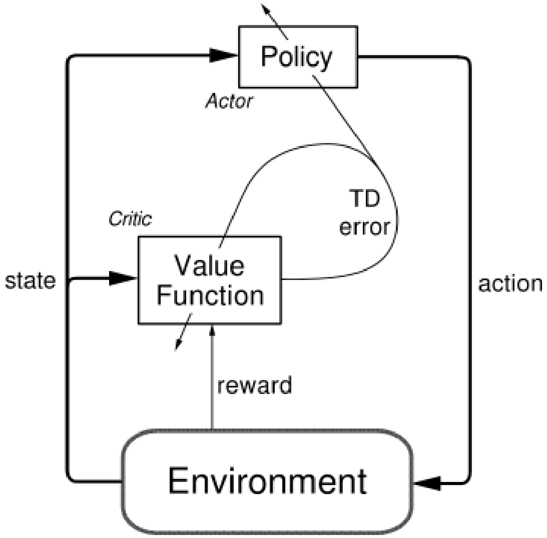
Actor-critic approaches: Idea

- 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



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.

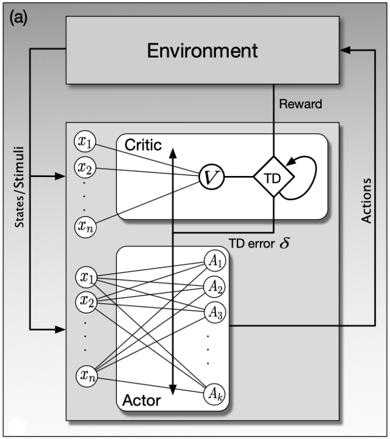
If δ_t is positive (negative), action a_t should become more (less) probable in state s_t

For example, with

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

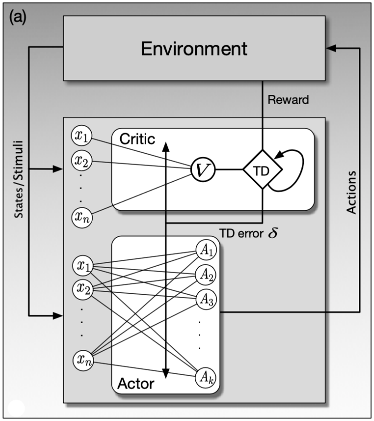
parameters θ adjusted so $f_\theta(s_t, a_t)$ increases (decreases)

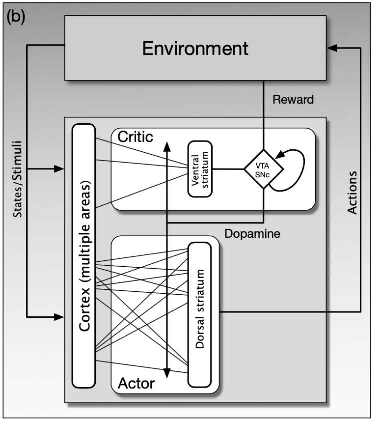
Neuroscience connection



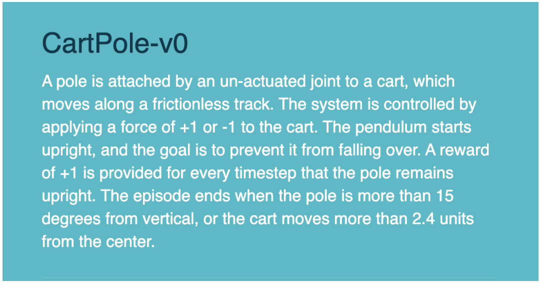
Y. Takahashi, G. Schoenbaum, and Y. Niv, "Silencing the critics: Understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an Actor/Critic model.

Neuroscience connection



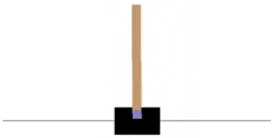


Y. Takahashi, G. Schoenbaum, and Y. Niv, “Silencing the critics: Understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an Actor/Critic model.



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

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 all learning
- Connections to neuroscience of behavior and reward