Day 3. Policy Gradient and Actor-Critic

NPEX Reinforcement Learning

July 28, 2021

Jaeuk Shin, Minkyu Park

CORE
Control + Optimization Research Lab

Policy Gradient - Recap

Cast Reinforcement learning problem into optimization problem

$$\max_\theta J(\theta) := E_{\tau \sim p_\theta(\tau)}\left[\sum \gamma^t r_t\right]$$

Where policy is represented with parameter $\theta \rightarrow \pi_\theta$

We can solve above optimization problem with *gradient ascent*

$$\theta \leftarrow \theta + \alpha \underline{\nabla_\theta J(\theta)}$$

How?

REINFORCE Algorithm

Recall the gradient of parametric policy π_θ for objective function J

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \Big[\ \Big(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \Big) \ \Big(\sum_{t=0}^{T} r(s_t, a_t) \Big) \ \Big]$$

REINFORCE Algorithm

Recall the gradient of parametric policy π_θ for objective function J

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\ \left(\ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \ \right) \left(\ \sum_{t=0}^{T} r(s_t, a_t) \ \right) \ \right]$$

Approximate with sample mean by sampling N trajectories $\tau = (s_0, a_0, \cdots, s_T, a_T)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \ \right) \left(\ \sum_{t=0}^{T} r(s_t, a_t) \ \right)$$

CORE
Control + Optimization Research Lab

REINFORCE Algorithm

Recall the gradient of parametric policy π_θ for objective function J

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \Big[\Big(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Big) \Big(\sum_{t=0}^{T} r(s_t, a_t) \Big) \Big]$$

Approximate with sample mean by sampling N trajectories $\tau = (s_0, a_0, \cdots, s_T, a_T)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \Big(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Big) \Big(\sum_{t=0}^{T} r(s_t, a_t) \Big)$$

$\Downarrow$

REINFORCE Algorithm

REINFORCE Algorithm

Additional approximation and baseline

Constant

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\ \sum_{t=0}^{T}\nabla_\theta \log\pi_\theta(a_t|s_t)\ \right)\left(\ \sum_{t=0}^{T} r(s_t,a_t)\ \right)$$

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\left[\left(\nabla_\theta \log\pi_\theta(a_t|s_t)\right)\left(\sum_{t=0}^{T} r(s_t,a_t)\right)\right]$$

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\left[\left(\nabla_\theta \log\pi_\theta(a_t|s_t)\right)\left(\sum_{t'=t}^{T} r(s_{t'},a_{t'})\right)\right]$$

Subtract baseline!

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\left(\nabla_\theta \log\pi_\theta(a_t|s_t)\right)\left(Q^{\pi_\theta}(s_t,a_t) - v^{\pi_\theta}(s_t)\right)$$

$$= A^{\pi_\theta}(s_t,a_t)$$

CORE

Control + Optimization Research Lab

REINFORCE - Implementation

How to get $\nabla_\theta \log \pi_\theta(a_t|s_t)$?

```
class Policy(nn.Module):
    def __init__(self, state_dim, num_action, hidden_size1, hidden_size2):
        super(Policy, self).__init__()
        self.fc1 = nn.Linear(state_dim, hidden_size1)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.fc3 = nn.Linear(hidden_size2, num_action)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        action_score = self.fc3(x)
        return F.softmax(action_score, dim=1)
```

NN returns $|\mathcal{A}|$ dimension vector

Return softmax value of length $|\mathcal{A}|$, which can be used as probability to choose action $\rightarrow \pi_\theta$

REINFORCE - Implementation

How to get $\nabla_\theta \log \pi_\theta(a_t|s_t)$?

```python
def select_action(state):
    state = torch.from_numpy(state).float().unsqueeze(0)
    state = state.to(device)
    probs = pi(state)

    m = Categorical(probs)
    action = m.sample()

    return action.item(), m.log_prob(action)
```

NN output - probability for each action

This function builds probability distribution for discrete action space and give $\log \pi_\theta(a_t|s_t)$, $\nabla_\theta \log \pi_\theta(a_t|s_t)$

REINFORCE - Implementation

Create $\nabla_\theta J(\theta)$

```python
def calculate_PG(pi_returns_discounted, dataset):
    pi_loss = 0
    for data in dataset:
        advantage, DCR = [], 0
        for r in reversed(data['reward']):
            # TODO : Caculate discounted redataset.append(data)turn from t=i
            # Hint : reversed() will give saved rewards in reversed order
            DCR = r + gamma * DCR

            # Q(s,a) is replaced with discounted sum of rewards (DCR)
            # v(s) is replaced with empirical v(s_0)
            advantage.insert(0, DCR - np.mean(pi_returns_discounted))

        # TODO : alternate between two losses to see difference!
        pi_loss_vanilla = [log_pi * DCR for log_pi in data['log_pi']]
        pi_loss_baseline = [log_pi * a for log_pi, a in zip(data['log_pi'], advantage)]

        # Take mean value
        pi_loss += torch.cat(pi_loss_vanilla).sum()

    return pi_loss / num_trajs
```

Reversed order

$$Q^{\pi_\theta}(s_t, a_t) \approx \sum_{t=t'}^{T} \gamma^{t-t'} r(s_{t'}, a_{t'})$$

baseline with $E_{s_0,\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$

Choose between loses!

CORE
Control + Optimization Research Lab

REINFORCE - Implementation

Using baseline increaes performance dramatically!

```
Epoch 0 Return_mean: 22.74      Return_std: 12.03       Time
Epoch 5 Return_mean: 25.18      Return_std: 13.00       Time
Epoch 10        Return_mean: 30.49      Return_std: 15.26
Epoch 15        Return_mean: 38.63      Return_std: 19.31
Epoch 20        Return_mean: 44.66      Return_std: 22.34
Epoch 25        Return_mean: 60.22      Return_std: 30.04
Epoch 30        Return_mean: 57.98      Return_std: 28.14
Epoch 35        Return_mean: 66.45      Return_std: 31.69
Epoch 40        Return_mean: 85.14      Return_std: 43.28
Epoch 45        Return_mean: 111.59     Return_std: 59.01
Epoch 50        Return_mean: 126.32     Return_std: 55.62
Epoch 55        Return_mean: 155.78     Return_std: 65.45
Epoch 60        Return_mean: 191.32     Return_std: 77.60
Epoch 65        Return_mean: 205.11     Return_std: 79.55
Epoch 70        Return_mean: 208.84     Return_std: 88.27
Epoch 75        Return_mean: 158.31     Return_std: 53.31
Epoch 80        Return_mean: 141.18     Return_std: 53.16
Epoch 85        Return_mean: 193.22     Return_std: 58.09
Epoch 90        Return_mean: 160.01     Return_std: 45.93
Epoch 95        Return_mean: 307.46     Return_std: 101.98
```

Without baseline

```
Epoch 0 Return_mean: 19.76      Return_std: 8.49        Time(
Epoch 5 Return_mean: 29.37      Return_std: 18.32       Time(
Epoch 10        Return_mean: 40.56      Return_std: 25.14
Epoch 15        Return_mean: 53.07      Return_std: 26.29
Epoch 20        Return_mean: 76.31      Return_std: 44.41
Epoch 25        Return_mean: 131.40     Return_std: 71.74
Epoch 30        Return_mean: 192.59     Return_std: 94.73
Epoch 35        Return_mean: 279.06     Return_std: 117.43
Epoch 40        Return_mean: 325.63     Return_std: 113.61
Epoch 45        Return_mean: 399.72     Return_std: 122.64
Epoch 50        Return_mean: 429.90     Return_std: 97.34
Epoch 55        Return_mean: 441.48     Return_std: 86.38
Epoch 60        Return_mean: 456.22     Return_std: 86.65
Epoch 65        Return_mean: 474.78     Return_std: 66.90
Epoch 70        Return_mean: 493.40     Return_std: 33.79
Epoch 75        Return_mean: 483.57     Return_std: 57.95
Epoch 80        Return_mean: 489.79     Return_std: 44.05
Epoch 85        Return_mean: 479.35     Return_std: 59.41
Epoch 90        Return_mean: 488.63     Return_std: 41.79
Epoch 95        Return_mean: 493.34     Return_std: 34.31
```

Without empirical mean baseline

Actor-Critic with Linear Architecture

Actor-Critic architecture

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(Q^{\pi_\theta}(s_t, a_t) - v^{\pi_\theta}(s_t)\right)$$

Fit advantage $A^\pi(s,a) = Q^\pi(s,a) - v^\pi(s)$ with parameter w!

With value function estimation $v^\pi \approx v_w^\pi$, $\rightarrow A^\pi(s,a) = r + \gamma v_w^\pi(s') - v_w^\pi(s)$

Here, w is called *critic*, θ is called *actor*

Actor-Critic with Linear Architecture

Example : Linear critic

Define linear feature set as follow for CartPole task

$$\phi(s) = (1, e^{-\frac{\|s-\mu_1\|^2}{2\sigma^2}}, \cdots, e^{-\frac{\|s-\mu_M\|^2}{2\sigma^2}}) \text{ (radial basis functions)}$$

$v_w^\pi(s) = \phi(s)^\top w$, where w is parameter vector of length $M+1$

Perform linear fitting with sampled (s, a, r, s')

$$x = [\phi(s^0), \cdots, \phi(s^N)]$$

$$y = [r^0 + \gamma\phi({s'}^0)^\top w_{old}, \cdots, r^N + \gamma\phi({s'}^N)^\top w_{old}]$$

Fit w_{new} to minimize $\|w^\top x - y\|_2^2$

Actor-Critic with Linear Architecture

Example : Linear critic

```python
def calculate_vf(dataset, vf):
    X, y = [], []

    for data in dataset:
        for s, next_s, r in zip(data['state'], data['next_state'], data['reward']):
            v = state2feature(s)
            Q = r
            if vf is not None:
                Q = r + gamma * vf.predict(state2feature(next_s).reshape(1, -1))[0]
            X.append(v)
            y.append(Q)

    return X, y
```

$\phi(s)$

$\phi(s')^\top w_{old}$

$$x = [\phi(s^0), \cdots, \phi(s^N)]$$

$$y = [r^0 + \gamma\phi({s'}^0)^\top w_{old}, \cdots, r^N + \gamma\phi({s'}^N)^\top w_{old}]$$

Actor-Critic with Linear Architecture

Algorithm summary

Sample trajectory $(s_0, a_0, \cdots, s_T, a_T)$

Fit w with TD error

Perform policy gradient with $\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(r + \gamma v_w^\pi(s_{t+1}) - v_w^\pi(s_t)\right)$

$$\updownarrow$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \left(\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(Q^{\pi_\theta}(s_t, a_t) - v^{\pi_\theta}(s_t)\right)$$

Note, we don't need full trajectory now (incremental online algorithm)

Actor-Critic with Linear Architecture

Algorithm summary

```python
def get_advantage(data, vf):
    advantage, baseline = [], []

    for s, next_s, r in zip(data['state'], data['next_state'], data['reward']):
        v = vf.predict(state2feature(s).reshape(1, -1))[0]
        v_next = vf.predict(state2feature(next_s).reshape(1, -1))[0]
        # TODO: Complete advantage calculation by calculating Q-value
        Q = r + gamma * v_next
        A = Q - v

        advantage.append(A)
        baseline.append(v)

    return advantage, baseline
```

$\phi(s)^\top w$

$\phi(s)$

$$\left(r + \gamma v_w^\pi(s_{t+1}) - v_w^\pi(s_t)\right)$$

CORE
Control + Optimization Research Lab

Actor-Critic with Linear Architecture

Slower learning due to linear fittng calculation time

```
Epoch 0 Return_mean: 19.76      Return_std: 8.49       Time(
Epoch 5 Return_mean: 29.37      Return_std: 18.32      Time(
Epoch 10        Return_mean: 40.56      Return_std: 25.14
Epoch 15        Return_mean: 53.07      Return_std: 26.29
Epoch 20        Return_mean: 76.31      Return_std: 44.41
Epoch 25        Return_mean: 131.40     Return_std: 71.74
Epoch 30        Return_mean: 192.59     Return_std: 94.73
Epoch 35        Return_mean: 279.06     Return_std: 117.43
Epoch 40        Return_mean: 325.63     Return_std: 113.61
Epoch 45        Return_mean: 399.72     Return_std: 122.64
Epoch 50        Return_mean: 429.90     Return_std: 97.34
Epoch 55        Return_mean: 441.48     Return_std: 86.38
Epoch 60        Return_mean: 456.22     Return_std: 86.65
Epoch 65        Return_mean: 474.78     Return_std: 66.90
Epoch 70        Return_mean: 493.40     Return_std: 33.79
Epoch 75        Return_mean: 483.57     Return_std: 57.95
Epoch 80        Return_mean: 489.79     Return_std: 44.05
Epoch 85        Return_mean: 479.35     Return_std: 59.41
Epoch 90        Return_mean: 488.63     Return_std: 41.79
Epoch 95        Return_mean: 493.34     Return_std: 34.31
```

With average reward baseline

```
Epoch 0 Return_mean: 19.53      Return_std: 8.33       Time
Epoch 5 Return_mean: 29.72      Return_std: 18.20      Time
Epoch 10        Return_mean: 38.70      Return_std: 24.50
Epoch 15        Return_mean: 54.86      Return_std: 32.84
Epoch 20        Return_mean: 72.85      Return_std: 34.04
Epoch 25        Return_mean: 121.04     Return_std: 67.89
Epoch 30        Return_mean: 206.30     Return_std: 101.12
Epoch 35        Return_mean: 268.13     Return_std: 112.62
Epoch 40        Return_mean: 343.16     Return_std: 110.35
Epoch 45        Return_mean: 411.13     Return_std: 95.30
Epoch 50        Return_mean: 428.35     Return_std: 100.25
Epoch 55        Return_mean: 446.02     Return_std: 91.36
Epoch 60        Return_mean: 476.90     Return_std: 57.94
Epoch 65        Return_mean: 477.56     Return_std: 55.69
Epoch 70        Return_mean: 495.10     Return_std: 21.86
Epoch 75        Return_mean: 468.04     Return_std: 64.06
Epoch 80        Return_mean: 483.19     Return_std: 50.66
Epoch 85        Return_mean: 475.75     Return_std: 62.50
Epoch 90        Return_mean: 492.98     Return_std: 28.44
Epoch 95        Return_mean: 497.29     Return_std: 19.28
```

With linear fitted value function baseline