REINFORCE, Policy gradient with baseline, Actor-Critic method

Lunar Lander -v2 풀어보기

REINFORCE, Policy gradient with baseline method는 잘 수렴되지 않음.

- solve reward 는 최근 100개 에피소드 리워드 평균 200 이상.
- REINFORCE 와 Policy gradient with baseline 모두 리워드 평균의 상승 하강 곡선을 그리며 unstable 하고 느리게 수렴해 가는데 평균 최고 리워드 130~ 140 근처에서 더이상 상승하지 않았습니다.
- **'reinforce_lunar_lander_output.txt', 'pg_lunar_lander_output.txt', 'pg_lunar_lander_output2.txt'** 파일에 로그를 저장하였습니다.

- 피팅에 문제가 있는 것일까 (lunar lander 게임이 하나의 네트워크 레이어만으로 probability를 나타내기에 너무 복잡한 것일까) 생각해 레이어를 한 개 더 늘려봤습니다.
- REINFORCE 는 수렴속도가 눈에 띄게 증가하였지만 여전히 130~140 최고 리워드에서 잘 증가하지 않았습니다.
- PG with baseline 은 피팅되지 않고 -300(최악의 리워드)를 계속 받는 방향으로 네트워크가 잘못 수렴되었습니다.
- **'pg_lunar_lander_wrong_fitting.txt'** 에 로그를 저장했습니다.

Actor-Critic method

- colab을 이용한 cuda 사용.
- 책의 저자가 제공하는 ptan library는 pytorch 0.4.1 버전에서 지원하므로 pytorch 다운그레이드 필요.
- 네트워크 수정.

```
class A2C_Box2D(nn.Module):
    def __init__(self, input_size, n_actions):
        super(A2C_Box2D, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, 128),
            nn.ReLU(),
        )

        self.policy = nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, n_actions)
            )

        self.value = nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
            )

    def forward(self, x):
        fx = x.float()
        net_out = self.net(fx)
        return self.policy(net_out), self.value(net_out)
```

잘 수렴되지 않았습니다.

- unstable 하고 느리게 수렴하였으며 평균 최고 리워드 120 ~ 140에서 더 오르지 않았습니다.
- 로그를 **'lunar_lander_a2c_output.txt'** 파일에 저장했습니다.

ptan.experience.ExperienceSourceFirstLast

```
class ExperienceSourceFirstLast(ExperienceSource):

    def __init__(self, env, agent, gamma, steps_count=1,
                 steps_delta=1, vectorized=False):
        assert isinstance(gamma, float)
        super(ExperienceSourceFirstLast,
            self).__init__(env, agent,
            steps_count+1, steps_delta,
            vectorized=vectorized)
        self.gamma = gamma
        self.steps = steps_count
```

```
def __iter__(self):
    for exp in super(ExperienceSourceFirstLast,
                     self).__iter__():
        if exp[-1].done and len(exp) <= self.steps:
            last_state = None
            elems = exp
        else:
            last_state = exp[-1].state
            elems = exp[:-1]
        total_reward = 0.0
        for e in reversed(elems):
            total_reward *= self.gamma
            total_reward += e.reward
        yield ExperienceFirstLast(state=exp[0].state,
            action=exp[0].action, reward=total_reward,
            last_state=last_state)
```

- iteration 마다 가장 처음 state와 마지막 state를 리턴한다.
- 내부적으로 gamma를 이용하여 discounted total reward를 리턴한다.
- 에피소드가 끝난다면 last_state 를 None 으로 리턴한다.

```
exp_source = ptan.experience.ExperienceSourceFirstLast(en
                            agent, gamma=GAMMA)
```

다음과 같이 초기화 후,

```
for step_idx, exp in enumerate(exp_source):
    batch_states.append(exp.state)
    batch_actions.append(int(exp.action))
    cur_rewards.append(exp.reward)
```

generator 를 이용하여 사용한다.