NCTU RL HW2-Problem3
REINFORCE and A2C

廖家鴻 0786009
2021/5/10

Outline

- Problem3-(a): CartPole-v0 and REINFORCE
  - Network
  - Hyperparameters
  - Results
- Problem3-(b): LunarLander-v2 and A2C
  - Network
  - Hyperparameters
  - Results
  - Remarks

Problem3-(a):
CartPole-v0 and REINFORCE

Prob3-(a): Network

```python
########## YOUR CODE HERE (5~10 lines) ##########
### Actor_Net ###
self.a_fc0 = nn.Linear(self.state_dim, self.hidden_size)
self.a_fc1 = nn.Linear(self.hidden_size, self.hidden_size * 2)
self.a_fc2 = nn.Linear(self.hidden_size * 2, self.hidden_size * 2)
self.a_fc3 = nn.Linear(self.hidden_size * 2, self.action_dim)

### Baseline_Net ### (for estimating value function)
self.c_fc0 = nn.Linear(self.state_dim, self.hidden_size)
self.c_fc1 = nn.Linear(self.hidden_size, self.hidden_size)
self.c_fc2 = nn.Linear(self.hidden_size, self.hidden_size)
self.c_fc3 = nn.Linear(self.hidden_size, 1)
########## END OF YOUR CODE ##########
```

1. The hidden_size here is 64

```python
def forward(self, state):
    ########## YOUR CODE HERE (3~5 lines) #######
    ### Actor_Net ###
    x = F.relu(self.a_fc0(state))
    x = F.relu(self.a_fc1(x))
    x = F.relu(self.a_fc2(x))
    x = F.softmax(self.a_fc3(x))
    action_prob = x

    ### Baseline_Net ###
    y = F.relu(self.c_fc0(state))
    y = F.relu(self.c_fc1(y))
    y = F.relu(self.c_fc2(y))
    y = self.c_fc3(y)
    baseline_value = y
    ########## END OF YOUR CODE ##########
    return action_prob, baseline_value
```

2. I implement the Actor_Net and Baseline_Net separately.

PS: I have tried another setting that is "one state input with two branches output." And this will also work. But here I only present the separated version.

Prob3-(a): Hyperparameters

```python
if __name__ == '__main__':
    # For reproducibility, fix the random seed
    random_seed = 20
    lr = 0.02
    env = gym.make('CartPole-v0')
    env.seed(random_seed)
    torch.manual_seed(random_seed)
    train(lr)
    test('CartPole_0.02.pth')
```

1. The learning rate is lr=0.02

2. The discount factor is gamma=0.99

```python
    def calculate_loss(self, optimizer, gamma=0.99):
            g_return = self.rewards[t] + gamma*g_return
```

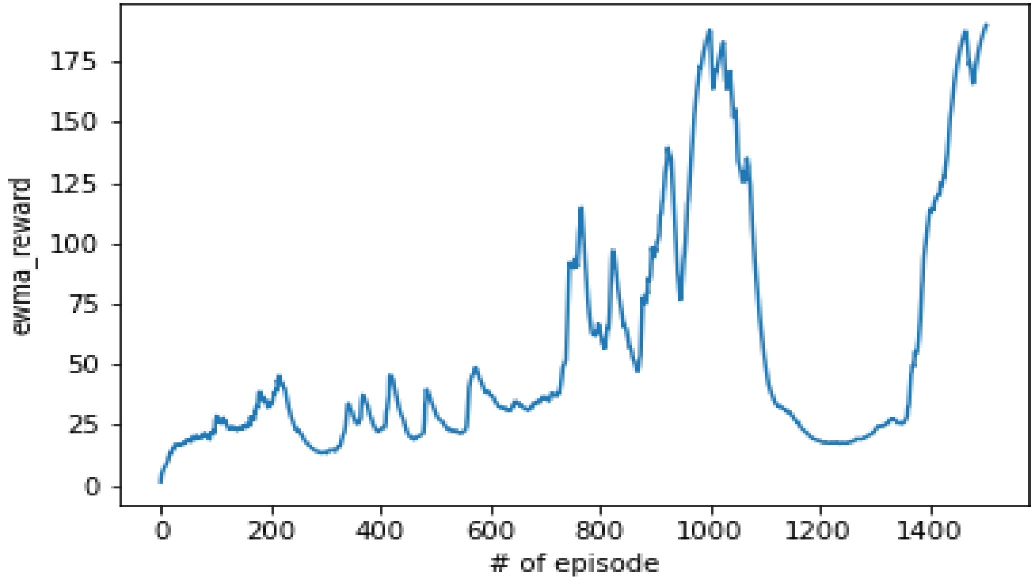
```python
    # Instantiate the policy model and the optimizer
    model = Policy()
    optimizer = optim.SGD(model.parameters(), lr=lr)
    scheduler = Scheduler.StepLR(optimizer, step_size=500, gamma=0.5)
```

3. The learning rate scheduler is set with step_size=500 and gamma=0.5

Prob3-(a): Results

```
# check if we have "solved" the cart pole problem
if ewma_reward >= 190.:
    torch.save(model.state_dict(), './CartPole_0.02.pth')
```

1. I save the best agent when the ewma_reward achieve 190.



2. The left figure shows the curve of ewma_reward for all training episodes.

```
Ep 1503 Length: 200 reward: 200.0000     ewma reward: 188.9586
Ep 1504 Length: 200 reward: 200.0000     ewma reward: 189.5107
Ep 1505 Length: 200 reward: 200.0000     ewma reward: 190.0352
Solved! Running reward is now 190.03516076169097 and the last episode runs to 200 time steps!
Episode 1   Reward: 200.0
Episode 2   Reward: 200.0
Episode 3   Reward: 200.0
Episode 4   Reward: 200.0
Episode 5   Reward: 200.0
Episode 6   Reward: 200.0
Episode 7   Reward: 200.0
Episode 8   Reward: 200.0
Episode 9   Reward: 200.0
Episode 10  Reward: 200.0
```

3. At this setting, the total training episodes are 1505. And the last episode runs to 200 steps.

4. I follow the sample code to test the trained agent for 10 episodes and all get the maximum reward.

Problem3-(b): LunarLander-v2 and A2C

Prob3-(b): Network

```python
        self.hidden_size = 64
        ########## YOUR CODE HERE (5~10 lines) ##########
        self.fc0 = nn.Linear(self.state_dim, self.hidden_size)
        ### Actor_Net ###
        self.a_fc0 = nn.Linear(self.state_dim, self.hidden_size)
        self.a_fc1 = nn.Linear(self.hidden_size, self.hidden_size * 2)
        self.a_fc2 = nn.Linear(self.hidden_size * 2, self.hidden_size * 2)
        self.a_fc3 = nn.Linear(self.hidden_size * 2, self.action_dim)

        ### Baseline_Net ### (for estimating value function)
        self.c_fc0 = nn.Linear(self.state_dim, self.hidden_size)
        self.c_fc1 = nn.Linear(self.hidden_size, self.hidden_size)
        self.c_fc2 = nn.Linear(self.hidden_size, self.hidden_size)
        self.c_fc3 = nn.Linear(self.hidden_size, 1)
        ########## END OF YOUR CODE ##########
```

1. The hidden_size here is 64.

```python
    def forward(self, state):
        ########## YOUR CODE HERE (3~5 lines) ##########
        # s = F.relu(self.fc0(state))
        ### Actor_branch ###
        x = F.relu(self.a_fc0(state))
        x = F.relu(self.a_fc1(x))
        x = F.relu(self.a_fc2(x))
        x = F.sigmoid(self.a_fc3(x))
        action_prob = x

        ### Baseline_branch ### (for estimating value function)
        y = F.relu(self.c_fc0(state))
        y = F.relu(self.c_fc1(y))
        y = F.relu(self.c_fc2(y))
        y = self.c_fc3(y)
        baseline_value = y
        ########## END OF YOUR CODE ##########
        return action_prob, baseline_value
```

2. Again, I implement the Actor_Net and Baseline_Net separately.

PS: I also tried another setting that is "one state input with two branches output", but it is hard work.

Prob3-(a): Hyperparameters

```python
if __name__ == '__main__':
    # For reproducibility, fix the random seed
    random_seed = 20
    lr = 0.01
    env = gym.make('LunarLander-v2')
    env.seed(random_seed)
    torch.manual_seed(random_seed)
    train(lr)
    test('LunarLander_0.01.pth')
```

1. The learning rate is lr=0.01

```python
    def calculate_loss(self, gamma=0.95):
```

```python
        baseline_loss = nn.MSELoss()
        value_loss = baseline_loss(value, reward+(gamma*next_value) )

        advantage = (reward + gamma*next_value)  - value
        policy_loss = -log_act_prob * advantage.detach()
```

2. The discount factor is gamma=0.95

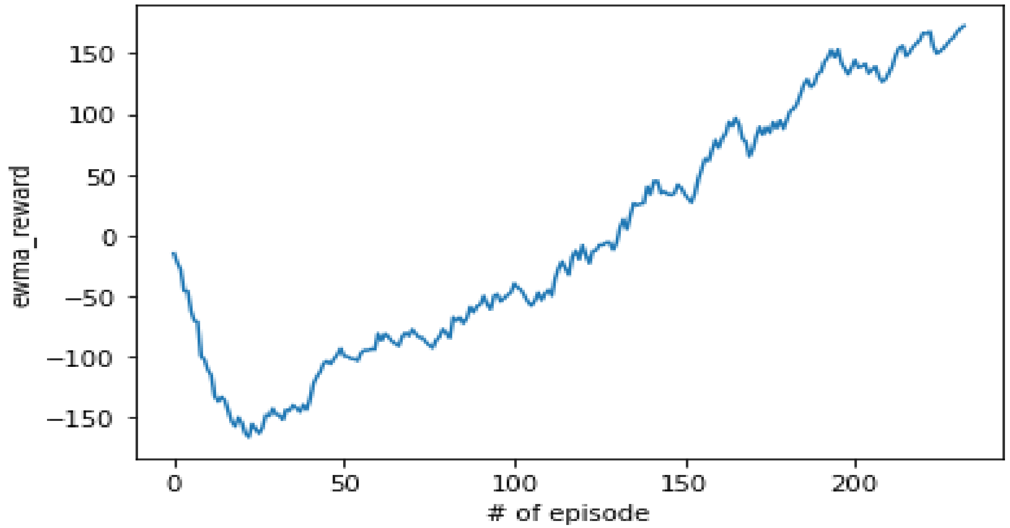
```python
    # Instantiate the policy model and the optimizer
    model = Policy()
    optimizer = optim.SGD(model.parameters(), lr=lr)
    # optimizer = optim.Adam(model.parameters(), lr=lr)
    scheduler = Scheduler.StepLR(optimizer, step_size=100, gamma=0.8)
```

3. The learning rate scheduler is set with step_size=100 and gamma=0.8

Prob3-(b): Results

```
        # check if we have "solved" the cart pole problem
        if ewma_reward >= 172.:
            torch.save(model.state_dict(), './LunarLander_0.01.pth')
            print("Solved! Running reward is now {} and "
                  "the last episode runs to {} time steps!".format(ewma_reward, t))
            break
```

1. I save the best agent when the ewma_reward achieve 172.



2. The left figure shows the curve of ewma_reward for all training episodes.

```
Ep 230  Length: 516 R: 200.5952  ewma reward: 162.8868
Ep 231  Length: 267 R: 247.0709  ewma reward: 167.0960
Ep 232  Length: 395 R: 233.3892  ewma reward: 170.4107
Ep 233  Length: 870 R: 212.3251  ewma reward: 172.5064
Solved! Running reward is now 172.506412644235 and the last episode runs to 870 time steps!
Episode 1   Reward: 247.8139565405427
Episode 2   Reward: 277.3608585658985
Episode 3   Reward: -1.5739138103989347
Episode 4   Reward: 202.5240771279636
Episode 5   Reward: 244.86838236388374
Episode 6   Reward: 235.36878862993836
Episode 7   Reward: 141.4510078744156
Episode 8   Reward: 239.24533874901292
Episode 9   Reward: 74.15252161222428
Episode 10  Reward: -191.0576380074957
```

3. At this setting, the total training episodes are 233.

4. I follow the sample code to test the trained agent for 10 episodes, only episode 3 and 10 are failedg.

Prob3-(b): Remarks

```python
########## YOUR CODE HERE (10-15 lines) ##########
for t in itertools.count(start=1):
    action = model.select_action(state)
    next_state, reward, done, _ = env.step(action)

    ep_reward += reward
    state = next_state
    model.rewards.append(reward)

    loss, policy_loss, value_loss = model.calculate_loss(reward, next_state)
    # optimizer.zero_grad()
    # loss.backward()
    # nn.utils.clip_grad_norm_(model.parameters(), 3)
    # optimizer.step()

    ### Update Actor_Net ###
    optimizer.zero_grad()
    policy_loss.backward(retain_graph=True)
    nn.utils.clip_grad_norm_(model.parameters(), 3)
    optimizer.step()

    ### Update Behavior_Net ###
    optimizer.zero_grad()
    value_loss.backward()
    nn.utils.clip_grad_norm_(model.parameters(), 3)
    optimizer.step()

    if done:
        break

model.clear_memory()
########## END OF YOUR CODE ##########
```

The key point that I finally success is that the network update frequency.

Originally, I update the network per episode as 3-(a), but it always fail no matter I tune the hyperparameters.

After I change the way to update the network by every step, the agent finally works!!!

The End