RL hw2 problem 4 : report

part (a)

hyperparameter

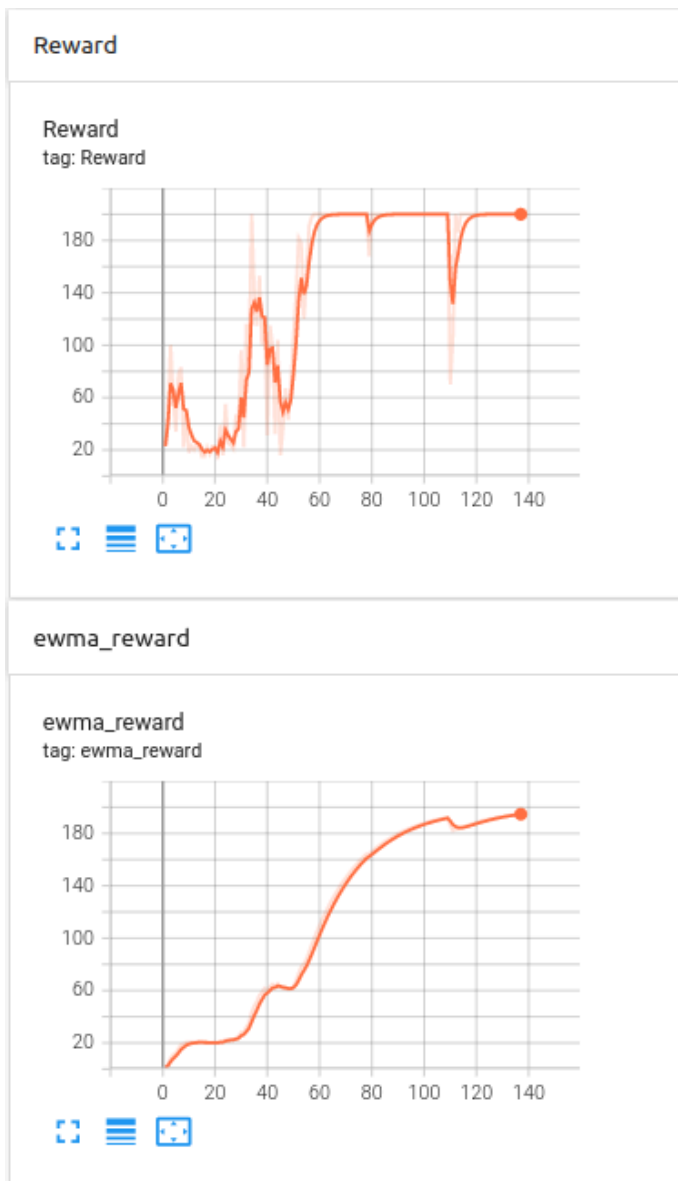
architecture	learning rate

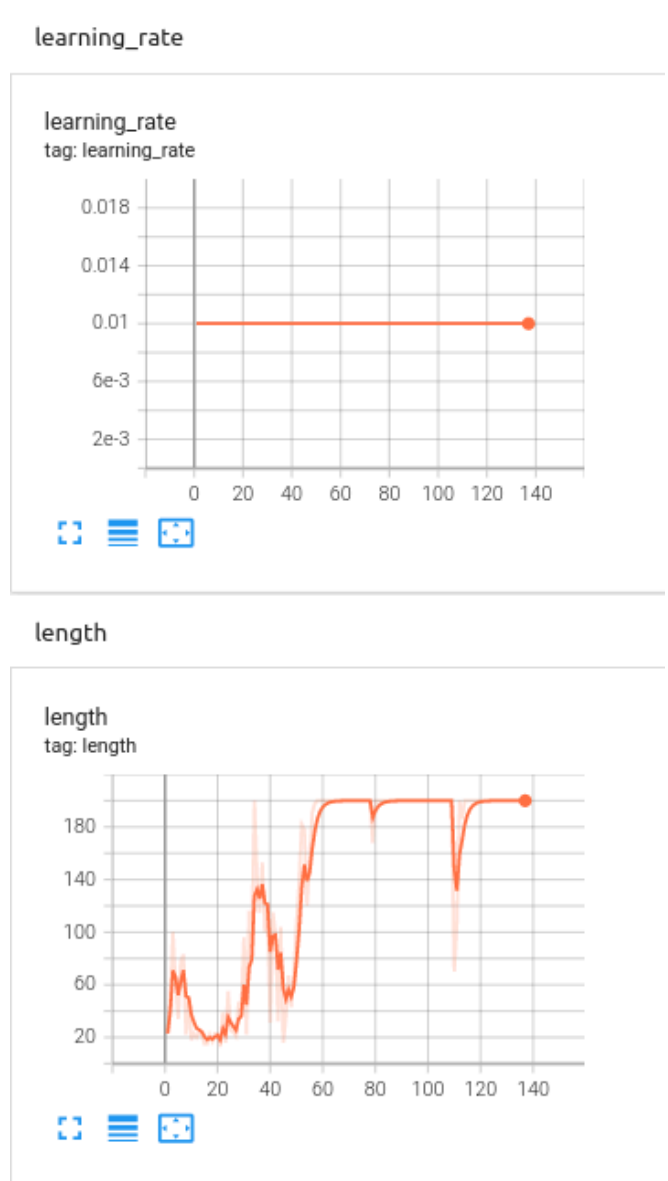
```python
########## YOUR CODE HERE (5~10 lines) ##########
# Shared layers
self.shared_layers = nn.Sequential(
    nn.Linear(self.observation_dim, self.hidden_size),
    nn.ReLU()
)

# Actor layers
if self.discrete:
    self.actor = nn.Sequential(
        nn.Linear(self.hidden_size, self.action_dim),
        nn.Softmax(dim=-1)
    )
else:
    self.actor = nn.Sequential(
        nn.Linear(self.hidden_size, self.action_dim)
    )

# Critic layers
self.critic = nn.Sequential(
    nn.Linear(self.hidden_size, self.hidden_size),
    nn.ReLU(),
    nn.Linear(self.hidden_size, 1)
)
```

```python
if __name__ == '__main__':
    # For reproducibility, fix the random seed
    random_seed = 10
    lr = 0.016
    env = gym.make('LunarLander-v2')
    env.seed(random_seed)
    torch.manual_seed(random_seed)
    train(lr)
    test(f'LunarLander_{lr}.pth')
```

result

reward & emma reward	learning rate & length



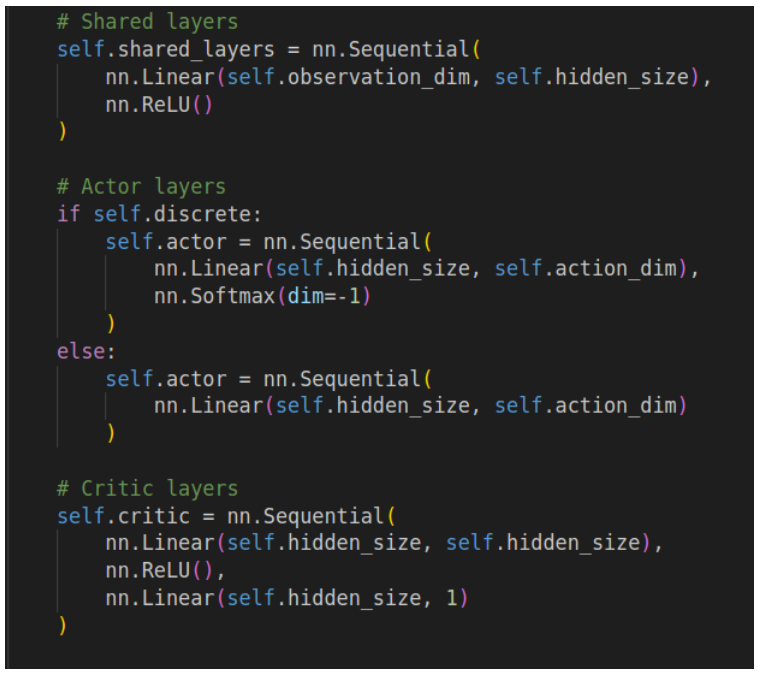
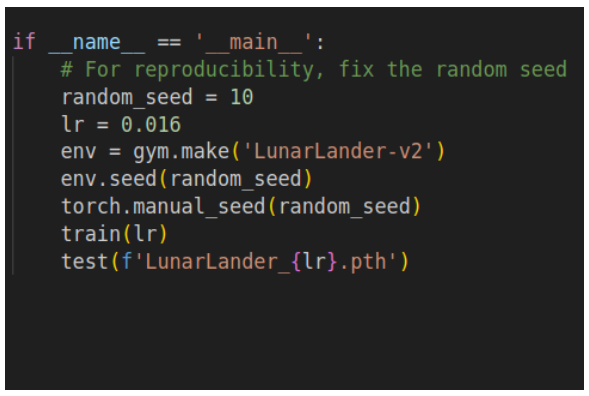


part (b)

explain : choose of baseline

I choose value function as our baseline, so I could implement policy gradient (P4) formula

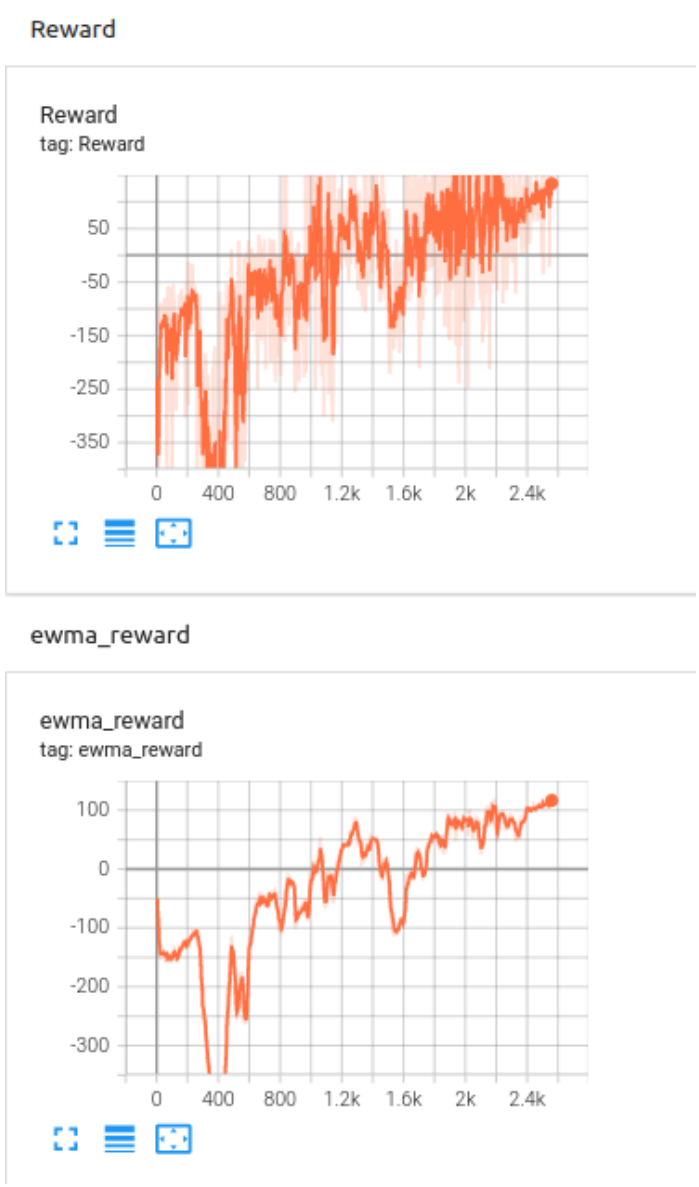
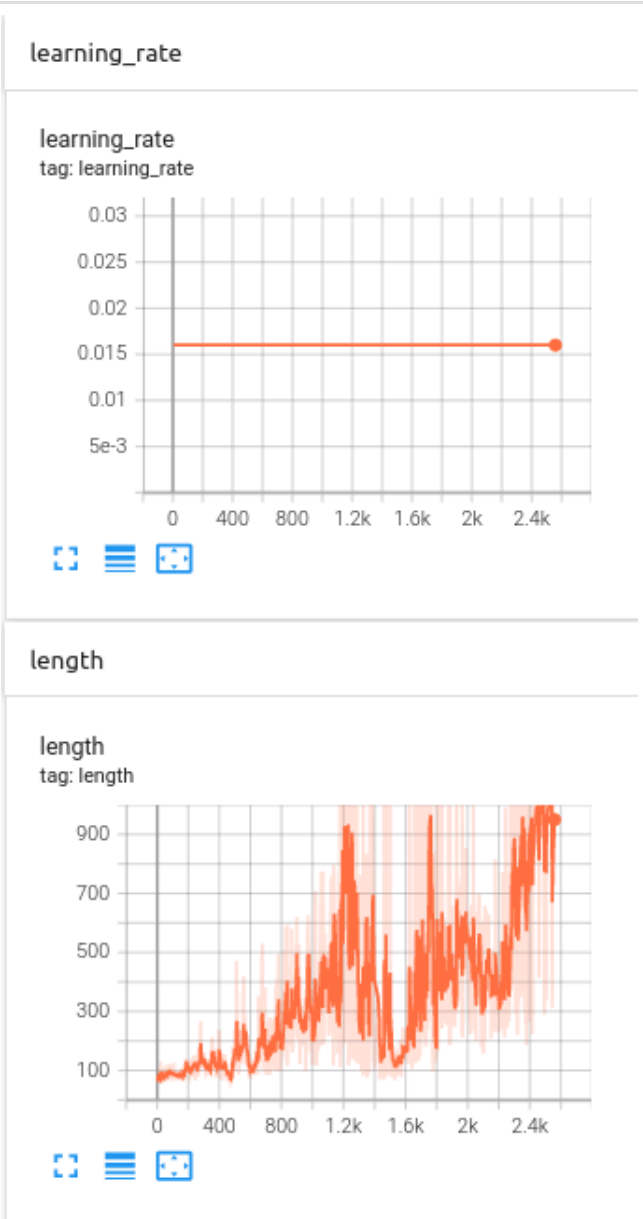
hyperparameter

architecture	learning rate
	! 

```
# Shared layers
self.shared_layers = nn.Sequential(
    nn.Linear(self.observation_dim, self.hidden_size),
    nn.ReLU()
)

# Actor layers
if self.discrete:
    self.actor = nn.Sequential(
        nn.Linear(self.hidden_size, self.action_dim),
        nn.Softmax(dim=-1)
    )
else:
    self.actor = nn.Sequential(
        nn.Linear(self.hidden_size, self.action_dim)
    )

# Critic layers
self.critic = nn.Sequential(
    nn.Linear(self.hidden_size, self.hidden_size),
    nn.ReLU(),
    nn.Linear(self.hidden_size, 1)
)
```

```
if __name__ == '__main__':
    # For reproducibility, fix the random seed
    random_seed = 10
    lr = 0.016
    env = gym.make('LunarLander-v2')
    env.seed(random_seed)
    torch.manual_seed(random_seed)
    train(lr)
    test(f'LunarLander_{lr}.pth')
```

result

reward & emma reward	learning rate & length
	

part (c) GAE

code

```python
def __call__(self, rewards, values, done):
    """
        Implement Generalized Advantage Estimation (GAE) for your value prediction
        TODO (1): Pass correct corresponding inputs (rewards, values, and done) into the function
        TODO (2): Calculate the Generalized Advantage Estimation and return the obtained value
    """
    ########## YOUR CODE HERE (8-15 lines) ##########
    advantages = []
    advantage = 0
    next_value = 0

    # gen_advantage(t) = sum((r * lambda)^(l) * td_error(t+l)) from l = 0 to infinity
    for r, v in zip(reversed(rewards), reversed(values)):
        td_error = r + self.gamma * next_value - v
        advantage = td_error + (self.gamma * self.lambda_) * advantage
        next_value = v
        advantages.insert(0, advantage)

    advantages = torch.tensor(advantages)
    return advantages
    ########## END OF YOUR CODE ##########
```

explain

I implement GAE with the formula $\sum_{l=0}^{\infty}(r \cdot \lambda)^l \cdot \delta(t+l)$ where $\delta(t+l)$ is td errorat time at t + l , by couting it reversely, we can get the Generized advantage in O(n)

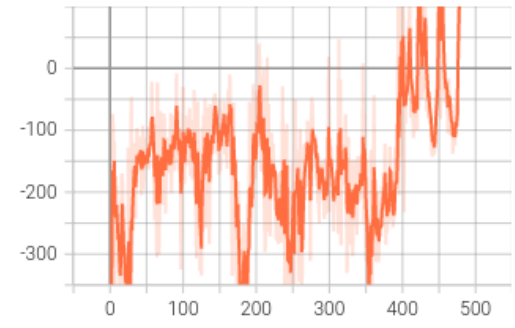
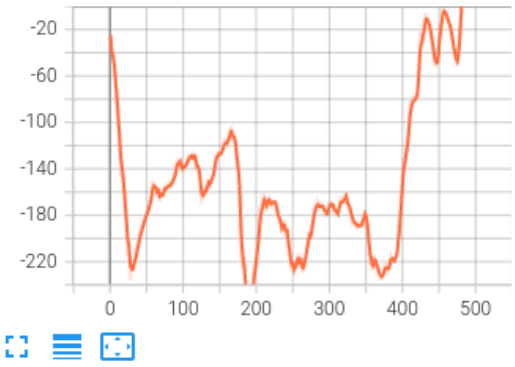
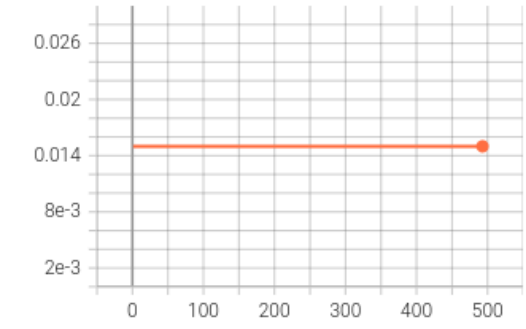
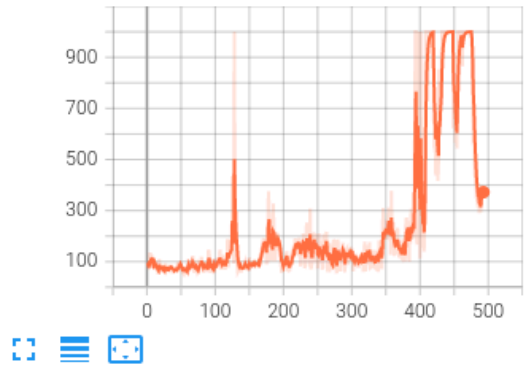
hyperparameter

architecture	learning rate & GAE parameter λ	

`# Shared layers`
`self.shared_layers = nn.Sequential(`
`  nn.Linear(self.observation_dim, self.hidden_size),`
`  nn.ReLU()`
`)`
 	
`# Actor layers`
`if self.discrete:`
`  self.actor = nn.Sequential(`
`  nn.Linear(self.hidden_size, self.action_dim),`
`  nn.Softmax(dim=-1)`
`  )`
`else:`
`  self.actor = nn.Sequential(`
`  nn.Linear(self.hidden_size, self.action_dim)`
`  )`
 	
`# Critic layers`
`self.critic = nn.Sequential(`
`  nn.Linear(self.hidden_size, self.hidden_size),`
`  nn.ReLU(),`
`  nn.Linear(self.hidden_size, 1)`
`)`
	`if __name__ == '__main__':`
	`  # For reproducibility, fix the random seed`
	`  random_seed = 10`
	`  lr = 0.015`
	`  lambda_ = 0.95`
	`  env = gym.make('LunarLander-v2')`
	`  env.seed(random_seed)`
	`  torch.manual_seed(random_seed)`
	`  train(lr, lambda_)`
	`  test(f'LunarLander_gae_{lr}_lambda_{lambda_}.pth')`	

1. GAE with λ = 0.95

result

reward & emma reward	learning rate & length	
Reward Reward tag: Reward  ewma_reward ewma_reward tag: ewma_reward 	learning_rate learning_rate tag: learning_rate  length length tag: length 	

2. GAE with λ = 0.80

reward & emma reward	learning rate & length
	

3. GAE with λ = 0.98

reward & emma reward	learning rate & length