HW2

Problem 1 (Baseline for Variance Reduction)

a. By Monte Carlo policy gradient, we can sample a trajectory τ to estimate V^{π_θ}. In this problem, there are only three possible trajectory, which are:

   1. $s \to a \to r(s,a) \to s_1(\text{terminal})$, simply denoted as a
   2. $s \to b \to r(s,b) \to s_1(\text{terminal})$, simply denoted as b
   3. $s \to c \to r(s,c) \to s_1(\text{terminal})$, simply denoted as c

   Let's calculate score function for each trajectory first:

$$\nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta\ \theta_a - \log(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \\ = \begin{bmatrix} 1 - e^{\theta_a}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \\ -e^{\theta_b}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \\ -e^{\theta_c}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \end{bmatrix}$$

   Similarly for action b and c, then we can get:

$$\nabla_\theta \log \pi_\theta(b|s) = \begin{bmatrix} -e^{\theta_a}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \\ 1 - e^{\theta_b}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \\ -e^{\theta_c}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \end{bmatrix}$$

$$\nabla_\theta \log \pi_\theta(c|s) = \begin{bmatrix} -e^{\theta_a}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \\ -e^{\theta_b}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \\ 1 - e^{\theta_c}/(e^{\theta_a} + e^{\theta_b} + e^{\theta_c}) \end{bmatrix}$$

   We already know that $\theta_a = 0,\ \theta_b = \ln 5,\ \theta_c = \ln 4$, so:

$$\hat{\nabla}_a = 100 \cdot \begin{bmatrix} 1 - 0.1 \\ -0.5 \\ -0.4 \end{bmatrix} = \begin{bmatrix} 90 \\ -50 \\ -40 \end{bmatrix},\ \hat{\nabla}_b = 98 \cdot \begin{bmatrix} -0.1 \\ 1 - 0.5 \\ -0.4 \end{bmatrix} = \begin{bmatrix} -9.8 \\ 49 \\ -39.2 \end{bmatrix},\ \hat{\nabla}_c = 95 \cdot \begin{bmatrix} -0.1 \\ -0.5 \\ 1 - 0.4 \end{bmatrix} = \begin{bmatrix} -9.5 \\ -47.5 \\ 57 \end{bmatrix}$$

   Then mean vector of $\hat{\nabla} V$ is:

$$\mathbb{E}[\hat{\nabla} V] = 0.1 \cdot \begin{bmatrix} 90 \\ -50 \\ -40 \end{bmatrix} + 0.5 \cdot \begin{bmatrix} -9.8 \\ 49 \\ -39.2 \end{bmatrix} + 0.4 \cdot \begin{bmatrix} -9.5 \\ -47.5 \\ 57 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.5 \\ -0.8 \end{bmatrix}$$

   To compute convariance matrix of $\hat{\nabla} V$, we need to compute $\sum_\tau P(\tau)(\hat{\nabla}_\tau - \mathbb{E}[\hat{\nabla} V])(\hat{\nabla}_\tau - \mathbb{E}[\hat{\nabla} V])^T$, here I run the following python code to help me calculate the value.

```
import numpy as np
a = np.array([90, -50, -40])
b = np.array([-9.8, 49, -39.2])
c = np.array([-9.5, -47.5, 57])
E = np.array([0.3, 0.5, -0.8])
Cov = 0.1 * np.outer(a - E, a - E) + 0.5 * np.outer(b - E, b - E) + 0.4 * np.outer(c - E, c - E)
print(Cov)
```

   Output:

```
[[  894.03  -509.75  -384.28]
 [ -509.75  2352.75 -1843.  ]
 [ -384.28 -1843.    2227.28]]
```

   So the covariance matrix of $\hat{\nabla} V$ is:

$$\begin{bmatrix} 894.03 & -509.75 & -384.28 \\ -509.75 & 2352.75 & -1843 \\ -384.28 & -1843 & 2227.84 \end{bmatrix}$$

b. When we introduce baseline $B(s)$ to policy gradient, it won't change the expection. So the expectation of $\tilde{\nabla} V$ is also:

$$\mathbb{E}[\tilde{\nabla} V] = \begin{bmatrix} 0.3 \\ 0.5 \\ -0.8 \end{bmatrix}$$

   Here we choose $B(s) = V^{\pi_\theta}(s) = 0.1 \cdot 100 + 0.5 \cdot 98 + 0.4 \cdot 95 = 97$, then for each trajectory:

$$\tilde{\nabla}_a = (100-97)\cdot\begin{bmatrix}0.9\\-0.5\\-0.4\end{bmatrix} = \begin{bmatrix}2.7\\-1.5\\-1.2\end{bmatrix},\ \tilde{\nabla}_b = (98-97)\cdot\begin{bmatrix}-0.1\\0.5\\-0.4\end{bmatrix} = \begin{bmatrix}-0.1\\0.5\\-0.4\end{bmatrix},\ \tilde{\nabla}_c = (95-97)\cdot\begin{bmatrix}-0.1\\-0.5\\0.6\end{bmatrix} = \begin{bmatrix}0.2\\1\\-1.2\end{bmatrix}$$

Similar to Part A, I run the following python code to help me calculate the value.

```
import numpy as np
a_ = np.array([2.7, -1.5, -1.2])
b_ = np.array([-0.1, 0.5, -0.4])
c_ = np.array([0.2, 1, -1.2])
E = np.array([0.3, 0.5, -0.8])
Cov_ = 0.1 * np.outer(a_ - E, a_ - E) + 0.5 * np.outer(b_ - E, b_ - E) + 0.4 * np.outer(c_ - E, c_ - E)
print(Cov_)
```

Output:

```
[[ 0.66 -0.5  -0.16]
 [-0.5   0.5   0.  ]
 [-0.16  0.    0.16]]
```

So the covariance matrix of $\tilde{\nabla} V$ is:

$$\begin{bmatrix}0.66 & -0.5 & -0.16\\ -0.5 & 0.5 & 0\\ -0.16 & 0 & 0.16\end{bmatrix}$$

c. Use python package `scipy.optimize` to find the optimal base:

```
import numpy as np
from scipy.optimize import minimize

def trace_of_cov(base):
    a = (100 - base) * np.array([0.9, -0.5, -0.4])
    b = (98 - base) * np.array([-0.1, 0.5, -0.4])
    c = (95 - base) * np.array([-0.1, -0.5, 0.6])
    return np.trace(0.1 * np.outer(a - E, a - E) + 0.5 * np.outer(b - E, b - E) + 0.4 * np.outer(c - E, c
- E))

base = minimize(trace_of_cov, x0=97)
print(f"Optimal base B(s) = {round(base.x[0],3)}")
```

Output:

```
Optimal base B(s) = 97.138
```

Problem 2 (Non-Uniform Polyak-Lojacsiewicz Condition in RL)

a. **Proof**:

$$\begin{aligned}
\left\|\frac{\partial V^{\pi_\theta}(\mu)}{\partial\theta}\right\|_2 &= \left[\sum_{s,a}\left(\frac{\partial V^{\pi_\theta}(\mu)}{\partial\theta(s,a)}\right)^2\right]^{\frac{1}{2}} \\
&\geq \left[\sum_{s}\left(\frac{\partial V^{\pi_\theta}(\mu)}{\partial\theta(s,a^*(s))}\right)^2\right]^{\frac{1}{2}} \\
&\geq \frac{1}{\sqrt{S}}\sum_s\left|\frac{\partial V^{\pi_\theta}(\mu)}{\partial\theta(s,a^*(s))}\right| \quad (\text{by Cauchy-Schwarz, } \|x\|_1 \leq \sqrt{S}\|x\|_2) \\
&= \frac{1}{1-\gamma}\cdot\frac{1}{\sqrt{S}}\sum_s\left|d_\mu^{\pi_\theta}(s)\cdot\pi_\theta(a^*(s)|s)\cdot A^{\pi_\theta}(s,a^*(s))\right| \quad (\text{PG under softmax policy parametrization}) \\
&= \frac{1}{1-\gamma}\cdot\frac{1}{\sqrt{S}}\sum_s d_\mu^{\pi_\theta}(s)\cdot\pi_\theta(a^*(s)|s)\cdot|A^{\pi_\theta}(s,a^*(s))| \quad (\text{because } d_\mu^{\pi_\theta}(s)\geq 0 \text{ and } \pi_\theta(a^*(s)|s)\geq 0)
\end{aligned}$$

b. **Proof**:

Define the distribution mismatch coefficient as $\alpha = \max_s \frac{d^{\pi^*}\rho(s)}{d_\mu^{\pi_\theta}(s)}$. We have,

$$\begin{aligned}
\|\nabla V^{\pi_\theta}(\mu)\|_2 &\geq \frac{1}{1-\gamma}\cdot\frac{1}{\sqrt{S}}\sum_s \frac{d_\mu^{\pi_\theta}(s)}{d_\rho^{\pi^*}(s)}\cdot d_\rho^{\pi^*}(s)\cdot\pi_\theta(a^*(s)|s)\cdot|A^{\pi_\theta}(s,a^*(s))| \\
&\geq \frac{1}{1-\gamma}\cdot\frac{1}{\sqrt{S}}\cdot\left\|\frac{d_\rho^{\pi^*}}{d_\mu^{\pi_\theta}}\right\|_\infty^{-1}\cdot\min_s \pi_\theta(a^*(s)|s)\cdot\sum_s d_\rho^{\pi^*}(s)\cdot|A^{\pi_\theta}(s,a^*(s))| \\
&\geq \frac{1}{1-\gamma}\cdot\frac{1}{\sqrt{S}}\cdot\left\|\frac{d_\rho^{\pi^*}}{d_\mu^{\pi_\theta}}\right\|_\infty^{-1}\cdot\min_s \pi_\theta(a^*(s)|s)\cdot\sum_s d_\rho^{\pi^*}(s)\cdot A^{\pi_\theta}(s,a^*(s)) \\
&= \frac{1}{\sqrt{S}}\cdot\left\|\frac{d_\rho^{\pi^*}}{d_\mu^{\pi_\theta}}\right\|_\infty^{-1}\cdot\min_s \pi_\theta(a^*(s)|s)\cdot\frac{1}{1-\gamma}\sum_s d_\rho^{\pi^*}(s)\cdot\sum_a \pi^*(a|s)\cdot A^{\pi_\theta}(s,a) \\
&= \frac{1}{\sqrt{S}}\cdot\left\|\frac{d_\rho^{\pi^*}}{d_\mu^{\pi_\theta}}\right\|_\infty^{-1}\cdot\min_s \pi_\theta(a^*(s)|s)\cdot[V^*(\rho)-V^{\pi_\theta}(\rho)]
\end{aligned}$$

Problem 3 (Monte Carlo Policy Evaluation)

Property 1: Show that the true value function at state S (denoted by $V(S)$) satisfies that:

$$V(S)=\frac{P_S}{P_T}R_S+R_T$$

Proof.

$$V(S)=(P_S+P_SP_S+P_SP_SP_S+\cdots)R_S+R_T=\frac{P_S}{1-P_S}R_S+R_T=\frac{P_S}{P_T}R_S+R_T$$

Property 2: Suppose we construct an every-visit MC estimate based on only 1 trajectory τ (denoted by$\hat{V}_{MC}(S;\tau)$). Then, please show that

$$\mathbb{E}[\hat{V}_{MC}(S;\tau)]=\frac{P_S}{2P_T}R_S+R_T$$

Proof.

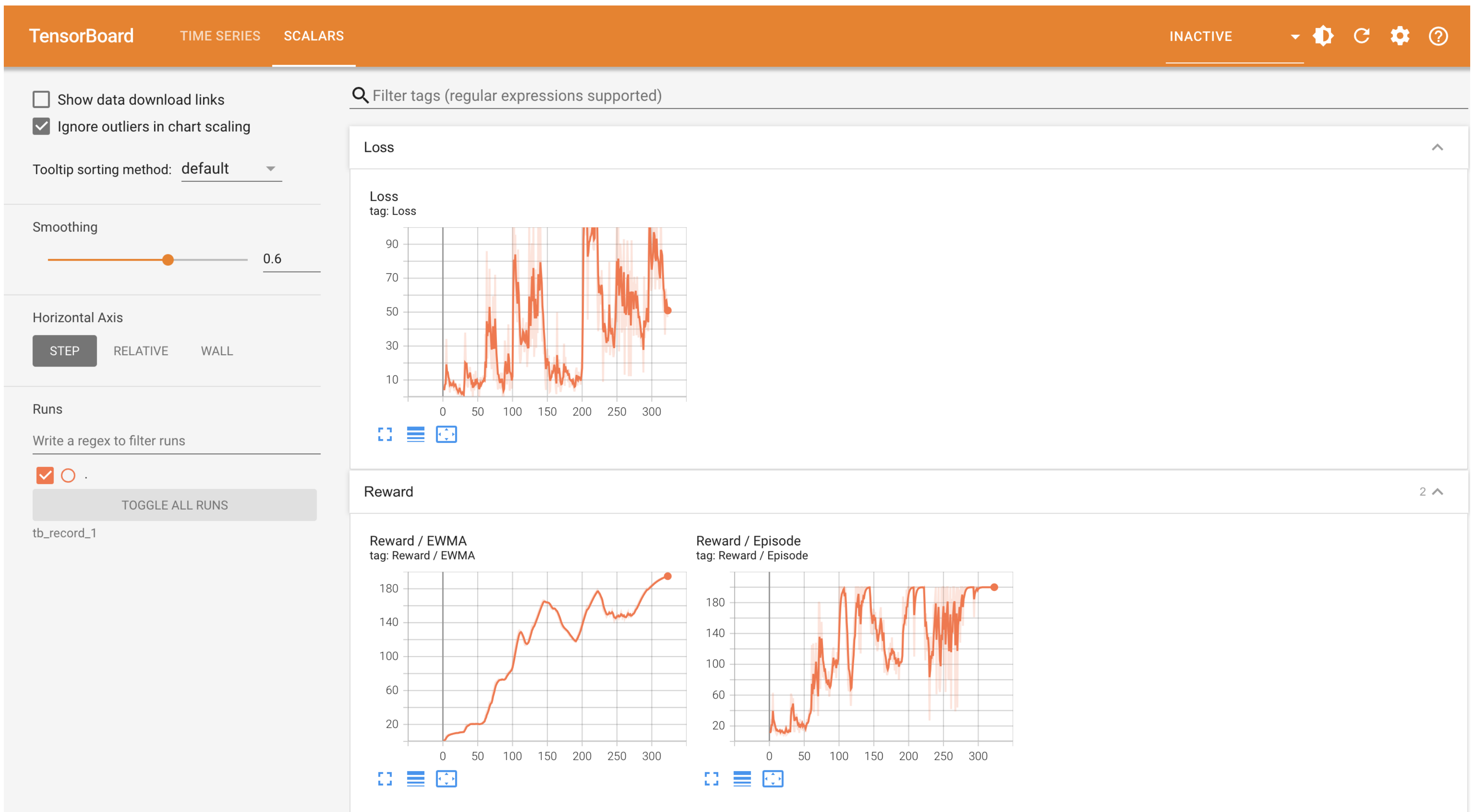
Consider all possible trajectories and the corresponding probabilities, we can get:

$$\begin{aligned}
\mathbb{E}[\hat{V}_{MC}(S;\tau)] &= \sum_{k=0}^{\infty}P_TP_S^k\left(\frac{R_S+2R_S+\cdots+kR_S+(k+1)R_T}{k+1}\right) \\
&= \sum_{k=0}^{\infty}P_TP_S^k\left(\frac{k\cdot(k+1)\cdot R_S}{2\cdot(k+1)}+R_T\right) \\
&= \sum_{k=0}^{\infty}P_TP_S^k\left(\frac{k\cdot R_S}{2}+R_T\right) \\
&= \frac{P_TP_SR_S\sum_{k=0}^{\infty}kP_S^{k-1}}{2}+\sum_{k=0}^{\infty}P_TP_S^kR_T \\
&= \frac{P_TP_SR_S}{2(1-P_S)^2}+P_T\cdot\frac{1}{1-P_S}R_T \\
&= \frac{P_TP_SR_S}{2(P_T)^2}+P_T\cdot\frac{1}{P_T}R_T \\
&= \frac{P_S}{2P_T}R_S+R_T
\end{aligned}$$

Problem 4 (Policy Gradient Algorithms With Function Approximation)

Vanilla REINFORCE

- NN architecture
  - Layer1: Shared layer, `nn.Linear( self .observation_dim, 128)` then `ReLU()`
  - For Actor: `nn.Sequential(nn.Linear(128, self.action_dim), nn.Softmax(dim=-1))`
  - For Critic: `nn.Linear(128, 1)`
- Learning rate: `lr = 0.01`

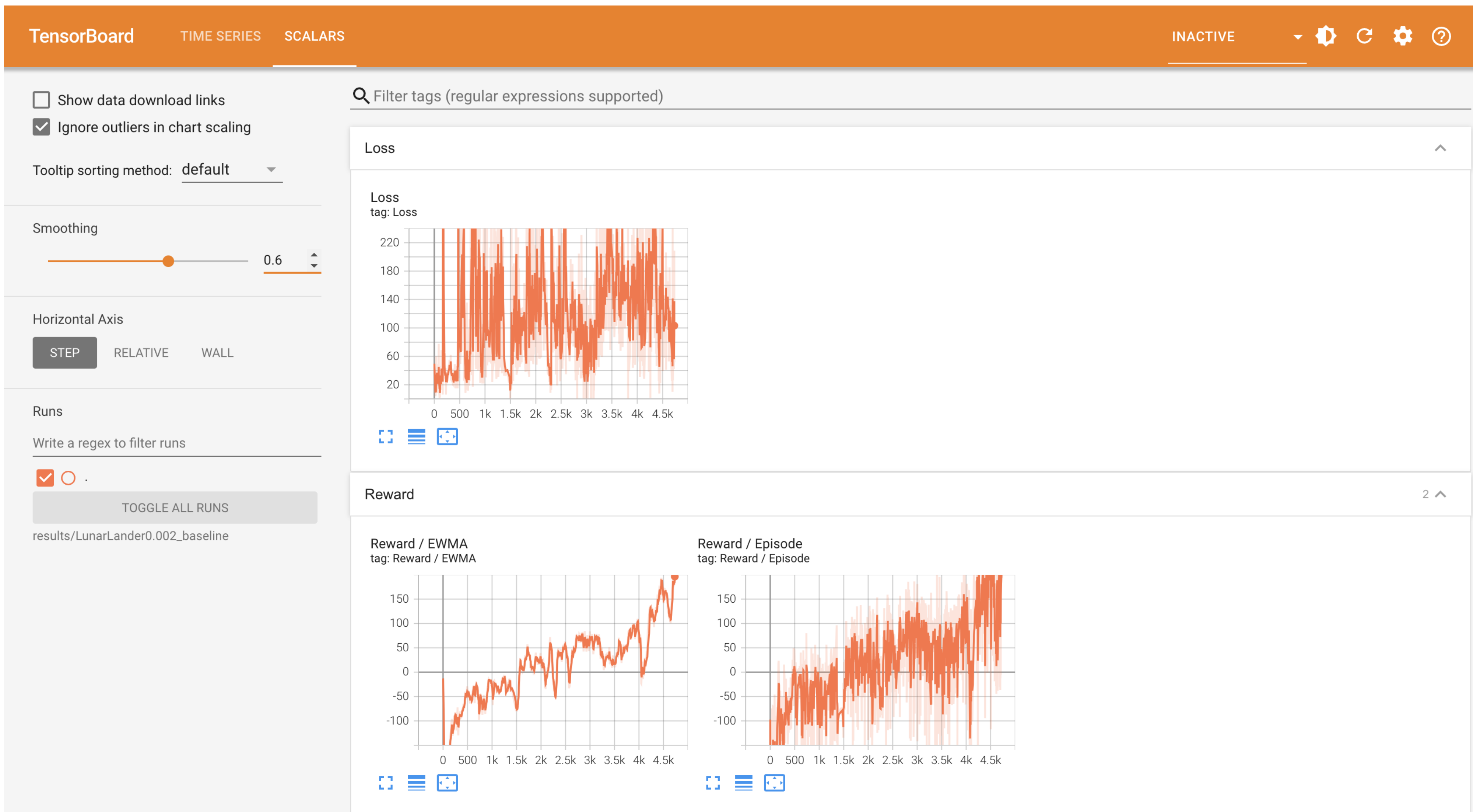


REINFORCE with baseline

- Choose state value as baseline
- NN architecture
  - Share layer: `nn.Sequential`

```
self.affine1 = nn.Sequential(
             nn.Linear(self.observation_dim, self.hidden_size),
             nn.ReLU(),
             nn.Linear(self.hidden_size, self.hidden_size),
             nn.ReLU()
        )
```

  - For Actor: `nn.Sequential(nn.Linear(128, self.action_dim), nn.Softmax(dim=-1))`
  - For Critic: `nn.Linear(128, 1)`
- Learning rate: `lr = 0.002`



REINFORCE with GAE

- NN architecture
  - The same with NN use in *REINFORCE with baseline*
  - Share layer: `nn.Sequential`

    ```
    self.affine1 = nn.Sequential(
                nn.Linear(self.observation_dim, self.hidden_size),
                nn.ReLU(),
                nn.Linear(self.hidden_size, self.hidden_size),
                nn.ReLU()
            )
    ```

  - For Actor: `nn.Sequential(nn.Linear(128, self.action_dim), nn.Softmax(dim=-1))`
  - For Critic: `nn.Linear(128, 1)`
- Learning rate: `lr = 0.002`
- Lambda: `0.99`
- **Advantage Calculation Process**

  The process of calculating the advantage estimates is as follows:

  1. **Initialization**: An empty list `advantages` is initialized to store the advantage values. Two variables, `advantage` and `next_value`, are initialized to zero. The `advantage` variable accumulates the discounted TD-errors, and `next_value` holds the value estimate for the state at the next time step.
  2. **Backward Pass Through Time**: The function iterates backward through each time step:
     - For each step `i`, it retrieves the reward `r` and the current value estimate `v`.
     - The Temporal Difference (TD) error is calculated as:

       $\text{td\_error} = r + (\gamma \times \text{next\_value}) - v$
     - The `advantage` is updated using the TD-error, discounted by both `gamma` and `lambda_`:

       $\text{advantage} = \text{td\_error} + (\gamma \times \lambda_ \times \text{advantage})$
     - This updated advantage is prepended to the `advantages` list to maintain the correct order (since the iteration is backward).
     - The `next_value` is updated to the current value `v` for use in the next iteration.
  3. **Tensor Conversion**: Finally, the list of advantage values is converted to a PyTorch tensor for compatibility with other calculations in neural networks or for use with gradient-based optimization algorithms.