Problem 4　0816170 郭建良

a. Vanilla reinforce algorithm

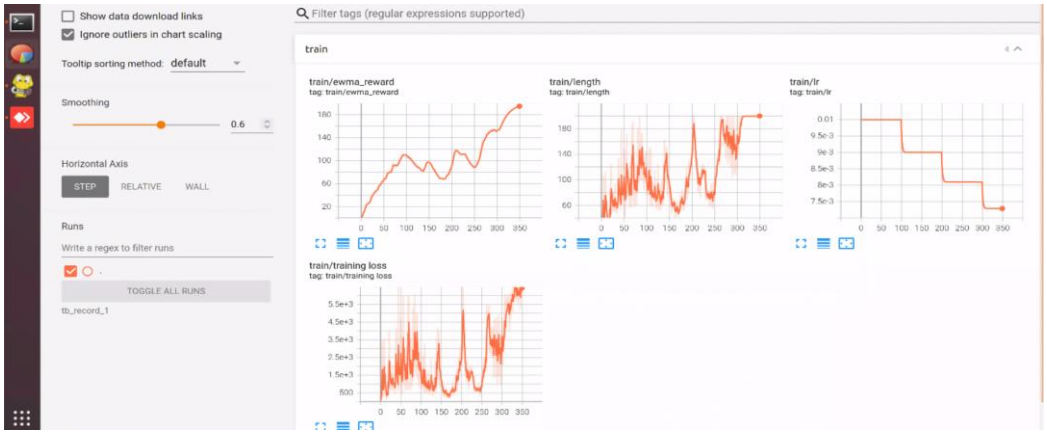
The network architecture is as follows. The value network is constructed but not used for training in this vanilla version. The only used network is the policy network, which has two layers with dropout mechanism.

```
self.linear1_for_shared = nn.Linear(self.observation_dim, self.hidden_size)
print(self.linear1_for_shared)
self.dropout = nn.Dropout(p = 0.5)
self.linear2_for_policy = nn.Linear(self.hidden_size, self.action_dim)
self.linear2_for_value = nn.Linear(self.hidden_size, 1)
```

```
shared_linear = self.linear1_for_shared(state)
shared_linear = self.dropout(shared_linear)
# shared_linear = F.relu(shared_linear)
action_scores = self.linear2_for_policy(shared_linear)
action_prob = F.softmax(action_scores, dim=1)
state_value = self.linear2_for_value(shared_linear)
```

As for hyperparameters, I use initial lr =0.01, which is derived from grid search, and the scheduler in the sample code to schedule lr. The gamma is 0.999, which is the same as the default in the sample code.

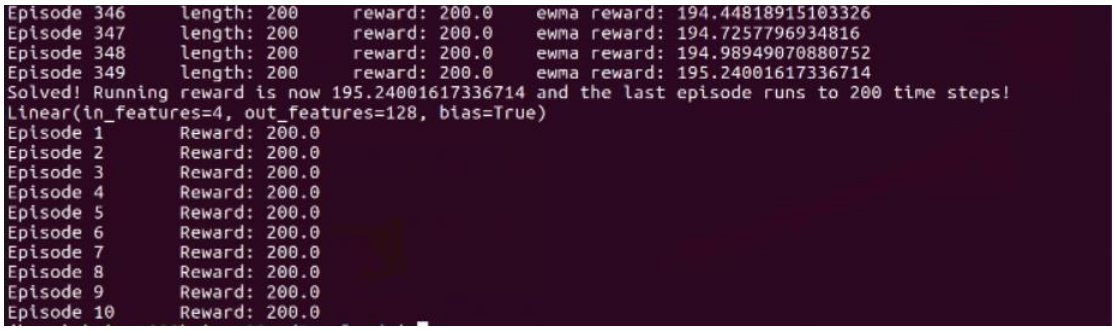
The figure below is the snapshot of the Tensorboard record.



We can observe that the ewma_reward is gradually getting higher as the number of trained episodes increases, which means the model is performing better and better in this task.

The lr is lower and lower due to the scheduler.

The test result is as follows.

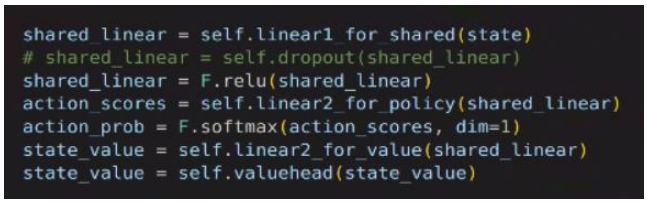


After 349 episodes, the ewma_reward reach the threshold 195, and the problem is solved.

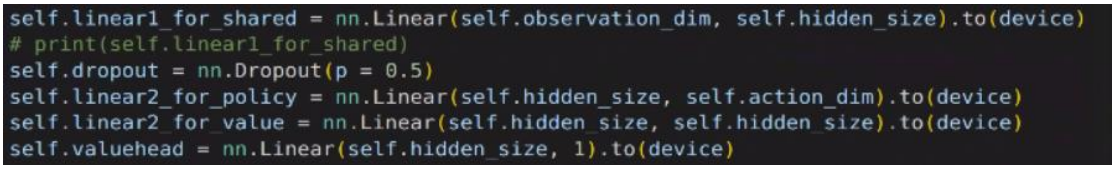
b. Reinforce algorithm with a baseline **The code submitted is the version using method 1

Method 1: Actor critic (use trained value function as baseline)

As the baseline discussion in class, I choose value function as baseline. Compared to vanilla Reinforce algorithm implementation in (a), I add a 3-layer NN network to construct the value network (1 shared, 2 independent). The network architecture is as follows.

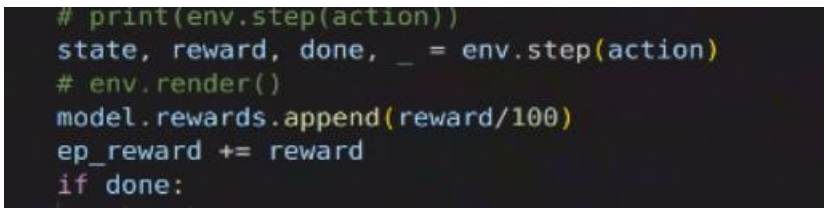


```
shared_linear = self.linear1_for_shared(state)
# shared_linear = self.dropout(shared_linear)
shared_linear = F.relu(shared_linear)
action_scores = self.linear2_for_policy(shared_linear)
action_prob = F.softmax(action_scores, dim=1)
state_value = self.linear2_for_value(shared_linear)
state_value = self.valuehead(state_value)
```



```
self.linear1_for_shared = nn.Linear(self.observation_dim, self.hidden_size).to(device)
# print(self.linear1_for_shared)
self.dropout = nn.Dropout(p = 0.5)
self.linear2_for_policy = nn.Linear(self.hidden_size, self.action_dim).to(device)
self.linear2_for_value = nn.Linear(self.hidden_size, self.hidden_size).to(device)
self.valuehead = nn.Linear(self.hidden_size, 1).to(device)
```

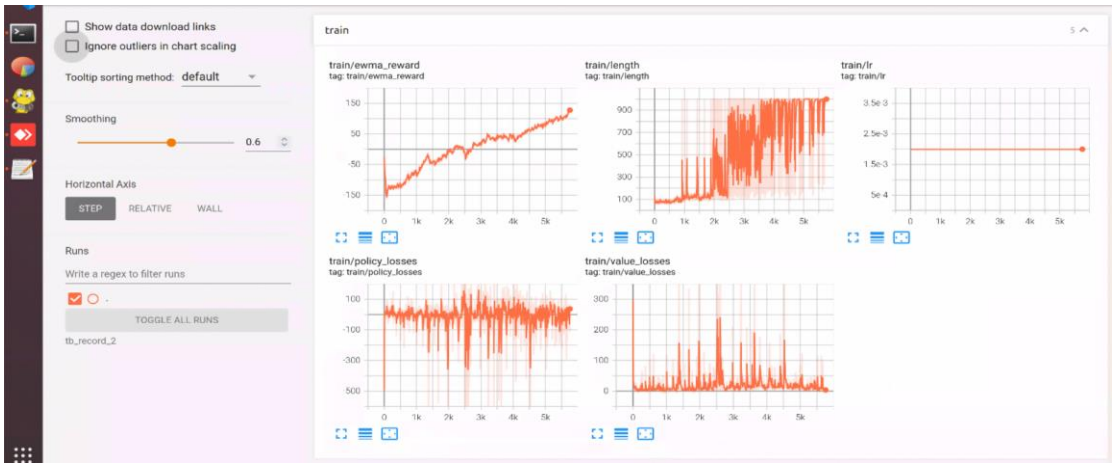
Besides, I scale the reward for training, so that the loss of value network is closer to the one of policy network.



```
# print(env.step(action))
state, reward, done, _ = env.step(action)
# env.render()
model.rewards.append(reward/100)
ep_reward += reward
if done:
```

As for hyperparameters, I use a fixed lr =0.002, which is derived from grid search. The gamma is 0.999, which is the same as the default in the sample code.

The figure below is the snapshot of the Tensorboard record.



We can observe that the ewma_reward is gradually getting higher as the number of trained episodes increases, which means the model is performing better and better in this task. In addition, the value loss converges to a small value, indicating that the prediction of value network is very close to ground truth value for the environment.

The test result is as follows.

```
Episode 5753    length: 1000    reward: 93.23093484535856       ewma reward: 123.87313741212377
Episode 5754    length: 1000    reward: 137.99969447416757      ewma reward: 124.57946526522596
Episode 5755    length: 1000    reward: 173.61620499169277      ewma reward: 127.03130225154929
Episode 5756    length: 1000    reward: 144.24395173884733      ewma reward: 127.89193472591418
Episode 5757    length: 1000    reward: 178.6216592045302       ewma reward: 130.428420949845
Solved! Running reward is now 130.428420949845 and the last episode runs to 1000 time steps!
/home/hcis-s09/miniconda3/lib/python3.7/site-packages/gym/core.py:50: DeprecationWarning: WARN: You
are calling render method, but you didn't specified the argument render_mode at environment initiali
zation. To maintain backward compatibility, the environment will render in human mode.
If you want to render in human mode, initialize the environment in this way: gym.make('EnvName', ren
der_mode='human') and don't call the render method.
See here for more information: https://www.gymlibrary.ml/content/api/
  "You are calling render method, "
Episode 1       Reward: 163.22112510188654
Episode 2       Reward: 28.197656077422252
Episode 3       Reward: 128.9131609971315
Episode 4       Reward: 168.9722949370005
Episode 5       Reward: 102.33815915101363
Episode 6       Reward: 99.39418517663863
Episode 7       Reward: 123.50910112131191
Episode 8       Reward: 127.10340850453257
Episode 9       Reward: 144.5258326342473
Episode 10      Reward: 125.54221805982264
```

After 5757 episodes, the ewma_reward reach the threshold 130, and the problem is solved.

I set the threshold to 130 in order to train a more stable model.

Method 2 : Whitening transformation (老師說實作上可行，在此 task 有用，但不是所有情況都通用)

I found that Whitening transformation is useful for reducing variance, so I conduct another simulation with this method. Note that the baseline in this method uses the information of the whole trajectory (mean and standard deviation). Compared to vanilla Reinforce algorithm implementation in (a), I only standardize Gt's to reduce variance. The screenshot of my code below is the modified part.

```
135     returns = torch.tensor(list(reversed(reversed_returns)))
136
137     #use whitening transformation to modify Gt as the result of Gt - B(st)
138     returns = (returns - returns.mean()) / (returns.std())
139
140     for log_prob_i, R_i in zip(saved_actions, returns):
```

The picture below further explain the detail of Baseline function

$B(s_t)$ use the information of the trajectories to perform standardization, so, that:

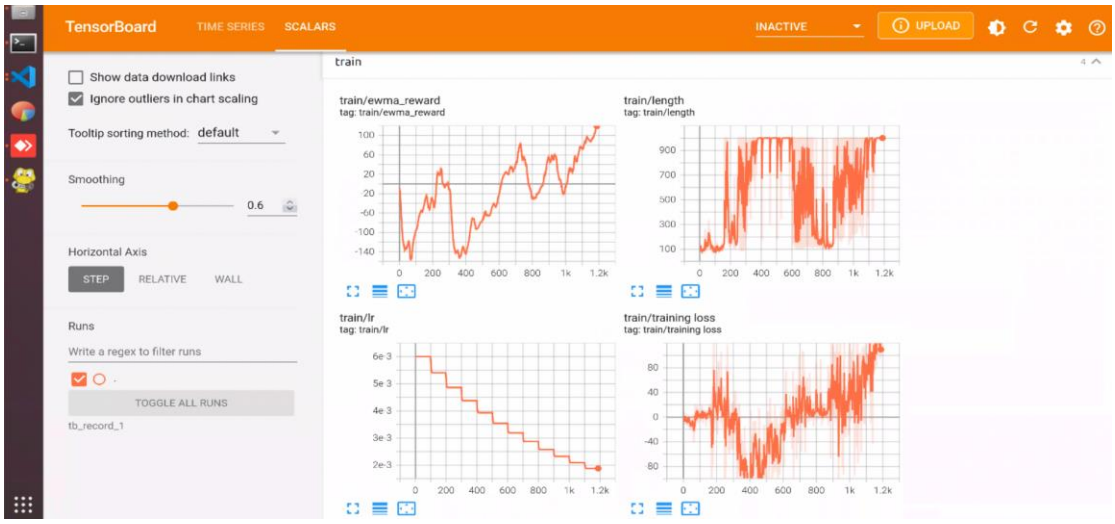
$$G_t - B(s_t) = \frac{G_t - \bar{G}}{\sigma_G}$$

$$\Rightarrow B(s_t) = \frac{\sigma_G G_t - G_t + \bar{G}}{\sigma_G},$$

where $\bar{G}$ is the arithmetic mean of G_t's in a trajectory,
σ_G is the standard deviation of G_t's in a trajectory

As for hyperparameters, I use initial lr =0.006, which is derived from grid search, and the scheduler in the sample code to schedule lr. The gamma is 0.999, which is the same as the default in the sample code.

The figure below is the snapshot of the Tensorboard record.



We can observe that the ewma_reward and training loss is gradually getting higher as the number of trained episodes increases, which means the model is performing better and better in this task.

The lr is lower and lower due to the scheduler.

The test result is as follows.

```
Episode 1186    length: 1000    reward: 102.42212057556586      ewma reward: 118.75730189400898
Episode 1187    length: 1000    reward: 134.28655074234123      ewma reward: 119.53376433642558
Episode 1188    length: 1000    reward: 140.6530261025518       ewma reward: 120.58972742473188
Solved! Running reward is now 120.58972742473188 and the last episode runs to 1000 time steps!
/home/hcis-s09/miniconda3/lib/python3.7/site-packages/gym/core.py:50: DeprecationWarning: WARN: You a
re calling render method, but you didn't specified the argument render_mode at environment initializa
tion. To maintain backward compatibility, the environment will render in human mode.
If you want to render in human mode, initialize the environment in this way: gym.make('EnvName', rend
er_mode='human') and don't call the render method.
See here for more information: https://www.gymlibrary.ml/content/api/
  "You are calling render method, "
Episode 1       Reward: 53.621102102468086
Episode 2       Reward: 148.78663954630494
Episode 3       Reward: 160.2569265288049
Episode 4       Reward: 126.35290998112879
Episode 5       Reward: 88.03023513621146
Episode 6       Reward: 154.16481961344854
Episode 7       Reward: 23.67573331462205
Episode 8       Reward: 142.36701086670985
Episode 9       Reward: 133.48308275262173
Episode 10      Reward: 163.69128776768702
```

After 1188 episodes, the ewma_reward reach the threshold 120, and the problem is solved.

c. Reinforce with GAE

Implement the GAE calculation function discussed in class (there is a code in the slide)

```
reversed_advantages = []
advantages = []
advantage = 0
next_value = 0

for true_reward, log_prob_and_value in zip(reversed(rewards), reversed(saved_actions)):
    td_error = true_reward + next_value * self.gamma - log_prob_and_value.value.item()
    advantage = td_error + advantage * self.gamma * self.lambda_
    next_value = log_prob_and_value.value.item()
    reversed_advantages.append(advantage)
advantages = list(reversed(reversed_advantages))
advantages = torch.tensor(advantages)
return advantages
```

Compared to vanilla Reinforce algorithm implementation in (a), I add a 3-layer NN network to construct the value network (1 shared, 2 independent) and a 3-layer NN network to construct the policy network (1 shared, 2 independent).

The network architecture is as follows.

```
self.linear1_for_shared = nn.Linear(self.observation_dim, self.hidden_size)
# print(self.linear1_for_shared)
self.dropout = nn.Dropout(p = 0.5)
self.linear2_for_policy = nn.Linear(self.hidden_size, self.hidden_size)
self.linear2_for_value = nn.Linear(self.hidden_size, self.hidden_size)
self.policy_head = nn.Linear(self.hidden_size, self.action_dim)
self.value_head =  nn.Linear(self.hidden_size, 1)
```

```
shared_linear = self.linear1_for_shared(state)
# shared_linear = self.dropout(shared_linear)
# shared_linear = F.relu(shared_linear)
action_scores = self.linear2_for_policy(shared_linear)
action_scores = self.policy_head(action_scores)
action_prob = F.softmax(action_scores, dim=1)

state_value = self.linear2_for_value(shared_linear)
state_value = self.value_head(state_value)
```

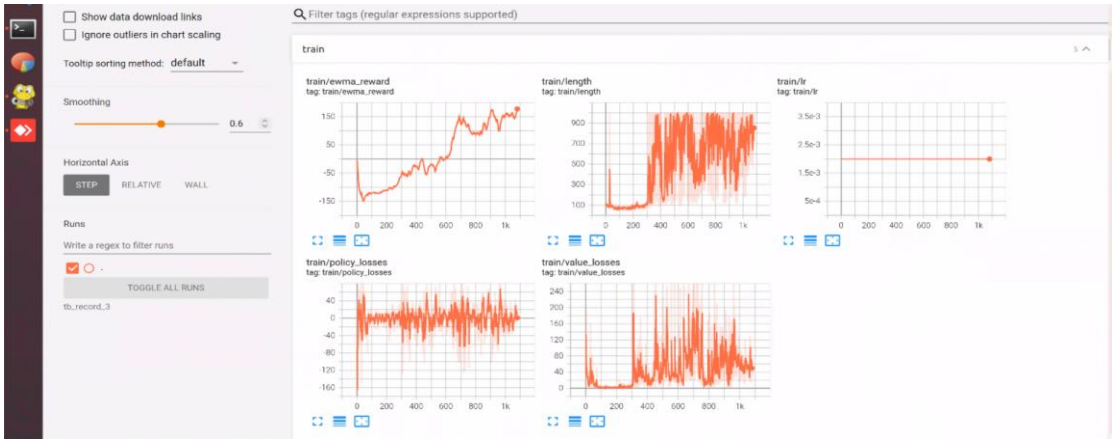
Besides, I scale the reward for training, so that the loss of value network is closer to the one of policy network.

```
# print(env.step(action))
state, reward, done, _ = env.step(action)
# env.render()
model.rewards.append(reward/100)
ep_reward += reward
if done:
```

As for hyperparameters, I use a fixed lr =0.002, which is derived from grid search. The gamma is 0.999, which is the same as the default in the sample code.

The figures below are the snapshot of the Tensorboard record and the running results with different lambda.

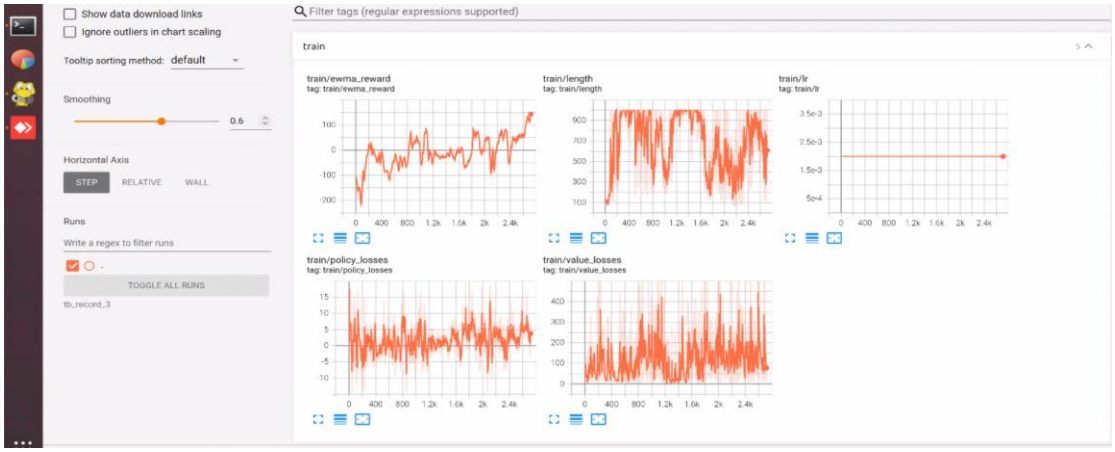
Lambda = 0.99

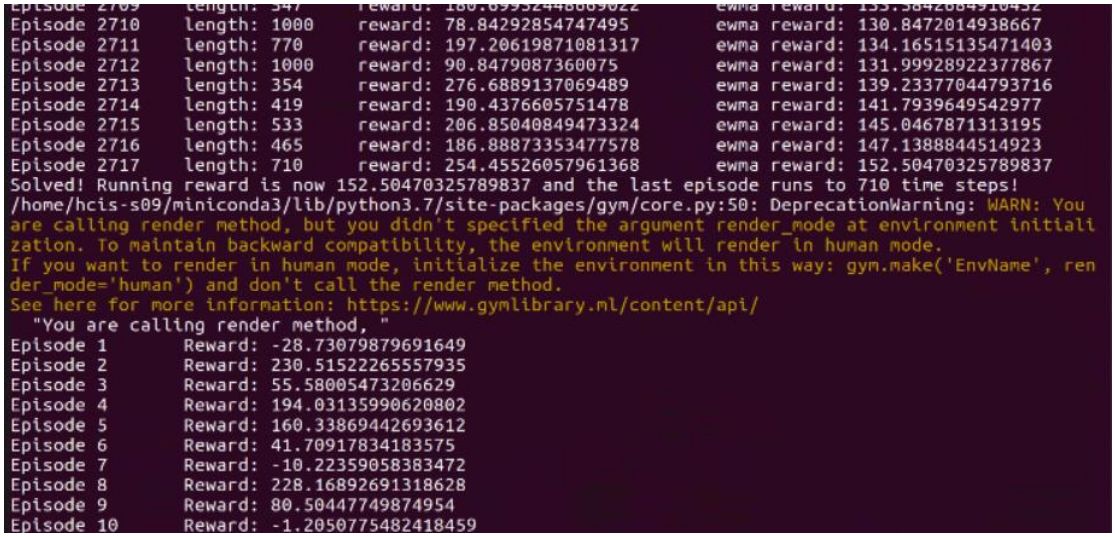



```
Episode 1074    length: 1000    reward: 127.304500660518        ewma reward: 160.9750955801389
Episode 1075    length: 593     reward: 274.2797106314875       ewma reward: 166.64032633270634
Episode 1076    length: 712     reward: 226.64563875097167      ewma reward: 169.64059195361958
Episode 1077    length: 748     reward: 262.6006170883862       ewma reward: 174.2885932103579
Episode 1078    length: 671     reward: 242.5571807588811       ewma reward: 177.70202258778406
Episode 1079    length: 998     reward: 212.3881126128077       ewma reward: 179.43632708903525
Episode 1080    length: 877     reward: 221.2241955846406       ewma reward: 181.52572051381551
Solved! Running reward is now 181.52572051381551 and the last episode runs to 877 time steps!
Episode 1       Reward: 274.5531920736165
Episode 2       Reward: 74.7884041712923
Episode 3       Reward: 159.49839575180152
Episode 4       Reward: 270.171470006407
Episode 5       Reward: 262.5499705815166
Episode 6       Reward: 240.72949094996594
Episode 7       Reward: 232.41605471726288
Episode 8       Reward: 123.95887975957176
Episode 9       Reward: 266.9844675856521
Episode 10      Reward: 186.3727285730632
```

We can observe that the ewma_reward is gradually getting higher as the number of trained episodes increases, which means the model is performing better and better in this task. In addition, the value loss converges to a small value, indicating that the prediction of value network is very close to ground truth value for the environment.

Lambda = 0.9



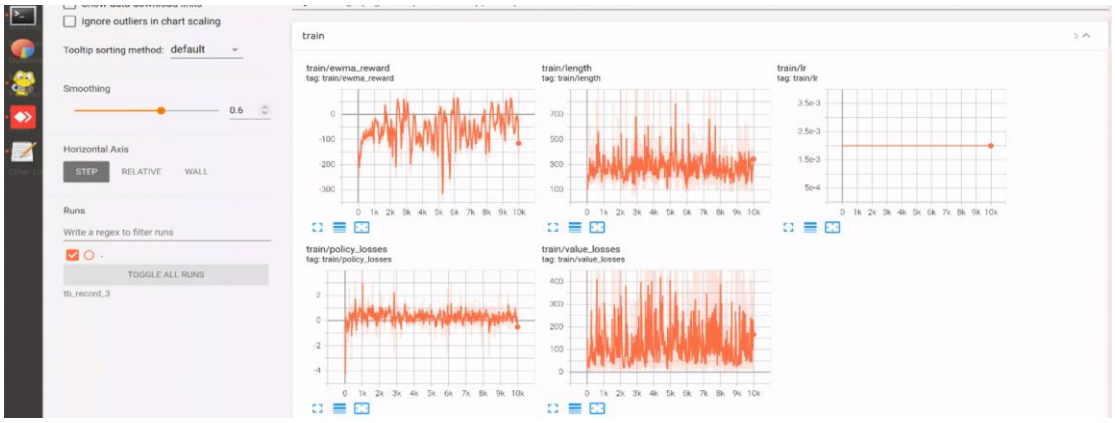


We can observe that the ewma_reward is gradually getting higher as the number of trained episodes increases, which means the model is performing better and better in this task.

However, the performance is slightly worse than the result of lambda=0.99.

Besides, it also uses 2717 episodes to converge, which is also slower than the convergence of lambda=0.99.

Lambda = 0.5




```
Episode 10046   length: 155     reward: -36.194119787387805      ewma reward: -81.94578335472762
Episode 10047   length: 197     reward: -31.265682526532416      ewma reward: -79.41177831331785
Episode 10048   length: 218     reward: -41.113712142681734      ewma reward: -77.49687500478603
Episode 10049   length: 476     reward: -119.17851085752325      ewma reward: -79.58095679742289
Episode 10050   length: 342     reward: -76.49914399906953       ewma reward: -79.42686615750522
Episode 10051   length: 377     reward: 247.4592586731922        ewma reward: -63.08255991597035
Episode 10052   length: 165     reward: -74.995012158415 3       ewma reward: -63.6781825280926
Episode 10053   length: 514     reward: 221.12723235061623       ewma reward: -49.43791178415715
Episode 10054   length: 287     reward: 240.07978075711858       ewma reward: -34.96202715709336
Episode 10055   length: 210     reward: -39.357876947085614      ewma reward: -35.18181964659297
Episode 10056   length: 1000    reward: 39.93007836224467        ewma reward: -31.426224746151085
Episode 10057   length: 144     reward: -147.34593858884202      ewma reward: -37.22221043828563
Episode 10058   length: 679     reward: 144.55620792872125       ewma reward: -28.133289519935282
Episode 10059   length: 130     reward: -175.45160528731338      ewma reward: -35.499205308304184
Episode 10060   length: 371     reward: -63.20597770816662       ewma reward: -36.884543928297305
Episode 10061   length: 137     reward: 6.974852127999469        ewma reward: -34.69157412548246
Episode 10062   length: 210     reward: -0.09497761950500205     ewma reward: -32.96174430018358
Episode 10063   length: 455     reward: 239.61040214329353       ewma reward: -19.333136978009726
Episode 10064   length: 496     reward: 203.10830083650575       ewma reward: -8.21106508728395
Episode 10065   length: 100     reward: -12.447518566592493      ewma reward: -8.422887761249378
Episode 10066   length: 375     reward: -259.66603058579864      ewma reward: -20.98504490247684
Episode 10067   length: 213     reward: -17.60090792181208       ewma reward: -20.815838053443603
Episode 10068   length: 323     reward: -64.67151742111427       ewma reward: -23.008622021827133
Episode 10069   length: 365     reward: -25.43433229003942       ewma reward: -23.12990753523775
Episode 10070   length: 1000    reward: 134.15191610036004       ewma reward: -15.265816353457858
Episode 10071   length: 264     reward: -68.44334951915724       ewma reward: -17.924693011742825
Episode 10072   length: 105     reward: -12.350415044351436      ewma reward: -17.645979113373254
Episode 10073   length: 277     reward: 13.985091474266397       ewma reward: -16.064425583991273
Episode 10074   length: 205     reward: -3.9587418551133453      ewma reward: -15.459141397547377
Episode 10075   length: 226     reward: -63.77433909728519       ewma reward: -17.874901282534267
Episode 10076   length: 150     reward: -0.011850234835080187    ewma reward: -16.981748730149306
Episode 10077   length: 555     reward: 195.6573812261688        ewma reward: -6.349792232333398
Episode 10078   length: 320     reward: -223.66375155341012      ewma reward: -17.215490198387236
Episode 10079   length: 433     reward: 226.52741400290782       ewma reward: -5.028344988322482
Episode 10080   length: 259     reward: -208.2014230975285       ewma reward: -15.186998893782786
Episode 10081   length: 210     reward: 31.75587601061204        ewma reward: -12.839855148563043
Episode 10082   length: 402     reward: -217.8746821548654       ewma reward: -23.09159649887816
Episode 10083   length: 245     reward: 6.856055816685142        ewma reward: -21.594213883099997
Episode 10084   length: 165     reward: 23.680151621000945       ewma reward: -19.33049560789495
Episode 10085   length: 156     reward: 28.437531091793886       ewma reward: -16.942094272910506
Episode 10086   length: 226     reward: -62.4942179236571        ewma reward: -19.219700455447835
Episode 10087   length: 103     reward: -11.190344088729574      ewma reward: -18.818232637111922
Episode 10088   length: 294     reward: -56.47973627671897       ewma reward: -20.701307819092275
Episode 10089   length: 156     reward: -59.91853290002305       ewma reward: -22.662169073138813
Episode 10090   length: 352     reward: -70.90060982585  ewma reward: -25.074091110774372
Episode 10091   length: 101     reward: 34.70244545926792        ewma reward: -22.08526428227226
Episode 10092   length: 232     reward: -182.66593586303395      ewma reward: -30.11429786131034
Episode 10093   length: 313     reward: -180.8371339122126       ewma reward: -37.65043966385545
Episode 10094   length: 499     reward: 173.0004781572414        ewma reward: -27.117893772800606
Episode 10095   length: 290     reward: -10.529270289859795      ewma reward: -26.288462598653563
```

Fail to converge before 10000 episodes :<