Multi Objective PPO

git@github.com:Yaishnavi/MultiObjective-PPO.git

Contents

Introduction

Multi-Objective RL problem with k different reward functions. Find a policy that performs well for all k reward functions.
Implemented the following algorithm:
Use PPO to find policy pi_1 with respect to rewards r_1
For k = 2 to K

Use PPO to find policy pi_k where reward is r_k + a*KL(pi_{k-1}|| pi_k)

The **Kullback-Leibler Divergence** score, or KL divergence score, quantifies how much one probability distribution differs from another probability distribution.
The KL divergence between two distributions Q and P is often stated using the following notation: KL(P || Q)
Where the "||" operator indicates "divergence" or Ps divergence from Q.

PPO is a policy gradient method for reinforcement learning, which alternates between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates.
Proximal Policy Optimization
PPO can be viewed as an approximation of TRPO, but unlike TRPO, which uses a second-order Taylor expansion, PPO uses only a first-order approximation, which makes PPO very effective in RNN networks and in a wide distribution space.

for $i \in 1, 2, \ldots, N$ do:
    Run policy π_θ for T timesteps, collecting s_t, a_t, r_t
    $\pi_{\text{old}} \leftarrow \pi_\theta$
    for $j \in 1, 2, \ldots, M$ do:
        $J_{\text{PPO}}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \hat{A}_t - \lambda \text{KL}[\pi_{\text{old}}|\pi_\theta]$
        Update θ by a gradient method w.r.t. $J_{\text{PPO}}(\theta)$
    end for
    for $j \in 1, 2, \ldots, B$ do:
        $L_{\text{BL}}(\phi) = -\sum_{t=1}^{T} (\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t))^2$
        Update ϕ by a gradient methor w.r.t. $L_{\text{BL}}(\phi)$
    end for
    if $\text{KL}[\pi_{\text{old}}|\pi_\theta] > \beta_{\text{high}} \text{KL}_{\text{target}}$ then
        $\lambda \leftarrow \alpha\lambda$
    if $\text{KL}[\pi_{\text{old}}|\pi_\theta] < \beta_{\text{high}} \text{KL}_{\text{target}}$ then
        $\lambda \leftarrow \alpha/\lambda$

The first half of Estimate Advantage is obtained through the rollout strategy, and the second half of V is obtained from a value network. (Value network can be trained by the data obtained by rollout, where the mean square error is used).

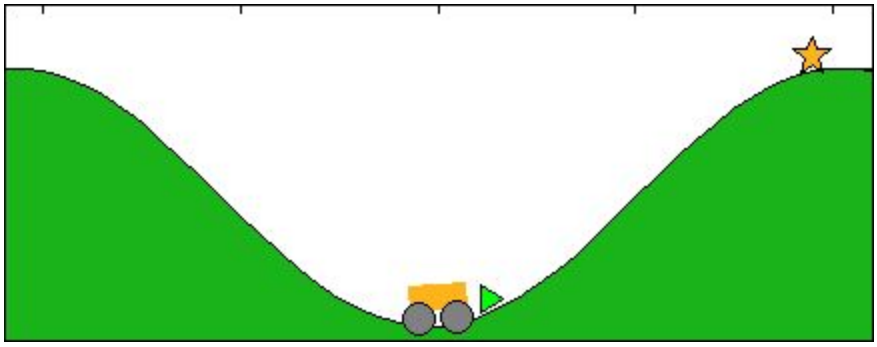
There is a clipped surrogate objective,

$$L^{\text{CLIP}}(\theta) = \hat{E}_t[r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$
$$\text{where } r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Dataset

Fruit API is a universal deep reinforcement learning framework, which is designed meticulously to provide a friendly user interface, a fast algorithm prototyping tool, and a multi-purpose library for the RL research community. External environments can be integrated into the framework easily by plugging into FruitEnvironment. Finally, we developed 5 extra environments as a testbed to examine different disciplines in deep RL:

1. **Mountain car** (multi-objective environment/graphical support)



The Mountain Car problem defines an environment where a car is required to escape from a valley as illustrated in figure. The car's engine is not powerful enough to climb up the mountain on the right side. Therefore, the car needs to reverse up the left side to obtain additional momentum. The learning algorithm's inputs are the car's current position and velocity whilst the action sets include forward acceleration, backward acceleration, and zero throttle (null action). The first objective of the problem is to minimize the number of steps taken by the car. Two other objectives include minimizing the number of backward and forward acceleration actions. As such, a penalty of -1 is applied to each time step and the same is applied to each backward (or forward) acceleration action.

The MO Mountain Car problem has three intrinsic rewards corresponding to the three objectives. In the implementation, we limit the time steps to 100; therefore, an episode terminates when the time step exceeds 100 or the car reaches the goal.

2. Deep sea treasure (multi-objective environment/graphical support)

DST takes advantage of predefined Pareto solutions so that it becomes a normative multiobjective environment to verify new methods. While the original DST proposed had a fixed spatial structure and rewards, here we use the generalised DST which allows the creation of parameterised instances, varying in terms of the size of the state space, and the values of the treasure rewards. The state output of DST can be a scalar (current position of the agent) or a graphical representation (image). Two grid DST environments are illustrated in figure, with the dimensions of 10x5.

5-column DST

The agent is designed to control a submarine that searches for treasure under a sea. Two objectives need to be optimized: maximize the treasure values and minimize the search time. Therefore, the DST problem has one extrinsic (treasure values) and one intrinsic (time penalty) reward. The submarine starts each episode at the top left state and ends when it finds a treasure 9 location, or the predefined maximum number of actions is reached. Four actions including move up, down, left, and right are available to the agent. The agent receives a reward characterized by a 2-element vector representing the treasure value and time penalty. The treasure value is 0 unless the agent reaches a treasure location. Each move returns -1 time penalty.

3. Tank battle (multi-agent/multi-objective/human-agent cooperation environment)

The world editor of Warcraft III has been selected to create a game to simulate a tankbattle game . At first, the whole map will be divided into areas from left to right. The NPC tank will move randomly towards the left or right side of the map. In the game, the player will fire at several fixed areas (see figure). Once the tank is hit by the player in a particular area, the probability of moving to the area will be reduced next time.

Here the two objective is the reward from two players playing simultaneously. Player one is the human and player 2 is the machine. Each reward is defined as below

$$f(x_i)=f(x_i)+0.2(R(x_i)+f(x_j)-f(x_i))$$

In this formula, f(xi) is the probability to move previously and f(xj) is the probability to move now. R(x) is the reward, and the parameter 0.2 is our learning parameter. After approximately 15 rounds, this tank will not move to the area where it will be attacked.

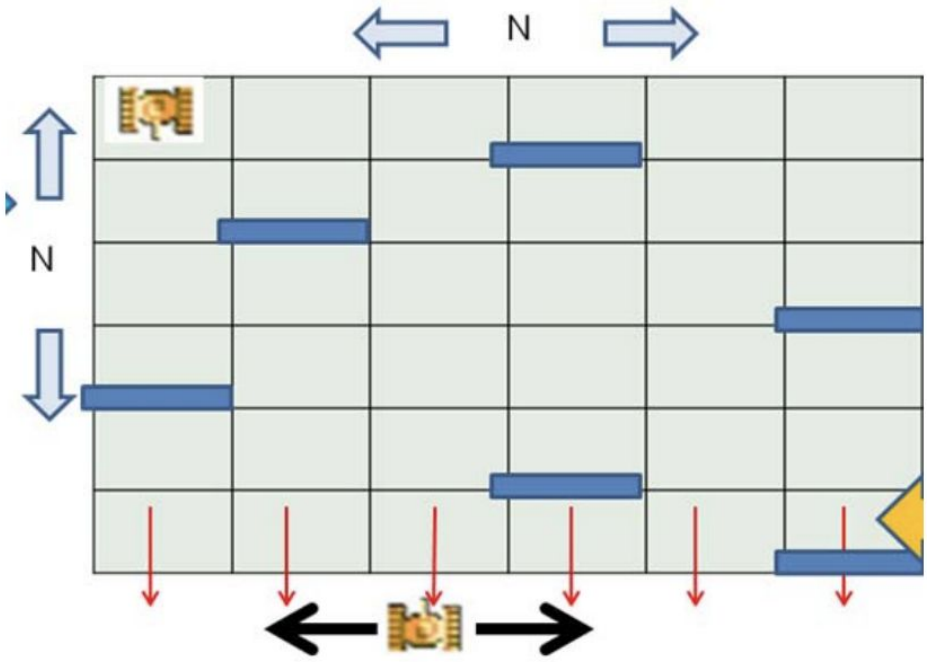
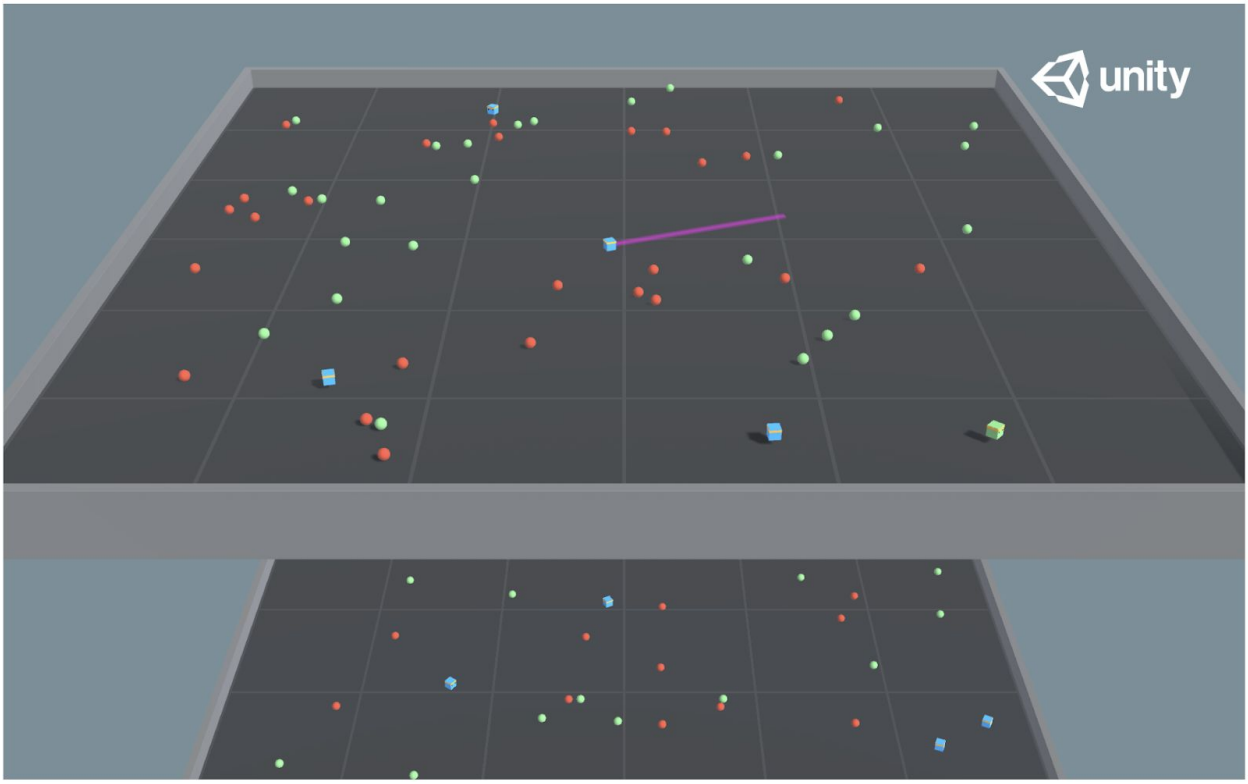


Figure 8. The concept graph of the tank game

4. Food collector (multi-objective environment)



The food collector game is about collecting certain objects (food) while avoiding others (poison).
Set-up: A multi-agent environment where agents compete to collect food.
Goal: The agents must learn to collect as many green food spheres as possible while avoiding red spheres.
Agents: The environment contains 5 agents with same Behavior Parameters.
Agent Reward Function (independent): +1 for interaction with yellow spheres -1 for interaction with blue spheres and the second reward is if the agent as eaten the food indicating if it was good +1 or bad -1.
Actions: 3 continuous actions correspond to Forward Motion, Side Motion and Rotation 1 discrete acion branch for Eat with 2 possible actions corresponding to eat or No Action.
Terminate if the player position reaches the door which is randomly placed among other particles.

5. Other environments used but not evaluated for this task are Milk factory and fruit tree.

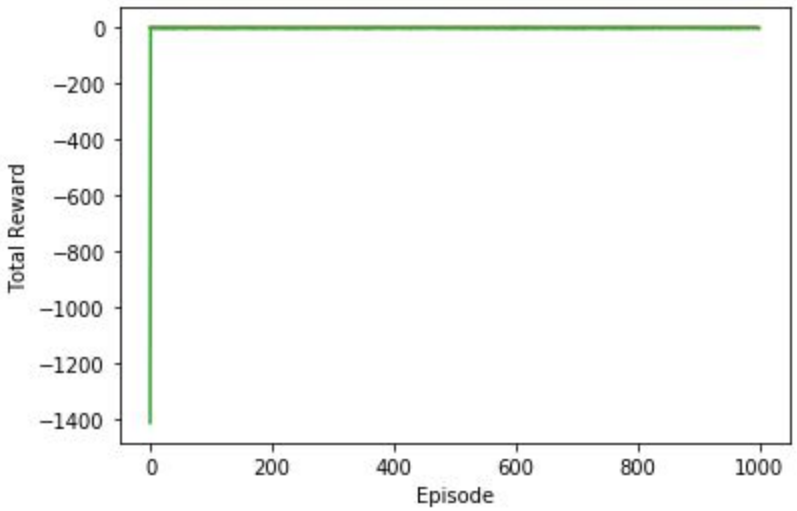
	Number of rewards/objectives	Number of actions	State Dimension
Mountain Car	3	3	1
Deep Sea Treasure	2	4	2
Tank Battle	2	5	650x650x3
Food Collector	2	6	300x325x3
Mountain Car 2	2	3	1

Experiment

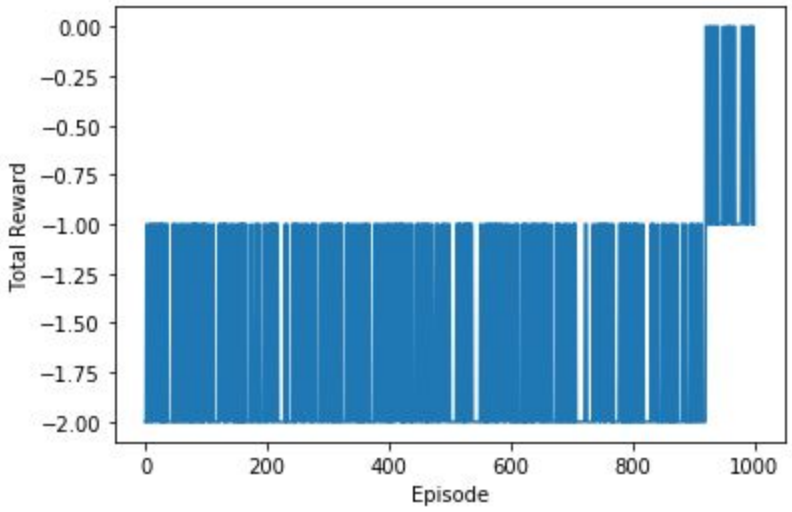
PPO algorithm is used to compute the policy for each objective/reward function. This objective is updated with the KL divergence of the previous policy. The baseline using a linear combination of the rewards to determine the best policy. Here dense neural network model is used for all of the tasks mentioned above.

Results

1. Mountain Car

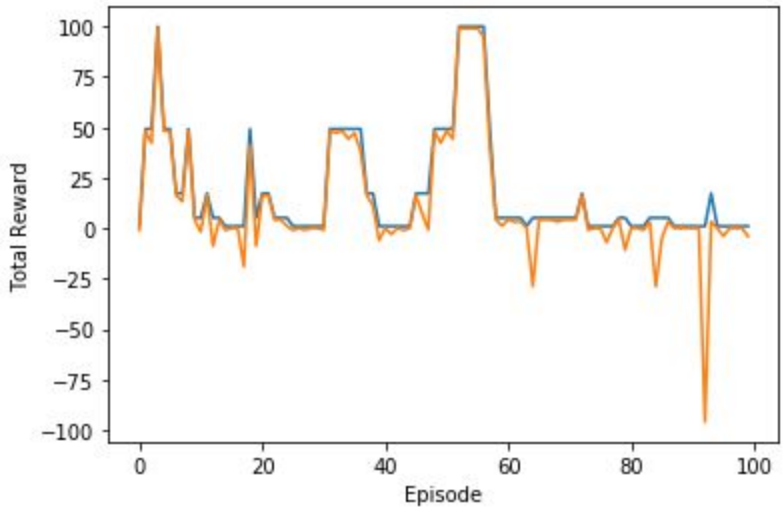


Proposed method

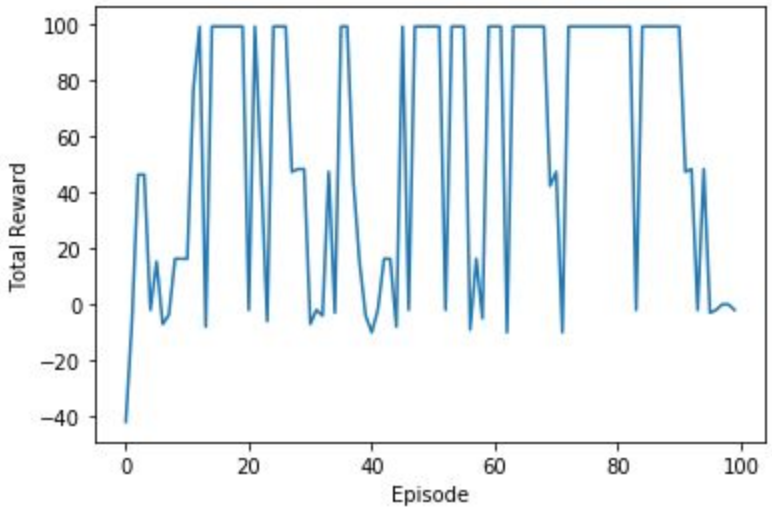


Baseline

2. Deep Sea Treasure

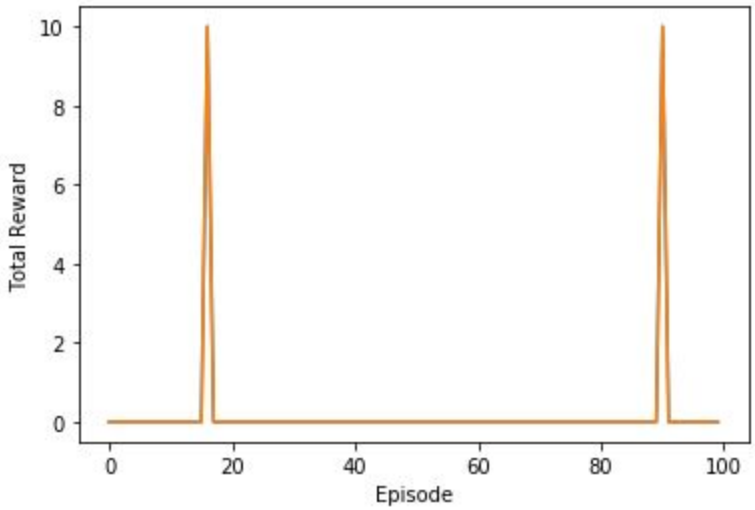


Proposed method

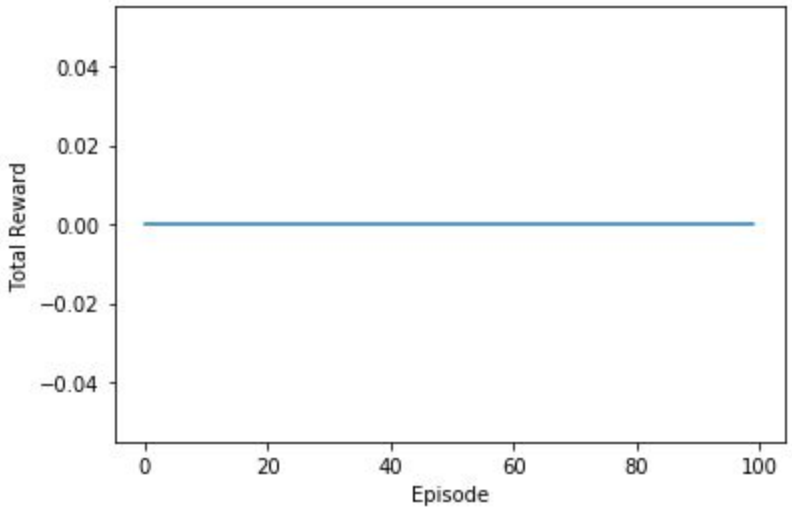


baseline

3. Tank Battle

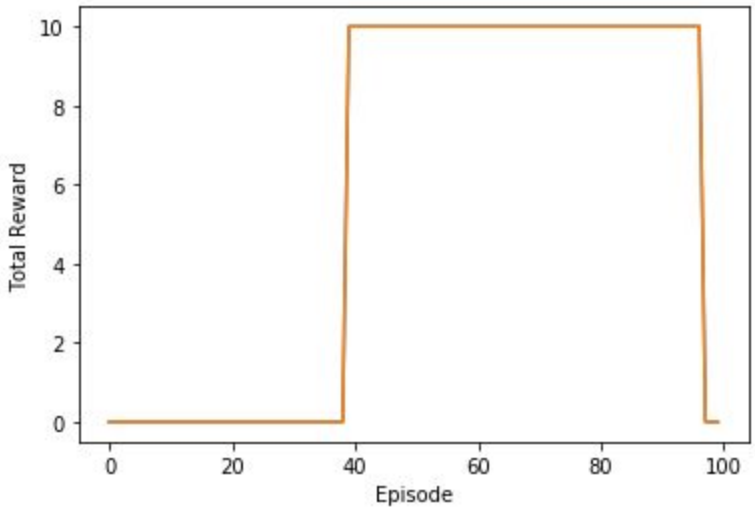


Proposed Method

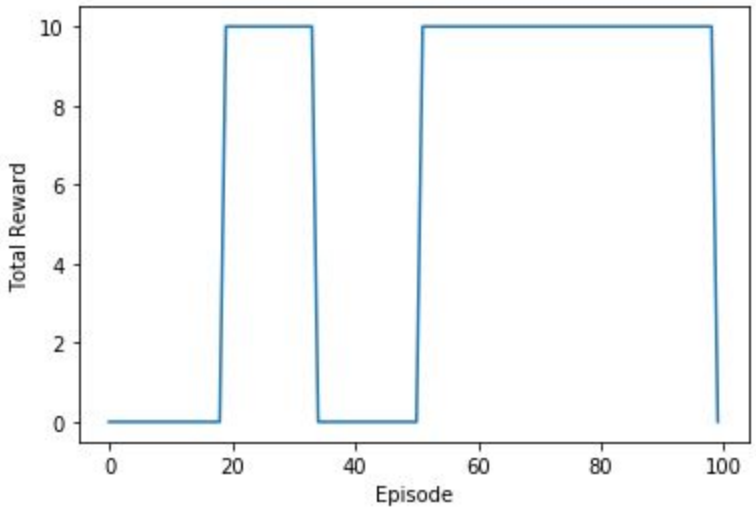


baseline

4. Food Collector

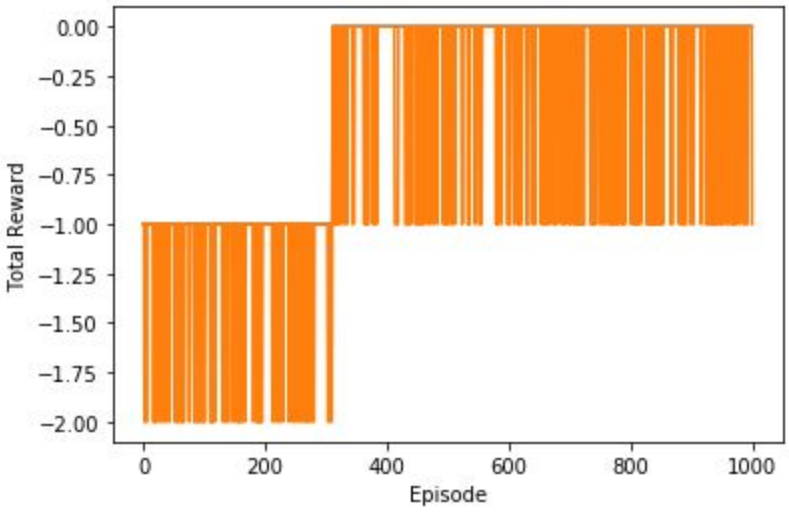


Proposed method

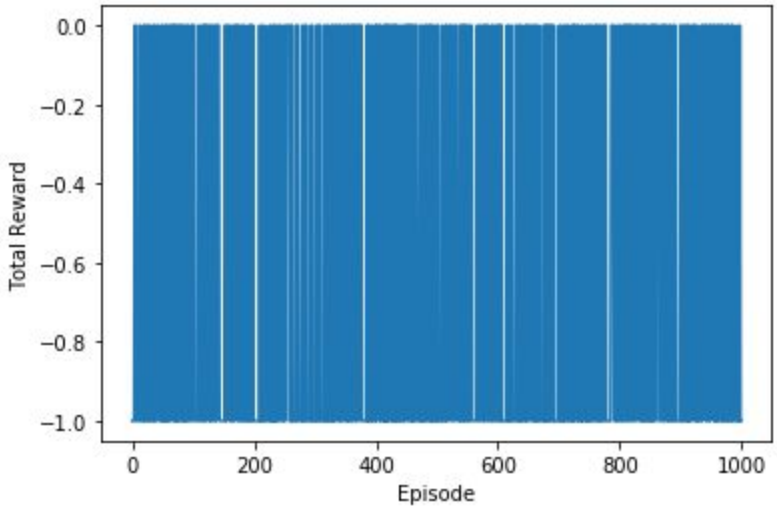


baseline

5. Mountain Car2



Proposed Method



baseline

Installation

1. Install fruit api to access environments
   Git clone https://github.com/garlicdevs/Fruit-API.git
   Open install path and run python setup.py install
2. Other packages needed: Python==3.6, Pytorch, Matplotlib, Numpy

References

- Dataset: FruitAPI https://fruitlab.org/
- Generalizing across Multi-Objective Reward Functions in Deep Reinforcement Learning https://arxiv.org/pdf/1809.06364.pdf
- Proximal Policy Optimization Algorithms https://arxiv.org/pdf/1707.06347.pdf
- Multi Objective Deep Reinforcement Learning https://arxiv.org/pdf/1610.02707.pdf
- Deep Sea Treasure and Mountain Car https://arxiv.org/pdf/1803.02965.pdf
- Tank battle http://www.jsoftware.us/vol5/jsw0512-3.pdf