Multi-action Mixed Loss Proximal Policy Optimization

Xiulei Song[*1] **Yizhao Jin**[*2] **Greg Slabaugh**[2] **Simon Lucas**[2]

Affiliations
[1] JumpW AI Group, Shanghai, China
[2] Queen Mary University of London Game AI Group, London, United Kingdom
songxiulei@jumpw.com, {acw596, g.slabaugh, simon.luacs}qmul@email

Abstract

PPO (Proximal Policy Optimization) is a state-of-the-art policy gradient algorithm that has been successfully applied to complex computer games such as Dota 2 and Honor of Kings. In these environments, an agent makes compound actions consisting of multiple sub-actions. PPO uses clipping to restrict policy updates. Although clipping is simple and effective, it is not efficient in its sample use. For compound actions, most PPO implementations consider the joint probability (density) of sub-actions, which means that if the ratio of a sample (state compound-action pair) exceeds the range, the gradient the sample produces is zero. Instead, for each sub-action we calculate the loss separately, which is less prone to clipping during updates thereby making better use of samples. Further, we propose a multi-action mixed loss that combines joint and separate probabilities. We perform experiments in Gym-μRTS and MuJoCo. Our hybrid model improves performance by more than 50% in different MuJoCo environments compared to OpenAI's PPO benchmark results. And in Gym-μRTS, we find the sub-action loss outperforms the standard PPO approach, especially when the clip range is large. Our findings suggest this method can better balance the use-efficiency and quality of samples.

PPO (Schulman et al. 2017) is a simple and efficient reinforcement learning algorithm inspired by TRPO (Trust Region Policy Optimization) (Schulman et al. 2015). A major challenge for PPO and other policy gradient algorithms is to estimate the correct step of policy updates, since input data is heavily dependent on the current policy (Schulman et al. 2015). TRPO solves this problem by imposing a trust domain constraint on the target function to control the KL divergence between the old and new policies. PPO significantly reduces complexity by employing a clipping mechanism, which attempts to eliminate incentives that push the policy away from the old policy when the likelihood ratio between the two exceeds a limit.

There are existing studies that examine the clipping mechanism of PPO. Engstrom et al. (Engstrom et al. 2020) conducted an experiment on whether a clip mechanism was needed in PPO. In their experiment, PPO code-level optimizations played a greater role than the clipping mechanism. Separately, Wang et al. (Wang, He, and Tan 2020) used a new clipping function to support rollback behavior to limit differences between new and old policies. The trigger condition of clipping was replaced by the condition based on the trust region, so that the agent's objective function generated by optimization can guarantee monotonic improvement of the final policy performance.

PPO has been successfully used in some complex games, such as Dota 2 (Berner et al. 2019) and Glory of Kings (Ye et al. 2020). The actions of these games' agent are compound actions that consist of several sub-actions. A common treatment of compound actions in the execution of PPO is based on the probability (or probability density) of the entire action, as shown in Eq. 12. However, one problem with this approach is that for compound actions, clipping will filter out the entire action data that does not meet the requirements, even if the compound action can be adjusted in the dimensions of some sub-actions. From another point of view, it is feasible to calculate the loss separately for each sub-action. A problem with considering each sub-action individually is that it does not consider the connections between the sub-actions. We address this by introducing a simple mixing of both approaches in this paper.

In the next part of the paper, the PPO algorithm will be introduced, with its approach to clipping as the focus of our research. Then, we analyze PPO for compound action environments and present our novel PPO loss functions. We conduct experiments with these loss functions in Gym-μRTS (Huang et al. 2021) and MuJoCo (Todorov, Erez, and Tassa 2012), which are environments for discrete compound actions and continuous compound actions, respectively. In addition to the common loss function for the entire combined action, we also study the loss function of each sub-action and the mixed loss function of the two.

Proximal Policy Optimization

The mathematical basis of the main idea of TRPO and PPO is importance sampling: x_i is sampled from the policy distribution $p(x)$. However, the value of $p(x)$ is often not directly available, and must be obtained through other distributions, namely the old policy $q(x)$ obtained by indirect sampling.

$$E_{x\sim p}[p(x)f(x)] = E_{x\sim q}[\frac{p(x)}{q(x)}f(x)] \tag{1}$$

The distributions $p(x)$ and $q(x)$ should be similar in order to achieve good results. Specifically, because the new strategy

[*]These authors contributed equally.

distribution in the original formula of the strategy gradient is difficult to obtain, the old strategy is used for indirect sampling. This results in the ratio $r_t(\theta)$ of PPO, expressed as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2}$$

where π_θ is the new policy, $\pi_{\theta_{old}}$ is the sample policy, a_t is the action at the time of sampling, and s_t is the state at the time of sampling. PPO (Schulman et al. 2017) is restricted by the clipping mechanism, whereas TRPO (Schulman et al. 2015) restricts the policy by explicitly imposing constraints, and Wang et al. (Wang, He, and Tan 2020) used a new clipping method to soft clip the ratio. In general, these methods are designed to enforce the trust region. If $\pi_\theta(a_t|s_t)$ and $\pi_{\theta_{old}}(a_t|s_t)$ are very different, this is not conducive to convergence. PPO uses clipping to cull out-of-range data. If the action is a good action, the PPO restricts it from over-updating in a favorable direction (probability increases), and if it is a bad action, it restricts it from over-updating in an unfavorable direction (probability decreases), so as to limit the magnitude of the policy update.

$$L^{CLIP} = min(r_t\hat{A}_t, clip(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{3}$$

where $\hat{A}_t$ is an estimator of the advantage function at time t and ϵ is a hyperparameter. Eq. 3 can be written as follows in Eq. 4. Once the ratio exceeds the clipping range, the gradient of the update strategy will be zero. In other words, data beyond the clipping range will be filtered out.

$$L^{CLIP} = \begin{cases} (1-\epsilon)\hat{A}_t & r_t \leq 1-\epsilon \quad and \quad \hat{A}_t < 0 \\ (1+\epsilon)\hat{A}_t & r_t > 1+\epsilon \quad and \quad \hat{A}_t > 0 \\ r_t\hat{A}_t & otherwise \end{cases} \tag{4}$$

The overall loss $J(\theta)$ of PPO consists of three parts: policy loss L^{CLIP}, value loss L^{VT}, and entropy loss S_{π_θ},

$$L^{VT} = (V_\theta(s_t) - V_t^{target})^2 \tag{5}$$

$$S_{\pi_\theta} = \sum -p_a log(p_a) \tag{6}$$

$$J(\theta) = E_t[L^{CLIP} - c_1 L^{VT} + c_2 S_{\pi_\theta}] \tag{7}$$

where V is the value, c_1 is value loss coefficient, and c_2 is entropy loss coefficient.

Compound Action

In many environments, the action of an agent is a compound action that is composed of multiple sub-actions. For example, often in real-time strategy (RTS) games, the action interacting with the environment is composed of a selected unit, action type, action direction and action target. And when controlling some robots, an action consists of the movement of each joint.

The optimization objective of the policy gradient can be expressed as Eq. 8. When a certain state action pair (s, a) in the trajectory is satisfactory, that is, $q(s, a) > 0$, increase the probability of $\pi_\theta(s, a)$, and vice versa.

$$J_\theta = E_{\pi_\theta}\left[\Psi_t \log \pi_\theta(a_t|s_t)\right] \tag{8}$$

For the compound action environment, action a_t in Eq. 8 is the entire compound action. Assuming that the sub-actions of the composite action are independent of each other, then $\pi_\theta(a_t|s_t) = \prod \pi^i(a_t^i|s_t)$, where a_t^i is the ith sub-action.

$$J_\theta^i = E_{\pi_\theta}\left[\Psi_t \log \pi_\theta^i(a_t^i|s_t)\right] \tag{9}$$

Under the same independence assumption, $\log \pi_\theta(a_t|s_t) = \sum_i \log \pi^i(a_t^i|s_t)$, so

$$E_{\pi_\theta}\left[\Psi_t \sum_i \log \pi_\theta^i(a_t^i|s_t)\right] = \sum_i E_{\pi_\theta}\left[\Psi_t \log \pi_\theta^i(a_t^i|s_t)\right] \tag{10}$$

$$J_\theta = \sum_i J_\theta^i \tag{11}$$

But the situation is different for PPO. We will discuss this in detail in the following subsection.

Compound ratio

For compound actions, a natural choice is to multiply the probabilities (or probability densities) of the sub-actions to obtain the probability (or probability density) of the compound action, thus obtaining the compound ratio (Eq. 12). This is a common method for PPO implementation (Schulman et al. 2017; Huang et al. 2021; Engstrom et al. 2020; Wang, He, and Tan 2020; Wang et al. 2019).

$$r_1 = \frac{\prod \pi_{new}^i(a_t^i|s_t)}{\prod \pi_{old}^i(a_t^i|s_t)} \tag{12}$$

Without considering clipping, the optimization objective is shown in the following equation (13).

$$J_\theta = E_{\pi_\theta}\left[\hat{A}_t \frac{\prod \pi_{new}^i(a_t^i|s_t)}{\prod \pi_{old}^i(a_t^i|s_t)}\right] \tag{13}$$

Sub-action ratio

An improvement of PPO is to change the KL divergence in the policy loss function to clip on the basis of TRPO to constrain the range of the ratio and remove the data generated by the policy with too large a gap. This method requires less computation than TRPO, but it will roughly filter out data that exceeds the clip range, which reduces the efficiency of sample usage. A simple way to improve the pass rate of data is to expand the range of clipping, but this will result in unnecessary data passing through the filter. Alternatively, one can calculate the ratio for each sub-action a_t^i, as shown in Eq. 14. With the compound ratio (Eq. 12), the algorithm's optimization goal is the entire compound action, and with the sub-action ratio of Eq. 14, the algorithm will optimize each sub-action. If each sub-action is optimized then the entire compound action is also optimized. In subsequent experiments, we verified that this method is feasible. As shown in Figures 1 and 2, more samples will be unclipped with sub-action-loss.

$$r_2^i = \frac{\pi_{new}^i(a_t^i|s_t^i)}{\pi_{old}^i(a_t^i|s_t^i)} \tag{14}$$

Without considering clipping, the optimization objective is shown in the following equation (Eq. 15). Obviously, Eqns. 13 and 15 cannot be the same as Eq. 11.

$$J_\theta = E_{\pi_\theta}\left[\hat{A}_t \sum \frac{\pi^i_{new}(a^i_t|s_t)}{\pi^i_{old}(a^i_t|s_t)}\right] \tag{15}$$

In the case of clipping, for Eq. 12, if the ratio of compound action exceeds the clipping range, the entire data of this action will be ignored. If Eq. 14 is adopted, each sub-action is clipped individually, and for a compound action, part of its sub-actions are filtered out and the rest will be optimized for the strategy. As mentioned in Section 2, the effect of clipping is that if the action is a good action, PPO restricts it from over-updating in a favorable direction, and if it is an unfavorable action, it restricts it from over-updating in the unfavorable direction. Compared with Eqns. 12 and 14 provides more effective data to adjust the sub-action. In the case where there is not much data filtered out by clipping, the policy can be more quickly optimized.

Mixed loss

We propose a mixed method that can be used to reduce the impact of excessively large ratios while increasing the sample efficiency. Combining equations 3, 12 and 14, the mixed ratio represented by Eqn. 16 Eq. 17 can be obtained, where w is used to weight the two ratios. We suppose that the two ratios work similarly; while the optimal weight could be found for a particular environment, we did not deeply explore this hyper-parameter. In the experiments in this paper, the two ratios have the same weight, that in Eq. 17 with $w = 0.5$, and the same in Eq. 20.

$$r_{mix} = wr_1 + (1-w)r_2 \tag{16}$$

$$L^{CLIP} = min(r_{mix}\hat{A}_t, clip(r_{mix}, 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{17}$$

Another option is to mix two different losses as Eq. 20. In this case, the sample would be invalid only if both ratios were out of range.

$$L^{CLIP}_1 = min(r_1\hat{A}_t, clip(r_1, 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{18}$$

$$L^{CLIP}_2 = min(r_2\hat{A}_t, clip(r_2, 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{19}$$

$$L^{CLIP} = wL^{CLIP}_1 + (1-w)L^{CLIP}_2 \tag{20}$$

Algorithm 1: PPO with mixed loss

Initialize π_θ
for iteration=1, 2, . . . **do**
  run π_θ in environment, get experiences
  Compute advantage estimate $\hat{A}_t$ with the experiences
  $r_1 = \frac{\prod \pi^i_{new}(a^i_t|s_t)}{\prod \pi^i_{old}(a^i_t|s_t)}$
  $r^i_2 = \frac{\pi^i_{new}(a^i_t|s^i_t)}{\pi^i_{old}(a^i_t|s^i_t)}$
  if mix ratio loss **then**
    $r_{mix} = wr_1 + (1-w)r_2$
    $L^{CLIP} = min(r_{mix}\hat{A}_t, clip(r_{mix}, 1-\epsilon, 1+\epsilon)\hat{A}_t)$
  end if
  if mix loss **then**
    $L^{CLIP}_1 = min(r_1\hat{A}_t, clip(r_1, 1-\epsilon, 1+\epsilon)\hat{A}_t)$
    $L^{CLIP}_2 = min(r_2\hat{A}_t, clip(r_2, 1-\epsilon, 1+\epsilon)\hat{A}_t)$
    $L^{CLIP} = wL^{CLIP}_1 + (1-w)L^{CLIP}_2$
  end if
  compute gradient
  update π_θ
end for

Experiments

We validated our idea in Gym-μRTS (Huang et al. 2021) and MuJoCo (Todorov, Erez, and Tassa 2012). In our experiments, we will mainly investigate four different losses, as shown in Table 1. In Gym-μRTS, we used the winning rate of the last 100 games and the average reward of the last 100,000 steps in the training process as indicators of model performance. In MuJoCo, the total reward of each round is used as a measure of model performance.

As far as the code is concerned, if the difference between the old policy and new policy is small, then the clip of PPO will not be applied. In a common serial architecture, reusing samples improves efficiency. The smaller sample re-use time and the number of samples in small batches may result in less clipping.

Considering the work of Engstrom et al. (Engstrom et al. 2020) on PPO, we did not use additional optimizations of the standard implementation of PPO in our code. And given that in the work of Engstrom et al. (Engstrom et al. 2020), PPO without and with clipping have similar results, we conduct experiments in our implementation. In our implementation, PPO with clipping is far superior to PPO without clipping, which means that clipping is important as shown in our experiments in Figures 4 and Figures 7. The hyperparameters we used are basically the same as those in the PPO paper (Schulman et al. 2017).

We mainly use an asynchronous PPO architecture as Algorithm 2, where the sampler and trainer are two separate processes. This structure can be deployed on a large scale, where the new strategy can be updated asynchronously. Using this structure, there will be a large gap between the old strategy when sampling and the new strategy when updating and clipping will play a role in such an architecture. In the experiments in MuJoCo we also use a serial OPP same as OpenAI baseline (Dhariwal et al. 2017) for supplement as shown in Figure1. The random seed for the experiment shown in this paper is set to 0. In the experiments, we did not use any GPU. We use an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz for our experiments.

Before testing the effectiveness of our proposed method, we conducted a pre-experiment to count the number of unclipped samples with different losses to verify our ideas in Section 3. The hyper-parameters of this experiment are the same as those in Section 4.1 and 4.2. The details are shown in Figure 2. The experimental results confirm that in training, when the compound action loss is used, the number of samples that are not clipped is the least; when the sub-action loss is used, the number of samples that are not clipped is

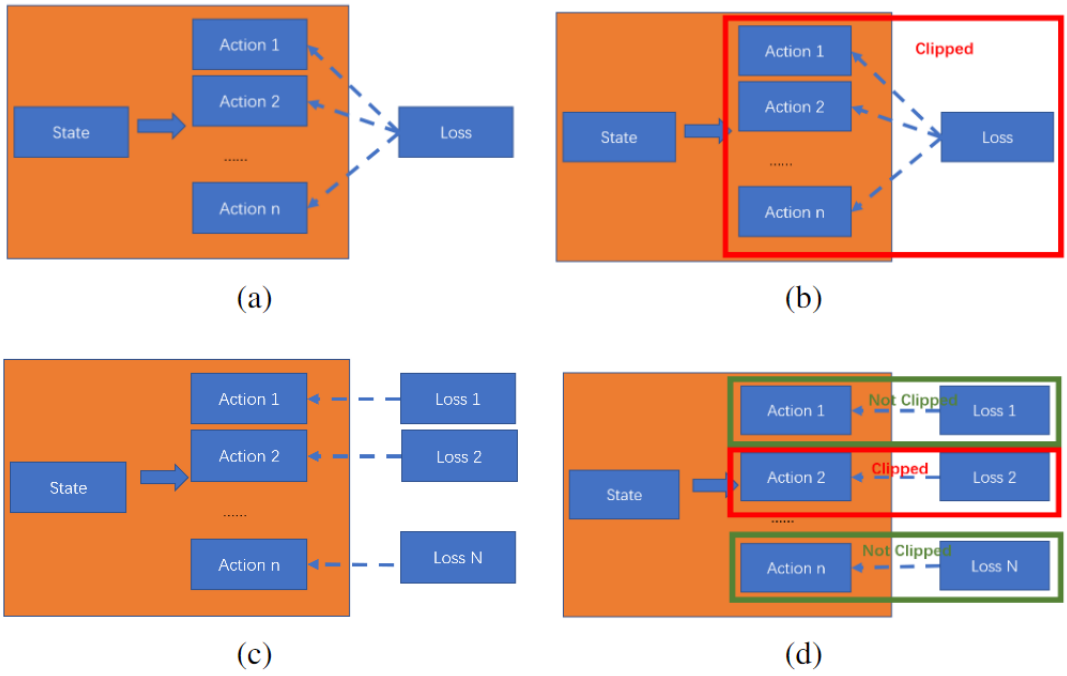


Figure 1: This figure shows the difference between the compound action loss and the sub-action loss. If the compound action loss is used, as shown in (a), there is one loss for the entire compound action to update the policy. With the sub-action loss, shown in (c), there is a loss for each sub-action in a composite action to update the policy. When using the compound action loss, if the ratio is out of range as shown in (b), the entire likelihood ratio is clipped. However with sub-action loss as shown in (d), for one sample, it is possible that part of the ratio will be clipped (red box) and the other parts will not be clipped (green boxes). This means that less clipping occurs, which improves the efficiency of sample use.

Loss	Equation	Description
compound action loss	(12)	Calculate the probability of the entire action (probability density)
sub-action loss	(14)	Calculate the probability of each sub-action (probability density)
mix ratio loss	(17)	Calculate loss using weighted average of two ratios
mix loss	(20)	Weighted average of compound action loss and sub-action loss

Table 1: List of losses studied in this paper.

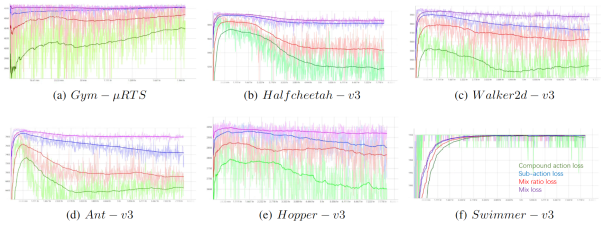


Figure 2: Number of unclipped samples in different environments during training.

larger; when the mix ratio loss is used, the number of samples that are not clipped is between two between them; and when using the mix loss, the number of samples that are not clipped is the largest, because it is a combination of the first two (the weight of two losses is 0.5, the influence of a sample will be less then in the other loss).

Experiments in MuJoCo

MuJoCo is a physics engine designed to facilitate research and development in different fields that require fast and accurate simulations. We conduct experiments in the MuJoCo environment made by OpenAI (OpenAI 2022). We use Version 1.31 of MuJoCo (distributed with an MIT License). MuJoCo's reward consists of three parts: the forward speed, action consumption and survival reward. That is, in this environment, the robot should move forward as quickly as possible and consume as little energy as possible. The proportion of the three rewards is different in different environments. The observation space and action space of the five MuJoCo environments in this experiment are shown in Table 2. Among them, the action is the torque applied to each joint of the robot, and observations include the position and speed of each part of the robot.

In the experiments of MuJoCo, the discount factor γ is 0.99, λ for the GAE is 0.95, clip ϵ is 0.2, the entropy regu-

Environment	Observation Dim	Action Dim	Type
Ant-v3	111	8	continuous
HalfCheetah-v3	17	6	continuous
Swimmer-v3	8	2	continuous
Hopper-v3	11	3	continuous
Walker2d-v3	17	6	continuous
Humanoid-v3	376	17	continuous
Gym-μRTS 10x10 map	(10,10,27)	8	discrete
Gym-μRTS 16x16 map	(16,16,27)	8	discrete

Table 2: Observation space and action space of different MuJoCo and Gym-μRTS environment

Algorithm 2: Asynchronous PPO

Sampler process:
Initialize $\pi_{sampler}$, exp buffer
for iteration=1, 2, . . . **do**
  copy neural network parameters θ from database to $\pi_{sampler}$
  run $\pi_{sampler}$ in environment, get experiences and send to database
end for
Trainer process:
Initialize π_{train}
for iteration=1, 2, . . . **do**
  get experiences from database
  Compute advantage estimate $\hat{A}_t$
  Compute gradient
  update π_{train}
  copy θ of π_{train} to database
end for

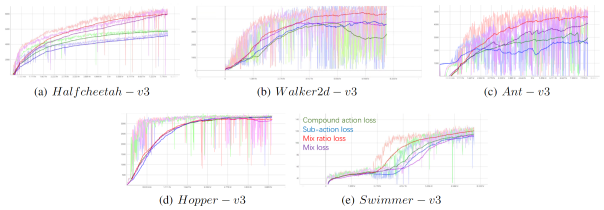


Figure 3: The performance of four losses during training in different MuJoCo environments. The mix ratio loss had the highest performance among Halfcheetah-v3, Walker2d-v3, and Ant-v3. Hopper-v3 and Swimmer-v3 have less space for action, Hopper-v3 has only 3 sub-actions and Swimmer-v3 has only 2 sub-actions. Their composite action loss and sub-action loss effects are similar, so the performance of the four losses in Hopper-3 and Swimmer-v3 is similar.

larization coefficient is 0.001, the value coefficient is 1, the learning rate of the optimizer is 2.5e-4, and the training time is 8 hours. To represent this policy, we use a 3 layer fully connected MLP with a 64 unit hidden layer, and there are only two additional noise layers after the MLP for exploration during training. We do not share parameters between policy and value function. We used asynchronous PPO2 and serial PPO1.

We compare the experimental results with the benchmark of OpenAI, as shown in Table 3. Table 3 records the average rewards of 100 episodes of four loss models after 8 hours of training in different MuJoCo environments and the benchmark of OpenAI.

As a common loss, the compound action loss has a good performance in our implementation (Figure 3 and Table3). Compared with the experimental benchmark of OpenAI, the average episode reward after training is higher. And the performance of the mix ratio loss has a higher score of more than 50% in Ant-v3 and Halfcheetah-v3 compared with the loss of compound action. Although the effect of using only the sub-action loss underperforms, the simple mixed loss combining it and the compound action loss can achieve a higher reward after training. The mix ratio loss received higher rewards in more experiments than the mix loss. In most of our experiments, the reward of the training curve using the sub-action loss is the lowest. We think this is because the sub-action loss does not consider the strategy of the whole action. The mixed loss is simply a combination of sub-action loss and compound action loss, that is, the learning rate is halved and using these two losses at the same time. The performance of this loss is also good. In the Ant-v3 and Halfcheetah-v3 environments, its reward is second only to the mix ratio loss. This experiment in different MuJoCo environments verifies the effectiveness of our proposed loss.

As shown in Equation 3, the clip coefficient is the main coefficient of the loss in PPO, so we investigate the influence of this parameter. We carried out experiments under different clipping coefficients including no clipping. Although the pass rate of samples can be increased by using sub action loss and increasing the clip range, the sub action loss will not use samples that will lead to overestimated for updating because of increasing the clip range. Then we compare different losses in Halfcheetah-v3 under different clipping coefficients. The training curve of the ratio-mix-loss has a faster growth rate and a final reward when the clipping coefficient is small (as shown in Figure 4 (a) (b)). And the sub-action loss and the mix-loss have higher rewards than the compound action loss when the clip coefficient is large (as shown in Figure 4 (c) (d) (e)). In the case of no-clip, the training curves of four different loss types are difficult to converge.

At the same time, we also use the common serial PPO al-

Loss	compound action	sub-action	mix ratio	mix loss	baseline
Ant-v3	2955[2525,3366]	2495[1711,2649]	4585[4480,5107]	4099[3702,4547]	-
HalfCheetah-v3	5166[4876,5332]	5734[5535,6051]	8065[7499,8224]	7931[7766,8479]	1668
Swimmer-v3	109[106,116]	108[106,116]	108[100, 110]	105[102,111]	111
Hopper-v3	3290[3065,3361]	3166[3024,3309]	3071[2984,3260]	3480[3338,3651]	2316
Walker2d-v3	2846[2685,2941]	3608[3402,3726]	4415[4123,4521]	3583[3386,3708]	3424

Table 3: Average rewards after training, 95% confidence intervals computed from 100 trajectory of trained model. Baseline data is from OpenAI Github web (Dhariwal et al. 2017).

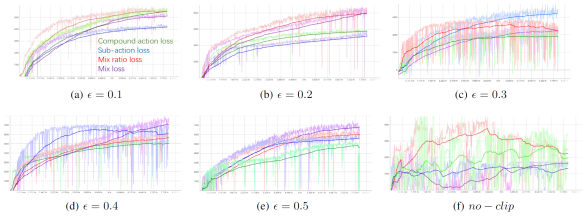


Figure 4: Performance in Halfcheetah-v3 during training with different clipping coefficient ϵ.

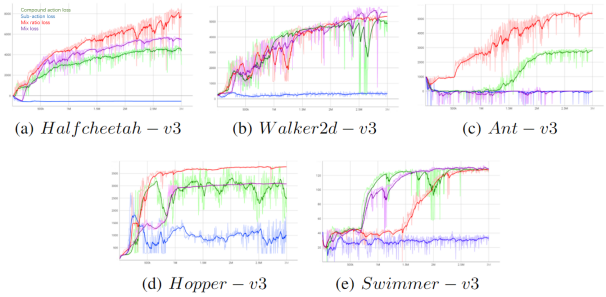


Figure 5: The performance of four losses during training in different MuJoCo environments with serial PPO algorithm.

gorithm to carry out experiments. In this experiment, we applied standard methods to achieve stable training. These include: advanced normalization, gradient clipping, value clipping, state normalization, and reward scaling. If these methods are not used, performance curve will struggle to rise or will drop during training.

In this experiment, the model sampled 3M steps. The experimental results are shown in Figure 5. Similar with the experiments with asynchronous PPO, the mix-ratio-loss still performs well with the serial PPO algorithm. It has the best performance in Halfcheetah-v3, Ant-v3 and Hopper-v3 compared with other losses. In the experiments with asynchronous PPO, the performance of the sub-action-loss and compound-action-loss are roughly the same. On the other hand, in the experiments with serial PPO, the performance curve of the sub-action-loss does not rise. The causes of the different performance of sub-action-loss in two experiment may be that in the asynchronous implementation more actors are used and there are more adequate sampling. And the mini-batch size of asynchronous PPO is larger than mini-batch size of serial PPO, which leads to more accurate gradients during updating.

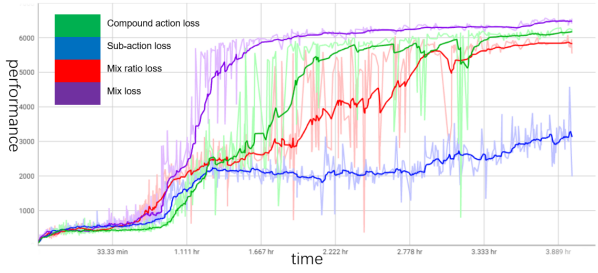


Figure 6: The training curve of four losses in Humanoid-v3.

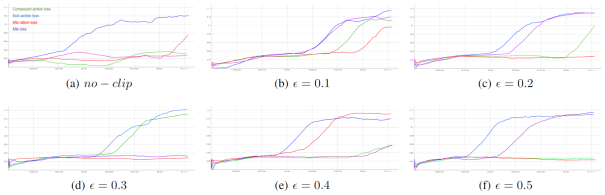
In addition, we have an experiment in Humanoid-v3, which is regarded as a complex high-dimensional continuous control problem. The result shown in Figure 6 is similar to results previously presented in this paper. And it is worth noting that sub-action-loss has significantly worse performance while mix-loss has best performance.

Based on the previous experiment, it can be seen that in the MuJoCo environment, the sub-action loss alone does not work well, while the mixed loss (mix rtaio loss and mix loss) combining the sub-action loss and compound action loss results in better model training. As mentioned above, the sub-action-loss increases the pass rate of the samples and optimizes the strategy at each sub-action level, ignoring the optimization at the action combination level, which will bias the direction of the strategy optimization. The mixed loss makes the optimization more stable and obtains better training results while increasing sample utilization.

In the next section of this paper, we will conduct experiments in the Gym-μRTS environment, to investigate the performance of the different losses in discrete action environment.

Experiments in Gym-μRTS

μRTS (Villar 2017) is a small implementation of an RTS game. Its action space and observation space are relatively small (compared to some complex environments such as StarCraft2), which allows agents to be trained with less resources and time. Gym-μRTS is the Python version of μRTS, which is based on OpenAI gym. Gym-μRTS provides different AI opponents, which can be used for testing and training. We use Version 0.3.2 of Gym-μRTS (GNU General Public License v3.0). The difference between Gym-μRTS and MuJoCo environment is that the actions in MuJoCo are continuous actions, while the actions in Gym-μRTS are dis-



(a) $no-clip$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.2$

(d) $\epsilon = 0.3$ (e) $\epsilon = 0.4$ (f) $\epsilon = 0.5$

Figure 7: This figure show the average step reward of recent 100000 steps during training with different clipping coefficients ϵ. In the Gym-μRTS environment, during the training process, the agent must explore to find a winning strategy, which will be optimized until the winning rate is 1. After the winning rate reaches 1, the winning efficiency will continue to be optimized and step reward will continue to increase slowly with the training.

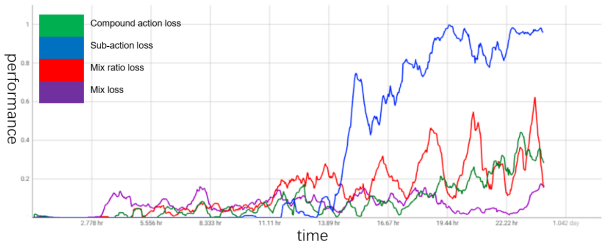


Figure 8: Winning rate of recent 1000 games against CoacAI during training in 16x16 map. Training model with sub-action loss can get nearly 100% winning rate after 24h training.

crete actions, the agent of Gym-μRTS must select a unit, decide what action to do and how to do it; MuJoCo will return reward at every step, whereas the reward of Gym-μRTS is sparse and related to game victory.

We experimented with different clipping ranges in the 10×10 map against CoacAI who is the champion of the 2020 μRTS competition. In this experiment, the discount factor γ is 0.99, λ for the GAE is 0.95, the entropy regularization coefficient is 0.01, the value coefficient is 0.5, the steps number of a experience is 512, the number of environments is 32, the number of trainers is 3, and the iterations per epoch is 1. To represent this policy, we used the same neural network as Huang et al. (Huang et al. 2021). The experimental results are shown in Figure 7.

In addition, we conducted experiments in a more complex 16x16 map. The training time of each experiment is 24 hours, with CoacAI as the opponent, and the winning rate of the last 100 games during training is used as the evaluation indicator. The experimental results are shown in Figure 8.

The experimental results in Gym-μRTS are different with the experiments in MuJoCo. Although the performance of PPO with the sub-action loss is not good in MuJoCo environments, it performs well in in Gym-μRTS, more quickly achieving a higher winning rate. When the range of clip increases, the training speed of PPO using the other three kinds of loss slows down and it is difficult to achieve a 100% winning rate within one hour of training. The reason for this result may be related to the action of the agent. In the Gym-μRTS environment, the correlation between sub actions is very high, which is different from the assumption that sub actions are independent of each other by compound action loss. In Gym-μRTS environment, when an agent executes a compound action, it first needs to select a unit, then select the action type to be executed by the unit based on the selected unit, and then determine how to execute the action. This means that it is not appropriate to calculate the joint probability of sub actions by multiplying in the compound action loss. For the sub action loss, each sub action is updated separately, which reduces the correlation between sub actions and the update of sub action strategies that should not have been updated.

Conclusion and Discussion

In conclusion, in this paper, we propose new loss functions for the PPO algorithm. We have shown that using the mixed loss of the sub-action and compound action can increase the efficiency of using sampled data, and take into account the optimization of compound action and sub-action at the same time, so as to obtain better training. Our experiments compare the training effects of the compound action loss, sub-action loss, mix ratio loss and mixed loss in Gym-μRTS and MuJoCo. In different experimental environments for continuous actions, the mix ratio loss can be used to obtain higher rewards in most cases. And in Gym-μRTS, the sub-action loss can be used to increase performance. We think it is better to increase the proportion of sub action loss in the environment with high correlation of sub actions. In our future work, we will explore more complex games. In addition, the weight of mixing of the two losses in this paper is the same. In fact, with the change of hyperparameters and the influence of the training environment, the weights should also change to achieve the optimal effect. Which loss should be weighted under what circumstances is a point worthy of further study, as is finding a better hybrid functional form instead of simply adding the weighted sum of ratio or loss. We suggest that if the correlation between sub actions is small, the weight of compound action ratio in mixed loss can be increased; otherwise, the weight of sub action ratio can be increased. Moreover, what causes the differences in the performance of different loss in different environments is also a problem worthy of in-depth study.

References

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. OpenAI Baselines. https://github.com/openai/baselines.

Engstrom, L.; Ilyas, A.; Santurkar, S.; Tsipras, D.; Janoos, F.; Rudolph, L.; and Madry, A. 2020. Implementation matters in deep policy gradients: A case study on PPO and

TRPO. *International Conference on Learning Representations*.

Huang, S.; Ontañón, S.; Bamford, C.; and Grela, L. 2021. Gym-μRTS: Toward Affordable Full Game Real-time Strategy Games Research with Deep Reinforcement Learning. In *2021 IEEE Conference on Games (CoG)*, 1–8.

OpenAI. 2022. Gym-MuJoCo. https://www.gymlibrary.ml/environments/mujoco/. Accessed: 2022.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *nternational conference on machine learning*, 1889–1897. PMLR.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 5026–5033. IEEE.

Villar, S. O. 2017. microrts. https://github.com/santiontanon/microrts. Accessed: 2017-07-13.

Wang, Y.; He, H.; and Tan, X. 2020. Truly proximal policy optimization. In *Uncertainty in Artificial Intelligence*, 113–122. PMLR.

Wang, Y.; He, H.; Tan, X.; and Gan, Y. 2019. Trust region-guided proximal policy optimization. *Advances in Neural Information Processing Systems*, 32.

Ye, D.; Liu, Z.; Sun, M.; Shi, B.; Zhao, P.; Wu, H.; Yu, H.; Yang, S.; Wu, X.; Guo, Q.; et al. 2020. Mastering complex control in moba games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 6672–6679.