1 Extended Abstract

Many Reinforcement Learning (RL) algorithms have been tested in the same or similar environment, but they can hardly generalize to unseen environments. A transfer learning contest was set up to encourage researchers to come up with solutions to the problem of generalization. This contest uses Gym Retro [1], a project providing a suite of classic video games that can be used for reinforcement learning. Specifically, the Sonic The Hedgehog™ series of games for SEGA Genesis are used in the contest, and participants will try to train agents that can play well on custom levels of the games without seeing those levels beforehand. The winning solution of this contest used a joint PPO algorithm, but it was only able to achieve less than half of the theoretical maximum score, so there is still a large margin we can improve on. In this project, we propose to add a replay buffer and train an additional reward prediction task to help facilitate the original RL objective. Moreover, we use entropy to sub-sample samples before storing those into the replay buffer so that reward predicition can learn from a "better" dataset.

Our first innovation is predicting the reward for each state with the use of a replay buffer. We want to use a neural network which has a shared encoder with the policy network but has a different fully connected layer on top to predict the reward. Several previous research have shown that model training will degrade if data is correlated so it is much better to be trained on identically independent distributed (i.i.d.) dataset. In order to generate the i.i.d. data and use the data more efficiently. We designed an replay buffer to store the previous trajectories and samples. As the model is being trained, states are pushed into the replay buffer and if the replay buffer is full, it will pop the earliest states which is first in first out (FIFO). Each time when we need to calculate the loss and update the reward network, we uniformly sample a batch of samples (observations and rewards) from the replay buffer. We add the reward loss to the original loss function to facilitate the original RL training. However, we add an loss scale for this reward loss, representing the importance of the reward loss. We conducted experiments with scale = 1, 0,5, 0,2 (Figure 3) and compare them with the baseline. When the scale = 1, the model performance decreases quickly at the beginning and cannot reach the same reward as the baseline. For scale = 0.5 and 0.2, the model outperforms the baseline. Experiment with scale = 0.5 has the highest reward. Moreover, with replay buffer based reward prediction, at first, the replay buffer does not has enough previous experience/ state, so the training reward decrease quickly. Then, after the replay buffer is filled and has enough randomness, reward starts increasing smoothly.

With the replay buffer added, we want to further optimize our model. It is common that people tend to do random sampling when dealing with replay buffer in RL, but we would like to explore another approach other than random selection, Entropy-Based Sub-Sampling. By taking the combination of 50% easiest states and 5% hardest states, where easy states have low entropy loss and hard states have high entropy loss, we achieved significant performance improvement from the baseline model (Figure 4). Before feeding the current trajectory into the replay buffer, we would like to perform entropy loss based sub-sampling on all of the states in the current trajectory. In this case, we are treating the entire trajectory as a whole dataset, and each sample in the dataset is a state along the trajectory, and we are calculating the state-level entropy value. Due to the significant cost of computing entropy values from Tensorflow discrete Categorical Distribution logits, we are calculating entropy values in mini-batches rather than one by one, and then we append each state with its corresponding entropy value. With every state associated with entropy loss, we then take the 50% of states with the lowest entropy loss values and 5% of states with the highest entropy loss values among the trajectory.

For the performance comparison with baseline model (Figure 4), at the beginning of the time steps, the performance of this Entropy Loss Based Sub-Sampling method dropped dramatically compared with the performance of baseline model. Initially, the replay buffer of this method was not full and it contained hard samples, which made the states in our replay buffer difficult to fit our model. However, around time steps 60, the performance of this method catched up with the performance of baseline as the states in the replay buffer were more representative to the trajectories. After time step 100, the performance of this method improved dramatically, and quickly outperformed the baseline because of the more representative states in the replay buffer.

With the combination of the two methods, we are able to achieve a better mean episode reward than the baseline of joint PPO, and we believe this method can be transferred to other tasks and domains.

Auxiliary Objective and Entropy in Learning Sonic Game Series

Zeyang Bao
University of California, Berkeley
`zeyang_bao@berkeley.edu`

Russell Wang
University of California, Berkeley
`russell.wang@berkeley.edu`

Zuang Yu
University of California, Berkeley
`zuang99@berkeley.edu`

Abstract

Many Reinforcement Learning (RL) algorithms are tested in the same environment as the training set, but they can hardly generalize to unseen environments. A transfer learning contest that uses Sonic The Hedgehog™ series of games was set up to encourage researchers to come up with solutions to the problem of generalization. The winning solution of this contest used a joint PPO algorithm, but it was only able to achieve less than half of the theoretical maximum score, so there is still a large margin we can improve on. In this paper, we propose two methods that build on top of the joint PPO algorithm. First, we train a reward prediction task as an auxiliary objective that can facilitate the original RL objective. This reward prediction task trains on data sampled from a replay buffer that stores samples/trajectories from previous policies. The second thing we propose is to further sub-sample 55% of the trajectory before being stored in the replay buffer using entropy, with low entropy data being easy samples and high entropy data being hard examples. With the two methods, we are able to outperform the baseline of joint PPO by a large margin.

2 Introduction

Many Reinforcement Learning (RL) algorithms have been tested in the same or similar environment, but they can hardly generalize to unseen environments. The next step for RL is to leverage seen environments to quickly adapt to new tasks and situations. Pursuing this next step, a transfer learning contest was set up to encourage researchers to come up with solutions to the problem of generalization. This contest uses Gym Retro [1], a project providing a suite of classic video games that can be used for reinforcement learning. Specifically, the Sonic The Hedgehog™ series of games for SEGA Genesis are used in the contest, and participants will try to train agents that can play well on custom levels of the games without seeing those levels beforehand. The winning solution of this contest used a joint PPO algorithm, but it was only able to achieve less than half of the theoretical maximum score, so there is still a large margin we can improve on.

In this project, we present a method predicting the reward for each state, sampled from a replay buffer that stores the previous experience, and add the prediction loss to original loss function to help the model training. We use a neural network to approximate the reward and trained the network by supervised learning against the ground truth reward. In order to facilitate the original RL policy training, we designed the architecture to have a shared encoder network between the policy network and reward prediction network.

To further improve the reward prediction task, we also implement an entropy filtering method to sub-sample states in the replay buffer. It is common that people do random sampling from replay

Fall 2021 CS285 Project, please see https://github.com/rwang97/cs285-project for the implementations.

buffer in RL, but we would like to explore some approaches other than random selection. We will use state-level entropy value to sub-sample some states and in particular, we only store a mix of 50% easy and 5% hard samples for each trajectory to the replay buffer. By adding both auxiliary objective and trajectory sub-sampling with entropy, we are able to surpass the baseline of using only joint PPO with a large margin. We believe that having explicit additional information via reward prediction and entropy about the environment dynamics can indeed help the reinforcement learning model to learn.

3 Related Work

Proximal Policy Optimization (PPO). There are in general two lines of work when it comes to reinforcement learning tasks: value-based algorithms such as DQN [2] and policy-based algorithms such as PPO [8]. In this project, we mainly look at PPO, which was set as the baseline approach for the contest. PPO [8] is a widely used policy gradient method, which builds on trust region policy optimization (TRPO) [7], and proposed a "surrogate" objective function. The resulting optimization objective is simple and easy to implement and enables multiple epochs of minibatch updates.

Replay Buffer & Auxiliary Objectives. In reinforcement learning, on-policy learning can be extremely inefficient, since each gradient step requires fresh samples and we cannot fully utilize parallel training. This is how off-policy methods come into play. Off-policy learning uses replay buffers that store trajectories that were executed by previous policies in an environment. During training, the buffer will be queried for a subset of a subset of samples (states, actions, etc.)or trajectories to replay the agent's experience.

Besides being used to train the policy itself, replay buffer can also be used to learn auxiliary objectives such as reward prediction, dynamic verification and reconstruction, as proposed in [9] and [3]. The environment dynamics is learned through these tasks, as the reply buffer contains a much broader spectrum of the observation and action space. These auxiliary tasks share a common encoder as the primary actor-critic network so that the knowledge can be transferred. Both RL objective and Auxiliary objectives are jointly optimized, so the dynamics knowledge can further help the RL objective during training.

Entropy Sub-sampling. In the classical neural architecture search (NAS) [4], one of the most common approaches to accelerate the process of searching optimal hyperparameter is to perform random sub-sampling on the whole dataset, to overcome the huge demand for computing resources. While another popular approach to accelerate the process of hyperparameter optimization (HPO) is to perform the entropy based importance sub-sampling by only taking the easy samples with low entropy values. However, by only taking the easy samples with low entropy values, the resulting sub-samples are only composed of easy samples with lower chaos degree among the whole dataset, which is not representative to the whole dataset. Indeed, recently there is a new method to solve the problem of unrepresentative sub-samples to the whole dataset, by taking samples with both lowest and highest entropy values [4]. Across different whole dataset, different total sampling size, and different easy-hard sampling portion, the newly proposed method outperforms the traditional random search and entropy based sampling.

4 Retro-Sonic Environment

Gym Retro. In order to run sonic games and train our agent, we make use of the Gym Retro [1] project. The project is aimed at creating emulated environments for reinforcement learning tasks.

Sonic Games. Gym Retro includes the Sonic The Hedgehog™ series, which we use for this project. There are a total of 47 training levels and 11 test levels in the Sonic The Hedgehog™ series. Each level is a tuple of *(ROM, zone, act)*. ROM is the data and code that make up a game. Each zone has a unique set of textures and objects, and it is further divided into acts, where different acts within a zone share textures and objects, but they tend to be different in terms of spatial layout.

(a) SpringYardZone.Act3 (b) GreenHillZone.Act3 (c) MarbleZone.Act2

Figure 1: Three example levels in Sonic games

An env step function will produce an observation which is a RGB image with a dimension of 320 by 224. An agent can produce actions that are a combination of buttons in the environment. The action space consists of the following buttons: B, A, MODE, START, UP, DOWN, LEFT, RIGHT, C, Y, X, Z. The actual combination of the buttons make up the action space, for example: {[LEFT], [RIGHT], [LEFT, DOWN], [RIGHT, DOWN], []}. The reward is associated with two components: the horizontal offset of the agent and a completion bonus. For the horizontal offset, going right always produces positive reward and going left will give a negative reward. The completion bonus will be added when the agent reaches the end of the level.

Environment Setup. We installed the required Gym-Retro package in order to import and run the ROMs of the games. The games have to be purchased through Steam platform and then can be downloaded and run by the package. We perform all these actions and conduct training on Google Cloud Platform, where we rented a virtual machine with a Tesla T4 GPU and several CPUs. We set our baseline to be the transfer learning contest winner and set up their code environment too. Specifically, it requires *tensorflow-1.4* for training and inference and *openai/baselines* [6] libirary for some additional operations.

5 Methodology

5.1 Baseline

We chose the winner of the transfer learning contest as our baseline. The winning solution modifies the joint PPO [5], a policy gradient algorithm to learn optimal polices. The joint training simply means that they train a single policy to play every level in the training levels. Parallel workers are used and assigned a training level, and at every gradient step, the gradients of all the workers are averaged to ensure policy is trained evenly on all the training levels. After the policy converges, it is fine-tuned on the test levels. This can be seen as a meta-learning approach, and it works effectively in the contest. Besides the algorithm, they made a few modifications such as using an augmented action space and augmented reward function. Specifically, they used a more common button combination as the action space: {[LEFT], [RIGHT], [LEFT, DOWN], [RIGHT, DOWN], [DOWN], [DOWN, B], [B], [], [LEFT, B], [RIGHT, B]}. For augmented reward, they encourage the agent to visit new states.

The pretrained model on all training levels is used as the starting point, and we fine-tune it on a specific test level as our baseline model. To be consistent, the following reward prediction and entropy sub-sampling method all use the same starting model and the same test level. All other settings are also kept the same including the observation space, the action space and the reward.

5.2 Replay Buffer

In order to approximate a complex and nonlinear reward approximate function $R(ob)$, we need to sample the independent and identically distributed (i.i.d.) data. However, the samples in the same trajectory can be highly correlated. In order to get rid of the risk of correlation, we apply a replay buffer to store and sample the saved observations and corresponding rewards. Specifically, the replay buffer has a fixed size and as more samples are pushed into the buffer, the earliest data will be removed. The data stored in the replay buffer is in the form of $(ob_t, a_t, ob_{t+1}, done, reward, entropy)$. ob_t is the resized image of a Sonic Game raw frame, and a_t is the action. During training, we can query the replay buffer randomly to get a batch of data. Replay buffer increases data efficiency through re-use of data from earlier policies. More significantly, it reduces variance as uniform sampling from it reduces the correlation among the samples used in the following updates.

5.3 Reward Prediction

In order to approximate the reward $R(ob)$, we use a neural network R-network: $R(ob; \theta)$ with parameter θ. The network shares the parameters with the policy network except the last fully connected layer. To estimate the network, we optimize the following sequence of loss functions at iteration i:

$$L_{reward}(\theta_i) = \mathbb{E}\left[\left((y_i - R(ob; \theta_i)\right)^2\right]$$

During the learning stage, we will uniformly sample a mini-batch of $(ob_t, a_t, ob_{t+1}, done, reward, entropy)$ from the replay buffer and particularly use (ob_t) and $reward$ for reward prediction.

$$L_{reward}(\theta_i) = \mathbb{E}_{(ob,a,ob',r)\sim\mathbb{U}(B)}\left[\left((y_i - R(ob; \theta_i)\right)^2\right]$$

where y_i is the reward of the batch of tuples we sample from the replay buffer, which is the ground truth and the ob_i is the observations from the batch of tuples we sample in this iteration i. Also, we do not want the reward value loss degrades the policy network training, we add a scaling parameter λ for L_{reward}. Then the total loss after we add the reward prediction is:

$$L = L_{PPO} + \lambda * L_{reward}$$

where L_{PPO} is the standard PPO Loss that includes the Clip Loss, Value Loss and an Entropy bonus.

5.4 Sub-sampling with Entropy

Before feeding the current trajectory into the replay buffer, we would like to perform entropy based subsampling on all of the states in the current trajectory. In this case, we are treating the entire trajectory as a whole dataset, and each sample in the dataset is a state along the trajectory. Similar to Neural Architecture Search (NAS), we are calculating the state-level entropy value. Due to the significant cost of computing entropy values from Tensorflow discrete Categorical Distribution logits, we are calculating entropy values in mini-batches rather than one by one. After calculating all of the state-level entropy values, we create an object for each state in the format of $(ob_t, a_t, ob_{t+1}, done, reward, entropy)$. Then, we form a trajectory by appending all of objects for states into an empty list. With the trajectory as a "whole dataset", a list of states associated with entropy values, we store 55% of the states in the current trajectory into our replay buffer. More specifically, we perform two set of experiments for our Entropy Based Sub-sampling. One of the experiments takes all of the easy sample/state to our replay buffer, with the low entropy value associated to it. While the other one takes 50% of the easy sample/state and 5% of the hard sample/state to our replay buffer, where easy samples/states have low entropy values and hard samples/states have high entropy values.

6 Evaluation

6.1 Baseline

The baseline model is trained on all 47 training levels and fine-tuned on *HydrocityZone* and *Act 1* of the zone. The training is done by using a joint PPO as in section 5. The mean episode reward plot shows that it goes from a reward of around 2950 to somewhere around 3600. The learning curve is a little bit noisy but in general it has an increasing trend.

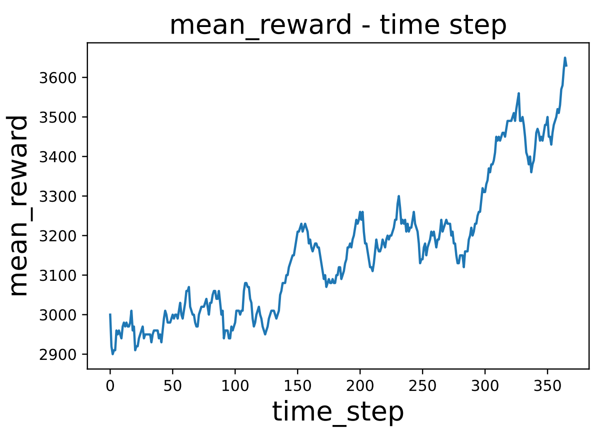

Figure 2: Result of the Baseline Model

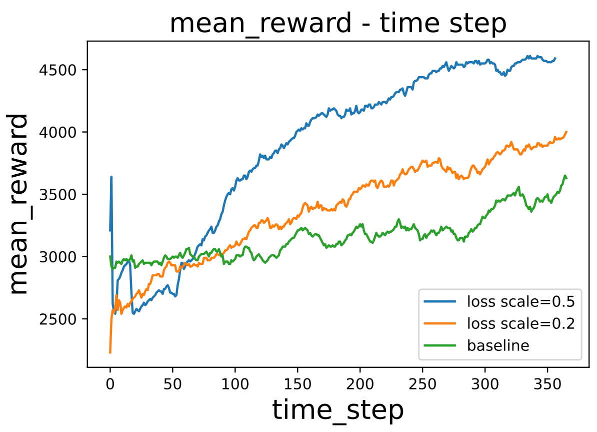

Figure 3: Result of reward prediction loss scale

6.2 Reward Prediction

The reward prediction is implemented based on the baseline. We implemented a replay buffer and a reward prediction neural network. Our replay buffer size is 30000 and sampling size is 5000. We trained with a loss scale of 1, 0.5 and 0.2.

Generally, from the result plot (Figure 3), reward loss helps the model converge to a higher reward. However, if the scale is very large, for example 1, although we did not finish training, early results show that the model cannot reach the same level of reward as baseline. loss scale is 1 means the reward loss has the same weight as the PPO loss. During the back-propagation, the gradient of reward loss has too large weight and the model will update to the direction decreasing the reward loss more instead of the PPO loss and that lead the training to degrade. Reward loss scale = 0.5 performs better compared to the reward loss scale = 0.2.

6.3 Subsampling with Entropy

For the part of Entropy Method based sub-sampling, we performed two sets of experiments for the comparison of "Mean Reward" with the baseline model (without replay buffer). The first experiment takes all of the easy samples, with low entropy values. The other experiment is to take 50% of the easy samples and 5% of the hard samples, with low and high entropy values respectively.

55% Easy Samples Entropy Sub-Sampling (Figure 4). At the beginning of the time steps, the replay buffer is not filled with states. Since the replay buffer size that we set is 30,000 and for each

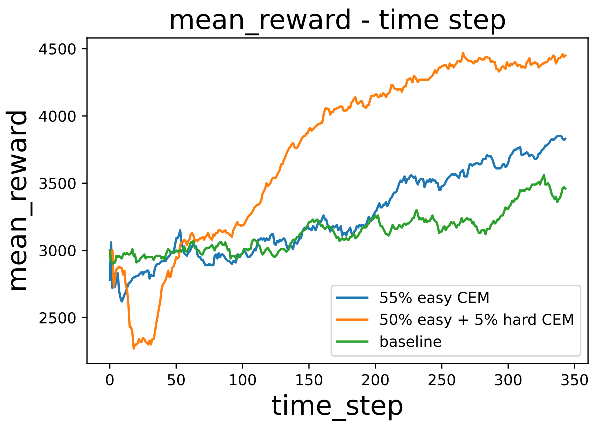

Figure 4: Result of Entropy Based Sub-Sampling

time step we only feed around 4,000 samples with low entropy values into the replay buffer, the replay buffer will be filled after a few time steps since the start. Therefore, at the very beginning of the time steps, the performance of full easy entropy based sub-sampling dropped and was worse than the baseline model performance in terms of mean reward because of not yet filled replay buffer. After 50 steps, the performance of full easy entropy based sub-sampling caught up with the performance of the baseline. From time steps 50 to 200, the performance of full easy entropy based sub-sampling and the performance of baseline are pretty similar. After time step 200, the full easy entropy sampling method started outperforming the baseline model, with enough collection of critical easy states/samples.

50% Easy and 5% Hard Samples Entropy Sub-Sampling (Figure 4). At the beginning of the time steps, the performance of this method dropped dramatically compared with the performance of the baseline and the full easy method. Initially, the replay buffer of this method was not full and it contained hard samples compared with full easy method, which made the replay buffer difficult to fit our model. However, around time step 60, the performance of this method caught up with the performance of baseline and full easy method as the states in the replay buffer were more representative to the previous trajectories. After time step 100, the performance of this method improved dramatically, and quickly outperformed the baseline and full easy method because of the more representative samples in the replay buffer.

7 Conclusion

Motivated by closing the gap between joint PPO and theoretical maximum score on the Sonic The Hedgehog™ game series, we propose training an auxiliary task and use trajectory filtering method to narrow down the gap. For the auxiliary task, we use reward prediction that predicts the actual reward given by the environment at every time step. A neural network is used to perform reward prediction, and we use a shared encoder network as the primary RL objective so that the knowledge can be transferred during training. Instead of learning from current states and current policies, reward prediction is learned by utilizing a replay buffer that stores trajectories executed by previous and current policies, because replay buffer covers a wider range of situations. To further improve the task of reward prediction, we propose to use entropy to sub-sample easy and hard samples and add those into the replay buffer so the model can learn from a better sampled dataset. With the combination of the two methods, we are able to achieve a better mean episode reward than the baseline of joint PPO, and we believe this method can be transferred to other tasks and domains.

References

[1] Gym retro. https://github.com/openai/retro, 2017.

[2] Hado Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[3] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks, 2016.

[4] Byunggook Na, Jisoo Mok, Hyeokjun Choe, and Sungroh Yoon. Accelerating neural architecture search via proxy data. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2848–2854. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[5] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl, 2018.

[6] Openai. Openai/baselines: Openai baselines: High-quality implementations of reinforcement learning algorithms.

[7] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[9] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *CoRR*, abs/1612.07307, 2016.