A Comparison of Policy Gradient Methods for Multitask Learning

Chris Nalty
University of Maryland
cnalty@umd.edu

Abstract

This paper compares two policy gradient methods using multitask learning (MTL) on the Atari visual environments. These environments are complex and take millions of time steps to learn. This paper investigates Advantage Actor-Critic (A2C) and Proximal Policy Optimization's (PPO) performance on one, two and four tasks from the Arcade Learning Environment. The results show that agents trained with both PPO and A2C have improved performance when trained on multiple tasks when compared to a single task. PPO showed the most consistent improvement and scored the best overall. However, A2C's improved the most on average compared to its baseline. This shows that the trust-region approximation of PPO may not be as beneficial in MTL as in a single task.

1. Introduction

Reinforcement learning (RL) is the field of machine learning where an agent is trained to perform one or multiple tasks in a given environment. Recent RL research has focused on neural network function approximators to learn complex tasks. The two main types of methods are Q-Learning [1] and policy gradients, including Advantage Actor-Critic (A2C) [2] and Proximal Policy Optimization (PPO) [3]. This paper will be comparing the performance of these two policy gradient methods, A2C [2] and PPO [3], on multitask learning (MTL) performance.

The goal of MTL is to learn multiple tasks simultaneously using partially shared network parameters between tasks. Generally the tasks share some underlying features. This allows the network to generalize upon these features and use them in multiple tasks [4]. In this paper there are four tasks considered taken from the Arcade Learning Environment (ALE) [5].

These environments are four games from the Atari 2600 system: Breakout, Pong, Space Invaders, and Q*bert. These environments all provide visual inputs to the agent and have discrete action spaces of varying sizes. Three of these environments are chosen since they share similar input styles and goals. Breakout, Pong, and Space Invaders all share a similar game style,only allowing one dimensional control by the player. Additionally they all involve either catching or dodging projectiles coming perpendicular to the player's movement. The goals do vary slightly between the games, in Breakout and Pong the goal is to bounce the projectile back towards the other side of the screen, where in Space Invaders the goal is to launch projectiles at enemies and avoid incoming projectiles. Q*bert was chosen as a challenge environment to the agents, since it varies significantly from the other three tasks. In this game the agent can move in many directions and has to avoid enemies that can move sporadically unlike the projectiles in the other games.

Since these games in general share many underlying features and concepts, they are ideal candidates for MTL. This comparison will test if PPO [3], a trust region approximation, is able to learn stronger in a more general environment than A2C [2] a basic policy gradient method. We hypothesis that PPO [3] will be able to benefit from MTL more than A2C[2], due to its trust region approximation. We expect without this, A2C[2] will suffer from training instability and catastrophic forgetting.

2. Background

The reinforcement learning problem formulated here will follow the traditional methods and notation. We represent the environments as Markov Decision Processes (MDPs) with a set of states, S, actions, A, and a finite set of rewards, $R \subset \mathbb{R}$. At each time step, t, the agent is given some state $s_t \epsilon S$, and choses some action $a_t \epsilon A$. After taking this action in the environment the agent is given some reward, $r_t \epsilon R$. This moves the environment to a new state, $s_{t+1} \epsilon S$. The values s_{t+1} and r_t come from discrete probability distributions based on s_t and a_t, P(S) and P(R) respectively

[6]. All of these elements, along with a probability distribution for the initial state, p_0, form the MDP {S, A, R, P(S), p_0, P(R)}.

Policy Gradient methods are the category of RL algorithms that estimate the gradient for a policy function approximator, $\pi_\theta(a_t \mid s_t)$, given weights θ. The policy takes in a state, s_t, and outputs an action probability distribution that is then used to select what action to take on the environment. The gradient estimated by these methods is used to update π_θ with any gradient descent method [3, 7], typically Adam [8] or RMSProp [9] are used. These methods work on-policy, meaning they alternate between stepping through a Markov Decision Process (MDP) gathering these steps into a batch and performing gradient descent with the transitions gathered. After gradient descent on these transitions is performed, which may involve multiple epochs, these transitions are discarded before gathering new ones. The values saved at each timestep, t, include the state, s_t, the action taken, a_t, the reward received, r_t, and the log probability, $\log\pi_\theta(a_t \mid s_t)$, of the action taken. In some cases additional information is stored, such as if the MDP ended after a given step [7].

A2C [2] and PPO [3] are both actor-critic [10] methods. This is a style of policy that is split into two portions, an actor that decides the actions to take and a critic that estimates the value expectancy, V_t , of each state. In this paper the actor and critic share all parameters except the final output layer. The values from the critic along with the rewards are used to calculate an Advantage estimate, A_t. This is an estimate for how much 'advantage' there is to choose an action over the estimates the critic is giving. A2C [2] and PPO [3] use different methods for calculating advantage, which will be discussed in the following sections. The advantage along with the log probabilities of chosen actions are used to calculate the loss for the actor. In both algorithms the loss for the critic is given as follows,

$$L_{critic}(\theta) = (R_t - V_t)^2$$

R_t is known as the return, and is given by the following equation,

$$R_t = A_t + V_t$$

Although it appears that the critic's loss could be simplified to be the A_t^2, the calculation for the Advantage, A_t may not include V_t , thus this additional step is needed to propagate the gradients properly. Getting an accurate value estimation is necessary for these methods to compute the advantage which will be used in the loss for the actors. Since these methods do not finish an entire episode before performing gradient descent this allows an estimation for the agent's performance over the remainder of the episode to train the actor.

2.1 Advantage Actor-Critic

Advantage Actor-Critic (A2C) [2] is one of the most basic policy gradient algorithms that is able to perform reasonably on tasks such as the ones in ALE [5]. It only uses the basic log probability, $\log\pi_\theta(a_t \mid s_t)$, for estimating the policy gradient. In A2C [2] the Advantage is calculated as follows,

$$A_t = \left(\sum_{t=0}^{H} \gamma^t r_t\right) - V_t$$

Where H is the minimum of the distance from time t to the horizon or the time until the episode ends. If the episode has not ended at the horizon r_H is taken as V_H, the value given by the critic at state s_H. γ is a hyperparameter used to discount future rewards. Using this advantage estimate we formulate the loss of the actor as

$$L_{actor}(\theta) = -\log\pi_\theta(a_t \mid s_t)\, A_t$$

In addition to the loss from the actor and the critic there is also an added loss for the entropy of the network's action selection. This loss is given by the negative entropy of the probability distribution output by the actor. To balance out these gradients we have coefficients, c_1 and c_2, to shirk the loss of the critic and the entropy. In total we have the following loss for the entire network

$$L_{total}(\theta) = L_{actor}(\theta) + c_1 L_{critic}(\theta) - c_2\, Entropy_t$$

To optimize this gradient, RMSProp [9] is used as in the original A3C [2] paper. In addition A2C [2] uses sixteen environments running in parallel. States from each environment are combined at the end of each horizon and batched together for training. For a complete list of hyperparameters see **Appendix A**.

2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [3] was a method developed in response to trust region methods such as Trust Region Policy Optimization (TRPO) [11]. TRPO [11] and other trust region methods were designed to remedy a major problem in policy gradient methods, catastrophic forgetting. This is where a policy's performance sharply decreases after a single update due to an update step being too large. TRPO [11] solved this by introducing a trust region to the gradient updates, which searches a small area near the current policy for an update that will improve the current policy's performance. However this method is difficult to implement and involves either taking second derivatives, or estimating them, which can slow down training speed. PPO [3] attempts to simulate a trust region without the need to take second derivatives. This leads to a quicker and easier to implement algorithm with little to no loss in performance compared to TRPO [11]. This estimation is done by clipping the range of the loss coefficient instead of constraining the loss to a trust region. Instead of using the log probabilities of actions TRPO [12] and PPO [3] use a ratio, $r_t(\theta)$, of the log probabilities of an updated policy to the old policy's log probabilities. That is,

$$r_t(\theta) = \frac{{}_{\theta}(a_t| s_t)}{{}_{\theta old}(a_t| s_t)}$$

Thus $rt(\theta_{old}) = 1$ [3]. PPO [3] uses multiple epochs per update, including using mini batches of the gathered trajectories. The clipped loss for the PPO [3] actor is given as follows

$$L_{clip}(\theta) = E_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)]$$

Where ϵ is a hyperparameter that determines the maximum and minimum clip range for $r_t(\theta)$. The optimizer used in PPO is the Adam [8] optimizer, as used in the original PPO [3] implementation. Similarly to A2C [2], PPO [3] runs eight environments in parallel during training.

PPO [3] uses Generalized Advantage Estimation (GAE) [12] to calculate its advantages. GAE [12] uses a running sum of delta values to calculate advantage given by,

$$\delta_t = r_t + \gamma V_{t+1} - V_t$$

Where r_t is the reward at time t, γ is a hyperparameter, and V_t is the value given by the critic at time t. This delta value is then used to fully calculate the GAE as follows,

$$A_t = \sum_{k=0}^{H-t} (\lambda\gamma)^k \delta_{t+k}$$

λ is another hyperparameter, similar to γ, used to discount the weight given to future rewards. PPO [3] adds together the losses in the same manner as A2C [2] with an added bonus for the entropy of the network.

$$L_{total}(\theta) = L_{actor}(\theta) + c_1 L_{critic}(\theta) - c_2\, Entropy_t$$

A complete list of hyperparameters for PPO [3] is also located in **Appendix A**.

3. Experiments

In this paper we perform four experiments to compare the results to the A2C [2] and PPO [3] with MTL versus without. We train on four atari games from ALE [5]: Pong, Breakout, Space Invaders and Q*bert. For the baseline comparison an actor is trained on each game for 10 million frames. The network used in all of the tasks is based on the original neural network from Mnih et al.'s Deep Q-Learning paper [1], pictured in **Figure 1**. For the MTL experiments the network is altered to have a unique set of linear layers for each task and shares all the convolutional layers between all tasks. The goal with this design is for the agent to learn a shared visual representation, while maintaining the ability to have fine tuned control for each individual task.

For both methods, A2C [2] and PPO [3], we perform symmetric experiments. Firstly, we run with Pong and Breakout as the two tasks running for 20 million steps total, 10 million on both environments. Second, we train all four environments for 40 million steps total, again with 10 million on each environment. For these experiments we take advantage of the fact these training methods already train on multiple copies of the environment at once. We use the same number of environments in total, and evenly fill the environment vector with each task. For example, PPO [3] uses a vector of eight parallel environments, thus in the two task experiments there will be four Pong environments and four Breakout environments running in parallel. The agent is told which environments in the vector belong to what tasks before execution. All hyperparameters are kept the same for the MTL training and each experiment was run for three trials with the results averaged over the trials.

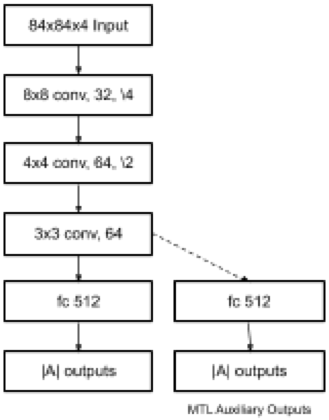


Figure 1: Network architecture. Convolutional layers denoted by kernel size, number of filters and stride

4. Results

There are two main points of discussion for this results section. Firstly, we explore how well the MTL methods compare to the baselines after 10 million training steps in total. This explores if MTL allows the agents to learn policies at a faster rate than normal, since this is the amount of timesteps that it would normally take to learn one task. Secondly to compare learning multiple games with a shared representation on the same number of time steps as it would take that baseline to learn each of these games individually, that is 10 million time steps per game.

4.1 Advantage Actor-Critic

At the end of the full 10 million timesteps per task execution A2C [2] improved significantly with MTL over a single task. It did on average 1.04 times as well with two tasks and 1.17 times as well with four tasks compared to a single task execution. Both tasks in the two task run had improved performance compared to a single task. However, when trained on four tasks two of the tasks improved performance while two decreased. Most of the increase in overall performance was attributed to the Q*bert game, with a 1.75 times increase in score using four tasks. Q*bert is the most unique gameplay of those chosen. This deviates significantly from our expected results. We expected this game would cause trouble for the agent to learn, but seemed to have the highest benefit from MTL. For the 10 million total frame executions, two tasks did only 0.85 times as well and four tasks did only 0.26 times as well as a single task. These are significant decreases in performance, showing A2C is not able to learn quicker with multiple tasks, and it is infact a serious detriment to performance over the same amount of total training as a single task. Full final results can be found in **Figure 2** below.

Game	1 Task	2 Tasks - 20m	2 Tasks - 10m	4 Tasks - 40m	4 Tasks - 10m
Pong*	18.02	18.42 (101.0%)	17.23 (98.0%)	18.53 (101.3%)	-14.68 (16.2%)
Breakout	368.99	399.70 (108.3%)	264.80 (71.8%)	348.80 (94.5%)	59.57 (16.1%)
Space Invaders	686.32	-	-	676.53 (98.6%)	354.6 (51.7%)
Q*bert	5961.83	-	-	10448.92 (175.3%)	1283.0 (21.5%)
Avg Percent	100%	104.7%	84.9%	117.4%	26.4%

Figure 2: Chart of scores on last 100 timesteps. Percentages compare performance to single task agents. * note when pong score percentages are calculated 21 is added to the score to account for possible negative values

4.2 Proximal Policy Optimization

At the end of execution PPO [3] did on average 1.06 times as well with two tasks and 1.09 times as well with four tasks when compared to a single task. This shows a significant improvement when training on multiple tasks over a single one using 10 million timesteps per game. With two tasks the agent performed nearly as well on Pong and improved significantly on Breakout. On the four task executions the agent was able to improve its performance on all four games. However, on only 10 million total steps, the multi-task agents extremely underperformed only doing

0.82 times as well with two tasks and half as well on average with four tasks. This shows PPO [3] is able to successfully extract data from multiple tasks to improve overall performance, however it is not able to learn multiple games significantly quicker than they would be learned by individual networks. Full results can be found in **Figure 3** below.

Game	1 Task	2 Tasks - 20m	2 Tasks - 10m	4 Tasks - 40m	4 Tasks - 10m
Pong*	20.01	19.77 (99.4%)	17.96 (95.0%)	20.30 (100.7%)	11.71 (79.8%)
Breakout	297.84	340.04 (114.2%)	206.24 (69.2%)	349.76 (117.4%)	57.78 (19.4%)
Space Invaders	826.25	-	-	865.03 (104.7%)	484.95(58.7%)
Q*bert	12891.75	-	-	15008.67 (116.4%)	5467.67 (42.4%)
Avg Percent	100%	106.8%	82.1%	109.8%	50.1%

Figure 3: Chart of scores on last 100. Percentages compare performance to single task agents. * note when pong score percentages are calculated 21 is added to the score to account for possible negative values.

4.3 PPO to A2C Comparison

Overall PPO [3] generally performed better than A2C [2] during MTL as shown by **Figure 4**. Breakout was the only task where A2C [2] was able to outperform PPO [3]. It was also found that training on more tasks increased performance in general. PPO [3] trained on 4 tasks performed the best on every game, except for Breakout, which performed the best with A2C [2] using 2 tasks. A2C [2] had the largest percentage improvement for MTL agents compared to its baseline agent during its four task run, with a 1.17 times increase. This was better than PPO [3] with just under a 1.10 times increase using four tasks. It appears that A2C [2] does not suffer from catastrophic forgetting despite the lack of trust region approximation to stabilize training. A2C [2] performs worse on two of the games in the four task run compared to the baseline. However, it is not a significant decrease in performance. Full training graphs for all runs on each mode are shown in **Appendix B**. From visual inspection, training on A2C [2] does not

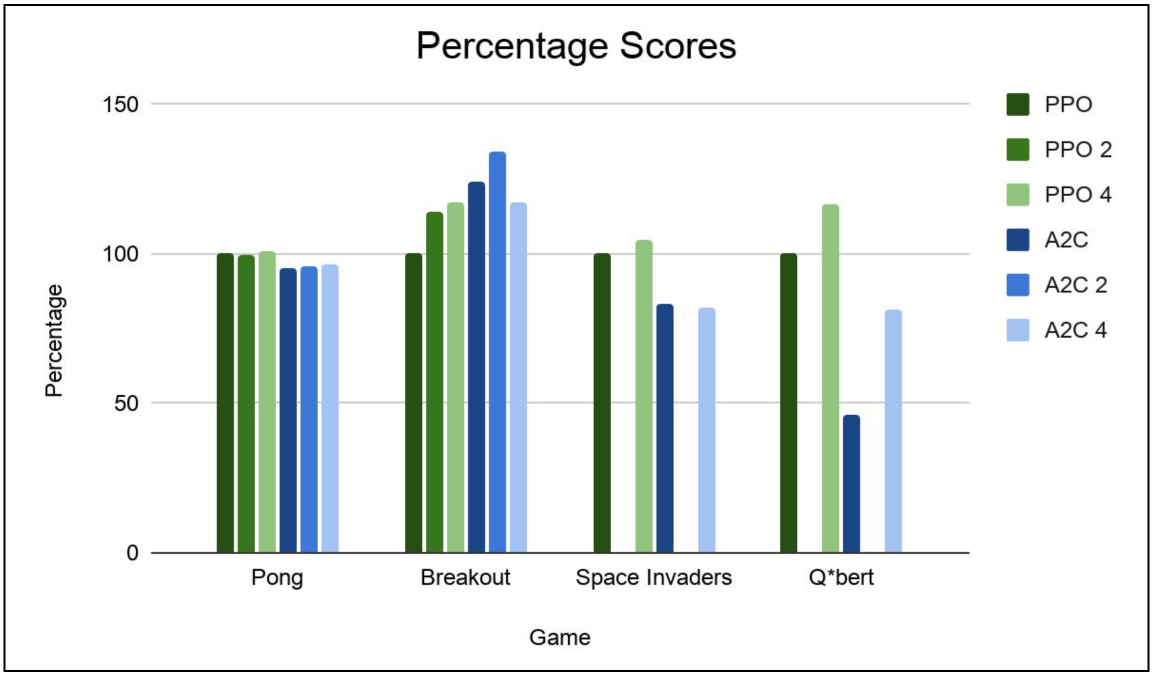


Figure 4: Graph showing the comparison of agents on all games trained on 10 million timesteps per task. They are compared on their percentage score in relation to PPO as a baseline (100%).

seem any less stable than PPO [3]. In fact, in some cases A2C [2] appears to have a more stable increase in performance. So, it may be the case that the benefits of trust region approximation do not carry over strongly to MTL in preventing catastrophic forgetting and may even be a hindrance.

5. Conclusion

This work has shown policy gradient methods such as A2C [2] and PPO [3] can benefit from MTL. These improvements were obtained without the need to alter hyperparameters or algorithmic design. Even though A2C [2] does not have the benefit of a clipped objective to approximate a trust region it still obtained massive gains from the use of MTL. However, overall PPO [3] was still able to outperform A2C [2] in this setting. The use of multiple tasks seems to allow A2C [2] to overcome some of its limitations and fine tune strong policies without the use of a trust region or trust region approximation. With more tasks, both methods were able to perform the stronger compared to their baselines, with nearly a 10% improvement over a single task using PPO[3] on four tasks and a 17% improvement for A2C [2] on four tasks.

The next step for this work is to explore the results with an increased number of tasks to see if performance keeps improving. It was observed in this work that the four task agents performed even stronger than the two task agents. It is possible that the overall scores may continue to increase with a larger number of tasks. This would be straightforward to implement up to 8 and 16 tasks respectively on PPO [3] and A2C [2], but may require some alterations to the algorithm for more tasks, or a change in the hyperparameter for the number of environments. There may also be some benefit to using a deeper convolutional neural network as the number of tasks increase to allow for more fine grained extraction of features from images that may vary greatly from task to task.

Another direction of interest is testing more policy gradient methods, or testing on different task types. Many policy gradient methods perform poorly on ALE [5], however it may be the case these methods benefit from MTL to learn these tasks. Most policy gradient methods are designed for continuous control environments, such as robotics. It is possible these generalized representations from MTL may also benefit policy gradient methods in these other domains. Finally since A2C [2] appeared to be more stable in training than PPO [3], it would be an area of interest to rerun PPO [3] in these experiments without clipping the objective to estimate a trust region. It may be the case that trust region approximation is misleading in MTL and PPO [3] may improve even further by removing the approximation.

References

[1] Mnih, V., Kavukcuoglu, K. and Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 7540 (2015), 529-533. https://doi.org/10.1038/nature14236

[2] Mnih, V., Badia, A. and Mirz, M. et al. Asynchronous methods for deep reinforcement learning. ICML 48, (2016), 1928-1937. https://dl.acm.org/doi/10.5555/3045390.3045594

[3] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. Proximal Policy Optimization Algorithms. *arXiv.org*, 2017. https://arxiv.org/abs/1707.06347.

[4] Zhang, Y. and Yang, Q. A Survey on Multi-Task Learning. *arXiv.org*, 2017. https://arxiv.org/abs/1707.08114.

[5] Bellemare, M., Naddaf, Y., Veness, J. and Bowling, M. The arcade learning environment: an evaluation platform for general agents. Journal of Artificial Intelligence Research 47, (2013), 253-279. https://doi.org/10.1613/jair.3912

[6] Sutton, R., Bach, F. and Barto, A. *Reinforcement Learning*. MIT Press Ltd, Massachusetts, 2018.

[7] Sutton, R., McAllester, D., Singh, S. and Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Papers.nips.cc*, 2000. https://papers.nips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[8] Kingma, D. and Ba, J. Adam: A Method for Stochastic Optimization. *arXiv.org*, 2014. https://arxiv.org/abs/1412.6980.

[9] Hinton, G., Srivastava, N. and Swersky, K. Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch Gradient descent. 2012.

[10] Konda, V. and Tsitsiklis, J. Actor-Critic Algorithms. *Papers.nips.cc*, 2000. https://papers.nips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[11] Schulman, J., Levine, S., Moritz, P., Jordan, M. and Abbeel, P. Trust region policy optimization. ICML 37, (2015), 1889–1897. https://dl.acm.org/doi/10.5555/3045118.3045319

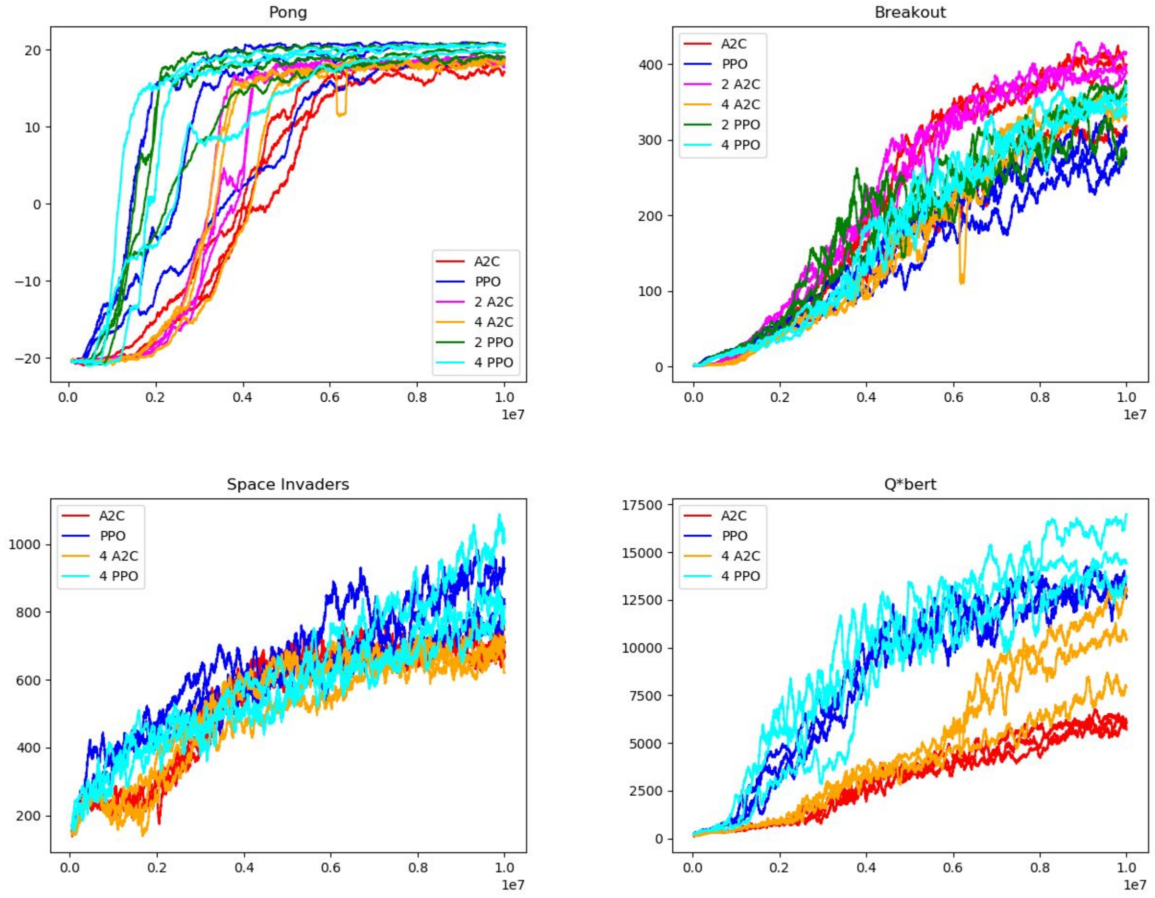
[12] Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv.org*, 2015. https://arxiv.org/abs/1506.02438.

A. Hyperparameters

Hyperparameter	A2C	PPO
Horizon	5	128
Learning Rate	$2.5 \cdot 10^{-4} \cdot \alpha$	$2.5 \cdot 10^{-4} \cdot \alpha$
Epochs	1	4
Minibatch size	-	$32 \cdot 8$
Discount (γ)	.99	.99
GAE parameters (λ)	-	.95
Parallel Environments	16	8
Clipping parameter (ϵ)	-	$0.1 \cdot \alpha$
Value Coefficient (c_1)	0.5	0.5
Entropy Coefficient (c_2)	0.01	0.01

α is annealed linearly from 1 to 0 through the course of training.

B. Additional Graphs



Graphs showing all training runs. The graphs x-axis is the number of training frames on the specified game and the y-axis contains the average score over the last 100 episodes of the game.